Model-Based Design and Adaptive Scheduling of Distributed Real-Time Systems

vom
Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades eines

Doktor der Ingenieurwissenschaften (Dr.-Ing.)

genehmigte Dissertation

von
Ali Abbas Jaffari Syed
geboren in Hyderabad, Pakistan

D 386

Eingereicht am:	25.04.2018
Tag der mündlichen Prüfung:	22.05.2018
Dekan des Fachbereichs:	Prof. Dr.-Ing. Ralph Urbansky
Promotionskomission	
Vorsitzender:	Prof. Dr.-Ing. Wolfgang Kunz
Berichterstattende:	Prof. Dipl.-Ing. Dr. Gerhard Fohler
	Prof. Dr.-Ing. Rolf Ernst

Erklärung gem. § 6 Abs. 3 Promotionsordnung

Ich versichere, dass ich diese Dissertation selbst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt und die aus den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Dissertation wurde weder als Ganzes noch in Teilen als Prüfungsarbeit für eine staatliche oder andere wissenschaftliche Prüfung eingereicht. Es wurde weder diese noch eine andere Abhandlung bei einem anderen Fachbereich oder einer anderen Universität als Dissertation eingereicht.

(Ort, Datum)

(Ali Abbas Jaffari Syed)

Abstract

The complexity of modern real-time systems is increasing day by day. This inevitable rise in complexity predominantly stems from two contradicting requirements, i.e., ever increasing demand for functionality, and required low cost for the final product. The development of modern multi-processors and variety of network protocols and architectures have enabled such a leap in complexity and functionality possible. Albeit, efficient use of these multi-processors and network architectures is still a major problem. Moreover, the software design and its development process needs improvements in order to support rapid-prototyping for ever changing system designs. Therefore, in this dissertation, we provide solutions for different problems faced in the development and deployment process of real-time systems. The contributions presented in this thesis enable efficient utilization of system resources, rapid design & development and component modularity & portability.

In order to ease the certification process, time-triggered computation model is often used in distributed systems. However, time-triggered scheduling is NP-hard, due to which the process of schedule generation for complex large systems becomes convoluted. Large scheduler run-times and low scalability are two major problems with time-triggered scheduling. To solve these problems, we present a modular real-time scheduler based on a novel search-tree pruning technique, which consumes less time (compared to the state-of-the-art) in order to schedule tasks on large distributed time-triggered systems. In order to provide end-to-end guarantees, we also extend our modular scheduler to quickly generate schedules for time-triggered network traffic in large TTEthernet based networks. We evaluate our schedulers on synthetic but practical task-sets and demonstrate that our pruning technique efficiently reduces scheduler run-times and exhibits adequate scalability for future time-triggered distributed systems.

In safety critical systems, the certification process also requires strict isolation between independent components. This isolation is enforced by utilizing resource partitioning approach, where different criticality components execute in different partitions (each temporally and spatially isolated from each other). However, existing partitioning approaches use periodic servers or tasks to service aperiodic activities. This approach leads to utilization loss and potentially leads to large latencies. On the contrary to the periodic approaches, state-of-the-art aperiodic task admission algorithms do not suffer from problems like utilization loss. However, these approaches do not support partitioned

scheduling or mixed-criticality execution environment. To solve this problem, we propose an algorithm for online admission of aperiodic tasks which provides job execution flexibility, jitter control and leads to lower latencies of aperiodic tasks.

For safety critical systems, fault-tolerance is one of the most important requirements. In time-triggered systems, modes are often used to ensure survivability against faults, i.e., when a fault is detected, current system configuration (or mode) is changed such that the overall system performance is either unaffected or degrades gracefully. In literature, it has been asserted that a task-set might be schedulable in individual modes but unschedulable during a mode-change. Moreover, conventional mode-change execution strategies might cause significant delays until the next mode is established. In order to address these issues, in this dissertation, we present an approach for schedulability analysis of mode-changes and propose mode-change delay reduction techniques in distributed system architecture defined by the DREAMS project. We evaluate our approach on an avionics use case and demonstrate that our approach can drastically reduce mode-change delays.

In order to manage increasing system complexity, real-time applications also require new design and development technologies. Other than fulfilling the technical requirements, the main features required from such technologies include modularity and reusability. AUTOSAR is one of these technologies in automotive industry, which defines an open standard for software architecture of a real-time operating system. However, being an industrial standard, the available proprietary tools do not support model extensions and/or new developments by third-parties and, therefore, hinder the software evolution. To solve this problem, we developed an open-source AUTOSAR toolchain which supports application development and code generation for several modules. In order to exhibit the capabilities of our toolchain, we developed two case studies. These case studies demonstrate that our toolchain generates valid artifacts, avoids dirty workarounds and supports application development.

In order to cope with evolving system designs and hardware platforms, rapid-development of scheduling and analysis algorithms is required. In order to ease the process of algorithm development, a number of scheduling and analysis frameworks are proposed in literature. However, these frameworks focus on a specific class of applications and are limited in functionality. In this dissertation, we provide the skeleton of a scheduling and analysis framework for real-time systems. In order to support rapid-development, we also highlight different development components which promote code reuse and component modularity.

TO MY LOVING PARENTS ASIF AND GUL-E-ZEHRA

"The world cannot defeat you,
unless you accept the defeat."

— Ali Ibn-e-Abi Talib

Preface

During the four years of my work as a researcher and a PhD student, I had a chance to learn a lot about different cultures, work ethics and technical skills. This learning experience also helped in developing a research aptitude and presentation skills, which I lacked at the start. During my working period, I had a chance to meet and work with people from all around the world, who helped me expand the horizon of my knowledge as a researcher. At this point, it would not be an exaggeration to say that in the technical world, whatever I know now, I learned it from Technische Universität Kaiserslautern and, more specifically, from the Chair of Real-Time Systems.

At this point, I would like to thank all the people who helped me reach where I am today. First and foremost, I would like to thank Prof. Gerhard Fohler for providing inspiration, be it for research work or for dealing with everyday life. At times, he has been so tough that I thought of giving up. Other times, he has been so patient and supportive that he became an ideal of mine. I believe that this union of yin and yang was necessary for keeping me active and thriving for future. And I am thankful to Prof. Fohler for providing me just that. Other than the work related issues, it has been a pleasure to spend time with him due to his open personality and friendly nature, which is perceived by everyone around him. After having a chance to know about different working environments in Europe, I don't think that I will ever be as close to my boss as I am to Prof. Fohler.

Secondly, I would like to thank Prof. Rolf Ernst and Prof. Wolfgang Kunz for being part of my PhD defense committee and providing valuable feedback which helped improve the quality of the contributions presented in this dissertation.

Moreover, I would like to thank all the people who had never-ending but fruitful discussions with me in order to expand my perspective of thinking and, therefore, to improve the work I presented in this dissertation. Specifically, I would like to thank my former colleagues Stefan Schorr, Mitra Nasri, Raphael Guerra and Jens Theis; and my current colleagues Rodrigo Coelho, Gautam Gala, Kritin Krüger, Florian Heilmann, Steven Dietrich and Ankit Agrawal.

I am also grateful to Stephanie Jung (aka Steffi) for her support in the world of bureaucracy and thankful to Markus Müller (aka Striker) for his support with extra-ordinary technical acumen. Thank you very much Steffi for helping me out with all the official documentation and for explaining German laws, regulations and bureaucracy,

which help me in everyday life in Germany. My utmost respect and thanks go to the Striker and the technical assistant at the Chair of Real-Time Systems for fixing all my computer or lab related disasters.

During my research work, I also supervised many master theses and student HiWis (Hilfswissenschaftler). Many of these students have somewhat contributed to this dissertation in the form of discussions, tools and tutorials. Therefore, I would like to thank John Gamboa, Gokul Vasan, Sireesha Basavaraju, Chao Huo, Habib Ahmed and Melvin Richard John.

Although I had a chance to work in a European project, I could not get an opportunity to thank all the contributing partners. Writing all of their names is impractical here, but I would like to use this opportunity to thank the ones who had a great impact on this dissertation. I would like to thank Daniel Gracia Pérez from Thales Research & Technology (TRT, France), Simon Barner and Alexander Diewald from fortiss GmbH (Germany), Jörn Migge from RealTime-at-Work (RTaW, France), Claire Pagetti from Office national d'études et de recherches aérospatiales (ONERA, France), Prof. Alfons Crespo from Universitat Politècnica de València (Spain) and Prof. Roman Obermeisser from Universität Siegen (Germany).

Other than the project work, I also got an opportunity to collaborate with other people due to their close ties with the Chair of Real-Time Systems. For their help and support, I would like to thank Silviu Craciunas and Ramon Serna Oliver from TTTech GmbH (Austria), Sergey Tverdyshev and Don Kuzhiyelil from SYSGO AG (Germany), Anton Cervin and Prof. Karl-Erik Årzén from Lunds Universitet (Sweden) and Eilert Johansson and Prof. Jakob Axelsson from RISE SICS AB (Sweden).

Last but not least, I would like to thank my family, especially my parents, who have always been and will always be my strength and a source of light in darkness. I appreciate their efforts and struggles for helping me turn into what I am today. I am also thankful to my brother who has always motivated me to be a better person and, especially, for helping and supporting my parents during the time I stayed in Germany. I am also grateful to all my sisters for providing me sisterly love and care, without which my life in Germany would have not been the same.

Ali Abbas Jaffari Syed
Kaiserslautern, Germany

The work presented in this dissertation is partially supported by the European Commission under the European IST-FP7 project *Distributed Real-time Architecture for Mixed Criticality Systems* (DREAMS, FP7-ICT-2013.3.4-610640).

Publications

I have authored or co-authored the following publications:

Book chapters

- Ali Abbas Jaffari Syed, *Distributed Real-Time Architecture for Mixed-Criticality Systems*, chapter *Algorithms and Tools*. In Hamidreza Ahmadian, Roman Obermaisser and Jon Perez, editors, CRC Press, May 2018.

Journal papers

- Ali Abbas Jaffari Syed, Daniel Gracia Pérez and Gerhard Fohler, *Job-Shifting: An Algorithm for Online Admission of Non-Preemptive Aperiodic Tasks in Safety Critical Systems*, Special issue *Real-Time Embedded Systems Design and Analysis (RTESDA)* of Journal of Sysetms Architecture (JSA), 2018.

- Ali Abbas Jaffari Syed and Gerhard Fohler, *Efficient Offline Scheduling of Task-Sets with Complex Constraints on Large Distributed Time-Triggered Systems*, Journal of Real-Time Systems (RTSJ), May 2018.

Conference and refereed workshop papers

- Simon Barner, Alexander Diewald, Jörn Migge, Ali Abbas Jaffari Syed, Gerhard Fohler, Madeleine Faugère and Daniel Gracia Pérez, *DREAMS Toolchain: Model-Driven Engineering of Mixed-Criticality Systems*, ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS), September 2017.

- Ali Abbas Jaffari Syed, Daniel Gracia Pérez and Gerhard Fohler, *Online Admission of Non-Preemptive Aperiodic Mixed-Critical Tasks in Hierarchic Schedules*, 23rd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), August 2017.

- Florian Heilmann, Ali Abbas Jaffari Syed and Gerhard Fohler, *Mode-Changes in COTS Time-Triggered Network Hardware without Online Reconfiguration*, 14th Workshop on Real-Time Networks (RTN) in conjuction with 28th Euromicro International Conference on Real-time Systems (ECRTS), July 2016.

Technical Reports

- Ali Abbas Jaffari Syed and Gerhard Fohler, *Search-Tree Exploration For Scheduling using PIDA**, Chair of Real-Time Systems, Technische Universität Kaiserslautern, August 2014.

Contents

List of Figures

List of Tables

List of Abbreviations

AUTOSAR	AUTomotive Open System ARchitecture
B&B	Branch and Bound (algorithm)
BE	Best-Effort (network traffic)
BSW	Base SoftWare
CC	Cluster Cycle
CM	Compression Master
COTS	Commercial Off-The-Shelf
CPS	Cyber-Physical Systems
DC	Device Configuration (file)
DHP	DREAMS Harmonized Platform
DOM	Document Object Model
DREAMS	Distributed REal-time Architecture for Mixed Criticality Systems (EU FP7 Project)
DS	Device Specification (file)
DTM	Device Target Mapping (file)
ECU	Electronic Control Unit
EDF	Earliest Deadline First (Scheduler)
ES	End-System
ET	Event-Triggered
GRG	Global Reconfiguration Graph
GRM	Global Resource Manager
GUI	Graphical User Interface
HEX	Hexadecimal Binary (file)
HP	Hyper-Period
IC	Integration Cycle
IDA*	Iterative Deepening A*
ISR	Interrupt Service Routine
LCM	Least Common Multiple
LIN	Local Interconnect Network
LRG	Local Reconfiguration Graph
LRM	Local Resource Manager
LRS	Local Resource Scheduler

MAF	Major Application Frame
MCAL	Micro-Controller Abstraction Layer
MDE	Model-Driven Engineering
NC	Network Configuration (file)
ND	Network Description (file)
OEM	Original Equipment Manufacturers
PCF	Protocol Control Frames (for TTEthernet) or Partition Configuration File
PDT	Processor Demand Test
PDU	Protocol Data Unit
PL	Physical Link
PN	Processing Node
RC	Rate-Constrained (network traffic)
RPi	Raspberry Pi
RTE	Run-Time Environment
RTOS	Real-Time Operating System
RTSet	Response-Time Set
RX	Reception
SAS	Symmetry Avoidance Scheduler
SC	Synchronization Client (for TTEthernet) or Scheduling Cycle
SM	Synchronization Master
SMT	Satisfiability Modulo Theories
SOM	System-On-Module
SPI	Serial Peripheral Interface
SWaP	Size, Weight and Power
SWC	SoftWare Component
TG	Task Graph
TGFF	Task Graph For Free (tool)
TGS	Task Graph Scheduler
TT	Time-Triggered
TTEthernet	Time-Triggered Ethernet
TX	Transmition
V&V	Verification and Validation
VFB	Virtual Function Bus
VL	Virtual Link
WCET	Worst-Case Execution Time
WCMCD	Worst-Case Mode-Change Delay
XML	Extensible Markup Language
XSD	XML Schema Definition

Chapter 1

Introduction

To cope with highly dynamic modern life styles and fast growing social networks and economic infrastructures, modern computing systems are in constant need of extension and improvements. Surging environmental changes, which are often result of economic competition and intentional or unintentional damaging activities, is also one of the major driving forces for enhancing technical capabilities in modern societies. In this regard, computing systems play a major role and are required to improve their efficiency while delivering maximum output.

Real-time systems are special type of computing systems where the time of delivery of a result is often just as important as its value. Such systems focus on providing guarantees for delivery of certain services before their deployment. For instance, an avionics system needs to provide guarantees that the aircraft will remain controllable in mid air even in harsh environmental conditions. Of course, the extent to which these guarantees are ensured depends on the application. For instance, a failing multimedia system does not lead to safety hazards. Therefore, a statistical study of system related faults and hazards is performed, which is used to identify verification and validation techniques to evaluate guarantees for a system service, as defined by (often application dependent) industrial standards.

In order to guarantee deterministic and safe operation of real-time systems, the hardware and software components of a real-time system are certified based on their respective design and safety constraints. Considering that the certification process is expensive in terms of time and money, new design technologies based on (meta-)models need to be developed which target at providing modularity, concern separation and ease of certification. Such design technologies should provide uniform and deterministic component behaviour across multiple platforms, in turn supporting portability and code reuse.

In this dissertation, we propose solutions to many problems related to system efficiency and usability, which are faced in the development of real-time systems. These solutions can be employed in several application domains, for instance, health care, railway, avionics and automotive systems. In this regard, we present solutions for both processors and their interconnect based on model driven engineering (where possible), with special focus on certification process and industrial practices. Moreover, we propose optimization approaches in order to maximize or minimize one or more system parameters depending on the available platform or system design.

In this dissertation, before we discuss the problems which we provide solutions to, we need to introduce terminology and common practices for designing a real-time (or cyber-physical) system. Therefore, the rest of this chapter is organized as follows; In Section 1.1, we introduce common terms used in real-time (or cyber-physical) systems. In Section 1.2, we provide a description of problems with real-time systems design which we focus in this dissertation. In Section 1.3, we present a summary of contributions which we propose in this work and, in Section 1.4, we provide an outline of this dissertation.

1.1 Terminology & Background

In this section, we provide a description of the terms which we use throughout this dissertation. We also explain what we understand by certain terms, the meanings of which are either debated amongst the real-time community or vary depending on the application. Moreover, we provide a brief description of TTEthernet and the DREAMS project, which we use for the evaluation of contributions presented in this dissertation.

1.1.1 Hardware Architecture

Computing systems consist of processors, memory, input/output (I/O) modules and other components. A processor consists of data registers which are controlled based on a cyclic clock. By controlled manipulation of data and control commands, a processor can perform data processing and control the operation of the computing system including its components. Furthermore, a memory component is used to store data and program instructions which can be used by the processor. Moreover, I/O components are used to transfer data between the processor and the external environment [SP98], e.g., communication interfaces.

In order to speed-up data processing and save power, a modern processor is built with multiple processing elements, called cores, which process different commands at a single point in time, i.e., multiple cores are capable of executing multiple programs. These individual cores may share other sub-systems (e.g., system bus, shared caches) on the processor and may have dedicated resources (e.g., scratch-pad, local caches). Computing systems built on such processors are termed as multi-core systems. Although there exist various alterations of processor designs in market today (e.g., many-cores), we will focus on single core processors in this dissertation. However, since the modern processors anyway contain more than one core (in order to increase processing speed and save power), we cannot avoid their use during the evaluation process of our contributions. Therefore, we will discuss processor core related issues only when necessary.

Based on the locality of the communication components, a communication sub-system or a network is classified in two types: on-chip and off-chip. As the name suggests, an on-chip network is built on top of the same silicon wafer where the processor resides and often consists of simple elements, e.g., buses. Although bus sub-systems provide necessary bandwidth required for most operations, they become sources of throughput bottleneck for large data traffic and therefore limit system scalability. Due to extreme scalability and flexibility available in switched networks (exhibited by the Internet),

simple switching elements are also used for on-chip communication. On the contrary to an on-chip network, an off-chip network is intended to carry information way beyond the processor wafer.

A distributed system is a collection of independent processors, connected together via an off-chip network, that appear to its users as a single coherent system [TVS07]. The software components in such systems may expose their services to each other and often share information via message passing. This design allows system extensions (e.g., addition of new hardware components), improves component availability and performance. The off-chip network in a distributed system can make use of bus architectures (e.g., TDMA-TTP [KNH+98], CAN [Law97]) or switch architectures (e.g., AFDX [AFD09], TTEthernet [TTT18b]). It is important to mention here that each processor in a distributed system has its own local clock and, therefore, special synchronization services are used when synchronized execution of programs on multiple processors is required. Although the scope of distributed systems is not limited to real-time or cyber-physical systems, we will assume a real-time network as off-chip network in this dissertation and provide network related information when necessary.

1.1.2 Application Model

In computing systems, an application or a task represents an execution instance of a piece of code on a target platform. An application performs the transformation of input data to the output data. In order to exploit modern platforms or to execute synchronized activities, an application may create a number of sub-tasks. The sub-tasks of an application/task represent execution of separate pieces of code (possibly) communicating with each other to achieve a common goal. These communication mechanisms may be blackboards, scratch-pads, message passing, etc. The sub-tasks of an application can be implemented using either a thread or a process [SP98]. The implementation of a sub-task using a thread or a process is subjective [Bra11], therefore in this dissertation, we will use these terms only when required.

The main characteristic of real-time systems, which separates them from conventional computing systems, is the satisfaction of timing constraints. Consequently, an application or a task in real-time systems defines a relationship between the code and its execution in time. As it is difficult to capture exact execution patterns in all possible application scenarios and processor states, task models are used to abstract the timing information. In the following, we provide a brief description of common task models and application properties like criticality and mode, which we use throughout this dissertation.

Task Models

The most common task model used in real-time systems design is the periodic task model [LL73]. The periodic task model defines a task as a workload which needs to be handled periodically. Each instance of this periodic workload is termed as a *job* and all infinite jobs of a task feature the same known upper bound on their execution time, which is termed as *worst-case execution time* (WCET). The point in time when a job

can be started is termed as the job *release* time, while the point in time until when a job must finish the workload is termed as *absolute deadline*. When the deadline is specified relative to the release time of a job, the deadline is termed as *relative deadline* and is associated with a task (i.e., all jobs of the task have the same relative deadline). The periodic task model defines the temporal difference between two consecutive releases of jobs to be equal to the task period. In literature, the term *task utilization* is used to refer to the ratio WCET/period (which identifies periodic load generated by the task on the system) and the term *hyper-period* is used to represent the least common multiple (LCM) of all task periods in the system.

Based on the relationship between timely parameters of a periodic task, different variants of the periodic task model are available in literature. For instance, when the relative deadline of a task is equal to its period, the task is said to have an *implicit* deadline. Similarly, when the relative deadline of a task is less or equal to its period, the task is said to have a *constrained* deadline and, when the deadline can be less, equal or greater than its period, the task is said to have *arbitrary* deadlines. Moreover, another extension of the periodic task model found in literature assumes that the first job of the task is released with non-zero offset known as *release phase*.

The sporadic task model [Mok83] is another common model based on the periodic task model. This model relaxes the strictly periodic release of jobs in periodic task model by defining the term *minimum inter-arrival time* which represents the lower bound on releases of two consecutive sporadic jobs. Similarly, an aperiodic task model removes all temporal restrictions on releases of consecutive jobs and is used to model workloads in order to process rare events. There exist several other task models in literature (e.g., adaptive variable-rate (AVR) task model [ABB16], recurring task model [Bar03], timed automata [FKPY07]), however, these models are often used for specific applications and require special measures for scheduling.

Depending on the points in time when a job can be interrupted, real-time tasks can be classified in three categories: non-preemptive, limited-preemptive and fully-preemptive. Preemption refers to the fact that a task is temporarily interrupted with the intention of resuming it at a later point in time. For instance, non-preemptive tasks cannot be interrupted and must finish once started, while limited-preemptive tasks can be interrupted only at predetermined or predefined points in time [NNF16].

In the past, there has been a debate in real-time community regarding the hard, firm and soft classification of real-time tasks, which is based on data delivery time and its consequence. Therefore, different communities tried defining these classifications in order to standardize them. For instance, IEEE TCRTS community defined (i) hard real-time task as *a task for which missing the timing constraint of any of its job may jeopardize the correct behavior of the entire system*, (ii) firm real-time task as *a task for which missing a timing constraint does not jeopardize the correct system behavior but it is completely useless for the system*, and (iii) soft real-time task as *a task for which missing a timing constraint does not jeopardize the correct system behavior and has still a reduced value for the system*. However, these definitions are only popular among a group of academic researchers and might not represent industrial task models. For instance, the automotive engine control task is defined as a hard real-time task (representing the

importance of its result) but it often misses its deadline and the processor is restarted to save the engine from jeopardizing the system behaviour[1]. These classifications are also shadowed by the notion of a time utility function (TUF) [JLT85]. The TUFs define the significance of the result of a task computation as a function of time. In this dissertation, we will use the firm class of tasks (as defined by IEEE TCRTS community, termed as *hard deadline process* by Jensen et al. [JLT85]), unless stated otherwise.

Criticality

In 2007, a theoretical mixed-criticality task model was proposed by Vestal and, therefore, termed as Vestal('s) model [Ves07]. The Vestal model extends the periodic task model by defining a criticality level π and a list (of length π) of worst-case execution times (WCETs) for each task. Each WCET of a task in Vestal model is calculated/estimated with different levels of pessimism, where the WCET is often considered to be increasing monotonically with increasing criticality level. The list of WCETs of a task is utilized online to increase the criticality level γ of the system, i.e., the larger the WCET, the higher the system criticality γ. When the system criticality γ is increased online, the tasks with lower criticality level (i.e., $\pi_i < \gamma$) are dropped to accommodate pessimistic WCET of high criticality tasks (i.e., $\pi_i \geq \gamma$). Upon reaching a certain online state, the system criticality level γ is restored to a lower value. A number of extensions of the Vestal model have been proposed, which took the concept of variation in a task parameter based on criticality (e.g., WCETs in Vestal model) to task period and relative deadline [BB11, Bar12].

In the industrial mixed-criticality task model, a task/component only has a single WCET and its criticality level defines the level of assurance applied during the development of the application (IEC61508 [IEC10], DO-178C [DC11], ISO26262 [ISO11]). In industrial model, the criticality level of a task/component is defined based on the *severity* of the associated failure, the *probability* of occurrence of the failure and *controllability* of the faulty task/component [ISO11]. In general, the tasks/components with least severity and probability of failure are defined as non-critical.

There have been several debates in real-time community asserting that the Vestal mixed-criticality task model [Ves07] is not a practical model, e.g., [GB13, ENNT15, EN16]. The reasons for this assertion are manifold. For instance, it is impractical to employ various WCET analysis techniques when the certification authority only accepts one, a higher task criticality does not mean greater importance (compared to lower criticality tasks) and, therefore, does not warrant rejection of lower criticality tasks. Considering these important issues, we use the industrial mixed-criticality model in this dissertation.

In the industrial mixed-criticality task model, a component is not supposed to have an impact on the execution of other components. This indicates that the task criticality designation, as per industrial standards, cannot be exploited by the scheduler as long

[1]Example given by Arne Hamann from Bosch GmbH during discussion on the keynote presentation by Rob Davis in International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS), 2016. See `https://www.ecrts.org/forum/viewtopic.php?f=26&t=79`.

as critical and non-critical tasks remain temporally and spatially segregated. Usually, non-critical tasks are complex tasks (for instance, multimedia [SKF+11], java virtual machine [NKA14], etc.), which handle a large amount of data and consume a large number of resources. When critical tasks execute alongside non-critical tasks without temporal and spatial isolation, (i) the non-critical tasks need to be certified at the highest criticality level of the non-isolated tasks [ENNT15] and (ii) the critical tasks become vulnerable to attacks and, perhaps, compromise system operation. Therefore, executing critical and non-critical tasks without segregation/isolation is not recommended by industrial standards.

On multi-core processors, the task of ensuring isolation between different criticality tasks becomes challenging since these processors can execute multiple tasks (potentially from different criticality levels) in parallel. In other words, executing multiple tasks at a single point in time may increase WCET of some or all tasks due to inter-core interferences. Such an execution configuration can inflate the WCET of a (potentially critical) task manifolds [SCM+14] and therefore may lead to a deadline miss. On the contrary, by restricting the execution of single criticality tasks at a given point in time (e.g., by time-triggered scheduling), the temporal isolation between tasks increases but this execution configuration leads to significant utilization loss. In safety critical systems, the execution of tasks, belonging to different criticalities, on different cores of a platform at a single point in time is avoided (to enforce isolation), however, the trend is gradually changing [Man17].

Modes

Modern real-time systems require different sets of tasks to be executed during different phases of operation. For instance, an avionics control system needs to execute different sets of applications during take-off and auto-pilot. Moreover, an aircraft may also define different flight plans for different routes or destinations, for instance, emergency landing plans. When the system is designed without the knowledge of these (often) mutually exclusive operation phases, high priority tasks may suffer significant jitter and large latencies, which might (potentially) deem the system unfeasible. Moreover with such an all-in-one design, any minor change in the system design (addition/modification of tasks/constraints) requires the whole (sub-)system to be rescheduled. On the contrary, when a system is scheduled based on the information of these operation phases, the system can perform better in terms of task response-times, jitter and latencies. Moreover with such an approach, additional functionality in a single mode can be supported as the system utilization for a given mode would be significantly smaller compared to an all-in-one design approach.

In real-time systems, a *mode* is used to implement an explicit operation phase of a system. Consequently, a mode identifies a set of tasks which needs to be feasibly scheduled by the system during a distinct operation phase. In 1994, Fohler [Foh94] used time-triggered (TT) schedules for each mode to handle mutually exclusive phases of operations, which account for large changes in the environment or in the system itself, as opposed to small changes which are handled by the online scheduler. Similarly, Kopetz et al. [KNH+98] used TT mode schedules to account for different phases of

operations in TDMA-TTP based networks. On the contrary, Real et al. [RSC16] used modes to handle system overload/emergency situations. In summary, there exists no globally defined purpose and implementation of real-time *modes* and, therefore, Graydon et al. [GB13] classified them based on their use. In the following, we summarize the classification of modes presented by Graydon et al., which we will use throughout this dissertation:

Operation Modes [GB13]: An operation mode defines a system phase in which the system is being used by its operators (e.g., take-off, auto-pilot, landing) including emergency situations.

Design Modes [GB13]: The modes in which the system is being designed or maintained are termed as design modes (e.g., normal, maintenance, firmware debugging, software debugging).

Service Modes [GB13]: A service mode defines a system phase in which the system is reconfigured for either survivability, after a loss of resource (e.g., core failure, deadline missed), or improving efficiency (e.g., processor sleep modes, frequency scaling modes).

Mode-Change

When a system uses multiple modes to represent different operation phases, an online trigger or transition is required to switch between them. These switching events/requests are released at an arbitrary point in time by the environment or the schedulable entities. Once a switching request is detected by the system scheduler, the system undergoes a transition from a source mode to a destination mode. This gradual transition process between modes requires time. In literature [RSC16, Foh93, KNH+98], this transition process is termed as a mode-change, while the release of a switching event is termed as a mode-change request. For instance, an avionics system can generate a mode-change request upon detection of a resource failure, which consequently leads to a mode-change in order to circumvent drastic effects on aircraft dynamics.

In order to define how ongoing activities in a system are handled during a mode-change, Jahanian et al. [JLM88] identified three types of mode-changes which can be triggered upon detection of a mode-change request, i.e., (i) all ongoing activities need to finish before executing a mode-change, (ii) all ongoing activities need to abort before executing a mode-change (termed as *immediate* mode-change by Fohler [Foh94]), and (iii) some ongoing activities need to finish, but not all (implemented with two segregated *immediate* mode-changes by Fohler [Foh94]). For distributed time-triggered (TT) systems, Kopetz et al. [KNH+98] declared a mode-change *deferrable* when all system nodes change their modes at the end of the hyper-period, while they declared a mode-change *immediate* when only one system node changes mode soon after the release of a mode-change request, i.e., before the end of the hyper-period. They defined consistency as the motivation for deferrable/deferred mode-changes and speed for immediate mode-changes.

Kopetz et al. [KNH+98] and Fohler [Foh94] pointed out that an immediate mode-change may lead to consistency problems and therefore a number of tasks or messages must be aborted but should not go in an undefined state. In order to solve these consistency problems, Fohler proposed task graph-based scheduling approaches for dis-

tributed TT systems. Note that the notion of an immediate mode-change used by Kopetz et al. [KNH+98] is quite different from the notion used by Fohler [Foh94]. An immediate mode-change, as defined by Kopetz et al., can occur at any point in time and always starts the new TT mode schedule from the beginning (i.e., offset 0). This type of an immediate mode-change requires careful application design and leads to a strong codependency between the application and the scheduler. However, an immediate mode-change, as proposed by Fohler [Foh94], can only occur at predefined points in time and always executes the new mode from the same offset as in currently executing mode. This notion of a mode-change decouples the application from the scheduler and hence is more suited for complex systems.

In distributed TT systems, when a deferrable mode-change request leads to a mode-change by all nodes only at safe points in time (e.g., the end of the hyper-period for task-sets with implicit deadlines), data inconsistency issues are not encountered. However, as pointed out, an immediate mode-change request may lead to inconsistency issues since its scope is limited to a single node and it can be released at any point in time. In order to solve this issue, the execution of an immediate mode-change is restricted in time. In this regard, two different approaches can be observed in literature, i.e., mode-change blackout slots [Foh93] and mode-change partition slots [RSC16].

Fohler [Foh93] defined an annotation for each TT slot (further explained in Section 1.1.3) of a scheduling table, which he called mode-change blackout. This annotation is calculated offline based on the state of the task and message queues at a specific point in time. According to Fohler, an immediate mode-change can only be executed online when the currently executing slot in the scheduling table is not a mode-change blackout slot. If the slot is indeed a blackout slot, the mode-change is postponed until the end of the mode-change blackout. Note that in this approach, the online scheduler or dispatcher is responsible for executing the mode-change.

Recently, Real et al. [RSC16] defined mode-change partition slots (further explained in Section 1.1.3) which are generated by the offline scheduler at regular intervals. The offline scheduler also makes sure that, during the mode-change partition slot, no executing task or pending message leads to inconsistency issues. According to Real et al., during the online execution of a mode-change partition slot, a task checks for a pending immediate mode-change request and executes it when required.

1.1.3 Software Design & Implementation

In this section, we will describe foundations of real-time systems design, which are necessary for understanding the contributions of this dissertation. Moreover, we will also provide an introduction to common real-time design practices and discuss their impact on the capabilities of the system.

Time-Triggered VS Event-Triggered

A system in which all activities are initiated as a consequence of the occurrence of a significant event is called an event-triggered (ET) system [Kop91]. An ET system takes immediate action upon detection of a significant state change, e.g., user action,

message detection, task release, etc. Since the upcoming event is not known before its occurrence and the occurred event needs to be managed quickly (i.e., registered and queued for later processing) to avoid missing future events, an ET system utilizes simple scheduling approaches which require small run-time. As a consequence, complex constraints like latency, jitter and data age are usually not controlled in an ET system. Instead, worst-case bounds on application latency and jitter are estimated before system deployment and compared against the constraints. Due to simple scheduling approaches used in ET systems, they can adapt to changes in the workload and they are capable of handling heterogeneous task models (see Section 1.1.2).

On the contrary to an ET system, a system in which all activities are initiated by the progression of time is called a time-triggered (TT) system [Kop91]. Consequently, a TT system requires all run-time activities to be known before system deployment. Since the activation of sporadic and aperiodic activities cannot be known with certainty, TT systems assume all activities to be periodic. During the offline phase (i.e., pre-deployment development phase), a table or schedule of all known (deterministic) activities is created, which defines when an activity needs to be initiated. The generated schedule is then verified and validated against system constraints and, if valid, the schedule is stored in the system nodes (i.e., processors and network interfaces) for online execution (i.e., post-deployment execution phase). Since storing the schedule or scheduling table for the complete lifetime of the system is impractical, a schedule for a comparatively small portion of the system lifetime is stored in system nodes, which is repeated over and over for the complete system lifetime. For instance, the length of the scheduling table for a task-set, consisting of periodic tasks with implicit deadlines, is the least common multiple (LCM) of the tasks' periods, since after the duration of LCM units, the job release pattern repeats itself. Since TT systems require every activity to be planned before system deployment, TT systems require adaptation techniques in order to accommodate changes in the system workload or hardware components. However, since every activity is planned offline, complex constraints (e.g., latency, jitter) can be handled and validated.

In literature, two distinct variants of the implementation of TT systems can be found; TT with dense time-base and TT with sparse time-base [Kop92]. During the offline schedule generation process, if no assumption is made on the granularity of scheduling quantum, the TT system is said to have a dense time-base. Contrarily, if the schedule generation process assumes a minimum granule of scheduling quantum (termed as *slot* by Fohler [Foh94] and *macrotick* by Steiner [Ste10] and Craciunas et al. [CO16]), the TT system is said to have a sparse time-base. For digital computing systems, the sparse time-base is implemented by the scheduler using a scheduling quantum significantly larger than the processor clock cycle length, i.e., the scheduler can be invoked at the start/end of each scheduling quantum. However, when the scheduler does not enforce a scheduling quantum, the system is subjected to dense time-base, where the scheduling quantum is equal to the processor clock cycle length.

In 1992, Kopetz [Kop92] showed that nodes of a distributed TT system (i.e., processors and network interfaces) need to synchronize their local clocks to establish a global clock with defined precision and that sparse time-base is required to establish global

causal order of events. Although clock synchronization is essential for any distributed TT system, the global causal order of events is apparently not important for industry [AUT16a, ARI03]. From the perspective of industrial practice, local causal order is enough for servicing an event. For this purpose, the synchronization semantics in industry define that a TT scheduled component is held responsible for detecting an event. Upon online execution of this component, as per TT schedule, if the event is detected, the component services it. Note that the component, which services the event, can either be a task (e.g., a periodic task in cyclic executive [ARI03]) or the online dispatcher (e.g., scheduler in slot shifting [Foh94]). The effect of this synchronization semantics on the temporal behaviour of an application (e.g., latency, jitter, data-age) is then evaluated offline by timing analysis tools [HHR+04, BDM+16].

Flat VS Partitioned Design

In legacy avionics systems, dedicated distributed hardware components were required to execute individual software components [WW07]. In such a design (termed as Federated architecture in avionics), components executing on different processing nodes had no or little influence on each other as dedicated resources were available for each software component. However, over the last few decades, avionics systems are designed to execute multiple software components on the same hardware platform in order to reduce size, weight and power (SWaP) of the final product. When multiple software components execute on the same platform without temporal and spatial isolation (termed as *flat* design in literature, e.g., [BMR+10]), execution of some or all components can be delayed since a large number of platform resources are now shared. In order to keep unrelated components isolated from each other, hardware resources (i.e., processor time and address space) are reserved or statically partitioned among software components. This design based on reservation of resources provides guarantees that the reserved resources are always available to the software component. This design is called *partitioned* or *hierarchical* design, which is considered an important part of integrated modular avionics (IMA) architecture in avionics. Due to its improved isolation capabilities and compositional nature, the partitioned design is now used in several other domains, e.g., automotive, networking.

In a partitioned design, a guaranteed set of partitioned resources, which can be used in isolation by a software component, is termed as a partition or a virtual processor. The software which is responsible for creating and running these partitions on a platform is called a hypervisor. A hypervisor enforces temporal and spatial isolation between different partitions and manages platform resources. A hypervisor can be one of two types, which are called *type 1* and *type 2*. A type 1 hypervisor (often termed as native or bare-metal hypervisor) runs on bare-metal and uses a micro-kernel on top of which partitions and their hosted software are executed. Examples of type 1 hypervisors include Xen [BDF+03] and PikeOS [SYS17]. A type 2 hypervisor (often termed as hosted hypervisor) executes partitions on top of a fully-fledged operating system (OS). KVM[2] is an example of a type 2 hypervisor.

[2] `https://www.linux-kvm.org/`

To enforce temporal isolation between software components or partitions, the hypervisor scheduling problem can be seen as a two-tier problem, where tier-1 is responsible for scheduling *partitions* and tier-2 is responsible for scheduling *tasks* inside the partitions. Since partitions are isolated from each other, each partition contains its own task scheduler. Depending on the requirement and the analysis technique, the partition scheduler (i.e., tier-1 scheduler) can be implemented using either online priority-based or offline time-based (or TT) scheduling. Both approaches have their advantages and disadvantages. In general, online priority-based scheduling (e.g., periodic servers) leads to loss of processor utilization [LWVH12], requires compositional task analysis but is rather simple to implement. On the contrary, offline time-based scheduling is easy to analyze and certify but it is difficult to implement (due to synchronization issues). When offline time-based scheduling approaches are used, a contiguous interval of processor time, which is reserved for a particular software component or partition, is termed as a *partition slot* [BMR+10]. Note that the response-times of the tasks (i.e., schedulable components for tier-2 scheduler) depend on both tier-1 and tier-2 scheduling strategies.

In order to provide spatial isolation between components, the memory address ranges accessible by multiple partitions are kept disjoint or non-overlapping (managed by hypervisor memory management services), even when multiple partitions access memory from a common memory chip. Note that in modern processor architectures, spatial isolation also affects temporal behaviour of the applications due to a shared system bus. Therefore, scratch-pad memories (i.e., core-local memory which does not use the system bus) are preferred for safety critical software design [MGUU11, PP07].

An important point to mention here is that the allocation or the task mapping problem (i.e., which task executes on which processor/core) in hierarchical systems is also seen as a two-tier problem, where tier-1 is responsible for allocating tasks to partitions and tier-2 is responsible for allocating partitions to processors/cores. In the safety critical domain, multiple tasks which achieve a single goal form a complete software component [ISO11] and therefore a component should be isolated from other components. As a result, a common practice in safety critical design is to map each component to a partition [BDM+17]. This approach simplifies the tier-1 allocation problem but leaves the tier-2 allocation problem open. Note that in safety critical applications, dynamic task migration from one processor/core to the other is not recommended due to incurred migration delays [Sar12, Sch15].

Real-Time Scheduling Algorithms

There exists a large variety of real-time scheduling algorithms in literature. Based on the activation paradigm, the scheduling algorithms can be classified in online priority-based algorithms for event-triggered (ET) systems and offline time-based algorithms for time-triggered (TT) systems. Since in time-based scheduling algorithms, there exists no (explicit) task priority and such algorithms cannot be specified with simple rules (due to inclusion of search algorithms and optimizations, for instance), here we only discuss priority-based algorithms.

As the name suggests, priority-based scheduling algorithms require a priority level associated with each task/job. Based on the priority level of ready-to-execute tasks/jobs,

these algorithms decide which task/job to execute next (i.e., the task/job with the highest priority executes next). Priority-based algorithms can further be classified in fixed priority and dynamic priority algorithms. In fixed (task) priority scheduling (FPS), a task and all of its jobs are assigned a fixed (same) priority level which does not change at run-time. Rate-monotonic (RM) [LL73] algorithm is an example of FPS algorithms, where the task priority is directly proportional to its rate of execution (or inversely proportional to its period). On the contrary, if each job of a task can have different (but fixed) priority level at run-time, the scheduling algorithm is termed as dynamic (task) priority algorithm. Earliest deadline first (EDF) is an example of dynamic priority algorithms, where the priority of a job is inversely proportional to its absolute deadline.

A set of hard or firm real-time tasks is said to be *feasible*, if there exists an algorithm which can guarantee that none of the jobs of these tasks can lead to a deadline miss (i.e., have pending workload even after passing the absolute deadline). On the contrary, a hard or firm task-set is said to be *schedulable with respect to algorithm A*, if algorithm A can guarantee that none of the jobs of this task-set will lead to a deadline miss. In order to give guarantees whether a task can be scheduled by an algorithm, analysis techniques peculiar to the scheduling algorithm are utilized. For instance, the processor demand test [BRH90] is utilized to guarantee schedulability of an implicit deadline task-set with respect to EDF algorithm.

Over the last few decades, a number of attempts have been made to merge priority-based online algorithms with time-based offline algorithms (or TT with ET) to exploit benefits of both design approaches. For instance, the slot shifting algorithm [Foh94] executes an EDF scheduler online and employs an annotated scheduling table to provide online admission test for aperiodic tasks.

1.1.4 Meta-Models

In general, a model is an abstract representation of a real object. For example, a geographical map represents the locations of interest and routes between them. Such a map can be used to find a path from one place to the other. An important point to mention here is that the details shown in the model are only relevant for the intended use of the model. In the context of model-driven engineering (MDE), a model defines artifacts and their relationships relevant to a real object being modeled.

A meta-model is defined as a model of a model. A meta-model can be used to validate a model or build automated tools to facilitate design and development of a product. Although the scope of the term *meta-model* is not limited, we will restrict our understanding of this term in this dissertation to MDE.

From the perspective of MDE, a meta-model defines a number of entities, their properties, their relationships with other entities and their constraints. For instance, consider the following periodic task meta-model: an entity *periodic task* has a property *period* and contains other entities, called *jobs*, which have a property *release time*, while there exist a constraint that *exactly one job must be released during the task period*. From such a meta-model, an instance or a model of a periodic task can be constructed (not to be confused with the periodic task model in real-time systems defined by Liu and Lay-

land [LL73]). For instance, a task T_1 with period 5 units and jobs j_1 with release time 0, j_2 with release time 5, and so on. Given such a model of a periodic task, the periodic task meta-model can be used to validate the model. For instance, we can validate that each job of the task T_1 is indeed released exclusively during 5 units of time and each period of 5 units has a released job.

In the context of real-time systems, meta-models are used to automate or simplify artifact generation for testing, evaluation, validation and deployment of real-time components. In order to generate artifacts, models can be transformed to acquire other (possibly more detailed) model(s). This way, the model transformation process reduces errors and simplifies artifact validation process. For instance, time-triggered scheduling can be considered a model transformation process which converts input models (e.g., platform model, task model) into a scheduling table model.

1.1.5 TTEthernet

Time-Triggered Ethernet (TTEthernet) [TTT18b] is a multi-speed bi-directional multi-hop switched Ethernet network. It is employed in safety critical systems (e.g., NASA Orion Multi-Purpose Crew Vehicle [TTT17]) in order to deliver mixed-critical traffic to and from distributed nodes. TTEthernet supports three traffic classes for mixed-criticality applications: Time-Triggered (TT), Rate-Constrained (RC) and Best-Effort (BE). To support TT traffic, all network devices (i.e., processing nodes and switches) are synchronized by a fault-tolerant clock synchronization protocol. Contrary to TT traffic, RC traffic does not require synchronization but requires offline analysis methods, e.g., Network Calculus, to verify the timing behaviour and validate frame latency constraints. Furthermore, non-critical BE traffic is transmitted when a network link is not used by TT or RC traffic.

The TTEthernet network consists of TTEthernet End-Systems (ES) or processing nodes connected via TTEthernet switches through physical links. Similar to other Ethernet based networks, TTEthernet utilizes Ethernet frames to transport the data payload. In TTEthernet, the network bandwidth for a TT or an RC frame is reserved by using the concept of a Virtual Link (VL). A VL is defined by a period (or bandwidth allocation gap), message size, a source node, a set of destination nodes and a route through the physical links. The set of destination nodes capture the multicast communication paradigm. An example TTEthernet network is shown in Figure 1.1 with two switches and six end-systems. In the figure, the arrows indicate the direction and the route of a VL.

In TTEthernet, the TT frame transmission takes place based on an offline generated TT scheduling table. For each VL, the scheduling table defines scheduling windows, i.e., time intervals during which a TT VL frame uses a physical network link. Based on these scheduling windows, the online dispatcher for the network port sends out-going TT frames (when ready) and filters in-coming TT frames, i.e., discards them if not received during the defined time interval.

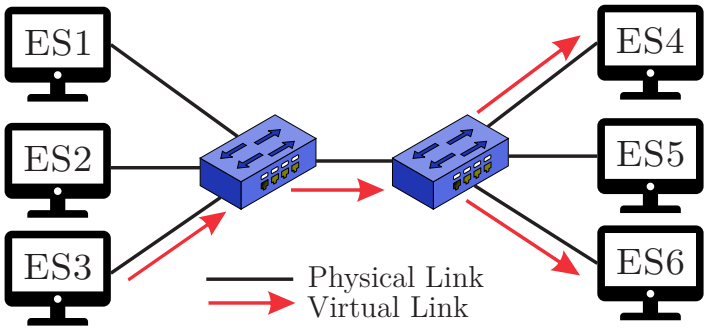


Figure 1.1: *A TTEthernet Network*

Design Methodology

Figure 1.2 shows the design methodology employed by TTEthernet networks. In the figure, crosshatched thick arrows represent processes achieved using the TTEthernet toolchain (discussed in the next section), black thin arrows indicate the data flow and boxes represent the files (or collections of files). In TTEthernet model-based design approach, six types of model files are required to define the TTEthernet network at different stages of the development process. The Network Description (ND) file captures the high level network aspects, such as topology, virtual links (VL), synchronization settings, etc. This ND file is used to schedule the network. After network scheduling, a set of files is generated. The Network Configuration (NC) file defines the low level network aspects such as devices, physical/virtual links, synchronization parameters, etc. The Device Target Mapping (DTM) file provides a mapping between the devices in ND and the physical devices. The Device Specification (DS) file defines the time-triggered (TT) schedule and the VL routes for the (egress or ingress) messages from a specific device in the network. From the NC, DTM and the DS files, the Device Configuration (DC) files for the individual devices (i.e., processing nodes and switches) in the network are generated which can then be used to build equivalent hexadecimal binary (HEX) images. Each built HEX file can be loaded to a TTEthernet device.

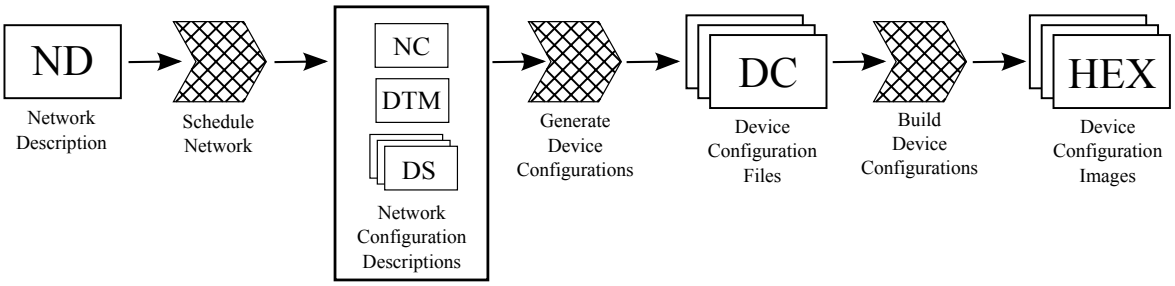


Figure 1.2: *TTEthernet design methodology*

TTEthernet Toolchain

The TTEthernet toolchain is used to accomplish design processes shown in Figure 1.2. The toolchain consists of four tools, namely the TTEthernet Eclipse [EF18b] plugin, TTPlan, TTBuild and TTLoad. The TTEthernet Eclipse plugin is used to create the network description (ND) file either from scratch or from a preconfigured template. Once the ND file is generated, a command-line tool called TTPlan is invoked. The tool TTPlan is responsible for (i) generating TT schedule, virtual link routes and synchronization parameters (in other words, converting ND files to NC, DTM and DS files), and (ii) generating DC files. The tool TTBuild converts the DC file of each TTEthernet device to an equivalent HEX file, while the tool TTLoad (or the TTEthernet Eclipse plugin) transfers the generated HEX files to the TTEthernet devices.

1.1.6 DREAMS Project

The European FP7 project *Distributed REal-time Architecture for Mixed-Criticality Systems* (DREAMS) aims at developing cross-domain architecture and design tools for complex networked systems. Such an architecture can be used to provide support for application sub-systems from different criticalities executing on distributed network with multi-core processor nodes. The DREAMS project provides architectural concepts, model-driven development technologies, tools, adaptation strategies and certification methodology for modern safety critical applications. These services enable seamless development and integration of mixed-criticality applications with safe and secure real-time performance.

The distributed system architecture in DREAMS is physically structured into a set of *clusters*, where each cluster consists of processing *nodes* that are interconnected via an off-chip network. Inter-cluster gateways serve as the communication connection between clusters. Each *node* in a cluster is a multi-core processor containing *tiles* which are interconnected by an on-chip network. A *tile* in a node can be a processor with several cores, caches, local memories and I/O resources or it can be a single core processor or an IP core (e.g., memory controller that is accessible using the on-chip network and shared between several tiles). Processor *cores* within a tile can run a hypervisor that creates and maintains partitions, each of which executes a corresponding software component [DRE14].

As porting process of a service component on a specific hardware platform requires substantial efforts, the scope of the DREAMS project was restricted to a number of platforms. In this dissertation, we will make use of only two platforms; DHP and T4240. The DREAMS Harmonized Platform (DHP) is a heterogeneous multi-core platform developed on top of the Xilinx Zynq®-7000 All Programmable SoC ZC706 board. The Zynq platform provides a dual-core ARM Cortex-A9 processor running at 400Mhz. The DHP design extends the ARM processor with 3 MicroBlaze cores. Different cores on the DHP are interconnected through a dedicated on-chip network designed to provide support for time critical communication. Moreover, the NXP QorIQ® T4240 is a multi-core platform which provides 12 PowerPC e6500 cores running at 1.8GHz. The T4240 platform is extended with a TTEthernet Peripheral Component Interconnect (PCI) card

to allow time critical communication between the platform and the rest of the network.

In order to provide a cross-domain temporal and spatial partitioning environment for DREAMS based mixed-critical applications, the XtratuM hypervisor [CRM+09] was ported to run on top of ARM, PowerPC and Intel platforms. For mixed-critical application design, the DREAMS project uses industrial mixed-criticality task model [IEC10, DC11, ISO11] but allows dropping of non-critical tasks in favor of competing critical tasks (executing on different cores at the same time). When a critical task inside a partition (is about to) miss a deadline, the non-critical partitions are interrupted. The execution of the interrupted partitions continues after the partition slot has finished.

Resource Management

To support execution of mixed-criticality applications, cross-domain resource management services were developed which were implemented using an abstraction layer between the XtratuM hypervisor and the user applications. This layer also provides services like online system reconfiguration [DFG+16] and secure communication [KGGP+16]. To provide fault-tolerance and deadline-overrun management, three resource management system partitions were developed: the global resource manager (GRM), the local resource manager (LRM) and the monitor (MON). Each processing *node* in the system executes at least one LRM and MON, while the GRM component runs on a single isolated node. In order to ensure fault-tolerance against core failures, the resource management components allow changing the mode schedule of certain node(s), such that the applications from the failed core are then handled by a working core. As the name suggests, the GRM software (executing in the GRM partition) is responsible for changing system wide modes, i.e., they affect two or more system nodes. On the contrary, an LRM is only responsible for changing resource local modes (for deadline-overrun handling and mode-changes) and informing GRM when the local reconfigraution is not enough. The MON component is responsible for monitoring the resource for ill-behaved execution and failures, and reporting LRM when they happen. Additionally, the user partitions (on which the applications are deployed) implement a local resource scheduler (LRS), which provides the applications with a cyclic executive [ARI03] intra-partition scheduler (CEIPS). A graphical representation of the hierarchy of resource management components and their communication interfaces is shown in Figure 1.3, where white boxes represent resource management components, gray boxes represent resources, solid arrows represent communication between resource management components and dashed arrows indicate monitoring or controlling actions.

DREAMS Toolchain and Design Methodology

In order to ease the deployment process and support product updates, the DREAMS project provides a toolchain [BDM+17] for industrial mixed-criticality applications. In this dissertation, we only provide a brief description of the relevant parts of the DREAMS toolchain and introduce only relevant tools. A detailed description of the

complete toolchain can be found in [BDM+17].

Figure 1.4 shows an abridged version of the DREAMS toolchain with its design methodology, where the boxes represent models, tools or files (see legends), while the numbers inside the boxes represent the tool invocation order and the solid arrows indicate the data flow. The figure shows four main tools (i.e., `af3`, `Xoncrete`, `GRec` and `MCOSF`) which are used for acquiring scheduling tables for processing nodes with support for fault-tolerance and aperiodic task admission control.

In Figure 1.4, the tool AutoFOCUS 3 (`af3`) [AVT+15] represents the main application development environment. `af3` is an Eclipse [EF18b] based modeling tool, which is used to create models for the application, platform, partitions and deployment (shown as DREAMS models in the figure) as defined by the DREAMS meta-models [DRE15]. Once models are created, text file generators can be used to generate input for a specific tool in the toolchain. After the invocation of the desired tool, the output file generated by the tool can be imported back in `af3` which updates the DREAMS models. When all the required tools are finished updating the models, the platform configuration file(s) can be generated for each processing node in the distributed network.

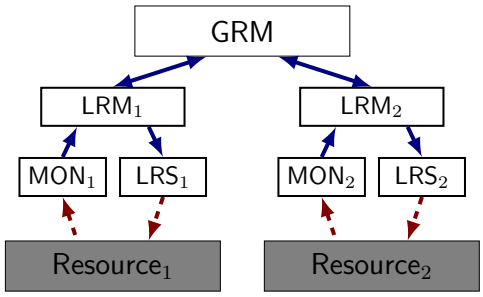


Figure 1.3: *Hierarchy of resource management components in DREAMS [DFG+16].*

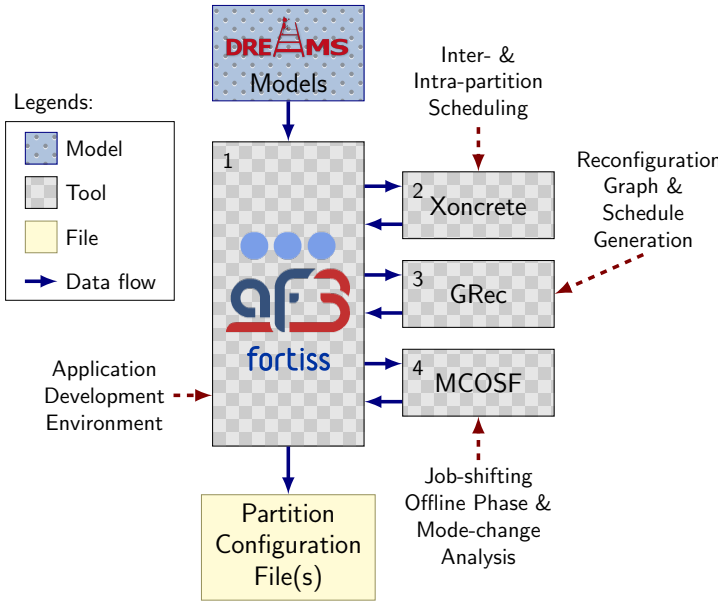


Figure 1.4: *Abridged version of the DREAMS toolchain [BDM+17].*

The tool `Xoncrete` [BMR+10] (see Figure 1.4) is used to generate partition scheduling tables (aka tier-1 or inter-partition schedules) and task scheduling tables (aka tier-2 or intra-partition schedules) for the first *service* mode (see Section 1.1.2). `Xoncrete` is a web-based scheduling tool which creates partition slots and task execution patterns (based on a variant of EDF [BMR+10]) for all cores of a single processing node.

The tool `GRec` [DFG+16] (see Figure 1.4) creates reconfiguration graphs, which are used by LRMs and GRM in order to provide fault-tolerance, and inter- and intra-partition scheduling tables for each *node* and each *service* mode (other than the first one). The tool `GRec` is a combinatorial optimization tool based on integer programming.

The tool `MCOSF` (see Figure 1.4) is used to generate flexibility coefficients for each job on each processing *node* as defined by the job-shifting offline phase in Chapter 4. The tool `MCOSF` also performs mode-change analysis and generates blackout slots and transition modes as defined in Chapter 5. `MCOSF` is a command-line tool which represents the DREAMS *module* in the GAIA framework as defined in Chapter 7.

1.2 Problem Statements

In this dissertation, we propose solutions for six problems which are faced during the development of distributed real-time systems. The following sections elaborate on these problems.

1.2.1 Run-time and Scalability of TT Schedulers

Modern real-time applications like avionics are required to incorporate more features and functionality with less production costs. Future applications like CityAirbus and autonomous driving require large distributed systems to deliver expected services and performance. For such systems, scheduling large set of applications may lead to contradicting problems, e.g., resource scarcity and SWaP (i.e., size, weight and power).

The time-triggered (TT) computation model has the potential to ease the way for solving all these issues. However, TT scheduling is known to be an NP-hard problem [JSM91, TBW92] and, therefore, poses several challenges including complex network architectures, co-synthesis of allocation/scheduling and complex constraints (e.g., precedence, latency, reliability, etc.). Although state-of-the-art TT scheduling approaches can be used to generate TT schedules for small to medium sized systems, they fail to provide solutions for large distributed systems within reasonable time.

In order to tackle run-time and scalability problems for TT scheduling, a number of problems are faced. The first and foremost problem is the selection of the scheduling approach (e.g., combinatorial optimization or meta-heuristics), as different approaches offer different run-time and scalability for the same problem instance. Once an approach is selected, the second problem is its implementation. For instance, several authors demonstrated that different constraint formulations and tools for combinatorial optimization have different run-time and scalability, e.g., [CO16, PSRNH15]. Another problem in TT scheduling is the long run-time required to show that there exists no solution for a task-set, since it requires traversal of the complete search-space. Of course,

due to the NP-hard nature of the scheduling problem [JSM91, TBW92], there exists no optimal polynomial-time TT scheduling algorithm (unless *P=NP*). However, intelligent approximation and heuristic approaches can provide adequate scalability for modern and future application designs.

1.2.2 Scheduling of Large TTEthernet Networks

In distributed real-time systems, a number of processing nodes are interconnected via a deterministic network. For such a network, different sets of guarantees are often required for different traffic types. For instance, a message on the network might require low jitter (e.g., for control applications) or a message might require low latency or high throughput. A number of real-time network types are available in market today which provide guarantees for different traffic types (also termed as mixed-criticality traffic). TTEthernet is one of these technologies based on switched Ethernet. It provides three traffic classes for mixed-criticality applications; time-triggered (TT), rate-constrained (RC) and best-effort (BE). In this dissertation, we focus on multi-mode TT scheduling of tasks and messages in TTEthernet based networks.

In order to generate a TT schedule for each mode in large TTEthernet based networks, a number of challenges are faced. First, preemptive task scheduling on distributed systems is an NP-hard problem [JSM91]. Second, a valid set of phases for each virtual link on each physical link needs to be generated. Even for a single physical link, this problem is NP-complete in the strong sense [KALW91]. Therefore over the last few decades, heuristics are proposed, e.g., [MS10, Coe17]. Third, designing a heuristic requires a lot of care since (i) task and message scheduling are interdependent, (ii) the problem grows exponentially with increasing number of virtual links, e.g., [CO16], and (iii) a large number of parameters affect the heuristic performance, e.g., virtual link ordering [Coe17].

In large distributed systems, a number of modes are often required to execute different sub-sets of software components. Existent TTEthernet network devices do not yet support multiple modes of operation. In order to support mode-changes in TTEthernet devices, a possible solution is to reserve bandwidth for all virtual links from all modes of operation in a single schedule. However, this approach leads to high bandwidth loss since the reservations for the messages, which are not active during the current mode, remain idle. Moreover, these reservations cannot be utilized by the offline scheduler to add more functionality. Another solution to this problem might be providing explicit support for mode-changes in network devices and interfaces, e.g., TDMA-TTP [KNH$^+$98]. However, with this approach, the communication protocol and the network devices need to convey extra parameters to and from network devices, e.g., mode identification number, time and type of mode-change, etc. Considering that modifications in the communication protocol and network devices leads to deprecation of existing system hardware, this approach for providing mode-change support is not plausible.

1.2.3 Aperiodic Task Execution

Modern safety critical systems require certification in order to guarantee correct operation before system deployment. The certification process requires rigorous verification and validation, the efforts for which can be greatly reduced by using resource partitioning. However, Lackorzyński et al. [LWVH12] demonstrated that bandwidth reservation for event-triggered (ET) activities in partitioned systems may lead to significant bandwidth loss. In contrast, the online admission of ET activities can prevent bandwidth loss. However, the state-of-the-art approaches for online admission of ET activities fail to fulfill the requirements of safety critical systems as they do not support (i) partitioning, (ii) the industrial mixed-criticality task model or (iii) non-preemptive task execution.

In order to provide an aperiodic admission test for a partitioned system, a number of problems are faced. First, hierarchical scheduling introduces a number of blockings which prohibit the execution of tasks. Such blockings make it difficult to provide task completion guarantees at partition boundaries since the tasks might be paused (due to partition switch) even though the tasks are non-preemptive. The second problem is non-preemptive scheduling itself, as it is known to be an NP-hard problem [JSM91] even for the simple case of uni-processor scheduling [NBFK14]. Moreover, existing priority-based non-preemptive scheduling strategies (e.g., npEDF, npRM, etc.) are known to be non-optimal [GRS96] with or without work-conserving approach [NBFK14]. Third, most of the aperiodic admission strategies (e.g., slot shifting [Foh94], slack stealing [Leh92], etc.) either rely on event-triggered approach or time-triggered approach *with* sparse time-base [Kop92]. Even though both sparse and dense time-bases [Kop92] can be implementation in a time-triggered system (based on the requirement of global causality relationship between events, see Section 1.1.3), the aperiodic admission approaches with dense time-base seams to be ignored in the literature. Clearly, providing yet another strategy just for the dense time-base would be an easy task, however the challenge is to unify these different facets of time-triggered approach. Another common problem with algorithms based on admission control is to compete against reservation/server based approaches (e.g., constant bandwidth server [AB98], sporadic server [SSL89], etc.) and provide at least the same task execution guarantees. Since the reservation based approaches have been available for more than three decades and have a mature theoretical standing, any algorithm which schedules less number of jobs compared to such algorithms will be unacceptable in the real-time community.

1.2.4 Reduction of Mode-change Delays

For a multi-mode distributed system, TT scheduling tables are often large (due to varying task periods). When a mode-change request is released, waiting for the end of the remaining scheduling table might lead to longer mode-change delays. In the worst-case, this might lead to a deadline miss by one of the tasks in the new mode. In order to reduce mode-change delays and maintain data consistency, mode-changes can be executed before the end of the scheduling table at safe time points (as demonstrated by Fohler [Foh94]). For an independent constrained deadline task-set, safe points for a mode-change can be trivially defined at time points when no task results in partial

execution upon mode-change (e.g., at the end of the hyper-period). During a mode-change, a job should either be completely executed or not executed at all [Foh94]. However in the case of communicating tasks, defining this point in time is not as trivial.

In this dissertation, we focus on reducing mode-change delays in distributed mixed-criticality applications with non-preemptive tasks executing in hierarchical scheduling environment. Although for non-preemptive tasks, determining safe points for mode-changes is comparatively easy, however reducing mode-change delays is still a difficult problem because of a strong correlation between the partition, task and communication schedules. Despite the fact that the hierarchical resource management services (similar to the one in the DREAMS project) help in determining the global state of the system at a given point in time, these services also limit the possible mode-change instants. For instance, if the mode-change execution is implemented by special resource management partitions, the instants at which the mode-change can occur is limited to the number of resource management partitions (similar to the mode-change partition slots by Real et al. [RSC16]). Therefore, deciding where to implement the mode-change execution strongly affects the opportunities for reducing the mode-change delays.

1.2.5 Development of Automotive Software

Due to ever increasing demand for functionality and complexity of software architectures, real-time applications require new design and development technologies. Other than fulfilling the technical requirements, the main features required from such technologies include scalability, re-usability and modularity. These features allow low cost of certification in terms of both time and money. AUTOSAR is one of these technologies in automotive industry, which defines an open standard for software architecture of a real-time operating system. However, being an industrial standard, the available proprietary tools do not support model extensions and/or new developments. These limitations prevent the development, evaluation and integration of new algorithms and approaches by third-parties and therefore hinder the software evolution. A solution to this problem is to develop an open-source AUTOSAR toolchain which supports application development and code generation for common modules.

In order to provide a cross-platform toolchain for AUTOSAR, a large number of problems are faced. First and foremost, adequate knowledge of the AUTOSAR architecture and meta-model is required to write working code. A good understanding of the AUTOSAR design is difficult to acquire since only AUTOSAR tools can be used to create sound examples. In this regard, the open AUTOSAR standard only provides understanding of the modules' capabilities and APIs, without referring to any specific implementation. As AUTOSAR implementations vary depending on the decisions made by the tool vendor, an API defined in the standard might not be implemented in the available AUTOSAR core. Similarly, other open documents (e.g., webinars similar to the ones by Vector [Vec17b]) only provide a shallow understanding of individual modules and often use commercial tools to provide an explanation. Another problem in creating AUTOSAR toolchain is that the architecture of the toolchain must be defined such that it is easy to extend without sacrificing the features of the AUTOSAR meta-

model. Moreover, the code generation framework must be kept independent of the generation methodology such that any existing code generation approach can be used, e.g., Xpand [EF13]. The code generation process must be designed with special focus on flexibility to enable code generation of all AUTOSAR modules, including run-time environment, base software and micro-controller abstraction layers. Another major problem in creating such a toolchain is that it requires a long time to write code and even longer time to keep it updated.

1.2.6 Rapid-Prototyping of Scheduling and Analysis Algorithms

Modern real-time systems are evolving day by day in order to cope with complex constraints and, at the same time, reducing the design, development and running cost. Moreover, the hardware platforms are also improving in order to induce determinism for real-time application development. Such diverse and evolving system designs require scheduling and analysis tools for verification and validation of their performance. Since a scheduling or analysis technique is usually tailored for a specific system (responsible for handling a specific set of constraints), rapid-prototyping of scheduling and analysis tools is required. In order to solve this issue, a number of scheduling and analysis frameworks were proposed, e.g., [LRSF04, LSST02]. However, these frameworks focus on a specific class of applications (POSIX-based applications by Li et al. [LRSF04], control applications by Lu et al. [LSST02]) and therefore only consider a sub-set of real-time task constraints.

Designing a modular real-time scheduling and analysis framework is a daunting task, as there exist large variety of real-time constraints and task models. Moreover, constantly evolving system architectures and network protocols make it even more difficult to devise standard interfaces between components/classes of such a framework. Therefore, the software architecture of such a framework needs to provide flexibility by design without sacrificing simplicity. Moreover, a number of components are expected from such a framework in order to facilitate rapid-development of scheduling and analysis algorithms. For instance, search algorithms, processor supply/demand test and response-time analysis components can greatly reduce programming efforts for offline scheduling and analysis of real-time systems. However, construction of such components is time consuming and requires large efforts in order to preserve simplicity.

1.3 Contributions

In the following, we provide a brief description of the solutions which we propose in this dissertation for the problems mentioned in the previous section.

1.3.1 Search-Tree Pruning Technique for TT Scheduling

In this dissertation, we propose a modular, scalable and flexible scheduler in order to solve large run-time and scalability problem for large distributed time-triggered systems. Our scheduler provides generic interfaces and utilizes a graph-based search algorithm in

order to allocate and schedule TT tasks. In order to reduce scheduler run-times, we propose search-tree node spawning technique and introduce a novel search-space pruning technique based on job response-time symmetries. In this dissertation, the process of generating only a sub-set of immediate search-tree nodes is termed as spawning. Moreover, a set of sub-schedules are termed as symmetric if they lead to the same response-times for all ready jobs. By generating only a sub-set of asymmetric sub-schedules at each scheduler invocation in the search-tree, the run-time of the scheduler can be greatly reduced since the number of asymmetric sub-schedules is very small compared to all sub-schedules. With thorough evaluation, we demonstrate that our scheduler is capable of exploring the search-space and that it provides adequate scalability for large distributed time-triggered systems.

1.3.2 Highly Scalable TTEthernet Scheduler

In order to schedule tasks and network traffic in large distributed TTEthernet networks, we propose a scalable multi-mode TTEthernet scheduler based on our modular scheduler design. For the sake of generating network schedule for each physical link in each mode, our TTEthernet scheduler utilizes an optimized implementation of phase generation method defined by Marouf and Sorel [MS10]. Moreover, we generate multi-mode schedules by stacking scheduling windows of tasks (on processing node schedules) and messages (on physical link schedules) which belong to different modes. Once stacked, the system can undergo deferred mode-changes seamlessly without requiring any support from the network hardware. Through extensive evaluation, we demonstrate that our scheduler is capable of fulfilling the demands of modern and future real-time applications and that it dominates the state-of-the-art time-triggered scheduling approaches for TTEthernet in terms of schedulability and run-times.

1.3.3 Job-Shifting Algorithm for Aperiodic Admission

In order to solve aperiodic task execution problem in partitioned non-preemptive time-triggered execution environment, in this dissertation, we present an algorithm (called job-shifting algorithm) for online admission of non-preemptive aperiodic tasks. Our approach circumvents bandwidth loss due to partitioning, and provides guarantees similar to the bandwidth reservation technique such that the certification process of safety critical systems need not be modified. Our approach can be implemented on top of variety of hypervisors, supports mixed-criticality task execution and can provide jitter control and lower task response-times. In order to demonstrate the applicability of our approach, we implemented job-shifting algorithm on DREAMS harmonized platform (DHP, see Section 1.1.6) in the safety critical demonstrator of the DREAMS project. Through evaluation, we demonstrate that our approach efficiently utilizes processor bandwidth and only incurs small scheduling overheads.

1.3.4 Mode-change Analysis and Delay Reduction Technique

For the sake of reducing mode-change delays, we present a novel approach for defining safe time points for mode-change execution in partitioned industrial mixed-criticality environment. Our approach generates mode-change blackout intervals for each mode-change between *service* modes (see Section 1.1.2) and generates intermediate scheduling tables, termed as transition mode schedules [Foh94], between these service modes (where possible). Based on the generated blackout intervals and transition mode schedules, our technique guarantees task-set feasibility during a mode-change and calculates the worst-case mode-change delays. We implement our mode-change analysis and reduction technique in the DREAMS project and demonstrate that the developed approach leads to significant improvements in mode-change delays.

1.3.5 Open-source AUTOSAR Toolchain

For the sake of encouraging automotive software evolution, we present an open-source toolchain for AUTOSAR, which we call AUTO-XTEND. The toolchain consists of a real-time operating system core with a low cost System-On-Module (SOM) support, a tool based on a graphical user interface (GUI) for automotive application development and a GUI tool for system configuration and code generation. AUTO-XTEND provides code generators for a large number of modules including run-time environment (RTE), communication manager (ComM) and protocol data router (PduR). To enable easy code generation, AUTO-XTEND also provides a code generation framework and an extension of the AUTOSAR meta-model which enables it to connect to third-party toolchains. To exhibit the capabilities of the toolchain, we present two case studies (i.e., inverted pendulum control via LIN bus and software-based TTEthernet end-system) and demonstrate that AUTO-XTEND generates valid artifacts and supports automotive application design.

1.3.6 Real-Time Scheduling & Analysis Framework

In order to enable rapid-prototyping of scheduling and analysis algorithms, we present a modular cross-platform framework for design and development of real-time scheduling and analysis tools; we call this framework GAIA. GAIA enables rapid-development of new scheduling and analysis algorithms for modern and the next generation of real-time systems. GAIA provides a rich set of development components (e.g., parsers, file generators, logger, visualizers, etc.) to facilitate rapid-prototyping and -development of scheduling and analysis algorithms. It also provides scalable search algorithms and analysis tools for offline scheduling of distributed, multi- and/or many-core systems. GAIA efficiently utilizes platform resources and allows verification and validation of timely constraints for large complex systems.

1.4 Dissertation Outline

The rest of this dissertation is organized as follows:

In Chapter 2, entitled Time-Triggered Scheduling and Pruning Techniques, we present our scalable scheduler for time-triggered distributed systems. We motivate the necessity of a scheduler based on response-time symmetries and node spawning technique. Based on these techniques, we propose a scheduler implementation which is capable of allocating and scheduling tasks on large distributed systems.

In Chapter 3, entitled Time-Triggered Ethernet and Scheduling, we extend our scheduler from Chapter 2 and develop a scalable scheduler for applications and time-triggered traffic in TTEthernet based distributed systems. We present an optimized implementation of the phase generation technique defined by Marouf and Sorel [MS10], which we use to generate single mode scheduling tables. Moreover, we extend our single mode scheduler and describe how time-triggered windows can be stacked to generate multi-mode scheduling tables.

In Chapter 4, entitled Online Admission of Aperiodic tasks, we present an algorithm for online admission control of aperiodic tasks in hierarchical mixed-critical environment. We describe our methodology for offline estimation of job flexibility, based on which we propose an algorithm for online admission of aperiodic jobs. We provide a detailed discussion on implementation of our algorithm for different hypervisors, free resource management and jitter control.

In Chapter 5, entitled Mode-Changes and Fault-Tolerance, we present an approach for reduction in mode-change delays in distributed hierarchic systems. We provide a description of how the fault-model and implementation affects the size of a reconfiguration graph and how mode-change blackout intervals and transition modes can be generated to reduce mode-change delays. Based on the blackout intervals and transition modes, we define how our approach can be used for mode-change delay reduction and analysis in the DREAMS project.

In Chapter 6, entitled AUTOSAR Toolchain, we present an AUTOSAR toolchain which we designed for evaluation and evolution of automotive software. Our toolchain utilizes AUTOSAR meta-model from a proprietary AUTOSAR core and provides project configurator and code generators for several AUTOSAR modules. Moreover, we propose an extension of AUTOSAR meta-model which allows it to connect to third-party toolchains. At the end, we demonstrate the code generation and extension capabilities of our toolchain on two real-time case studies.

In Chapter 7, entitled Real-Time Scheduling & Analysis Framework, we present a framework for offline scheduling and analysis of real-time systems. We provide the architecture of a typical scheduler or analysis module in our framework and describe which develop-

ment components can be exploited to speed-up algorithm development process.

In Chapter 8, we summarize the contributions of this dissertation and present the future work.

In Appendix A, we provide detailed description of the case study, which we used to elaborate extension capability of our AUTOSAR toolchain (presented in Chapter 6).

Chapter 2

Time-Triggered Scheduling and Pruning Techniques

Real-time systems are required to use state-of-the-art scheduling strategies to accommodate ever-increasing amount of functionality and provide guarantees before the system's deployment. For applications like wind farms and avionics, these real-time systems may consist of a large number of distributed processing nodes communicating through a deterministic network, e.g., 16 switches and 80 processing nodes in Airbus A380 [Iti07], several hundred nodes in Internet of Things [XHL14]. As the size and load of the distributed real-time network is increased, it becomes more and more difficult to feasibly schedule applications. To tackle these difficulties, new techniques need to be developed focusing on scalable, flexible and modular scheduler design.

A number of recent publications provide solutions for time-triggered (TT) scheduling of real-time tasks and messages on a distributed system, e.g., [ZGSC14, CO16]. Although the ideas proposed in these publications work well for small to medium sized distributed systems, however their performance decreases to a great extent when scheduling large distributed systems. The reasons for such deterioration include high processing and memory requirements, an exponential increase in run-time and low scalability. Although, the problem of high processing and memory requirements is somewhat alleviated by modern processors, the exponential run-time and the scalability problems still need adequate treatment.

From the application perspective, the use of heterogeneous processing nodes and network types (e.g., automotive networks with TDMA-TTP, CAN and LIN buses) can maximize resource utilization and reduce costs for modern applications. However, state-of-the-art TT scheduling techniques only focus on singular network types and are mostly defined for specific system architectures, e.g., [HBS10, CO16]. Consequently, it is difficult to provide network wide guarantees without a modular and scalable scheduler design.

Over the last few decades, a number of methods have been used to schedule TT tasks. These methods include meta-heuristics (e.g., Simulated Annealing by Tindell et al. [TBW92], Tabu Search by Tamas-Selicean et al. [TSPS12]), formal scheduling approaches (e.g., Timed-Automata by Waez et al. [WDR11]), constraint programming (e.g., Integer Programming by Zhang et al. [ZGSC14], Satisfiability Modulo Theory by

Craciunas et al. [CO16]) and graph-based search algorithms with heuristics (e.g., [PS93, BFR75]) or didactics (e.g., [EJ01, AK98]). Meta-heuristics are stochastic processes and strongly depend on the input parameters. Therefore, without specific measures, meta-heuristics may explore the same or similar search-space multiple times ([GK03]). Moreover, formal scheduling approaches often only scale up to a few real-time tasks executing on a few processing nodes (e.g., [AB10]).

Approaches based on constraint programming perform well for small to medium sized problem instances but their run-time explodes for large problem instances. Because of this run-time explosion, it is often difficult to incorporate all parameters of a complex task-set into constraints or equations (e.g., *switch buffer constraints* by Craciunas et al. [CO16]). One way to avoid run-time explosion using such approaches is to divide the problem into sub-problems and then compose the sub-solutions. However, such approaches are very sensitive to sub-problem division methodology/parameters (e.g., *segment size* by Pozo et al. [PRNHS17]), often explore same or similar search-space multiple times (e.g., *backtrack* by Steiner [Ste10]) and lose optimality (e.g., [PRNSH16]). Furthermore, in constraint programming, the scheduler designer can merely specify what relationships must be satisfied by the scheduling table but cannot define how to generate such a schedule (aka *declarative* programming). These generation approaches are defined by general-purpose tools which use problem agnostic approaches to solve the problem instance and are treated as black-boxes. Due to such a black-box approach, the performance of the scheduler becomes strongly dependent on the performance of the tool. Therefore, schedule generation methodologies and problem-specific optimizations via such general-purpose tools are either very difficult or impossible to implement.

In contrast, graph-based search algorithms using heuristics can quickly find a solution for moderately utilized systems. However, they only provide good average performance (i.e., for some problem instances they may take days or years to find the solution) and often explore similar search-spaces due to symmetries in task-sets. Graph-based search algorithms with didactics, i.e., pruning techniques, aid heuristics in finding the solution by reducing the search-space up to several orders of magnitude [Jon99]. Nonetheless, state-of-the-art search-tree pruning techniques only consider a limited set of symmetries (e.g., *node equivalence* by Ahmad et al. [AK98] and *scheduling symmetries* by Ekelin et al. [EJ01]) and are often used for task allocation. In other words, a large number of symmetric schedules are traversed multiple times by state-of-the-art algorithms, leaving room for improvement in terms of run-time and scalability.

Almost two decades ago, Ahmad et al. [AK98] and Ekelin et al. [EJ01] presented the concept of task-set symmetries to prune the search-tree for the allocation of real-time tasks. In this work, we introduce a new type of task-set symmetry based on the job response-times, i.e., the sub-schedules leading to the same response-times of ready jobs are symmetric. This new symmetry type can be used with or without task allocation and it enables quick search of TT schedules for large distributed systems without compromising the search-space determinism. To demonstrate that the proposed pruning technique can cope with large distributed systems, we present a modular scheduler which utilizes the symmetry information to generate partitioned TT schedules for task-sets with complex constraints. Through extensive evaluation, we hypothesize that pruning of

infeasible search-tree paths is as important as looking for feasible ones.

The rest of the chapter is organized as follows; In Section 2.1, we discuss the related work from the literature. Section 2.2 provides the system model and notations used in this chapter. In Section 2.3, we present an overview of the state of the art and its limitations. In Section 2.4, we present a detailed description of the response-time based pruning technique. Section 2.5 provides the design of a modular graph-based search algorithm. And finally, Section 2.6 provides the evaluation of the proposed pruning technique and the modular TT scheduler.

2.1 Related Work

Throughout the research history of real-time systems, a large number of publications have provided solutions to TT scheduling. Recently, Guasque et al. [GBC16] presented a strategy for offline scheduling of partitions, where the global level scheduling strategy is unknown. They presented an approach to analyze the schedulability of a set of tasks on shared processors on a multi-core platform. In contrast to their work, we allocate tasks to the heterogeneous resources and provide a generic approach to reduce the apparent search-space.

As it is difficult to summarize all major advancements in TT scheduling, from this point on we will only focus on the publications involving search-tree pruning. Moreover, we classify TT scheduling strategies in three categories based on the differences in their complexities: scheduling-only strategies, allocation-only strategies and strategies combining both scheduling and allocation techniques.

Among the scheduling-only strategies, Abdelzaher et al. [AS99] generated TT schedules for fully-preemptive task-sets with constrained deadlines using the Branch-and-Bound (B&B) approach proposed by Land et al. [LD60]. They provided optimal schedules for task-sets with communication, precedence and exclusion constraints. They used the cost of the search-tree node to bound or prune the search-tree.

Among the allocation-only approaches, Peng et al. [PS89] used B&B to optimally allocate tasks with precedence and communication constraints and provided a polynomial time algorithm. Due to the utilization of the B&B approach, they also pruned based on search-tree node cost. Almost a decade later, Ahmad et al. [AK98] provided a strategy based on the A* algorithm [HNR68] to allocate non-preemptive tasks to processing nodes. They used pruning based on processor isomorphism, node equivalence and cost bounds.

In the late 90s, Jonsson [Jon99] defined an allocation and scheduling strategy for non-preemptive jobs with precedence constraints. He also used the B&B algorithm and pruned the search-tree based on deadline misses and search-tree node cost. He asserted that the search complexity can be reduced by several orders of magnitude using depth-first search and assigning deadlines that are non-overlapping fraction of the end-to-end deadline. Two years later, Ekelin et al. [EJ01] defined an allocation and scheduling strategy for similar task-sets with fault-tolerance and heterogeneous constraints. They defined task-set symmetries which lead to similar search paths in the search-tree, i.e., exploring one of these paths is equivalent to the exploration of all similar paths. To

distinguish between the source of symmetry, they defined scheduling and allocation symmetries (earlier defined by Ahmad et al. [AK98] as *node equivalence* and *processor isomorphism*, respectively).

The work presented in this chapter resembles to the allocation strategy utilized by Ahmad et al. [AK98]. However, we utilize, along with other symmetries, response-time based symmetries. Therefore, the traversal of symmetric schedules leading to same response-times of tasks are avoided during search. We provide allocation and scheduling strategies for preemptive and non-preemptive tasks on heterogeneous processor and network architectures. Our approach is scalable and utilizes the IDA* algorithm [Kor85] to reduce the memory requirements and the scheduler run-times. Through evaluation, we demonstrate that the proposed scheduler design can fulfill the requirements of modern and future distributed systems.

2.2 System Model

In this chapter, all the activities in the system are assumed to be triggered by the passage of time, i.e., Time-Triggered (TT). Moreover, we assume a sparse time base [Kop92] (see Section 1.1.3) with a granularity defined by the smallest observable time change for each resource. Each processor time granule, termed as a *slot*, can either be used to execute a job or be left idle. All the temporal parameters in the system are considered integer multiples of the *slot* length. Moreover, the *slot* length includes the scheduling overhead, which is usually in the order of a few microseconds [Sch15].

A task τ is defined by the tuple $\langle C, D, \phi, L, P \rangle$, where C is the list of (known and bounded) worst-case execution times (WCETs) for all processor types on which τ can be allocated (which are defined by the location constraint L). D and ϕ represent the deadline and the release phase respectively, while the preemption type P can either be non-preemptive or fully-preemptive.

A task graph (TG) or a directed acyclic graph (DAG) of tasks is defined by the tuple $\langle G, V, T \rangle$, where G represents the set of graph nodes (i.e., tasks), V represents the edges between the graph nodes (i.e., communication and/or precedence relationship) and T defines the graph period (or the minimum inter-arrival time). Note that all the tasks of a TG inherit the same period T. The messages (if any) are assumed to be received at the start and sent at the end of the tasks. By definition, a TG is disjoint (i.e., no two tasks in two different TGs communicate or have a precedence relationship). For simplification, we assume that the deadlines of the tasks are constrained and the task phase ϕ is less than the period T.

In this work, we consider a real-time distributed system consisting of heterogeneous processing nodes (PNs) connected via a TT network (switched or otherwise). Fault-tolerance is achieved using task-replication. Tasks are statically allocated to PNs and do not change during system execution. The intra-node communication between the tasks is assumed to be carried out using shared memory, while the inter-node communication takes place through the TT network.

The least common multiple (LCM) of all the periods T of all TGs is termed as the hyper-period (HP) of the task-set. The TT schedule is generated offline for the length

Table 2.1: *Summary of used notations and symbols*

Symbol	Description	Symbol	Description
τ	Task	T	Period of a task graph (TG)
C	List of task WCETs	SC	Scheduling cycle
D	Task deadline	HP	Hyper-period
ϕ	Task release phase	L_{bp}	Length of busy-period
P	Full- or non-preemptive flag	$g \in \mathcal{G}$	Search-tree nodes
L	Task location constraint	S	Sub-schedules

of a scheduling cycle (SC), defined by the intervals below:

$$\mathrm{SC} = \begin{cases} [0, \mathrm{HP}] & \forall \tau : \phi = 0 \\ [0, p + 2 \times \mathrm{HP}] & otherwise \end{cases} \tag{2.1}$$

where, p represents the maximum phase ϕ of all tasks in the task-set [LM80]. During the online execution of the schedule, the second HP is the repetitive part [LM80]. The generated TT schedule is considered *feasible*, when all jobs of all the tasks, released during SC, finish their execution prior to their deadlines. A summary of notations defined in this section is provided in Table 2.1.

In order to help understand the proposed approach, in the following, we define a number of terms. A *busy-period* is defined as an interval during which a resource is utilized without any idle time. Note that the total length of the *busy-period* depends on the release of tasks/messages during the same interval. A task-set *symmetry* is defined as varying configurations - for instance, job execution pattern or task allocation - leading to equivalent schedules. Moreover, a *response-time symmetry* is defined as a *symmetry* due to equivalent response-times of jobs in a *busy-period*.

It is important to note that the pruning technique, mentioned in this chapter, is not limited to the models mentioned in this section and can be trivially extended to encompass other system requirements, e.g., on- and off-chip network traffic, shared resources, TT with dense time base [Kop92], etc.

2.3 Background

As mentioned in the introduction of the chapter, graph-based search offers some benefits over other approaches, including problem specific optimizations. In this section, we first provide an introduction to the graph-based search algorithms in Section 2.3.1. We present a brief summary on how the scheduling problem is mapped to the graph-based search problem in Section 2.3.2. Moreover, in Section 2.3.3, we give an overview of the techniques which have been used to reduce the search-tree in graph-based search algorithms.

2.3.1 Graph-based Search Algorithms

Graph-based search algorithms are used in a multitude of applications ranging from system parameter optimizations to architectural exploration. These algorithms use search-tree DAG $\langle \mathcal{G}, \mathcal{V} \rangle$ to store the explored search-space, where $\mathcal{G}$ represents the search-tree nodes which define the state of the search, while $\mathcal{V}$ represents the search-tree edges which define the decisions made to reach the nodes $\mathcal{G}$. A search-tree node without successors is termed as a *leaf node*, while a search-tree node without any predecessor is termed as a *root node*. Moreover, the *depth* of a search-tree node (starting from 0 for the *root node*) defines the number of nodes on the path leading to the *root node*. The search progresses by generating successors of a search-tree node (starting from the *root node*), until a solution *leaf node* is found. In other words, the graph-based search algorithms keep multiple complete (or incomplete) correct (or incorrect) solutions in memory and explore the search-space by enumerating all possible ways in which existing solutions may be modified.

Graph-based search algorithms are inherently deterministic (in the sense that they keep track of the explored search-space) and may provide guarantees if the solution exists for a specific problem instance. These algorithms are usually only capable of *exploitation* (i.e., local optimization) as opposed to *exploration* (i.e., global optimization). To enable *exploration* in these algorithms, either swapping [TFB13] or heuristics (defined in Section 2.3.3) can be utilized. Usually, graph-based search algorithms are used to solve complex problems and therefore have a tendency to explode in space and time.

2.3.2 Scheduling Viewed as Graph Search

In order to solve the TT scheduling problem using graph-based search algorithms, a mapping between the scheduling problem and the graph $\langle \mathcal{G}, \mathcal{V} \rangle$ is required. Note that the mapping methodology profoundly affects the performance (i.e., run-time and schedulability) of the scheduler. In general, a (sub-)schedule is mapped to a search-tree node $g \in \mathcal{G}$, while a (set of) scheduling decision(s) corresponds to a search-tree edge $v \in \mathcal{V}$.

In order to get a rough idea how scheduling is performed using graph-based search algorithms, consider a set of tasks $\tau_1, \tau_2, \ldots$ with $C_i = \{1\}$ which needs to be scheduled

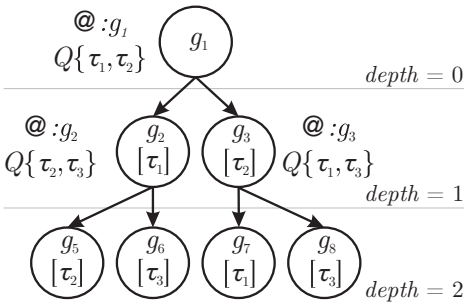


Figure 2.1: *Offline scheduling using graph-based search*

on a single processor. The search starts at *depth* 0 with the root search-tree node g_1 as shown in Figure 2.1. Assume that the ready-queue (shown as Q in Figure 2.1) at node g_1 consists of τ_1 and τ_2. Irrespective of the scheduling decision, the node g_1 will have two successor nodes g_2 and g_3, executing τ_1 and τ_2 at the start of the schedule, respectively. Now assume that a new task τ_3 becomes ready to execute at *depth* 1. The ready-queues of nodes g_2 and g_3 are updated accordingly. Similarly to node g_1, the nodes g_2 and g_3 will have two successors executing different tasks in the ready-queue. Note that with such a mapping, the left-most path in the search represent the schedule τ_1 followed by τ_2, while the right-most path correspond to the schedule τ_2 followed by τ_3. In this way, all possible schedules (*feasible* or *infeasible*) can be represented by the graph $\langle \mathcal{G}, \mathcal{V} \rangle$.

It is important to mention here that a scheduling decision taken at *depth* x of a search-tree node may lead to an *infeasible* solution at *depth* $x + y$, where the relative *depth* $y > 0$. In a purely depth-first approach, the higher the value of y and the more successor nodes, the more iterations are required to backtrack and modify the decision made at depth x. This suggests that the depth-first approaches are very sensitive to the way in which the cost of a node is defined, especially for large task-sets. Generally, the maximum *depth* x of a search-tree node is proportional to the schedule length SC, while the maximum relative *depth* y is proportional to the maximum length of the busy-periods. Moreover, the relative *depth* y can be very large for highly utilized task-sets, making these task-sets more difficult to schedule.

Throughout the research history, several techniques are used to map the scheduling problem to the graph search problem. Fohler [Foh94] mapped the search-tree nodes $\mathcal{G}$ to scheduling decisions at time t and the edges $\mathcal{V}$ to the alternatives of scheduling decisions (similarly to the mapping shown in Figure 2.1). This type of mapping makes it easier to track the explored search-tree. The heuristic function in such a mapping can make simple local decisions, however the average performance of such a heuristic is not good because of large number of search-tree nodes. Therefore, such a mapping results in large run-times and fails to provide schedules for highly utilized task-sets.

Peng et al. [PS93] and Jonsson [Jon99] mapped the search-tree nodes $\mathcal{G}$ to complete valid (or invalid) schedules and the edges $\mathcal{V}$ to sets of scheduler decisions. In such a mapping, the heuristic functions represent complex local or global decisions resulting in good performance for small to medium sized task-sets, but bad performance for large task-sets. Furthermore, the performance of such a mapping degrades to a minimum when the task-sets are highly utilized.

Abdelzaher et al. [AS99] defined another way of mapping scheduling problem on graph search problem based on solution sub-space. The characteristics of such a mapping strongly depend on the definition of the term *sub-space*. Abdelzaher et al. [AS99] mapped the search-tree nodes $\mathcal{G}$ to collections of schedules and the edges $\mathcal{V}$ to divisions of these collections. By defining such a mapping, keeping track of the explored search-tree becomes easier. However, without proper measures, this type of mapping may lead to no progress [AS99]. In this work, we used similar mapping by defining sub-space as a set of response-times for ready jobs at time t. Further details related to our mapping approach will be discussed in Section 2.4.3.

2.3.3 Search-Space Reduction Techniques

Prior to the explanation of the state-of-the-art techniques for reducing the search-space, it is important to note here that the techniques mentioned in this section do not reduce the complexity of the problem itself. Instead, these techniques help reduce the apparent search-space for the problem instance. More specifically, the techniques mentioned in this section guide the search such that a part of the search-space, which may not lead to a solution, is left unexplored.

In the following, we classify search-tree reduction techniques in two categories; didactics (aka pruning) and heuristics. In real-time scheduling theory, search-tree pruning is often considered a search-space reduction technique which does not eliminate the possible future solution (e.g., [EJ01]) as the pruning rule utilize an intrinsic property of the sub-solution to determine if it might or might not lead to a solution, e.g., deadline miss or processor symmetry. On the contrary, heuristics may lead to the elimination of possible solution, as these techniques use a search parameter to determine search-tree branch cut-off, e.g., number of backtracks, scheduler granularity.

Heuristics

Heuristics are utilized to reduce the apparent search-space as they are easy to implement and perform good on average. However, as the name suggests, heuristic approaches may not always lead to a solution (even when there exists one). These heuristic approaches reduce the scheduler run-time on average and are usually tailored to specific system models. A wide variety of heuristics can be found in publications. For example, Peng et al. [PS93] limited the number of preemptions, Bratley et al. [BFR75] limited the number of backtracks, while Fohler [Foh94] limited the number of generated successor nodes. Similarly, a number of intuitive heuristics are also found. For example, increase the scheduler decision granularity [Foh94, CO16] or use a divide-and-conquer approach [AS00, CO16] to reduce the apparent size of the search-tree.

Pruning Techniques

When scheduling hard real-time tasks, the most trivial approach for pruning is to prune the search-tree node when a job misses a deadline [Jon99]. The complexity of such a pruning technique is $O(n)$, where n is the number of ready jobs on all the processing nodes (PNs). The pruning technique usually utilized in B&B [LD60] is termed as cost bounding. Jonsson [Jon99] pruned the search-tree nodes which have higher cost than the cost of *infeasible leaf node*s. Similarly, Abdelzaher et al. [AS99] and Peng et al. [PS89] pruned the search-tree nodes based on cost bounding after generation of the successors for a search-tree node.

A number of effective pruning techniques are defined by Ahmed et al. [AK98] and later, by Ekelin et al. [EJ01]. Following the terminology used by Ekelin et al., we call them symmetry-based pruning techniques. When scheduling real-time tasks, a number of symmetries in the search-tree can be found. A processor symmetry is encountered when a task τ_k can be allocated to two or more identical PNs. In a general sense,

two PNs are identical when they lead to the same worst-case response-time for task τ_k. In a special sense, two PNs are identical when no tasks are allocated to them yet. Often, processor symmetries are not used in a general sense because of the increased time complexity [AK98]. When a processor symmetry is found, only one PN can be used to allocate the newly released task and the rest of the PNs can be excluded for allocation as exploring the search-tree for all the remaining identical PNs will result in similar scheduling tables. Such symmetries are common in homogeneous systems.

In contrast to the processor symmetries, task symmetry is encountered when two identical tasks τ_i and τ_j need to be allocated to identical PNs. Two tasks are identical when they are subject to the same set of constraints [AK98]. In such a scenario, the allocation of one task per PN or both tasks on one PN will lead to two similar scheduling tables. Therefore, in such cases, search-tree paths leading to the similar scheduling tables are pruned. Task symmetries are commonly found in systems utilizing task-replication based fault-tolerance.

2.4 Response-Time Symmetries

In Section 2.3.3, we provided a brief description of pruning techniques introduced by Ahmed et al. [AK98], which were later coined as (processor and task) symmetries by Ekelin et al. [EJ01]. In this section, we identify a new type of task-set symmetry based on the response-times of the tasks. We first motivate the necessity of pruning based on response-time symmetries with an example and discuss its advantages and disadvantages in Section 2.4.1. In Section 2.4.2, we describe how the response-time symmetries can be calculated. We provide a description of the approach we used to map the scheduling problem on the search-tree in Section 2.4.3. Then, we discuss how the information of response-time symmetries can be utilized to prune the search-tree in Section 2.4.4.

2.4.1 Motivation

In order to understand how the search-tree for a scheduling problem evolves, we need to calculate the size of search-tree sub-space for a set of ready jobs at a specific time. In the following, an execution pattern of the ready jobs at time t_s in a busy-period is termed as a *sub-schedule*. For a single PN, the number of sub-schedules at time t_s is proportional to the length of the busy-period L_{bp}. For busy-periods involving only *non-preemptive* jobs, the number of sub-schedules S_{np} is $n!$, where n denotes the number of ready jobs[1]. For a busy-period involving only *fully-preemptive* jobs ready to be executed on PN_m, the number of preemptive sub-schedules S_p can be found[2] using Equation 2.2.

$$S_p = \binom{L_{bp}}{C_1^m, C_2^m, \ldots, C_n^m} = \frac{L_{bp}!}{C_1^m! \times C_2^m! \times \ldots \times C_n^m!} \tag{2.2}$$

[1]The number of arrangements of n distinct objects is $n!$ [Tuc06].

[2]The number of arrangements of n objects with r_1 of type 1, r_2 of type 2, ..., and r_m of type m with $r_1 + r_2 + \ldots + r_m = n$ is $n!/(r_1! \times r_2! \times \ldots \times r_m!)$ [Tuc06].

where, L_{bp} denotes the length of the busy-period, while n represents the number of ready jobs. Note that, for larger values of L_{bp}, the number of preemptive sub-schedules is very large compared to the number of non-preemptive sub-schedules, i.e., $S_p \gg S_{np}$.

Assume three fully-preemptive jobs of tasks τ_1, τ_2 and τ_3 with $C_i = \{i\}$ and $D_i \geq 6$ are ready to be scheduled at time t_s. According to Equation 2.2, $S_p = 6!/(1! \times 2! \times 3!)$, i.e., 60 possible sub-schedules exist for ready jobs of tasks τ_1, τ_2 and τ_3, which are shown in Figure 2.2. Since the schedulability of a hard real-time job solely depends on its response-time and its deadline, the figure also shows all possible response-times. In the figure, the response-time sets of non-preemptive sub-schedules S_{np} are shown with underlines, while the set of sub-schedules leading to the same response-time sets are grouped together by curly braces. For the example shown in Figure 2.2, note that out of total 60 sub-schedules S_p, there exists only 30 sub-schedules S_u which lead to unique sets of response-times.

Similarly, assume four jobs with $C_i = \{2\}$ and $D_i \geq 8$ are ready to be scheduled at time t_s. For this set of ready jobs, there exist 24 non-preemptive sub-schedules S_{np}, a total of 2520 possible preemptive sub-schedules S_p, while there exist only 336 preemptive sub-schedules S_u which lead to unique sets of response-times. Based on these examples,

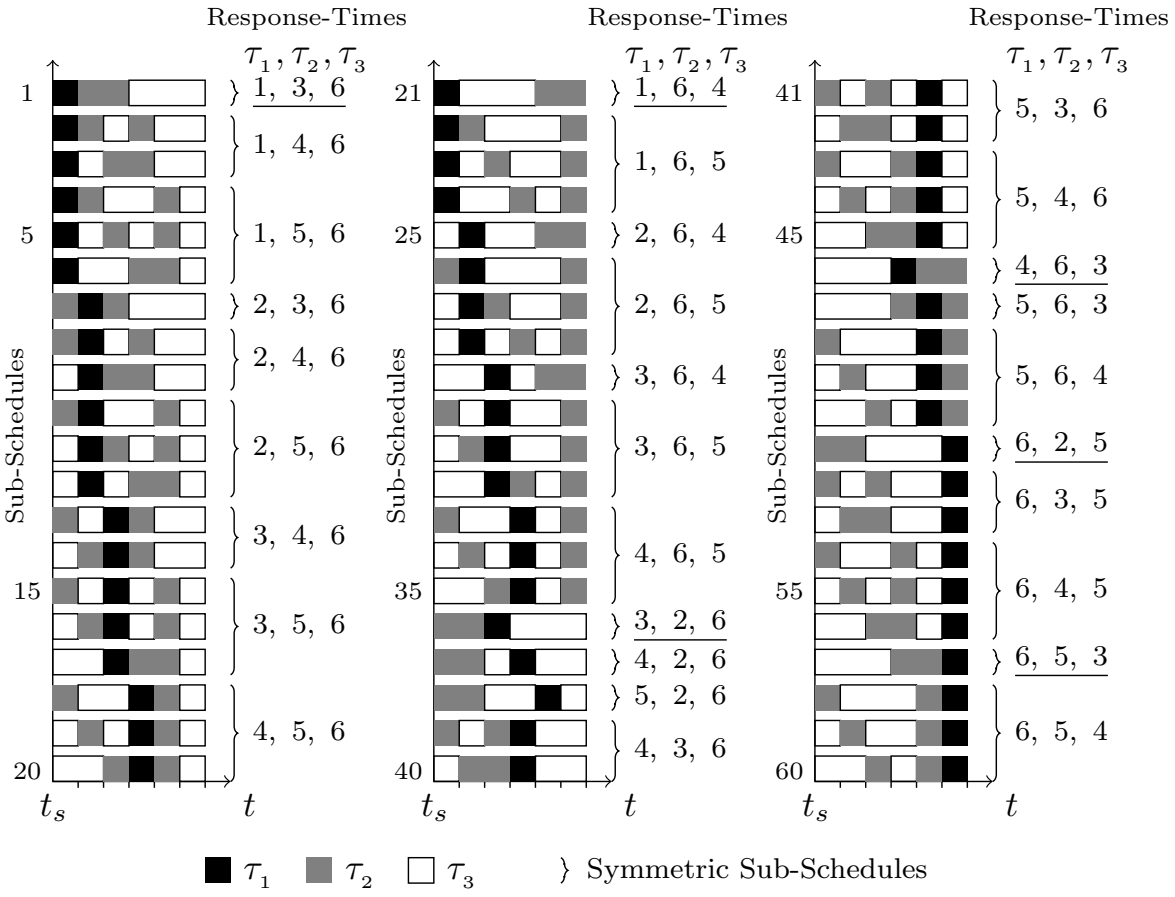


Figure 2.2: *Response-times for jobs of tasks τ_1, τ_2 and τ_3 with $C_i = \{i\}$*

it is trivial to see that, for an arbitrary set of ready jobs, $S_p > S_u > S_{np}$.

As the response-time of a job is representative of its *feasibility* [LM80, GD99], the sub-schedules leading to equivalent sets of response-times for a set of hard real-time ready jobs are symmetric. Therefore, the search for unique sets of response-times is faster compared to the search for all sub-schedules. In order to generate TT schedules, all unique sets of response-times need to be generated and checked against the deadlines. Through this process, the *feasible* TT schedule can be found, when there exists one.

In Section 2.3.3, processor and task symmetry based pruning techniques were briefly explained. These pruning techniques can only be utilized when allocating real-time tasks and are not efficient when identical processors are considered in the general sense (see Section 2.3.3). On the contrary, response-time symmetries can be utilized even when task allocation is not considered and can help reduce the apparent search-tree for a problem instance to several orders of magnitude. However, the adoption of response-time symmetries requires generation of unique sets of response-times, which is computationally expensive compared to the detection of processor and task symmetries.

2.4.2 Generation of Unique Sets of Response-Times

In this section, we first provide a general description of the procedure to generate unique sets of response-times. Moreover, we present an example implementation and discuss some performance issues.

Generation Procedure

A simple and trivial approach to generate all response-time sets is to use Processor Demand Test (PDT by Baruah et al. [BRH90]) and a sort operation. However, this approach is computationally expensive since a sort operation and a demand test is required to generate every single response-time set. As mentioned in Section 2.4.1, the number of unique sub-schedules or response-time sets is very large for trivial number of jobs, a response-time set generation approach based on PDT and a sort might cast significant impact on the scheduler run-time. Therefore, in the following, we present an approach which modifies response-times of jobs to generate valid preemptive response-time sets.

A graphical representation of the unique response-time sets S_u for the example in Section 2.4.1 is shown in Figure 2.3. The figure shows the response-times of the three jobs of tasks τ_1, τ_2 and τ_3 on x-, z- and y-axis, respectively. Since we have three jobs and the last job will always have a response-time equal to the length of the busy-period L_{bp} (see Section 2.4.1), the plot corresponds to a cube. Due to the 3-dimensional representation, we denote the response-time set (RTSet) as an ordered triple (τ_1, τ_2, τ_3). In the figure, the solid lines enclose sets of RTSets in what we call a response-times set *cluster*, while the circles highlight the non-preemptive RTSets termed as *seed* RTSets. The RTSets adjacent to the *seed* RTSets in the same *cluster* are termed as *stem* RTSets. For example, the sub-schedules 51 to 56 in Figure 2.2 are represented by RTSet *cluster A* in Figure 2.3. Similarly, the first 20 sub-schedules in Figure 2.2 correspond to RTSet

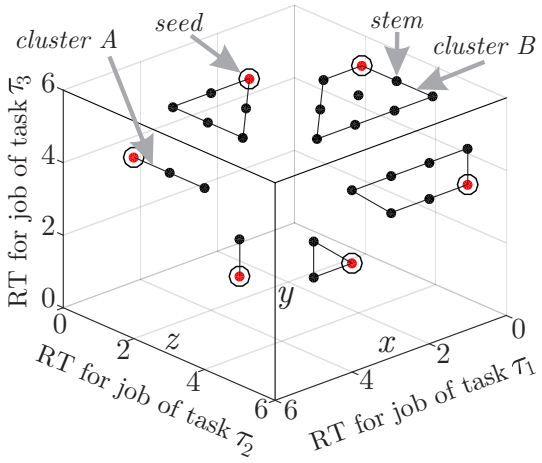


Figure 2.3: *Representation of response-time (RT) sets for the example in Figure 2.2*

cluster B. Inside *cluster A*, the non-preemptive RTSet $(6, 2, 5)$ is termed as a *seed* RTSet, while the adjacent preemptive RTSets $(6, 3, 5)$ and $(6, 4, 5)$ in the same *cluster* are termed as *stem* RTSets. Note that the number of *stem* RTSets from a given *seed* RTSet depends on the distribution of the WCETs C_i^m of the ready jobs.

To generate all preemptive sets of response-times without PDT and sort operations, one must generate (i) all *seed* RTSets (i.e., arrangements of jobs) and (ii) all *stem* RTSets from each *seed* RTSet (i.e., all multi-set permutations) by iteratively increasing the response-times until all the response-times in the RTSet become equivalent to the maximum response-time or L_{bp}.

Implementation

Algorithm 1 provides the pseudo-code for the function `NextRTSet()`, which iteratively generates preemptive response-time sets (RTSets). The function `NextRTSet()` calls `GenerateSeedRTSet()` (Algorithm 1:4) to generate non-preemptive RTSets and `GenerateStemRTSet()` (Algorithm 1:7) to generate preemptive RTSets based on the state of the input RTSet. To generate all non-preemptive or *seed* RTSets, various libraries are available which can be utilized off-the-shelf to generate all arrangements of a set of ready jobs. For example, the C++ Standard Template Library (STL) provides a template based function `next_permutation()` to iteratively generate arrangements, however it uses lexicographic ordering which requires more than one swap operation to get the next arrangement [Sed77]. For an optimized implementation, Heap's Algorithm [Hea63] is recommended as it requires only one swap operation per arrangement.

In order to generate preemptive RTSet from an input RTSet, the function `GenerateStemRTSet()` starts by finding the second highest response-time (Algorithm 1:8) in the RTSet and increments it to generate a new RTSet as long as any two response-times are the same (Algorithm 1:18). After the increment, if the response-time reaches the end of the busy-period L_{bp}, the algorithm successively increments the next lower response-times (Algorithm 1:14) and resets the older response-time to the

```
1  Function NextRTSet(s)
       Input: s = response-times set
       Output: next response-times set
2      nextSet ← GenerateStemRTSet(s);
3      if nextSet is invalid then
4          nextSet ← GenerateSeedRTSet(s);
5      end
6      return nextSet;
7  Function GenerateStemRTSet(s)
       Input: s = response-times set
       Output: next stem RTSet
8      get index i of second highest response-time in s;
9      repeat
10         Increment s[i];
11         j = i;
12         while s[j] = L_bp do
13             get index k of the next lower response-time than s[j];
14             Increment s[k];
15             s[j] ← max(s.seed[j], s[k]);
16             j = k;
17         end
18     until any two response-times are same or s is max. response-time ;
19     return s;
```

Algorithm 1: Pseudo-code for the generation of response-time sets

value from the *seed* RTSet. For example, when the function `GenerateStemRTSet()` is fed with the seed RTSet $(6, 2, 5)$ from the example in figures 2.2 and 2.3, the second highest response-time, i.e., 5, is increased to 6. Since the new response-time is equal to L_{bp}, the next lower response-time 2 is increased to 3 and the previous response-time is reset to the value in the *seed* RTSet, i.e., 5, to yield the next *stem* RTSet $(6, 3, 5)$. Note that the function `GenerateStemRTSet()` at Algorithm 1:7 generates a *stem* RTSet within polynomial time. Also notice that the presented algorithm to generate preemptive RTSets might not be as efficient as Heap's algorithm [Hea63] (which can only be used to generate non-preemptive RTSets). However, the presented algorithm has lower complexity than the demand based approach.

In order to get the response-time sets with the least cost first (for depth-first search, termed as *Priority Assignment* by Ahmad et al. [AK98]), the ready jobs must be sorted in accordance with the heuristic (e.g., absolute deadline for EDF heuristic). As the number of ready jobs is usually small for a single PN, insertion sort performs better than quick sort [CLRS01]. Moreover, as shown by the example in Section 2.4.1, the number of unique sets of response-times S_u can be very large for trivial sets of ready jobs. Therefore, it is recommended to keep only a fixed number of response-time sets in memory and generate more when required, i.e., upon backtrack. As the generation function is iterative, implementing such an optimization is quite trivial.

2.4.3 Mapping between the Scheduling Problem & the Search-Tree

Although pruning based on response-time symmetries can be used irrespective of the scheduling problem mapping, we map a search-tree node to the solution sub-space (see Section 2.3.1). In such a mapping, a search-tree node $g \in \mathcal{G}$ is mapped to one of the system wide response-time sets for a set of ready jobs at time t_s, while an edge $v \in \mathcal{V}$ is mapped to simple local scheduler decisions (i.e., which successor g to explore at schedule time t_s). The advantage of such a mapping is that one does not need to care about the actual execution pattern of the jobs, leaving room for further optimizations in post-processing (discussed in Section 2.5.1). Moreover, with such a mapping it becomes easier to keep track of the explored search-tree, which enables deterministic *exploration*. The disadvantage of such a mapping is that it leads to an enormous number of immediate successors of a search-tree node g. This means that the scheduling heuristic has to be very good in guiding the search (discussed in Section 2.5.2). Note that, in general, the large size of the search-tree is inevitable as the scheduling problem is NP-hard [TBW92].

2.4.4 Pruning Based on Response-Time Symmetries

Due to the mapping approach defined in the previous section, the response-time symmetries are automatically eliminated and, therefore, individual job based pruning is not required. Nonetheless, the response-time sets need to be created for the next set of ready jobs in the queue as the search-tree is explored *deeper*. This process may lead to symmetries as shown in Figure 2.4. Suppose the ready-queue (shown as Q in Figure 2.4) consists of jobs of tasks τ_1, τ_2 and τ_3 at time instant t_s, similarly to the example in Figure 2.2. Creating successors for the initial node g_1 leads to the expansion of the search-tree by nodes g_2, g_3, g_4 and so on. At time $t_s + 1$, a new job of task τ_4 with $C_4 = \{1\}$ becomes ready to execute. Assuming that the heuristic chose g_2 to expand the search-tree at time $t_s + 1$, the search-tree nodes g_5, g_6 and so on are generated. Note that the symmetric search-tree nodes (i.e., g_7, g_8 and so on) would have been generated when the heuristic would have chosen g_3 at time $t_s + 1$. This leads to symmetric sub-schedules (e.g., g_5 and g_7 are symmetric in Figure 2.4) and can be avoided by pruning

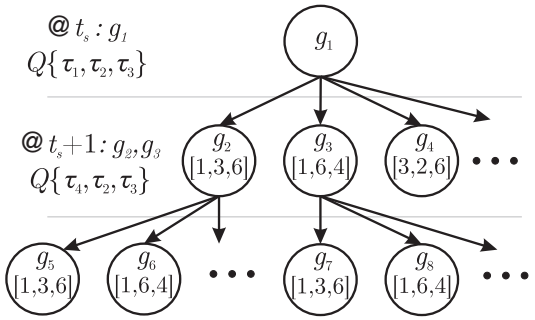


Figure 2.4: *Response-time symmetries with search-tree expansion (g_5 is symmetric to g_7, g_6 is symmetric to g_8, ...)*

node g_3.

For a system with multiple PNs, each response-time set for PN_i forms a combination with each response-time set for PN_j. Each of these combinations represents a unique sub-schedule for the complete system. When the search-tree is explored for schedule time $t_s + k$ (similarly to Figure 2.4), generating system wide sub-schedules for all possible combinations may lead to further symmetries. These symmetries can be avoided with a simple approach. The procedure for generating system wide unique response-time sets S_u^S is as follows (where p^r represents PNs on which one or more new ready jobs are released, while p^o represents PNs on which no new job is released):

1. For all p^r, generate all response-time sets S_u at time $t_s + k$.
2. Generate combinations of partial response-time sets S'_a only for p^r.
3. Deduce the response-time set S''_a for all p^o at time $t_s + k$ from the response-time set selected at time t_s (by subtracting k from all response-times).
4. Merge S''_a with all S'_a to form system wide response-time sets S_u^S at time $t_s + k$.

2.5 Symmetry Avoidance Scheduler

In this section, we provide a concrete scheduler based on response-time symmetries introduced in Section 2.4. First, we present a detailed description of the methodology we used to generate TT schedules in Section 2.5.1. Then in Section 2.5.2, we discuss the construction of the cost function and their complexities. Later in Section 2.5.3, we provide a heuristic for pruning in order to speed-up the search process and finally, in Section 2.5.4, we provide the pseudo-code for the scheduling algorithm.

2.5.1 Methodology

In order to provide a modular scheduler, we propose a scheduler design for hard real-time tasks which utilizes the information of all task-set symmetries (i.e., response-time symmetries introduced in Section 2.4 and task and processor symmetries introduced by Ahmad et al. [AK98]). We call our implementation of this modular scheduler *Symmetry Avoidance Scheduler* (SAS). As recommended by Jonsson [Jon99], we used a depth-first approach called Iterative Deepening A* (IDA*) [Kor85] to generate TT scheduling tables for hard real-time tasks. IDA* utilizes a heuristic function to estimate the cost of a search-tree node g, i.e., $f(g) = j(g) + h(g)$, where $j(g)$ represents the cost of the path from the *root node* to g and $h(g)$ is an estimate of the remaining cost to reach the *leaf node*. The algorithm starts search-tree traversal with an initial cost-bound. When the algorithm exhausts the search-tree without a solution with the current cost-bound, the cost-bound is increased by a minimum value and the algorithm reiterates over the expanded search-tree. The algorithm IDA* requires less memory than its predecessor A* [HNR68] and finds the optimal solution when the function $h(g)$ never overestimates the actual cost.

After the proposal of IDA* algorithm by Korf [Kor85], many researchers attempted to devise its parallel/distributed implementation, e.g., [RKR87, AK98, HS04, Kur09], etc. For our work, we chose the algorithm proposed by Rao et al. [RKR87] due to its dynamic search-tree nodes sharing capability, exclusive search-tree traversal by each processor and a linear speed-up. Following the terminology used by Rao et al. [RKR87], we call their parallel version of the IDA* algorithm PIDA*.

PIDA* defines each processor participating in the search as a *worker* with its own stack. Each *worker* processes the search-tree nodes from its stack in a depth-first fashion. The processing of a search-tree node means (i) pop a node from the top of the stack, (ii) create its children, (iii) check if one of the children is the solution, and (iv) if the solution is not found, push the children back to the stack. To start the search, an arbitrary *worker* j is fed with the *root* node. When there are enough nodes to process on the stack of *worker* j, the idle *worker*s get some of the unprocessed nodes from *worker* j. When none of the *worker*s has nodes to process (i.e., after termination), all the *worker*s calculate the new cost-bound and re-initialize themselves.

In order to speed-up the search, we propose a number of optimizations for PIDA*. During the search-tree node processing phase of PIDA*, instead of generating all children nodes of a search-tree node, we generate only a sub-set of children in increasing costs (see Section 2.4.2). Upon backtracking, we generate the next sub-set of children when necessary. This optimization decreases the required memory and enables design-space exploration (e.g., reallocation of tasks) of large search-trees without loosing time during the generation of unnecessary child nodes. In this work, the generation of a sub-set of c child search-tree nodes is termed as *spawning*. To enable search-tree *exploration* (see Section 2.3.1), we keep track of the number of explored search-tree nodes e for each *depth* d. When the search is stuck in a local minima, the number of explored nodes e resemble the shape of a Christmas tree. As the bottom e_b of the Christmas tree reaches a specific threshold, the local minima can be avoided by *exploring* a search-tree node close to the top of the Christmas tree. Last but not least, to achieve linear speed-up on modern processor architectures, we use one *worker* for each L1 cache block on processors with multiple cores [SF14].

The Symmetry Avoidance Scheduler (SAS) uses the scheduling problem mapping to the graph search problem as defined in Section 2.4.3. For the defined mapping technique, the search-tree node cost $f(g)$ represents the scheduling cost only (discussed in Section 2.5.2) while the initial cost-bound is set to SC. Furthermore, the task is allocated to a processing node (PN) upon the release of its first job and can be varied upon re-*spawning*.

In SAS, the drawbacks of the graph-based search algorithms (see Section 2.3.1) are circumvented by utilizing *spawning*, variable scheduler decision granularity and a divide-and-conquer approach. Furthermore, instead of utilizing pruning based on individual deadline misses (see Section 2.3.3), SAS uses pruning based on the Processor Demand Test (PDT) [BRH90] for each busy-period. If the ready-queue sorting is not considered (which is already sorted in the case of SAS), the PDT based pruning is not computationally expensive. For an arbitrary scheduler, if sorting the ready-queue is required before PDT, insertion sort is preferred over quick sort for small sized queues [CLRS01].

Pre-processing performed by SAS includes optimistic task-set transformation [RS95, CSB90], task replication for fault-tolerance and unrolling of the periodic tasks. When a valid solution is found by SAS, the post-processing step converts the response-time sets to a complete schedule. During post-processing phase, further optimization is possible, e.g., minimizing the number of preemptions.

2.5.2 Cost Functions

As mentioned in Section 2.4.2, the number of immediate children of a search-tree node generated by response-time based mapping is very large. To guide the search, SAS utilizes two levels of hierarchical scheduling costs (i.e., only when the Level-1 costs for a set of search-tree sibling nodes are equal, the Level-2 cost is used to sort them). The Level-1 cost defines the priority of the first job in the response-time set for all PNs, while the Level-2 cost represents the progression of the response-times in the response-time sets, i.e., monotonically increasing response-times result in lower Level-2 cost. The time complexity for calculating the Level-1 scheduling cost is $O(n)$, where n is the number of PNs in the system. Moreover, the complexity of calculating Level-2 scheduling cost is $O(nm)$, where m represents the number of ready jobs on a PN.

In SAS, the cost of task allocation on a PN is defined by its worst-case response-time on that PN, excluding communication (i.e., optimistic task-set transformation [RS95, CSB90]). Although, the actual response-time with communication may vary, the response-time based analysis provides a necessary condition for allocation. The computational complexity for calculating the allocation cost is $O(jk) \times O^r$, where j is the number of valid asymmetric PNs for the task to be allocated to, k is the number of tasks on the PN, while O^r represents the complexity of the utilized response-time analysis technique. In summary, the allocation heuristic is computationally expensive. However, it is important to mention here that the invocation of allocation heuristic is not required at each search-tree node, instead only at the release of a new task.

2.5.3 Search-Tree Reduction Heuristic

In order to avoid excessive number of immediate children and to reduce the scheduler run-time, the strictly preemptive sets of response-times can be ignored at the cost of slightly lower schedulability ratio. In order to implement such a pruning heuristic, Algorithm 1 can be replaced with Heap's algorithm [Hea63]. Note that this heuristic results in S_{np} sub-schedules instead of S_u sub-schedules (see Section 2.4.1). Moreover, this heuristic can also generate preemptive system schedules as the response-time sets are generated at the arrival of each new job. Furthermore, this heuristic may also prune a valid sub-schedule and therefore might not be able to find the only solution. However, such a situation is seldom encountered, as will be shown by the experimental evaluation in Section 2.6.

2.5.4 Implementation

Algorithm 2 provides the pseudo-code for the SAS functions which can be plugged in the IDA* implementation defined by Rao et al. [RKR87]. The function `isSolution()` checks if a search-tree node is the solution node, while the function `CreateChildren()` creates the immediate children nodes of a search-tree node. The function `SchedulingCostFunction()` calculates the hierarchical scheduling cost as defined in Section 2.5.2, while the function `NodeDiscard()` implements node re-*spawning* and is called before popping the search-tree node out of the *worker* stack (see Section 2.5.1). It is important to mention that the function `ScheduleNetwork()` at Algorithm 2:11 creates the TT schedule for the ready messages (if any) based on the network type(s)

```
1  Function isSolution(S)
       Input: S = search-tree node
       Output: true if S is the solution node, false otherwise
2      if t_s = SC ∧ h(S) = 0 then
3          return true;
4      end
5      return false;
6  Function CreateChildren(S)
       Input: S = search-tree node
       Output: Children of S
7      Update ready-queue at time t_s;
8      if task of any ready job j not allocated then
9          Allocate(S,task_j);
10     end
11     ScheduleNetwork();
12     return Spawn(S,c);
13 Function SchedulingCostFunction(S)
       Input: S = search-tree node with a set of response-times
       Output: Scheduling cost
14     if any ready job missed deadline then
15         return ∞;
16     cost_1 ← Level-1 cost;
17     cost_2 ← Level-2 cost;
18     return cost;
19 Function NodeDiscard(S)
       Input: S = search-tree node
       Output: true, if S can be popped out of the stack. false, otherwise
20     S ← Spawn(S, c);
21     if |S| > 0 then return false ;
22     if no allocation decision on S then
23         return true;
24     else
25         if allocOrder of last allocated task τ_j is not exhausted then
26             Reallocate τ_j to the next PN in allocOrder; return false;
27         else
28             return true;
29         end
30     end
```

Algorithm 2: Pseudo-code of SAS helper functions for PIDA*

and serves as the placeholder for TTEthernet extension of the scheduler in Chapter 3.

In order to perform scheduling, the PIDA* algorithm is initiated by feeding an empty search-tree root node to the function `isSolution()`. If the function returns false, the search-tree node is fed to the `CreateChildren()` function. This function updates the global ready-queue and calls the `Allocate()` function (discussed later in this section) to allocate tasks and to generate the PN-local ready-queues. Before returning back from the function, c children search-tree nodes are *spawned* using `Spawn()` function (discussed later in this section). The generated children search-tree nodes are then sorted based on the scheduling cost calculated using `SchedulingCostFunction()` and are pushed on the PIDA* *worker* stack. The last child node on the stack is then fed to the `isSolution()` function and the process is repeated until this function returns true. If `CreateChildren()` function returns without any child node, the next sibling search-tree node is selected (i.e., backtracked). When all c children of a search-tree node are backtracked, the `NodeDiscard()` function is called to either *spawn* c new children nodes or reallocated the last allocated task to other PN, if possible. If the function `NodeDiscard()` returns false, the search-tree node is not popped from the *worker* stack and the new children nodes are pushed on top. Note that the variable `allocOrder` at Algorithm 2:25 defines the PNs on which the task τ_j can be allocated in an order defined by the allocation cost.

```
1  Function Allocate(S,τ)
       Input: S = search-tree node, τ = task to allocate
       Output: S = search-tree node with τ allocated
2      for each asymmetric PN j valid for τ do
3          Temporarily allocate τ on PN j;
4          for each task τ_n on j do
5              Calculate maximum response-time of τ_n;
6              if maximum response-time of τ_n changed after allocating τ then
7                  allocCost[j] += response-time of τ_n;
8              end
9          end
10     end
11     allocOrder ← increasing allocCost;
12     Allocate τ to first PN in allocOrder;
13     Add allocation decision to S;
14     return S
15 Function Spawn(S,c)
       Input: S = search-tree node, c = number of children to generate
       Output: A list of c children of S
16     Create c RTSets, called set;
17     for each i in set do
18         SchedulingCostFunction(i);
19         succ_i += Create search-tree node for i;
20     end
21     Sort succ;
22     return succ
```

Algorithm 3: Pseudo-code of SAS auxiliary functions

Algorithm 3 provides the pseudo-code for the auxiliary functions `Allocate()` and `Spawn()`. The function `Allocate()` calculates the order for a task in which the task will be allocated to asymmetric *valid* (i.e., as per location/heterogeneity/replication constraints) PNs based on their response-times (see Section 2.5.2). The function first allocates the new task on a valid PN and then calculates the maximum response-time of all allocated tasks. If any response-time of an allocated task is changed after the allocation of the new task, the changed response-time of the task is added to the allocation cost. The order `allocOrder` of PNs allocation for the new task is then defined in monotonically increasing allocation cost. The first PN in the `allocOrder` is then selected for the new task and the allocation decision is added to the search-tree node. Moreover, the function `Spawn()` in Algorithm 3 is used to create a list of c children of a search-tree node which is sorted based on the cost calculated using `SchedulingCostFunction()`. The function `Spawn()` first creates c RTSets using the RTSet generation approach defined in Algorithm 1. Then, this function calculates and stores the scheduling cost for each RTSet and creates a search-tree node data structure for each generated RTSet. At the end, `Spawn()` sorts the children nodes based on the scheduling cost.

2.6 Evaluation

In this section, we focus on the evaluation of SAS while the detailed analysis of the optimizations in PIDA* can be found in the technical report [SF14]. In Section 2.6.1, a description of the experimental setup is provided. Section 2.6.2 provides a brief insight into the improvements due to search-tree exploration using response-time based pruning. Finally, Section 2.6.3 demonstrates the influence of response-time based pruning technique on scheduling of hard real-time tasks.

2.6.1 Experimental Setup

The evaluation process of the proposed schedulers is divided in two experiments. In the first experiment, the exploration capability of SAS is evaluated. In the second experiment, the influence of response-time based pruning on scheduling of hard real-time tasks is demonstrated.

In order to generate task-sets for the two experiments, a tool called Task Graph For Free (TGFF) [DRW98] is used. The tool TGFF generates set(s) of parameters for the tasks and the platform, however it does not define any mapping between them. It is important to mention here that the tool TGFF only provides random distribution to generate task-set parameters.

For the two experiments, two different task schedulers were written in C++; Symmetry Avoidance Scheduler (SAS) and Task Graph Scheduler (TGS) [Foh94]. Section 2.5 provided a detailed description of the scheduler SAS. As opposed to SAS, TGS did not utilize response-time based pruning techniques and mapped the scheduling problem to graph-based search problem as defined by Fohler [Foh94]. Moreover, both the schedulers SAS and TGS used EDF heuristic, as recommended by Jonsson [Jon99].

To run the developed algorithms, two different hardware platforms were used. The first experiment was run on an Intel® Core™ i7-4790 (8 logical cores, 3.6MHz base frequency and 8MB L3 cache), while the platform used 4GB of RAM. The second experiment was run on an Intel® XEON E5-2670 (16 logical cores, 2.6GHz base frequency, 20MB of L2 cache), while the used platform was equipped with 20GB of RAM.

In order to verify that the generated schedules are indeed correct, schedule verification tools were written in C++. The verification of schedules generated by TGS and SAS is done with sanity checks and constraint validation checks. The sanity checks included verification that (i) the generated artifact is within defined range, e.g., a job does not start before earliest start time or a job does not finish after absolute deadline, (ii) no artifact is missing, e.g., each job executes for complete WCET C, and (iii) no artifact is extra, e.g., a task is only allocated to single PN. The constraints validation included checks like, each job finishes before its absolute deadline, each successor job executes after the predecessor job(s), etc.

2.6.2 Search-Tree Exploration

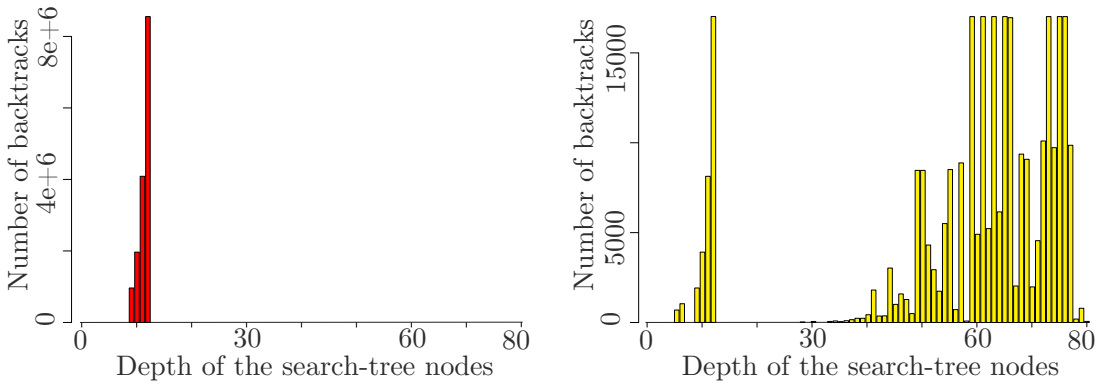
In order to evaluate the improvements in the search-tree exploration due to response-time based pruning technique, 3000 task-sets were generated using Task Graph For Free (TGFF) with the *old algorithm* defined by Dick et al. [DRW98] due to better task-set variability. The parameter ranges for generating task-sets were selected based on the behaviour of real applications [EJ01, CO16, TBW92] to generate high processing node (PN) utilization. Moreover, first 100 task-sets leading to backtracks were selected, since (i) the target of this experiment is exploration, instead of schedulability analysis, and (ii) some of the task-sets did not perform backtracking.

The selected task-sets contained 25 to 200 fully-preemptive tasks (in 5 TGs) with WCET C in the interval [5, 15], scheduling cycle SC length in the interval [200,1000), release phase $\phi = 0$ and location constraint $L = 0$ (i.e., task can be allocated to any PN), while the set {1, 2, 4, 5, 10, 20} was used as the period multiplier list (see [DRW98]) for the period T of each TG. Each generated task-set consisted of [5, 12] PN with mean PN utilization in the interval [0.6,0.9).

Each generated task-set was scheduled using SAS and TGS schedulers (see Section 2.6.1) for at most ten minutes, after which the scheduler was stopped and the task-set was deemed unschedulable. In order to observe the search-space *exploration* due to pruning, the search-tree *exploration* technique defined in Section 2.5.1 was disabled in both schedulers SAS and TGS. Moreover, for both used schedulers, only one PIDA* *worker* was used.

To observe the search-tree traversal by the two schedulers, we measured the number of backtracks[3] at all *depth*s of the search-tree nodes. Figure 2.5 shows the number of backtracks at each search-tree *depth* for a single task-set, with scheduling cycle length equal to 400, 8 PNs, 120 total tasks and 0.8571875 mean PN utilization. In Figure 2.5, the dispersed number of backtracks signify a global *exploration*, while the accumulated

[3]By backtrack, we mean traversing back to the immediate parent search-tree node.

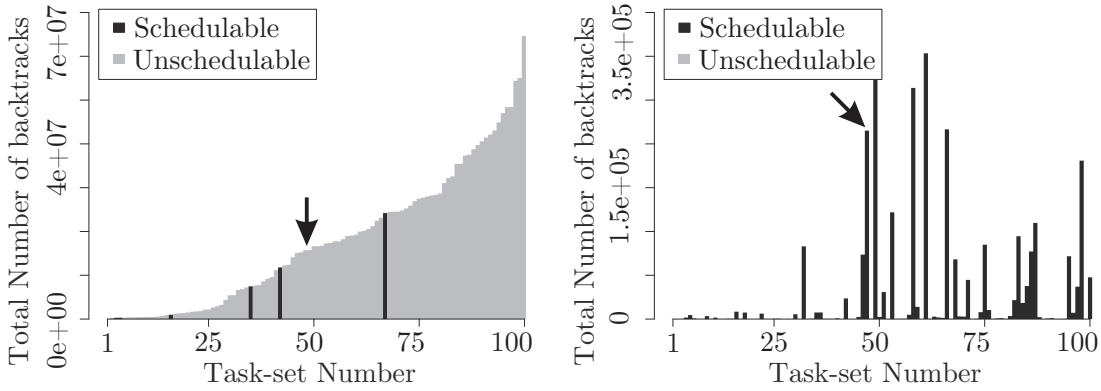


(a) *Backtracks with TGS* **(b)** *Backtracks with SAS*

Figure 2.5: *Search-tree exploration for a single task-set*

number of backtracks show that the scheduler is stuck in a local minima. We observe that, for the same task-set, SAS took very few backtracks compared to the number of backtracks by TGS. For the rest of the task-sets, the distribution of the number of backtracks showed similar trends.

For completeness, Figure 2.6 shows the total number of backtracks for each selected task-set, where the task-set list is sorted based on the total number of backtracks taken by TGS and the arrows indicate the task-set for which the backtracks are shown in Figure 2.5. In Figure 2.6, note that the number of backtracks by TGS are 200 times more than with SAS, due to which, SAS could schedule more task-sets than TGS within the same allowed execution time. Moreover, we observe that the number of backtracks taken by SAS are random, even though the task-sets are ordered in increasing number of backtracks by TGS. One would expect that the total number of backtracks by SAS would increase with increasing backtracks by TGS. However, this trend cannot be observed in Figure 2.6. The reason for this randomness is limited time for scheduling. If both



(a) *Total Backtracks with TGS* **(b)** *Total Backtracks with SAS*

Figure 2.6: *Search-tree exploration for all task-sets*

the schedulers are allowed to run as long as they can, the increasing trend will indeed be observed. In general, for a given task-set and a scheduling heuristic, the number of backtracks by TGS is always larger than the backtracks by SAS. Therefore, we can conclude that TGS spends a lot of time stuck in a local minima while SAS is capable of *exploring* the search-space and, hence, is more suitable for scheduling real-time tasks.

2.6.3 Pruning Evaluation

To evaluate the scheduler SAS for TT task scheduling, task-sets with medium and large system sizes were generated using TGFF with the parameter ranges as mentioned in Table 2.2. Since this experiment aims to quantify the difference in schedulability of tasks with and without response-time symmetries, message sizes were set to 0. As a result, the edges $v \in V$ of a TG represented simple precedences. We will extend our experimentation with real-time communication in Chapter 3.

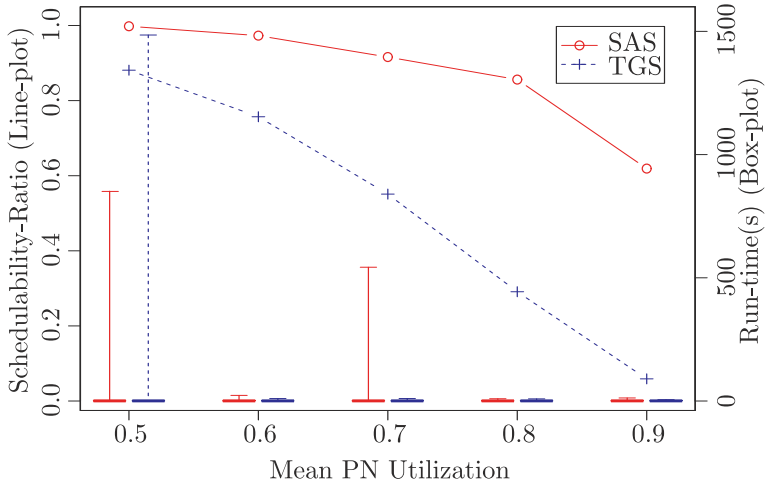
For each system size given in Table 2.2, 1000 task-sets were generated containing fully-preemptive tasks, where every task was free to be allocated to any PN, while the set {1, 2, 2.5, 3, 5, 10, 20} was used as the period multiplier list (see [DRW98]) for the period T of a TG. The period multiplier list was chosen to capture the behaviour of real applications [EJ01, CO16] and to limit the scheduling cycle length SC. As TGFF does not provide fine control over the task-set utilization, the number of PNs was set for the generated 1000 task-sets such that the mean PN utilization is 0.5. Similarly, the number of PNs was set for the 1000 task-sets to get task-sets with 0.6, 0.7, 0.8 and 0.9 mean PN utilization. This resulted in 5000 task-sets for each system size to be scheduled.

For this experiment, the schedulers SAS and TGS (see Section 2.6.1) ran for each generated task-set for at most one hour with one PIDA* *worker* thread per L1 cache block (i.e., 8 *workers* for XEON processor). Moreover, the number of *spawn*ed nodes c for SAS was set to 10.

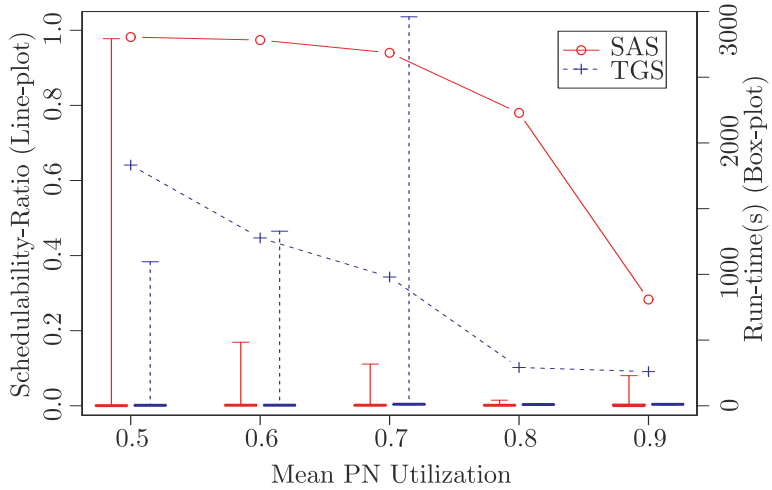
Figure 2.7 shows the schedulability ratios and run-times of SAS (with solid lines) and TGS (with dashed lines) for medium and large system sizes. The schedulability ratio of 1.0 signifies that valid feasible schedules for all 1000 task-sets were generated by the respective scheduler. The box-plots shown in the figure only represent the scheduler run-times for the feasible task-sets. For the rest of the task-sets, the schedulers ran for one hour, after which they were terminated. Note that the sizes of the boxes in the box-plot of Figure 2.7 are very small, signifying almost the same run-time for most of the scheduled task-sets. A number of task-sets took more than a few minutes (shown by the long whiskers) due to frequent backtracking. It is important to mention here that

Table 2.2: *System sizes and their parameter ranges for pruning evaluation*

System Size	PNs	Tasks	TGs	Tasks/TG	SC Length	C
Medium	[5, 25]	[25, 300]	5	[2, 60]	[200, 1000]	[5, 14]
Large	[26, 50]	[200, 700]	[20, 40]	[2, 60]	[1000, 4000]	[40, 60]



(a) *Medium system size, i.e., [5, 25] PNs*



(b) *Large system size, i.e., [26, 50] PNs*

Figure 2.7: *Comparison of SAS and TGS in terms of schedulability and run-times*

the run-times and the schedulability ratios shown in Figure 2.7 depend on the parameter c (i.e., the number of *spawned* search-tree nodes), the number of PIDA* *worker*s, the host CPU speed and the cache architecture [SF14].

Figure 2.7 shows that SAS provides much better schedulability ratios than TGS. For medium sized systems (i.e., Figure 2.7a), the largest difference between the schedulability ratios (i.e., 0.565) can be seen at 0.8 mean PN utilization. Moreover for highly utilized task-sets, TGS only resulted in the schedulability ratio of 0.059, whereas the SAS delivered a schedulability ratio of 0.619. For large system sizes (i.e., Figure 2.7b), SAS scheduled almost all task-sets until 0.7 mean PN utilization but could only schedule a few highly utilized task-sets. On the contrary to SAS, TGS could only schedule less than 50% of the task-sets beyond 0.5 mean PN utilization and dropped to its minimum at 0.9 mean PN utilization. Again, the largest difference between the schedulability ratios of the two schedulers (i.e., 0.678) can be observed at 0.8 mean PN utilization.

In order to perform this experiment, the search-tree *exploration* technique (see Section 2.5.1) was disabled and the processor and task symmetries [AK98, EJ01] were enabled in both the schedulers SAS and TGS. Moreover, both the schedulers utilized the same allocation strategy (see Algorithm 3). In other words, the only difference between the schedulers was the inclusion of response-time based pruning in SAS. Note that the difference of mapping, between the scheduling problem and the search-tree nodes, in SAS and TGS does not cast a major impact on the schedulability ratio, when the schedule and the response-time set generation approaches use the same sub-schedule ordering. In this experiment, lexicographic ordering was used to generate sub-schedules for TGS and response-time sets for SAS. This indicates that the improvement in the schedulability ratio can mostly be accredited to the response-time based pruning and node spawning techniques.

Chapter 3

Time-Triggered Ethernet and Scheduling

Safety critical applications are required to provide reliable, deterministic and predictable behaviour. For safety critical communication, time-triggered (TT) networks are often used to satisfy real-time requirements. The use of the TT paradigm [Kop92] also enables easy certification process and provides temporal isolation between network traffic from different tasks executing on multiple platforms. Moreover, as modern systems gain the ability to adapt to changes in the environment, the network also has to provide flexibility to support such adaptations. In order to provide this flexibility, modes of operations (see Section 1.1.2), which are switched online to adapt to specific application requirements, are used in TT networks.

In order to execute a TT schedule on a real-time network, all network links have to be *feasibly* scheduled for at least one mode of operation. To avoid interrupting a transmitting message and bandwidth loss when re-transmitting (or to eliminate preemption delay when resuming), the communication packets or frames are considered to be transmitted non-preemptively. Although TT scheduling problem for distributed systems and non-preemptive message scheduling problem are both NP-hard [JSM91, NBFK14], the benefits of non-preemptive TT scheduling (e.g., low overheads, deterministic behaviour) outweigh the difficulty of finding schedules for real problem instances. An important point worth mentioning here is that in some real-time networks, TT traffic is scheduled strictly periodic to reduce the memory requirement for network devices. The problem of finding strictly periodic schedules is NP-complete in the strong sense [KALW91].

To support adaptations to changing environmental conditions, the time-triggered paradigm proposes individual TT schedules for each mode of operation, which are switched online via a mode-change [KG94, Foh94]. In order to execute mode-changes on TT networks, the network protocol and network devices need to provide support for multiple modes of operation. However, existing TT network devices often do not support mode-changes in order to reduce size, weight and power (SWaP). Therefore, if mode-changes are required, the network protocol and the hardware need to be modified and re-certified.

There exists a large number of network types in market today, which provide TT scheduling with varying speeds and network scalability. For instance, the Time-Triggered Protocol (TDMA-TTP) [KNH+98] provides a 25Mbps bus speed and triple modular redundancy, FlexRay [KKM11] provides a 10Mbps bus speed, while TTEthernet [TTT18b]

allows 10/100/1000Mbps speeds over Ethernet. In order to support mode-changes in TDMA-TTP, Kopetz et al. [KNH+98] defined a mode-change protocol. However, TDMA-TTP supports only a few network nodes (due to bus architecture) and only supports time-triggered frames. Similarly, Klobedanz et al. [KKM11, KKMR11, KDMK10] provide support for mode-changes in FlexRay using online reconfiguration. However, they require a *coordinator* task to calculate a new schedule online, which might take a long time to finish. Despite offering high bandwidth and multiple network traffic classes, the TTEthernet protocol does not support mode-changes explicitly. Therefore, an efficient approach to schedule TT tasks and messages along with mode-change support is required without modifying network devices or the communication protocol.

In this chapter, we present an extension of our modular scheduler (presented in Chapter 2) for TTEthernet, to enable (i) efficient scheduling of TT traffic in large distributed networks and (ii) mode-change support in commercial off-the-shelf (COTS) TTEthernet devices. We call this extension of our modular scheduler *SAS-TTE*, where we generate a preemptive task schedule for each processing node and a schedule for all strictly periodic non-preemptive TT frames for each physical network link. In order to generate a multi-mode TT schedule for all modes of operation, we stack TT scheduling windows of tasks and frames which belong to different modes of operation. We evaluate the scheduler on industrial-sized synthetic but practical task-sets and highlight how SAS-TTE can be used to enable system wide mode-changes. Our evaluation shows that (i) our schedule generation approach dominates the state-of-the-art in terms of schedulability and run-time, and (ii) our stacking approach (supported by COTS TTEthernet devices) leads to less bandwidth loss compared to the non-stacking approach.

The rest of the chapter is organized as follows; In Section 3.1, we discuss the related work. Section 3.2 provides the system model and notations used in this chapter. In Section 3.3, we present an overview of the state-of-the-art and its limitations. In Section 3.4, we present a detailed description of the mode-stacking approach. Section 3.5 provides the details of the extension of the scheduler (from Chapter 2) required to implement the proposed TTEthernet scheduler. And finally, Section 3.6 provides the evaluation of the SAS-TTE scheduler from the perspective of single- and multi-mode schedule generation.

3.1 Related Work

There exists a large number of publications which focus on scheduling of networks based on TTEthernet. In this section, we will focus only on works related to the scheduling of time-triggered (TT) traffic. Moreover, we will classify the available strategies in two categories due to varying complexities: (i) network-only, and (ii) combined task and network scheduling strategies.

Among the network-only strategies, Steiner [Ste10] utilized C-API of an SMT solver to develop an incremental scheduler which iteratively generates sub-schedules for a subset of frames. These sub-schedules are then *placed* to form a complete schedule. When a frame sub-set cannot be scheduled, backtracking[1] is used by increasing the sub-set

[1]Steiner [Ste10] used the term *backtrack* to identify discarding procedure of one or more sub-

size. Due to the utilized backtracking approach, symmetric sub-schedules are traversed. The proposed approach was shown to schedule 1000 *uniform period* frames[2] within half an hour.

A similar network-only strategy was developed by Pozo et al. [PSRNH15], where they restricted possible assignments of phases in time domain for each sub-set of frames and then provided dependency ordering between frames of different sub-sets. Due to their decomposition methodology, their approach lost optimality at the cost of lower schedule synthesis time. However, they were able to schedule 100000 *uniform period* frames on networks with sizes up to 250 end-systems within an hour. Later, Pozo et al. [PRNSH16] proposed an extension of their previous work for arbitrary frame periods. They developed an iterative decomposition approach with frame ordering, and designed two variants with varying compositions and complexities. They were able to schedule their synthetic task-sets with networks of up to 64 end-systems with 10000 frames and 141000 frame instances within an hour. Similarly, Pozo et al. [PRNHS17] extended their previous approach for wireless TT links by providing an incremental approach with adaptive step- and segment-size. They demonstrated that their strategy is able to schedule task-sets with huge network sizes, i.e., 885 end-systems and 10000 frames, within 2 minutes, albeit with *harmonic* periods and without task schedule.

Recently, Coelho [Coe17] developed a search-tree based scheduler for TT network traffic. Using the schedulability conditions defined by Marouf and Sorel [MS10], he proposed two variants of his physical link scheduler: look-back and look-ahead. The look-back technique generates feasible phases for TT frames based on the previously scheduled traffic. While, the look-ahead technique also tries to generate feasible phases for frames ahead (i.e., next frames in the list), in order to detect infeasible phases at an earlier level in the search-tree. He demonstrated that the look-ahead technique was able to schedule up to 400 VLs on a single physical link within 80 seconds. Similar to the approach defined by Fohler [Foh94], he used branching factor heuristic to reduce apparent search-space.

Similar to the work by Steiner [Ste10], Craciunas et al. [CO16] defined an approach to schedule pre-allocated tasks and TT traffic in a large TTEthernet cluster. They also considered tasks with precedence constraints and switches with memory constraints. However, the number of assertions for their Satisfiability Modulo Theories (SMT) based solvers increased exponentially with increasing hyper-period length due to the fully-preemptive nature of tasks, resulting in large run-times. Similarly, Craciunas et al. [COCS16] presented yet another SMT based scheduler for (only) TT traffic in TSN (Time Sensitive Network) and evaluated its performance on medium sized networks. Similar to their previous scheduler, the run-time for their TSN scheduler was large due to large number of assertions.

In order to provide mode-change support, the state-of-the-art provides two approaches. One possible approach is the inclusion of adaptation support into the network specification itself. For instance, TDMA-TTP [KG94] includes support for adaptation through

schedules generated from the sub-sets of frames.

[2]Steiner [Ste10] (and Pozo et al. [PSRNH15]) used the term *frame* to identify a periodic message on a single physical link.

mode-changes in its specification [KNH+98] using two mode-change types; deferred and immediate (see Section 1.1.2). However in TDMA-TTP, the number of mode-changes is limited. Moreover, TDMA-TTP is not able to service event-triggered (ET) traffic alongside TT traffic which may be desired by modern mixed-criticality applications.

Another possible approach to provide mode-change support is to perform online reconfiguration by recomputing a suitable TT schedule on the fly. This approach is investigated by Klobedanz et al. [KDMK10, KKM11, KKMR11] for FlexRay. To support adaptation, they used frame-packing to dynamically alter the contents of scheduled messages. A problem with their approach is large incurred overheads due to online scheduling and re-configurations. Online re-computation of TT schedules may take a significant amount of time, during which the network may or may not be available. Another problem with their approach stems from their schedule generation methodology. Since their approach requires aggregation of all messages from all modes into a single schedule, we term this approach a *super-schedule* approach. A super-schedule wastes network bandwidth, as reserved bandwidth for inactive modes remains idle. However, this method allows the network to change modes seamlessly since all messages of all modes are assigned their own time slots in the TT schedule.

In this chapter, we present an efficient scheduler for real-time tasks and TT traffic for small to huge TTEthernet clusters. Our approach for generation TT traffic schedule is similar to that of Coelho [Coe17], however, we generate schedules for all physical links in the network and schedule tasks with complex constraints. Moreover, in order to support mode-changes in COTS TTEthernet devices, we highlight how the information about the mutually exclusive release of messages from different modes of operation can be exploited to create stacked schedules [HSF16]. Through evaluation, we demonstrate that the combination of the symmetry based scheduler (from Chapter 2) and the optimized implementation of phase generation method defined by Marouf and Sorel [MS10] performs very well for small to huge TTEthernet clusters. For evaluation, we generate task-sets similar to the evaluation done by Craciunas et al. [CO16] as they provide thorough understanding of the scheduler behaviour in different network architectures and system sizes. The evaluation demonstrates that our stacking approach can lead to better system utilization and schedulability compared to the super-schedule approach.

3.2 System Model

An introduction to the TTEthernet network, its design approach and its toolchain was presented in Section 1.1.5. In this chapter, we only focus on the time-triggered (TT) scheduling tables for TTEthernet and therefore consider no rate-constrained (RC) or best-effort (BE) traffic. In this section, we provide details of the system model and present the notation we use to explain the developed methodology. In order to provide a quick overview of the notations introduced in this section, a summary is provided in Table 3.1.

As we only consider TT traffic in this work, all activities are assumed to be triggered by the passage of time. Without loss of generality, we assume that all timely param-

Table 3.1: *Summary of used notations and symbols*

Symbol	Sub-element	*Description*
N	n	Network nodes, i.e., switches or end-systems
L	l	Network physical links (PL)
$\mathcal{V}$	v	Virtual links (VL)
Θ	O	Modes of operation
$\mathcal{S}$	s	Time-triggered (TT) transmission slots
μ		Physical link speed in bits-per-second (bps)
R		Virtual link route
p		Virtual link period
n		Size of the message in bytes on a virtual link
ϕ		Phase for a TT slot s
b		Breadth for a TT slot s
f		Frame transmission time
δ		Synchronization precision

eters for tasks are specified with an end-system granularity termed and *slot*, while all timely parameters for TT messages are specified with a network granularity termed as *macrotick*. Notice that the network granularity *macrotick* can differ from the end-system granularity *slot* and that there exists no direct link between their lengths. These granularities are usually selected based on best-practices and the tolerable scheduler run-time [CO16, Foh94] (as scheduler run-time decreases with increasing granularity).

We model a switched network as an un-directed graph $\langle N, L \rangle$, as shown in Figure 3.1, where the nodes N represent the switches and the end-systems (ES), while the edges L between the nodes represent full-duplex physical links (PL). A physical link $l \in L$ is further defined by the tuple $\langle a, b, \mu \rangle$, where a and b define the nodes connected by the link l, while μ defines the link speed in bits-per-second (bps). Note that the physical link speed μ can differ for each link, e.g., 1Gbps links between switches and 100Mbps links between end-systems and switches. This captures the multi-speed capability of the

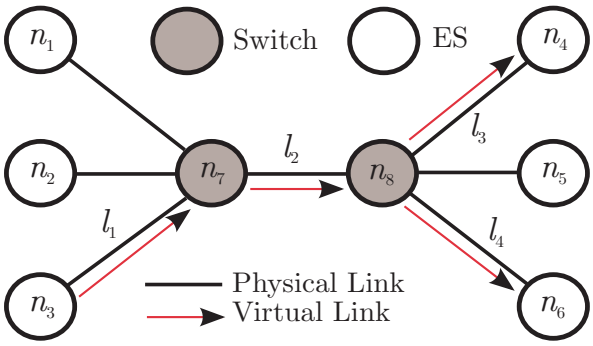


Figure 3.1: *Model of the TTEthernet network*

TTEthernet network.

In TTEthernet, network bandwidth for a TT or RC message or frame can be reserved using the concept of virtual links (VL). A virtual link $v \in \mathcal{V}$ is defined by the tuple $\langle c, D, R, p, n \rangle$, where c defines the source node/end-system (i.e., $c \in N$), D defines the set of destination nodes/end-systems (i.e., $D \subset N \wedge c \notin D$), R represents the virtual link route/path (shown as arrows in Figure 3.1), p represents the VL period and n represents the message size in bytes. In this chapter, we will use the terms virtual link (VL) and message interchangeably. Furthermore, we assume that the message deadlines are implicit (i.e., equal to the message period), and the source node s, the destination nodes D and the routes R are known and static for all virtual links.

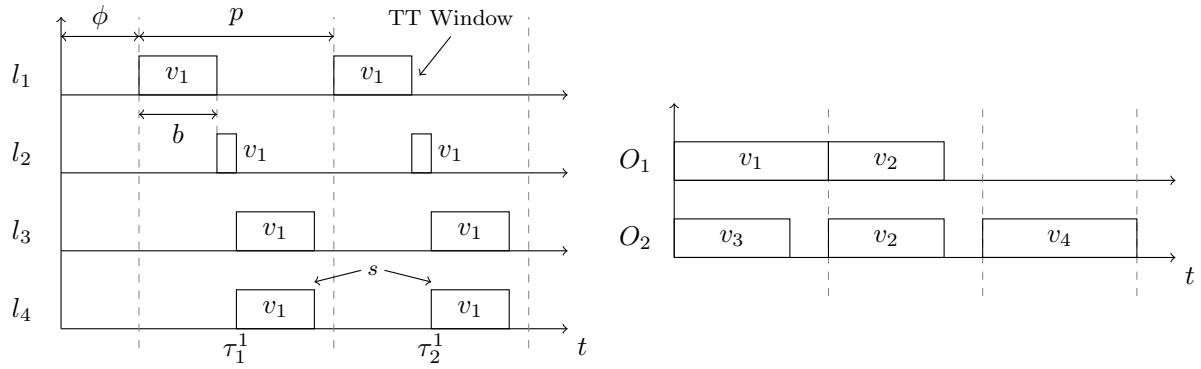
An application is represented by a set of modes Θ. A mode $O \in \Theta$ is defined as a set of virtual links which represent to-be-transmitted or -received TT messages in mode O. Note that this model of a mode does not encompass any task, as in this chapter, we focus on network traffic only. Although sections 3.5 and 3.6 use the task model from Chapter 2, any arbitrary task model can be considered here. In case the messages are generated by the tasks, they are assumed to be released at the end of the job (or at the end of transmission on a predecessor PL on the VL route).

In order to schedule the network, a scheduling table is defined for each physical link (PL) in each mode. The scheduling table for a PL $l \in L$ and mode $O \in \Theta$ is defined as a set $\mathcal{S}$ of non-overlapping TT transmission slots for all the virtual links (VL) $v \in O$ which use the PL l, i.e., $l \in v.R$. A TT transmission slot $s \in \mathcal{S}$ is defined by the tuple $\langle \phi, b, v \rangle$, where ϕ is the phase of the start of the scheduling slot s for VL $v \in O$, while b is the breadth/width of the scheduling slot s. It is important to note here that the TT transmission slot s represents an infinite sequence of TT transmission windows. A TT slot s can be unrolled with period $v.p$ to get all the TT transmission windows. This property of the TT slots $\mathcal{S}$ captures the strictly periodic constraint of the TT traffic in TTEthernet network (further discussed in Section 3.3.1).

For a TT transmission slot s, the phase ϕ is generated by the network scheduler, while the width b is calculated using the equation system below;

$$\begin{aligned} f &= \frac{n \times 8}{\mu} \\ b_i &= f_i + f_{\text{PCF}} + \delta + f_{\text{BE}} \end{aligned}$$

where, f represents the frame transmission time, b_i represents the TT slot width for virtual link v_i and δ represents the synchronization precision. The synchronization precision δ is a function of synchronization jitter and period of transmission of Protocol Control Frames (PCF). For scheduling, δ is considered a constant and usually has a value of 1μs [CO16]. For the above equations, the f_{PCF} defines the PCF transmission time, which is calculated with $n = 64$ [SAE11]. Moreover, the f_{BE} defines the maximum BE frame transmission time, where the frame size n is equal to the maximum Ethernet frame size, i.e., 1542. Note that the frame transmission times f_{PCF} and f_{BE} are included while calculating the width of each TT transmission slot s since, in the worst case, both of these PCF and BE frames might be occupying the physical link at the start of the transmission slot (assuming shuffling as traffic integration policy [SAE11]).



(a) *Uni-mode TT schedule for virtual link* v_1 **(b)** *Multi-mode TT schedule for physical link* l

Figure 3.2: *TT schedule examples for TTEthernet*

Figure 3.2a shows an example of the TT schedule for a virtual link $v_1 \in O$. In the figure, the TT transmission slots s represents an infinite number of TT windows (for infinite TT frames) for the virtual link v_1 on a specific physical link. Figure 3.2b shows the schedule of modes O_1 and O_2 for a physical link l. In the figure, note that the virtual link v_2 exists in both modes O_1 and O_2.

3.3 Background

In this section, we present an overview of the terms, which are required for understanding the scheduling problem. In Section 3.3.1, we provide a description of the strictly periodic constraint, its complexity and how is it solved in the state-of-the-art. Moreover, we summarize the available mode-change implementations and discuss their advantages and disadvantages in Section 3.3.2.

3.3.1 Strictly Periodic Constraint

In the real-time domain, the network traffic is often considered non-preemptive since preemption of a frame may require support from hardware and might lead to bandwidth loss when re-transmitting (or preemption delay when resuming). Although non-preemptive scheduling offers several benefits (e.g., reduced I/O delays for tasks [NBFK14], no division and padding of sub-messages), it is an NP-hard problem [JSM91]. From the perspective of scheduling, there exists almost no difference between a periodic task or a periodic message. Therefore, in the following, we will use the terms message and task (or frame and job) interchangeably.

In TT systems, the scheduling table entries can be defined in two ways, i.e., a single scheduling table entry (i) per *job* or (ii) per *task*. Since there may be more than one job of a task per hyper-period, the latter approach leads to smaller scheduling tables and, therefore, requires less memory. However, the latter approach puts forward a new constraint on the system schedule, which is usually termed as *strictly periodic constraint* [MS10, CO16]. This constraint asserts that the separation between the start of execution

of any two consecutive jobs of a non-preemptive task is equal to the task period, i.e., $\tau_k^i = \phi_i + (k-1) \times p_i$, where τ_k^i represents the start time of the kth TT window for virtual link v_i (as shown in Figure 3.2a). In order to reduce the required memory, the scheduling tables in TTEthernet are defined using a single scheduling table entry per virtual link (instead of multiple entries per virtual link). Therefore, a TTEthernet scheduler must be able to handle the strictly periodic constraint for all virtual links.

In 2010, Marouf et al. [MS10] proved that a set of non-preemptive tasks with strictly periodic constraint executing on single processor is schedulable when the sum of worst-case execution times is less or equal to the greatest common divisor (GCD) of their periods. In other words, there exists the following sufficient test for n virtual links:

$$\sum_{i=1}^{n} b_i \leq GCD(\forall i, p_i)$$

where, b_i represents the TT window width and p_i represents the period of the task (or virtual link) v_i. Based on the above equation, Marouf et al. [MS10] also provided Equation 3.1 to generate feasible phases for a task when the phase of a feasibly scheduled task is available. In the following, we provide the pair-wise phase generation approach presented by Marouf et al. [MS10] to compute the set of feasible phases ϕ_k for a strictly periodic non-preemptive task (or virtual link) v_k assuming that the task (or virtual link) v_i is scheduled at phase ϕ_i.

$$\forall n \in \mathbb{N}, m \in [b_i, GCD(p_i, p_k) - b_k] : \phi_k = \phi_i + n \times GCD(p_i, p_k) + m \tag{3.1}$$

A year later, Marouf et al. [MS11] proposed local schedulability conditions and their respective phase generation approach for strictly periodic non-preemptive tasks. However, due to the complexity of the equations and large expected run-time overheads, the implementation of this phase generation approach is debatable and, therefore, not used.

3.3.2 Mode-Change Implementations

In Section 1.1.2, we provided a brief introduction of the terms mode, mode-change and mode-change blackout. In this section, we discuss the different implementations of modes. To compare the bottlenecks of different mode-change implementations, multi-mode scheduling table implementations for a physical link l are shown in Figure 3.3 and 3.4. The figures show example schedules for mode O_1 with virtual links (VL) v_1 and v_2 and mode O_2 with VLs v_3 and v_4.

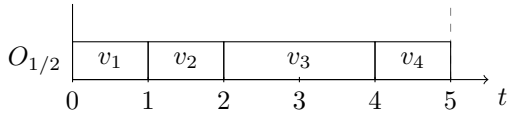
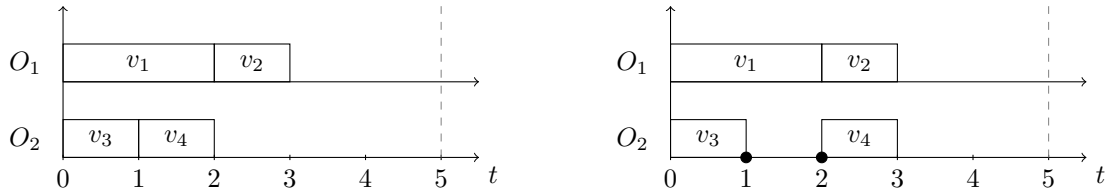


Figure 3.3: *Multi-mode physical link schedule using super-schedule approach*



(a) *Deferred mode-change schedule* **(b)** *Immediate mode-change schedule*

Figure 3.4: *Multi-mode physical link schedule using individual-schedule approach*

Figure 3.3 shows the PL schedule generated using the super-schedule approach, where all VLs from all modes are scheduled without any overlap between TT windows. The advantage of this approach is ease of implementation. Moreover, for such an implementation, the network devices and the communication protocol do not require modification in order to enable mode-changes. However, using this approach, scheduling is difficult for non-trivial network loads. Furthermore, this approach leads to bandwidth loss as TT slots for all VLs occupy bandwidth in the scheduling table, even when the message for the scheduled VL is never/seldom ready for transmission during run-time.

Figure 3.4 shows the PL schedule generated using the individual-schedule approach, where there exists a PL schedule for each mode of operation. This type of scheduling table is used where the network devices are capable of handling mode-changes. The figure shows the schedules for two different mode-change types (see Section 1.1.2). The dark circles on the time-line in Figure 3.4b represent mode-change blackouts [Foh94]. This approach is easy to implement and to schedule. However, such an implementation requires support from the network hardware and the communication protocol.

3.4 Mode-Stacking Approach

In this section, we present the mode-stacking approach to enable mode-changes in COTS network devices which do not explicitly support mode-changes. In Section 3.4.1, we present the generic concept of the mode-stacking approach. In Section 3.4.2, we provide details on how to implement deferred and immediate mode-changes using mode-stacking approach. In Section 3.4.4, we discuss the advantages and disadvantages of the mode-stacking approach. And in Section 3.4.3, we discuss how a stacked scheduling table can be generated.

3.4.1 Modes of Operation

In order to enable multiple modes of operation in network devices which do not support them, we schedule multiple TT slots at the same time if they belong to stackable virtual links. In the following, we provide the definition of the term stackability for a set of candidate virtual links as presented by Heilmann et al. [HSF16].

Definition 3.1. [HSF16] Two or more virtual links are termed as *stackable*, if, during run-time, only one message is ready to be transmitted at the same time from these virtual links.

Examples for stackable messages include messages sent from a sender node to different receiver nodes based on different system/environmental state (i.e., XOR constraint defined by Fohler [Foh94]) or messages sent during different modes of operations. Although Definition 3.1 is quite generic, it makes sense to limit the scope of stackability to *operation* modes or *service* modes, since *design* modes only affect the development phase of the system (see Section 1.1.2 for details on mode types).

By definition, two virtual links which belong to two different modes of operations are stackable. However, depending on the application requirements, multiple modes of operation may also share the same virtual links, e.g., the display management system in an aircraft is required during take-off, auto-pilot and landing. When a set of virtual links are found to be stackable, this property can be exploited to support multiple modes in COTS TT network devices. An example of stacked TT slots for multiple modes is shown in Figure 3.2b, where the transmission slots of virtual links v_1 and v_3 are stacked.

In order to enable virtual link stackability in COTS TT network devices, the following two conditions (as presented by Heilmann et al. [HSF16]) must be satisfied:

Condition 3.1. [HSF16] The stackability property for all pairs of virtual links hold during system operation.

Condition 3.2. [HSF16] The network devices are able to accept *and* service stacked schedules for all physical links during run-time.

Condition 3.1 can be satisfied by system design, for instance, by employing multiple modes and making sure that all network nodes switch modes simultaneously. Whereas, Condition 3.2 relates to the capabilities of the COTS network devices.

3.4.2 Mode-Changes

With the mode-stacking approach, the deferred mode-changes (see Section 1.1.2) can be implemented without major modifications in the schedule or the network devices. In order to guarantee a valid and deterministic network state, it is a good practice to allow mode-change execution only by a certain network node(s) [KG94]. In the following, we call such node(s) *master* node(s). This enables centralized resource management of the network similar to the hierarchical resource management defined by Koller et al. [KGGP+16].

In order to support deferred mode-changes, a number of virtual links to the master node for mode-change *request*s have to be added to the network. Similarly, a virtual link for a mode-change *decision* from the master node to the other nodes in the network needs to be added. The new virtual links are used to receive mode-change requests by the master node, and to carry the mode-change decision from the master node to the rest of the network. If a mode-change message is received on the mode-change decision VL by the slave (i.e., non-master) nodes, they change modes at the end of the hyper-period.

In order to support immediate mode-changes, the following conditions have to be satisfied in addition to conditions 3.1 and 3.2.

Condition 3.3. Mode-change blackout [Foh94] information is generated by the scheduler.

Condition 3.4. Changing the mode of operation in a certain network node and not in the other nodes also leads to a valid mode of operation.

In order to have a global notion of a mode, a deferred mode-change might be required after an immediate mode-change. On the contrary, when the mode-change is only required in a certain node (e.g., for fault-containment), a deferred mode-change is not necessary. In either case, Condition 3.4 must be satisfied. Note that, in case of an immediate mode-change, TT virtual links for mode-change requests and decision are not required.

3.4.3 Schedule Generation

The scheduling tables for mode-stacking approach are similar to the individual-schedule approach (see Section 3.3.2), however this approach differs in their implementation. During the scheduling process, the scheduler can overlap the transmission slot of a virtual link with a transmission slot of other virtual link(s), as long as the overlapping duration only contains TT windows of stackable virtual links. At the end of scheduling, all the (stacked and non-stacked) TT slots are merged in a single scheduling table, which utilizes network bandwidth efficiently compared to the super-schedule approach. As this approach results in a single scheduling table, the network devices do not require an online mode-change (contrary to the individual-schedule approach), however, the processing nodes or end-systems might require this information to maintain a global notion of a mode of operation. This information is communicated using the mode-change request and decision VLs (see Section 3.4.2). These VLs make sure that Condition 3.1 is satisfied for all end-systems in the network.

3.4.4 Pros & Cons

Other than providing support for mode-changes, the mode-stacking approach provides a number of advantages. Compared to the non-stacking approaches, stacking approach requires less network bandwidth (assuming that stackable virtual links or tasks exist). This enables the augmentation of new features and components or the allocation of more bandwidth to existing ones. Moreover, compared to the super-schedule approach, the mode-stacking approach can improve the response-times of messages in some cases by transmitting a message at an earlier point in time due to stacked TT windows. Furthermore, since the stacking approach requires stacked schedules to be written to a single scheduling table, the network does not need to undergo network reconfiguration.

Even though the mode-stacking approach efficiently utilizes network bandwidth (compared to the super-schedule approach), there might be situations in which stacking does not lead to maximum bandwidth reuse. Due to the difference in the TT slot width b

of different stacked virtual links, the stacking approach may lead to over-provisioning. Consider two stackable virtual links v_1 and v_3 as shown in Figure 3.2b. In the figure, notice that the network remains idle between the TT windows of virtual links v_3 and v_2 for mode O_2. However, this bandwidth cannot be utilized by v_2 (even when the message for v_2 is ready during this time) since v_1 and v_2 are not stackable. As a result, the TT scheduling window for v_2 in mode O_2 has to be shifted until the end of the largest TT window amongst all the stacked TT windows preceding v_2 (i.e., at the end of the TT window for v_1 in the figure).

3.5 TTEthernet Scheduler

In this section, we provide details of the schedule generation approach which we employ for the TTEthernet scheduler. In Section 3.5.1, we discuss how the TTEthernet scheduler can generate physical link schedules by generating a phase for each virtual link. In Section 3.5.2, we discuss how to extend the phase generation approach to generate multi-mode schedules. And in Section 3.5.3, we define how the Symmetry Avoidance Scheduler (SAS) (see Chapter 2) is extended to support TT traffic scheduling in TTEthernet networks.

3.5.1 Phase Generation

The Symmetry Avoidance Scheduler (SAS), presented in Chapter 2, traverses the search-tree by *iterating* over all possible schedules of a busy-period at a defined point in time t in the schedule. Due to the *iterative* nature of the scheduler, we use the pair-wise phase generation approach defined by Marouf et al. [MS10] to generate phases for all virtual links (VL) on each physical link (PL).

In the pair-wise phase generation method [MS10], a *pair-wise phase-set* $\Gamma_{(i,k)}$ is generated for each pair of strictly periodic VLs v_i and v_k using Equation 3.1. For a given set of phases for already scheduled VLs, a set of feasible phases for a virtual link v_i, having a newly released message on the physical link at time t, is termed as *comprehensive phase-set* Γ_i. The comprehensive phase-set Γ_i for v_i is calculated using pair-wise phase-sets as defined in the following equation:

$$\Gamma_i = \bigcap_{k=1}^{q} \Gamma_{(i,k)} \tag{3.2}$$

where, q represents the total number of scheduled VLs on the PL.

In a typical TTEthernet network, the *macrotick* size is usually very small, i.e., 50μs to 250μs [CO16], which can lead to large cardinality of Γ_i. Therefore, we generate *intervals* of valid phases instead of all valid phases for the VLs. This reduces the memory requirement and the execution time of the phase generation method.

In order to generate the nth *interval* of valid phases for a VL from Equation 3.1, the interval start is calculated by substituting $m = b_i$, whereas the interval end is calculated using $m = GCD(p_i, p_k) - b_k$. Generating all n valid intervals (i.e., $\forall n | \phi_k < GCD(p_i, p_k)$)

results in $\Gamma_{(i,k)}$. To calculate a comprehensive phase-set Γ_i using Equation 3.2, we use interval-trees to perform quick intersection of intervals [CLRS01].

3.5.2 Schedule Generation Methodology

In order to generate multi-mode schedules for TTEthernet, the pair-wise phase-set $\Gamma_{(i,k)}$ for two *stackable* VLs v_i and v_k is excluded when estimating comprehensive phase-set Γ_i using Equation 3.2. The exclusion of such phase-sets can lead to stacking/overlap of multiple TT windows and, therefore, low bandwidth loss is ensured.

In COTS TTEthernet devices, the TT slots of different VLs can overlap as long as their start is shifted by a small amount η [HSF16]. The required shift η depends on several factors, e.g., synchronization precision, device event processing speed, etc. This shifting feature of the COTS TTEthernet devices can be exploited to implement mode-changes by stacking/overlapping shifted TT slots, as long as the latency constraints are larger than the slot shift η. In real applications, the value of η is usually a few microseconds while the latency constraints are in the order of several milliseconds [CO16, BDNM16].

In order to satisfy Condition 3.2 for COTS TTEthernet devices, the stacked TT slots are shifted by η, depending on the stack height. Once the stacked scheduling tables for all modes are shifted, they can be merged together in a single scheduling table and written to the TTEthernet device specification (DS) file (see Section 1.1.5). An example of the TT schedule for a TTEthernet PL is shown in Figure 3.5.

Note that we are only concerned with generating schedules for deferred mode-changes in this work and, therefore, we do not generate mode-change blackout information. Since, in this case, Condition 3.3 is not satisfied, immediate mode-changes are not supported by our TTEthernet scheduler. This problem will further be explored in Chapter 5.

3.5.3 Scheduler Design

In this section, we consider the scheduling problem of both tasks and virtual links. For this purpose, we extend our Symmetry Avoidance Scheduler (SAS) presented in Chapter 2 using the same task model and the search-tree mapping. We call this scheduler extension for TTEthernet *SAS-TTE*. As mentioned in Section 3.2, the messages are assumed to be released at the end of a task (or at the end of transmission on a predecessor PL on the VL route). Therefore, response-time pruning based on ready jobs is enough and does not lead to traversal of all symmetric link schedules.

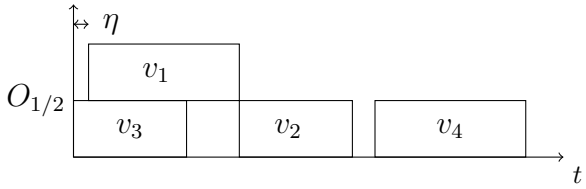


Figure 3.5: *Multi-mode TT schedule from Figure 3.2b modified for TTEthernet*

During the search-tree exploration, the `ScheduleNetwork()` function (in Algorithm 2 from Chapter 2) performs multiple operations in a specific order; (i) it updates the message queues for each PL based on the finished jobs, (ii) it sorts the message queues based on the message deadlines, (iii) it generates the comprehensive phase-sets Γ for all new messages in the message queues, and (iv) it selects the first phase from the generated comprehensive phase-sets as the phase ϕ for TT slot s. The phase for a slot s_i for VL v_i on a PL is changed in accordance with the comprehensive phase-set Γ_i upon backtrack and re-*spawn*.

A number of authors defined various types of constraints that a schedule for TTEthernet must satisfy, e.g., [Ste10, PRNSH16, CO16]. In the following, we specify how the constraints presented by Craciunas et al. [CO16] are guaranteed to hold by SAS-TTE. (i) Frame-constraint (i.e., phase of each frame is not negative and not large enough that the scheduling window is out of the period.) is satisfied by limiting the phases within the period. (ii) Link-constraint (i.e., no two scheduling windows on the same physical link overlap) is guaranteed by using pair-wise phase generation method developed by Marouf and Sorel [MS10]. (iii) Virtual-link-constraint (i.e., producer task is scheduled before the frame window and consumer task is scheduled after the frame window) is guaranteed by releasing the frame or consumer tasks only when the producer task finishes. (iv) End-to-end-latency-constraint (i.e., the difference between the last slot of the consumer task and the first slot of the producer task must be less than or equal to the latency constraint) is guaranteed by backtracking upon violation of latency constraint. (v) Task-constraint (i.e., all the tasks execute within their offset and deadlines) is guaranteed by releasing a job only after release offset and backtracking upon deadline miss. (vi) Virtual-frame-sequence-constraint (i.e., no two jobs on the same PN overlap) is guaranteed by specifying single job per slot. (vii) Task-precedence-constraint (i.e., the precedence relation between tasks is satisfied) is guaranteed by releasing a consumer task only after finishing the producer task. (viii) Memory constraints (i.e., the difference between phases of a VL on two consecutive PLs on the VL path is less than or equal to a bound) is guaranteed by checking that the generated phase is always less than or equal to the VL period, i.e., the bound is equal to the VL period ([Ste10, PRNSH16, CO16]). In order to verify that these constraints do hold, a schedule verification tool is written in C++ (discussed further in Section 2.6.1).

Note that when the task scheduling heuristic is selected without considering the VL scheduling approach, the scheduler may perform frequent backtracks due to an unsatisfied strictly periodic constraint. In order to minimize backtracks, jobs with equal scheduling costs (e.g., absolute deadlines for EDF heuristic) must be executed with a defined order. In this work, we use task-index based ordering. Furthermore, the new phase assignment upon backtrack (or re-*spawn*) skips a range of valid phases such that the sub-schedule problem responsible for the backtrack may be avoided.

3.6 Evaluation

In this section, we evaluate the SAS extension of our modular scheduler from Chapter 2. To define end-system (ES) load, in this section, we will be using notations from

Chapter 2. In the following, Section 3.6.1 provides a description of the experimental setup. Section 3.6.2 provides detailed results of the single mode scheduler. Finally, Section 3.6.3 presents the evaluation results of the mode-stacking approach.

3.6.1 Experimental Setup

In order to evaluate the performance of the SAS-TTE scheduler for scheduling TT tasks and network traffic, we follow the evaluation strategy used by Craciunas et al. [CO16] and generate task-sets with four different system sizes and three different network topologies (i.e., mesh, ring and tree) as mentioned in Table 3.2. To generate task-sets, we use the tool Config Generator (CG) developed by Craciunas et al. [CO16] and use their own generation parameters which, according to them, represent practical workloads. This allows a direct comparison between the results acquired from the SMT based scheduler and our scheduler extension SAS-TTE. The tool CG generates a set of task-set parameters (using random distribution) for real-time tasks, a TTEthernet network and a mapping between the two. It also generates the VL routes for each VL based on the mapping of the tasks.

In order to generate the input task-sets, we use three different sets of periods: $P_1 = \{50, 75\}$ms, $P_2 = \{10, 20, 25, 50, 100\}$ms and $P_3 = \{10, 30, 100\}$ms leading to scheduling cycle lengths of 150ms, 100ms and 300ms, respectively. Each end-system (ES) contains in total 16 tasks, half of which were set to communicate through the network[3]. Identical to the industrial applications, the communicating tasks in this experiment accounted for 25% of the ES utilization. The WCET C of a task was calculated using the period T of the task-graph (TG) and the desired ES utilization. The VLs were created between random pairs of communicating tasks executing on distinct ESs, limiting the number of tasks per TG to two[4]. Moreover, the message sizes n for the VLs were chosen between

Table 3.2: *System sizes of generated task-sets for the evaluation of SAS-TTE*

Size	Topology	Switches	ESs	Tasks	VLs
Small	Mesh, Ring	2	4	64	16
	Tree with depth = 1	4	6	96	24
Medium	Mesh, Ring	4	16	256	64
	Tree with depth = 2	13	36	576	144
Large	Mesh, Ring	8	48	768	192
	Tree with depth = 3	15	48	768	192
Huge	Mesh, Ring	16	192	3072	768
	Tree with depth = 2	43	432	6912	1728

[3]Note that a virtual link has at least two ends, a source and a destination. Four source ends of virtual links from one end-system also require destination end-system(s). This resulted in eight communicating tasks per PN, in total, by the task-set generator.

[4]The task-set generator developed by Craciunas et al. [CO16] limited the number of tasks per TG to two. Changing this limit may fail to provide a directly comparison between the evaluation done for

the minimum and maximum Ethernet frame sizes, i.e., between 64 to 1542 bytes. The PL speed μ between an ES and a switch was set to 100Mbps, while μ was set to 1Gbps for the PLs between two switches. For each generated task-set, the SAS-TTE scheduler was run for at most one hour, after which the scheduler was terminated and the task-set was deemed unschedulable.

To evaluate the scheduler, we generated 1000 task-sets for each configuration in Table 3.2 with a mean ES utilization of 50% and 75%. Moreover, the ES *slot* size was restricted to 250μs while the PL *macrotick* size was fixed at 1μs. Since the task-set generator by Craciunas et al. [CO16] allocates the tasks to ESs in order to create VL routes, SAS-TTE does not re-allocate them. Furthermore, the number of *spawn*ed nodes c was set to 10, while the experiments were run on an Intel® Core™ i7-4790 (8 logical cores, 3.6MHz base frequency and 8MB L3 cache).

In order to verify that the generated schedules are indeed correct, schedule verification tools were written in C++. The verification of task-only schedule generated by SAS-TTE is done with sanity checks and constraint validation checks. The sanity checks included verification that (i) the generated artifact is within defined range, e.g., a job does not start before earliest start time or a job does not finish after absolute deadline, (ii) no artifact is missing, e.g., each job executes for complete WCET C, and (iii) no extra artifact is present, e.g., a task is only allocated to single PN. The constraints validation included checks like, each job finishes before its absolute deadline, each successor job executes after the predecessor job, etc. Other than the task constraints validation (mentioned earlier), the constraints specified by Craciunas et al. [CO16] were used for TTEthernet schedule verification.

In the following, we divide the evaluation process in two experiments. The first experiment evaluates the performance of SAS-TTE for generating a single mode schedule for small to huge network sizes, while the second experiment compares the performance of the scheduler for huge network sizes (see Table 3.2) with and without the mode-stacking approach. The first experiment makes use of all the generated task-sets (defined above), while the evaluation for the second experiment was restricted to task-sets with period-set P_3, huge network size and tree topology (due to their lower-than-maximum schedulability ratio in the first experiment).

For the second experiment, the tasks and messages for each ES in a task-set (for huge networks with period-set P_3 and tree topology) were divided in two modes of operation. The division between the two modes was made such that random 30% of the tasks and messages were assigned to each mode, while the remaining 40% were assigned to both modes. This resulted in an average load of 70% (compared to the original task-set) per mode of operation on the ESs and the physical links.

3.6.2 Single Mode Scheduling

Figures 3.6, 3.7 and 3.8 show the evaluation results of the SAS-TTE scheduler, where different line types represent different period-sets. In the figures, the x-axis shows the system sizes, while the y-axis shows the SAS-TTE schedulability ratio with line-plots

SAS-TTE and the scheduler proposed by Craciunas et al. [CO16].

and their run-times with box-plots. In the figures, a schedulability ratio of 1.0 indicates that all of the 1000 generated task-sets were scheduled by the SAS-TTE scheduler.

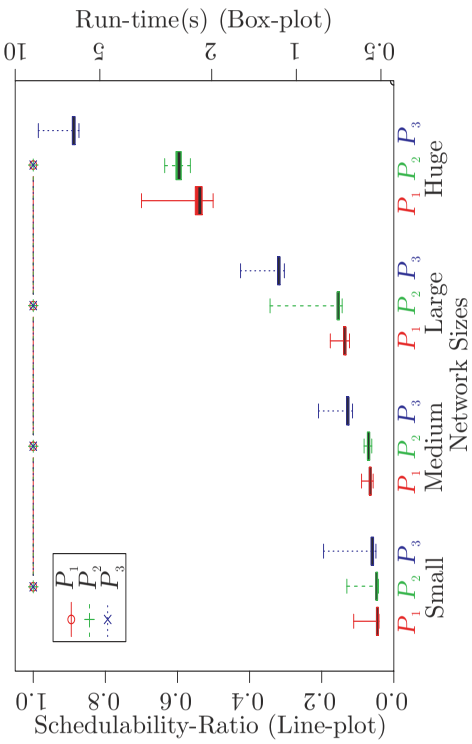
Figure 3.6 shows the schedulability ratio and run-times for the task-sets with mesh topology. The figure shows that SAS-TTE generated valid feasible schedules for each task-set within 8 seconds. Moreover, SAS-TTE only took around 2 seconds for small to large task-sets even for 75% mean ES utilization. In Figure 3.7, the schedulability ratio and run-times for task-sets with ring topology are shown. Note that SAS-TTE scheduled each generated task-set within 10 seconds, most of which finished well under 2 seconds even for 75% mean ES utilization. Likewise, Figure 3.8 shows the schedulability ratio and run-times for task-sets with tree topology. The figure shows that the scheduler took around 3 seconds to schedule small to large task-sets while the run-times for huge task-sets ranged from 7 to 35 seconds. It is worth noting that SAS-TTE could schedule all the task-sets except the combination of huge network sizes and period-set P_3 (where around 40% of the task-sets were scheduled).

In order to understand why SAS-TTE was unable to schedule a number of huge task-sets with the tree topology, figures 3.9 and 3.10 show the distribution of certain parameters related to the network load for the task-sets with 75% mean PN utilization and two different topologies (i.e., ring and tree). In Figure 3.9, the x-axis shows the period-sets while the y-axis shows the mean (Mn) and the maximum (Mx) PL utilization. Note that the difference between Mn and Mx utilization roughly identifies the distribution of the network load across all PLs. In Figure 3.10, the x-axis shows the period-sets while the y-axis shows the maximum (Mx) number of (Ethernet) frames per PL and the total (T) number of frames in the task-set. Note that the number of frames (termed as *frame instances* by Steiner [Ste10] and Pozo et al. [PRNSH16]) in the figure on the PLs signifies the number of constraints the scheduler needs to resolve.

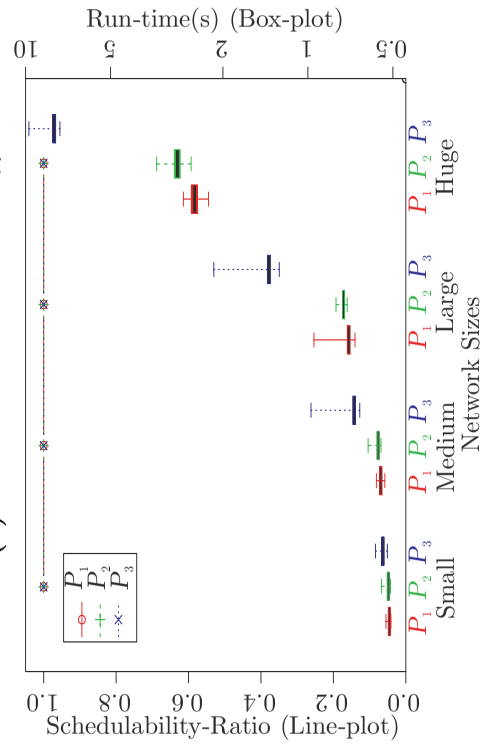
For the case of period-set P_3, figures 3.9 and 3.10 show that the deviation of the network load distribution across the PLs is significantly higher for task-sets with tree topology compared to that of ring topology. This indicates that, other than the period-set, the number of frames to be scheduled on a PL has a great impact on schedulability. Therefore, for such cases, it becomes difficult to find valid phases for all the messages on that particular PL. This justifies the results shown in Figure 3.8 and provides pointers for future work.

Comparing our results with those from Craciunas et al. [CO16], the symmetry based scheduler provides improved schedulability especially for huge system sizes. Note that for ring and tree topologies, Craciunas et al. [CO16] were unable to schedule task-sets with 50% mean ES utilization even after 10 hours of scheduler run-time. In contrast, this only took a few seconds with the symmetry based scheduler. Moreover, our scheduler generated valid TT schedules for task-sets with 75% mean ES utilization without using a demand-based scheduling technique [CO16]. These improvements in our scheduler stem from the fact that we used analytical techniques to generate valid phases for TT messages and utilized problem specific search optimization techniques.

In comparison with the results from Pozo et al. [PRNSH16], the symmetry based scheduler is able to generate schedules for huge task-sets within a few seconds. Notice that the number of frame instances in our huge task-sets with tree topology and period-

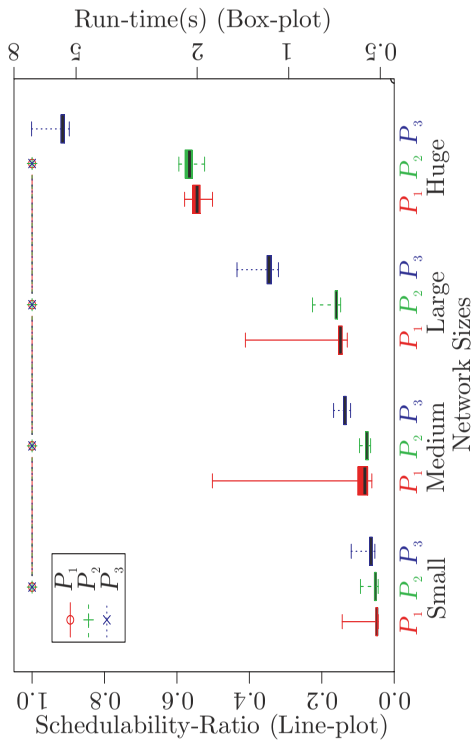


(a) *Mean ES utilization = 50%*

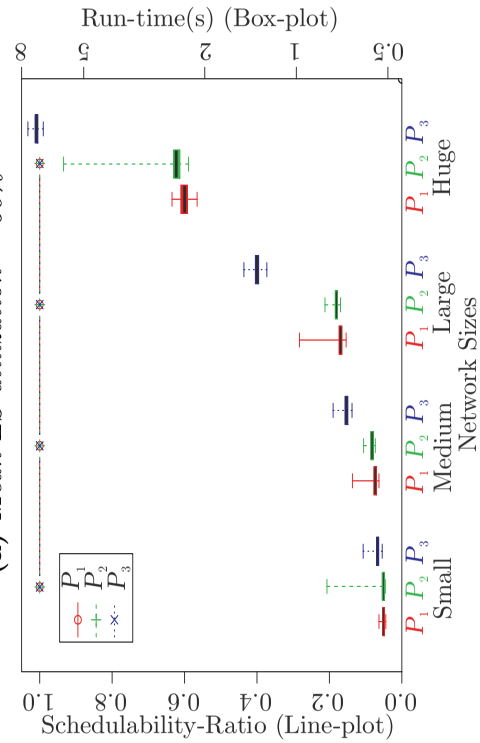


(b) *Mean ES utilization = 75%*

Figure 3.6: *Performance of SAS-TTE with mesh topology*

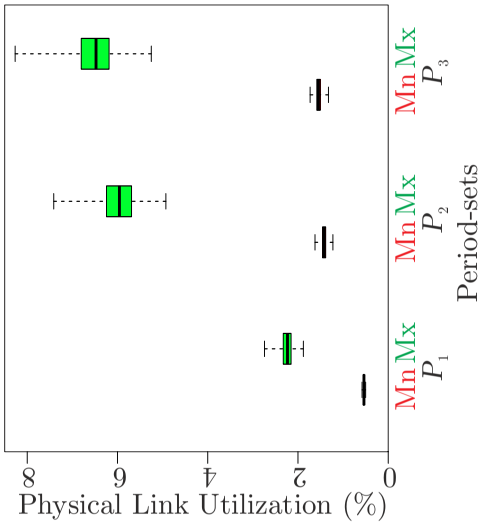


(a) *Mean ES utilization = 50%*

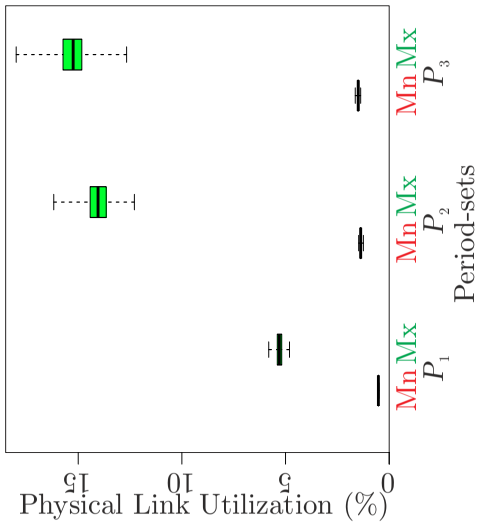


(b) *Mean ES utilization = 75%*

Figure 3.7: *Performance of SAS-TTE with ring topology*

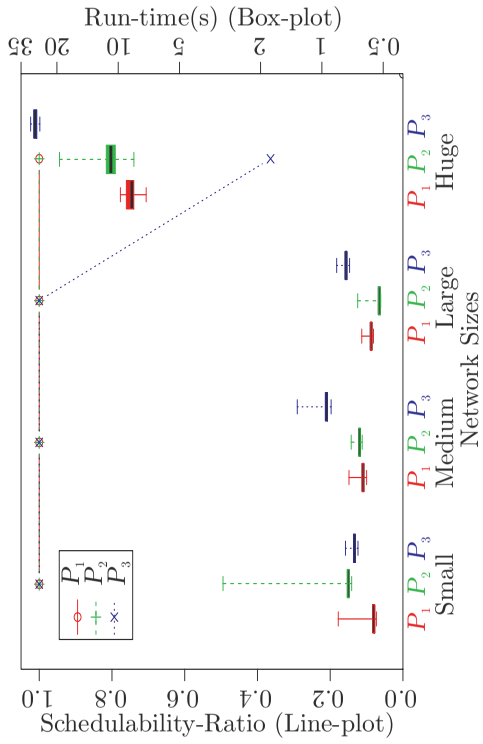


(a) *Ring topology*

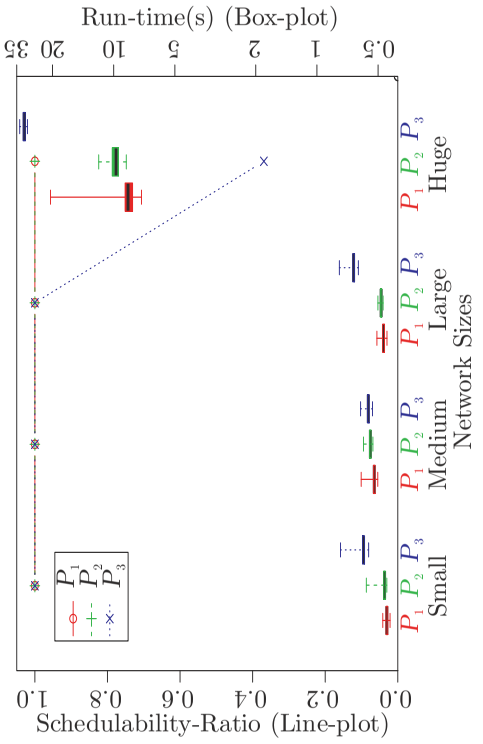


(b) *Tree topology*

Figure 3.9: *Physical link utilization (75% mean ES utilization, huge size)*

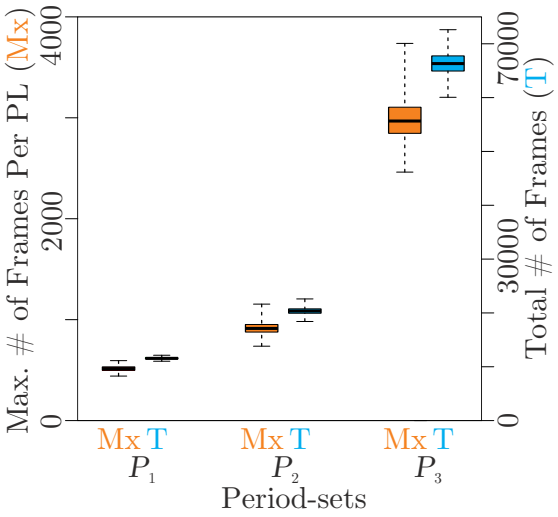


(a) *Mean ES utilization = 50%*

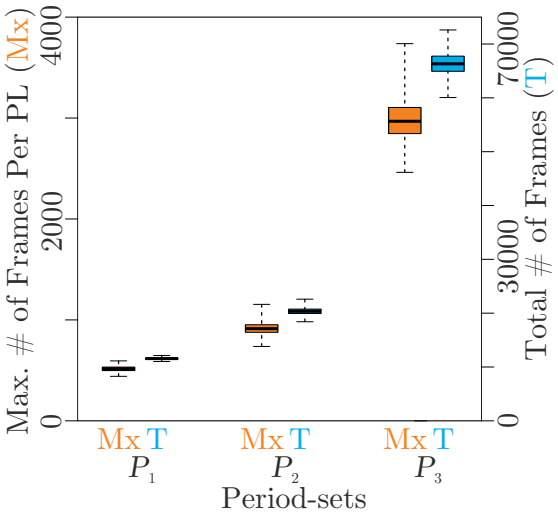


(b) *Mean ES utilization = 75%*

Figure 3.8: *Performance of SAS-TTE with tree topology*



(a) *Ring topology*



(b) *Tree topology*

Figure 3.10: *Ethernet frame count for task-sets (75% mean ES utilization, huge size)*

set P_3 are very close to their *normal low distribution* task-sets with 10000F, for which they require about 5 minutes to generate schedules. Note that the results from their adaptive extension [PRNHS17] are not comparable to ours, since they did not generate task schedules and used harmonic periods for evaluation. It is important to mention that the comparison between the results from Pozo et al. and our solution is very rough as there exist differences between the sets of constraints manageable by the two proposed schedulers: (i) they used artificial application constraints, while we generate the task schedules, (ii) they used large schedule *raster* in order to reduce the complexity (which also reduces the number of feasible schedules), while we do not, (iii) their approach might not find the only solution (due to parameters like *segment size*), while our approach will eventually find it, (iv) they consider simultaneous relay constraints, while

we do not, and (v) the physical link utilization in their synthetic task-sets, although non-realistic [CO16], is significantly higher than the ones we generated. Note that the ability to schedule task-sets with higher physical link utilization does not indicate superiority of an algorithm. The problem of scheduling strictly periodic tasks/messages is NP-Complete ([MS10, Coe17]), and therefore no guarantees can be provided that the scheduler by [PRNSH16] will be able to schedule network load from our synthetic but practical task-sets and vice-versa.

3.6.3 Stacking Evaluation

A comparison between the scheduling performance of SAS-TTE with super-schedule approach (from Figure 3.8) and mode-stacking approach (with two modes of operation) is shown in Table 3.3. The table presents the schedulability ratios and run-times of the two implementations. In the table, notice that (i) the mode-stacking approach leads to schedulability of all task-sets and (ii) the maximum run-times for the mode-stacking approach are less than the minimum run-times of the super-schedule approach. This evaluation demonstrates that the mode-stacking approach performs better than super-schedule approach for supporting multiple modes of operation in COTS TTEthernet devices.

Table 3.3: *Performance of SAS-TTE for multi-mode scheduling (huge tree topology and period-set P_3)*

	Super-Schedule		Mode-Stacking	
	Mean ES Utilization		Mean ES Utilization	
	50%	75%	50%	75%
Schedulability Ratio	0.364	0.369	1.0	1.0
Min. Run-Time (s)	24.258	31.058	19.932	21.556
Avg. Run-Time (s)	25.554	32.271	21.223	23.008
Max. Run-Time (s)	26.916	33.859	24.275	29.732

Chapter 4

Online Admission of Aperiodic tasks

Modern industrial applications are subject to several safety constraints. Verification and validation (V&V) processes utilized for such applications define if the product can be deployed for the desired application or sold in a particular market. However, the V&V process requires rigorous evaluation of the system, and therefore, it increases in complexity as the system size increases. Partitioning or virtualization enables certification of a sub-system irrespective of the behaviour of other sub-systems, and as a result, reduces the V&V efforts [ENNT15].

In addition to the safety constraints, a computing system may need to handle several real-time constraints depending on the application requirements. The characteristics of such systems strongly depend on the used model of computation, i.e., time-triggered (TT) or event-triggered (ET). In real-time systems, periodic tasks can easily be serviced using TT mechanisms due to predictable repetitive arrival patterns. However in the case of aperiodic tasks, TT mechanisms like offline bandwidth reservation may lead to over-provisioning, and therefore, may increase cost per feature. The over-provisioning problem becomes increasingly prohibitive with virtualization technologies as these technologies may lead to significant bandwidth loss [LWVH12]. Therefore, a new algorithm needs to be developed which provides efficient integration of periodic and aperiodic activities in hierarchic systems.

Besides bandwidth reservation, a number of techniques are proposed over the last few decades to integrate TT and ET activities. For instance, Slack Stealing [Leh92] is used for online admission of aperiodic tasks in fixed-priority systems, whereas, slot shifting [Foh94] is used with EDF (Earliest Deadline First). The basic operating principle for both of these approaches is to estimate the information about the amount and distribution of free resources offline and to accommodate aperiodic tasks online. A number of extensions were also proposed for both of these algorithms to compensate for new constraints, e.g., sporadic tasks [IF00], non-preemptive aperiodic tasks in a preemptive scheduling environment [Sch15] and Vestal's [Ves07] mixed-criticality task model [The15]. Although these extensions are viable solutions, they need to frequently keep track of free resources, they are computationally expensive and they cannot be applied on hierarchic scheduling schemes.

In this chapter, we present an algorithm which defines availability of free resources in an offline TT scheduling table of non-preemptive tasks, and employs the availabil-

ity information online to service non-preemptive aperiodic tasks in a partitioned system; we call this algorithm 'Job-Shifting'. Although non-preemptive scheduling offers several benefits (e.g., reduced I/O delays [NBFK14], precision in the estimation of WCET [RM08], etc.), it is an NP-hard problem [JSM91]. Consequently, the non-preemptive versions of EDF (e.g., npEDF) are not optimal [NBFK14]. Therefore, the job-shifting algorithm is not designed to confine the application integrator to the EDF strategy (unlike slot shifting [Foh94, The15, Sch15]) and can utilize either online or offline scheduling strategies. The job-shifting algorithm provides aperiodic task execution guarantees similar to the bandwidth reservation techniques, which means that there is no need to modify the certification process of the safety critical application. Furthermore, the job-shifting algorithm provides control over task jitter, can be used with sparse and dense time-bases [Kop92] and does not require computationally expensive slot-based record keeping mechanism.

We highlight the advantages and disadvantages of the job-shifting algorithm and provide a necessary condition for online aperiodic admission which can be checked offline. Furthermore, we also identify which factors affect the overheads and complexity of the algorithm implementation. Through simulation, we demonstrate that the proposed algorithm efficiently utilizes free resources in hierarchic schedules. Moreover, we evaluate the overheads introduced by the job-shifting algorithm on the Zynq ZC706 board[1], and demonstrate that the incurred overheads (i) are small and (ii) are comparable to the overheads incurred by the state-of-the-art *preemptive* approaches.

The reminder of this chapter is organized as follows; Section 4.1 presents an overview of the related work. Section 4.2 introduces the system model utilized by job-shifting algorithm. Section 4.3 provides a detailed description of the job-shifting algorithm, and highlights the viability of the algorithm. Section 4.4 presents a discussion on how to improve job-shifting performance. Section 4.5 provides the description of the experimentation for the job-shifting efficiency evaluation. Finally, Section 4.6 presents the overhead evaluation experiment and the job-shifting implementation in the EU FP7 DREAMS project[2].

4.1 Related Work

A trivial approach to service aperiodic tasks is to use interrupt mechanism. However with this approach, it is difficult to provide guarantees due to complex interference patterns [Bra11] and the methodology is strongly dependent on the hardware platform. A better strategy is to use background processing to service aperiodic tasks. However, background processing provides no guarantees for aperiodic task execution and leads to large aperiodic response-times.

Over the last few decades, a strong theoretical background has been established for the bandwidth reservation technique (aka aperiodic servers). The technique is employed in safety critical systems to service sporadic and aperiodic tasks [BDNM16] due to ease

[1] `http://www.xilinx.com/`
[2] `http://dreams-project.eu/`

of implementation and less V&V efforts. In the bandwidth reservation technique, the aperiodic or sporadic tasks are transformed to periodic tasks by defining the parameters *period* and *budget*. Although it is debatable how these parameters can be defined for an aperiodic activity, the values for these parameters are selected based on educated-guesses or best-practices. Moreover, the bandwidth reservation technique leads to bandwidth loss when (i) partitioning [ARI03] is used [LWVH12] and when (ii) the aperiodic activation frequency is less compared to the estimated frequency. Furthermore, the aperiodic response-time with this technique strongly depends on the server type, server priority and scheduling strategy (i.e., fixed- or dynamic-priority scheduling).

As the approaches mentioned above present several bottle-necks, from this point on we will focus on the online admission paradigm for aperiodic task execution. As there exist a significant number of algorithms which utilize online admission approach (e.g., [YG09, AE07]), we will only discuss two major ones (i.e., slack stealing [Leh92] and slot shifting [Foh94]) due to their support for variety of system models.

In the early 90s, Lehoczky et al. defined slack stealing algorithm to service soft [Leh92] and hard [TL94] aperiodic task in fully-preemptive fixed-priority systems. In this algorithm, they precomputed and stored the *slack function* of the task-set and used it online to service aperiodic tasks. The slack stealing algorithm was later extended by Tia et al. [TLS96] to efficiently compute the *slack function* online, since the size of the slack function table may be too large, depending on the periods of the tasks.

Similarly, Fohler [Foh94] defined slot shifting algorithm for aperiodic admission in fully-preemptive TT dynamic-priority systems (specifically EDF) with sparse time-base [Kop92]. In this algorithm, *capacity (or slack) intervals* and their *spare capacities* are calculated offline and stored in a table to be used online to service firm and soft aperiodic and sporadic tasks [IF00]. The slot shifting algorithm requires a significant amount of memory and utilizes an online mechanism to keep track of used resources activated at each *slot* boundary [Kop92].

Recently, Schorr [Sch15] extended the original slot shifting algorithm to allow admission of non-preemptive aperiodic tasks in a preemptive schedule. The extension utilized the original precomputed offline scheduling table defined by slot shifting [Foh94], however the aperiodic admission test (termed *acceptance test* [Foh94]) for non-preemptive aperiodic tasks was modified to calculate enough consecutive slots to execute the released aperiodic job. Moreover, the *guarantee algorithm* [Foh94] was modified to improve the response-time at the cost of flexibility [Sch15]. Similar to the original slot shifting algorithm, the extension by Schorr requires a significant amount of memory and uses an online slot based record keeping mechanism. Moreover, the slot shifting extension proposed by Schorr has higher complexity and larger run-time overheads.

Similarly, Theis [The15] extended the slot shifting algorithm to support mixed-critical tasks (based on Vestal's mixed-criticality task model [Ves07]) and mode-changes. Similar to the original slot shifting algorithm, the extension by Theis is based on EDF, however the memory requirements (and therefore the overheads) are increased proportional to the number of modes or criticalities.

In this chapter, we present job-shifting algorithm, which can be used in industrial hierarchical mixed-critical systems (*not* based on Vestal's mixed-criticality task

model [Ves07]) to admit non-preemptive aperiodic tasks. Unlike slot shifting algorithm [Foh94, Sch15, The15], the job-shifting algorithm does *not* require (i) slot-based record keeping mechanism, (ii) sparse time-base [Kop92] or (iii) EDF scheduling. The job-shifting algorithm respects separation of concerns (through partitioning), incurs small scheduling overheads compared to the state-of-the-art and provides better response-times for aperiodic activities. The job-shifting algorithm also provides offline guarantees and control over task jitter. However, job-shifting only supports non-preemptive scheduling, which is NP-hard even for the simple case of independent tasks with implicit deadlines [JSM91].

4.2 System Model

A partitioning kernel or hypervisor is used to provide strict temporal and spatial isolation [IEC10, DC11, ISO11], which enables independence of safety functions between applications. In order to fulfill strict safety requirements, cyclic executive scheduling [ARI03] is assumed to be the inter-partition scheduling strategy. We assume a distributed system with single core processing nodes in a non-preemptive execution environment. Moreover, all activities in the system are assumed to be triggered by the passage of time, i.e., Time-Triggered with sparse or dense time-base [Kop92] (see Section 1.1.3).

In order to apply job-shifting to a partition p_i in a partition set P, it is assumed that the intra-partition scheduling employs TT scheduling tables with a simple online dispatcher (relaxed in Section 4.4.6), and contains both periodic and aperiodic tasks. The idle time inside a partition p_i is also assumed to be non-zero (discussed further in Section 4.4.2). The time when the partition p_i is not available to service applications is defined by the blocking set V_i. Each blocking $v \in V_i$ is defined by the tuple $\langle b, m \rangle$, where b and m represent the absolute beginning and termination time of blocking v, respectively.

A task-set Γ is defined as a collection of tasks. Each task $\tau \in \Gamma$ is defined by the tuple $\langle \phi, C, T, D, Y \rangle$, where ϕ represents the task phase, and C represents the worst-case execution time (WCET) of the task τ. When the task is periodic, the parameter T defines its period; otherwise, $T = \infty$. Moreover, D defines the relative constrained deadline (i.e., $D \leq T$), and Y represents the task criticality (i.e., the safety level, see Section 1.1.2 for details). A task $\tau \in \Gamma$ consists of infinite jobs j, each of which is defined by the tuple $\langle r, d \rangle$, where r represents the absolute job release time, and d defines the absolute job deadline. The jobs j are assumed non-preemptive, however they can be paused at the partition boundary (further discussed in Section 4.6.1).

The scheduling table S_p for a partition p is constructed prior to the job-shifting offline phase (discussed in Section 4.3.1) for the length of the scheduling cycle (SC). The length of SC is defined by the following equation;

$$\mathrm{SC} = \begin{cases} [0, \mathrm{LCM}] & \forall \tau : \phi = 0 \\ [0, \phi_{max} + 2 \times \mathrm{LCM}] & \textit{otherwise} \text{ [LM80]} \end{cases} \tag{4.1}$$

Table 4.1: *Summary of used notations and symbols*

Symbol	Description	Symbol	Description
ϕ	Task phase	a	Job activation (absolute)
T	Task period	f	Job finish (absolute)
Y	Task criticality	r	Job release (absolute)
C	Task WCET	d	Job deadline (absolute)
x	Job flexibility	D	Task deadline (relative)
SC	Scheduling cycle	R	Aperiodic room
b	Blocking start	m	Blocking end

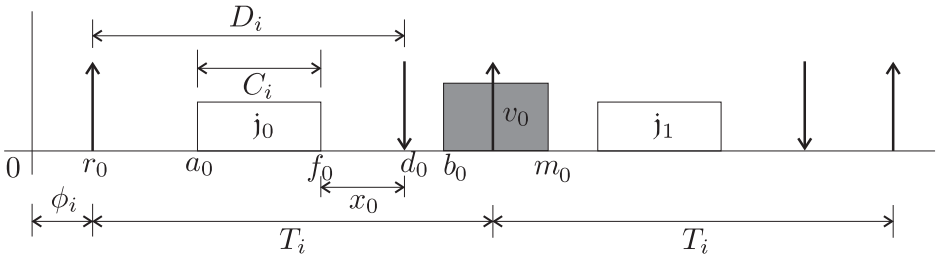


Figure 4.1: *Scheduling table example for a periodic task τ_i with blocking v_0*

where, ϕ_{max} represents the maximum phase ϕ of all tasks $\tau \in \Gamma$, while LCM represents the least common multiple of all the periods T of periodic tasks. For each job in SC, the scheduling table S_p defines the absolute job activation time a[3] and the absolute job finish time f. It is assumed that the partition scheduling table S_p, available as input to the job-shifting algorithm, is a valid feasible schedule. The tasks $\tau \in \Gamma$ may have precedence and/or mutual exclusion constraints. However, these constraints are assumed to be resolved either by modifying the job release time r and deadline d or by constructing the scheduling table S_p. An example scheduling table for a periodic task τ_i with blocking v_0 is shown in Figure 4.1, while a summary of the used notations is provided in Table 4.1.

4.3 Job-Shifting Algorithm

In this section, we provide details of the job-shifting algorithm to admit non-preemptive aperiodic tasks in flat and hierarchical scheduling models. Without loss of generality, in this section we assume that the sparse time-base [Kop92] is used. Moreover, all the jobs are assumed to execute for the complete worst-case execution time C every time (relaxed in Section 4.4.5). Furthermore, as the task criticality Y denotes the safety standard and not the importance [ENNT15] (see Section 1.1.2), its use will only be discussed in Section 4.4.1.

[3]Note that the job activation time a represents the job start time and not the job release time r. The term *activation* was used to avoid symbol overlapping.

4.3.1 Methodology

In order to apply the job-shifting algorithm, a feasible offline scheduling table S_p for partition p is required. To construct S_p, the tasks $\tau \in \Gamma$ are allocated to the processing nodes and the partitions P. The periodic tasks are then unrolled to create jobs for the length of SC and the scheduling table S_p is constructed utilizing the desired scheduler such that the feasibility is ensured. For the offline scheduled partitions, this necessarily means that the values for a_i and f_i are defined for each job j_i inside the partition such that $\forall \mathrm{j}_i \in S_p$: $r_i \leq a_i < d_i \wedge a_i < f_i \leq d_i \wedge f_i - a_i \geq C_i$[4].

Once the offline scheduling table S_p for a partition is ready, the job-shifting algorithm can be applied, which is divided in two phases; an offline phase and an online phase. In the offline phase, a parameter, we call the flexibility coefficient x_i, for each job $\mathrm{j}_i \in S_p$ is calculated (for example, see x_0 in Figure 4.1). We define the flexibility coefficient x_i as:

Definition 4.1. The flexibility coefficient x_i for job $\mathrm{j}_i \in S_p$ defines *the maximum delay, which can be added to the absolute activation time a_i of job j_i without changing the execution order of jobs in S_p and without missing any deadline.*

During the online phase of job-shifting algorithm, the scheduler checks the arrival of the aperiodic job(s). When a new aperiodic job is detected, the guarantee test is performed. If the guarantee test succeeds, the new aperiodic job is adjusted in the partition schedule by invoking the guarantee procedure. After the completion of the guarantee procedure, the job is considered *guaranteed*, i.e., it will finish its execution before missing its deadline. Contrarily, if the guarantee test fails, the aperiodic job is added to the best-effort queue. Upon each activation of the scheduler, a job on the best-effort queue can be executed if there exists enough aperiodic room R_i. We define the aperiodic room R_i as follows:

Definition 4.2. The aperiodic room R_i defines *the maximum contiguous processing node time, which can be spared for executing aperiodic job(s) prior to the activation of job j_i, without missing any deadline in the system.*

On account of the aperiodic room definition, it can be observed that the best-effort queue is not a background queue. Instead, a job on the best-effort queue is executed as soon as enough aperiodic room R_i can be reserved (further discussed in Section 4.4.2). This indicates that a job, which cannot be guaranteed to finish before missing its deadline, is not rejected at all by the job-shifting algorithm (as in slot shifting algorithm [Foh94]), instead such a job is put on a lower priority compared to guaranteed jobs. Of course, when the system model allows rejection of a job, the best-effort queue can be eliminated.

[4]Note that the condition $f_i - a_i = C_i$ holds, if there exists only one partition in the system. However for a system with multiple partitions, the condition $f_i - a_i \geq C_i$ might be true, since a job can be paused at the partition boundary.

4.3.2 Flat Scheduling Model

In this section, we assume that there exists only one partition which is available to service tasks at all times, i.e., $V_p = \emptyset$. Furthermore, the finish time f_i for a job $\mathtt{j}_i$ is not defined by the scheduling table S_p, instead f_i is directly calculated by the equation $f_i = a_i + C_i$.

Offline Phase

As mentioned earlier, during the offline phase, the flexibility coefficient x_i for each job $\mathtt{j}_i \in S_p$ is calculated starting from the job $\mathtt{j}_l$ (where l represents the index of the last job in S_p, i.e., the job with the maximum activation time a_l) and ending at the job $\mathtt{j}_f$ (where f represents the index of the first job in S_p, i.e., the job with the minimum activation time a_f). Assuming $a_{l+1} = d_l$ and $x_{l+1} = 0$, for each job $\mathtt{j}_i$ the following steps are performed;

i) Calculate the parameter O_i, which defines the overlap between the flexibility windows $[a, d]$ of job $\mathtt{j}_i$ and job $\mathtt{j}_{i+1}$, using the following equation;

$$O_i = \max(0, d_i - a_{i+1}) \tag{4.2}$$

ii) Calculate the flexibility coefficient x_i using the following equation;

$$x_i = d_i - a_i - C_i - O_i + \min(x_{i+1}, O_i) \tag{4.3}$$

In order to clarify that Equation 4.3 provides the correct job flexibility as per Definition 4.1, we provide the following lemma.

Lemma 4.3.1. The equations 4.2 and 4.3 provide the *maximum* value by which the job activation time a_i can be delayed without changing job execution order *and* without leading to a deadline miss.

Proof. Assume two consecutive jobs with activation times a and deadlines d as shown in Figure 4.2. Based on the occurrence of the deadline of job $\mathtt{j}_i$, three different intervals I_1, I_2 and I_3 can be identified.

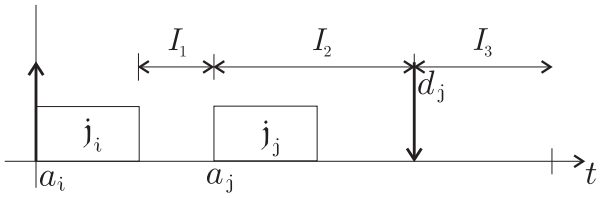


Figure 4.2: *Scheduling table for proving Lemma 4.3.1.*

When the deadline d_i lies during the interval I_1, the flexibility x_i of job $\mathtt{j}_i$ does not depend on $\mathtt{j}_j$ and therefore, the *maximum* delay is equal to the job slack [But11]. Due to a delay in a_i equal to the job slack, neither job $\mathtt{j}_i$ nor $\mathtt{j}_j$ will result in a deadline miss.

When the deadline d_i lies during the interval I_2, the job overlap O_i fall in the range $(0, x_j]$. Due to a non-zero overlap, the maximum delay in a_i gets a dependency on j_j and therefore, the overlap needs to be considered. In Figure 4.2, the *maximum* processing time which can be borrowed by j_i from the execution window of j_j is given by $\min(O_i, x_j)$. If more processing time is borrowed, the job j_j will lead to a deadline miss. In such a situation, executing j_j before j_i may or may not lead to a deadline miss, however, in such a case, the execution order of the schedule is violated.

For the case of interval I_3, the overlap O_i becomes larger than x_j (since the maximum flexibility of a job is given by the job slack [But11]). Therefore, the *maximum* processing time borrowed by j_i from j_j is given by x_j. Executing j_j before j_i may or may not lead to a deadline miss, however, the execution order of the jobs is violated.

Note that the equation 4.3 gives the *maximum* delay in all of these cases and, therefore, the lemma is proven. □

Example: Assume two jobs j_1 and j_2 with release time r, activation time a, deadline d and worst-case execution time C as shown in Figure 4.3 and defined in Table 4.2. During the offline phase of job-shifting algorithm, the flexibility coefficient x is calculated starting from job j_2 and ending at job j_1. The calculation is provided below;

For job j_2, $a_3 = d_2$ and $x_3 = 0$.

$$\begin{aligned} O_2 &= \max(0, d_2 - d_2) \\ &= 0 \\ x_2 &= d_2 - a_2 - C_2 - O_2 + \min(x_3, O_2) \\ &= 10 - 2 - 2 - 0 + \min(0, 0) \\ &= 6 \end{aligned}$$

For job j_1, $a_2 = 2$ and $x_2 = 6$.

$$\begin{aligned} O_1 &= \max(0, d_1 - a_2) \\ &= \max(0, 8 - 2) = 6 \\ x_1 &= d_1 - a_1 - C_1 - O_1 + \min(x_2, O_1) \\ &= 8 - 0 - 2 - 6 + \min(6, 6) \\ &= 6 \end{aligned}$$

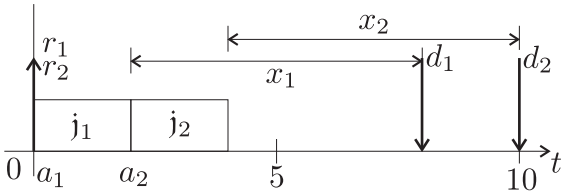


Figure 4.3: *Example for offline phase of flat scheduling model*

j	r	a	d	C	x
1	0	0	8	2	**6**
2	0	2	10	2	**6**

Table 4.2: *Example parameters for offline phase*

Online Phase

After finishing the offline phase, all the parameters for each job are passed to the online scheduler, which is activated at each job activation time a, finish time f and (when idle) at the release of a new aperiodic job.

The guarantee test is performed when the scheduler gets activated at time t and an aperiodic job release is detected. For the guarantee test, the job j_n is defined as the next job to be activated from the scheduling table S_p, while the job j_l is defined as the first job activated after the deadline of the released aperiodic job j_{ap}. During the guarantee test, the aperiodic room R_i for each job i from j_n to j_l is calculated using the following

set of equations;

$$\begin{aligned} s_i &= \begin{cases} a_{i-1} + C_{i-1}, & n < i < l \\ t, & i = n \end{cases} \\ R_i &= a_i - s_i + \min(d_{ap} - a_i, x_i) \end{aligned} \tag{4.4}$$

where, s_i represents the (potential) start of the released aperiodic job j_{ap} before job j_i. In order to guarantee that the equation system 4.4 provides the correct aperiodic job room as per Definition 4.2, we provide the following lemma.

Lemma 4.3.2. The equation system 4.4 provides the *maximum* processing time R_i available for the execution of an aperiodic job before job j_i without leading to a deadline miss.

Proof. Assume a set of guaranteed jobs j_n, j_{n+1}, ..., with activation times a and deadlines d as shown in Figure 4.4. When an aperiodic job j_{ap} is released at time t_{ap} with a deadline d_{ap} and worst-case execution time C_{ap}, three different intervals I_1, I_2 and I_3 can be identified.

When the deadline d_{ap} lies during the interval I_1, the *maximum* aperiodic room R_n for job j_n will be $d_{ap} - t_{ap}$. Executing an aperiodic job with C_{ap} larger than R_n may result in a deadline miss by job j_{ap}. Note that in this case, the difference $d_{ap} - a_n$ results in a negative value.

When the deadline d_{ap} lies during the interval I_2, the job overlap O_{ap} falls in the range $(0, x_n]$. Due to a non-zero overlap, the *maximum* aperiodic room R_n gets a dependency on j_n and therefore, the overlap needs to be considered. In Figure 4.4, the *maximum* processing time which can be borrowed by j_{ap} from the execution window of j_n is given by $\min(O_{ap}, x_n)$. If more processing time is borrowed, the job j_n will miss the deadline.

For the case of interval I_3, the overlap O_{ap} becomes larger than x_n (see Lemma 4.3.1). Therefore, the *maximum* processing time borrowed by j_{ap} from j_n is given by x_n. Executing a job with C_{ap} greater than $a_n - t_{ap} + x_n$ might result in a deadline miss by the job j_n.

Note that the same reasoning can be applied for all jobs $\mathrm{j}_{n+1}, \mathrm{j}_{n+2}, \ldots, \mathrm{j}_{l-1}$, however for these jobs, the maximum aperiodic room R will be defined as $a_{n+1} - f_n$ for the first case. Note that the equation system 4.4 gives the *maximum* room in all of these cases and, therefore, proves the lemma. □

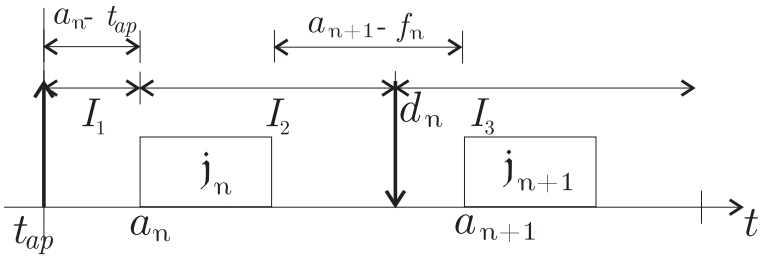


Figure 4.4: *Scheduling table for proving Lemma 4.3.2.*

The guarantee test passes when, for any job $\mathrm{j}_i | n \leq i < l$, the aperiodic room R_i is larger than or equal to C_{ap}. If the guarantee test passes, the aperiodic job j_{ap} can be guaranteed to finish before its deadline d_{ap} without missing any other deadline in the system. Once the guarantee test passes, the guarantee procedure is activated. The guarantee procedure requires the following steps to be performed in the defined order:

i) Insert the released aperiodic job j_{ap} in the schedule S_p before job j_i , i.e., $a_i = a_{ap} = s_{i+1}$.

ii) For each job j_k, such that $k > i \wedge a'_k > a_k$, perform the following steps starting from j_{i+1};
   a) $a'_k = a_k + \max(a_{k-1} + C_{k-1} - a_k, 0)$
   b) $x_k = x_k - (a'_k - a_k)$
   c) $a_k = a'_k$

iii) For each job j_p, such that $p \leq i$, calculate x_p similar to the offline phase starting from j_i and ending at j_n. The process of modifying x_p stops when the old x_p is equal to the new x_p.

At the scheduler activation time t, the scheduling decision can be made after handling the released aperiodic job(s). If the best-effort queue is empty, the next job with $a = t$ is scheduled from S_p. When there exists no such job, the processor is left idle. If there exists a job j_b in the best-effort queue and the next job aperiodic room $R_n \geq C_b$, the guarantee procedure is performed for job j_b with $a_b = t$ and it is scheduled.

Example: Assume an aperiodic job j_{ap} with $C_{ap} = 2$ and $d_{ap} = 3$ is released at time $t = 0$ in the example shown in Figure 4.3. The online scheduler performs the guarantee procedure with $n = 1$ and $l = 3$ (i.e., the first job of the next SC). For job j_i with $i = 1$, $s_1 = t = 0$. The aperiodic room R_1 is calculated as under;

$$\begin{aligned} R_1 &= a_1 - s_1 + \min(d_{ap} - a_1, x_1) \\ &= 0 - 0 + \min(3 - 0, 6) \\ &= 3 \end{aligned}$$

Since $R_1 > C_{ap}$, the newly released job j_{ap} can be guaranteed to finish before its deadline d_{ap}. Therefore, guarantee procedure is triggered. The steps are shown below;

1. Job j_{ap} is inserted at location 0, i.e., $a_0 = a_{ap} = s_1 = 0$, $C_0 = C_{ap} = 2$ and $d_0 = d_{ap} = 3$.

2. For job j_1:

$$\begin{aligned} a'_1 &= a_1 + \max(a_0 + C_0 - a_1, 0) = 0 + \max(0 + 2 - 0, 0) = 2 \\ x_1 &= x_1 - (a'_1 - a_1) = 6 - (2 - 0) = 4 \\ a_1 &= a'_1 = 2 \end{aligned}$$

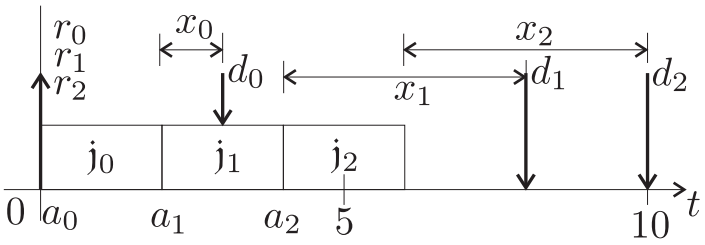


Figure 4.5: *Example for online phase of flat scheduling model*

j	r	a	d	C	x
0	0	**0**	3	2	**1**
1	0	**2**	8	2	**4**
2	0	**4**	10	2	**4**

Table 4.3: *Example parameters for online phase*

For job j_2:

$$\begin{aligned} a_2' &= a_2 + \max(a_1 + C_1 - a_2, 0) = 2 + \max(2 + 2 - 2, 0) = 4 \\ x_2 &= x_2 - (a_2' - a_2) = 6 - (4 - 2) = 4 \\ a_2 &= a_2' = 4 \end{aligned}$$

3. For job j_0:

$$\begin{aligned} O_0 &= \max(0, d_0 - a_1) = \max(0, 3 - 2) = 1 \\ x_0 &= d_0 - a_0 - C_0 - O_0 + \min(x_1, O_0) = 3 - 0 - 2 - 1 + \min(4, 1) = 1 \end{aligned}$$

The modified scheduling table is shown in Figure 4.5, while the modified parameters are highlighted in Table 4.3. After performing the guarantee test, the job j_0 is selected for execution since $a_0 = t = 0$ and the best-effort queue is empty. When the job j_0 completes, the scheduler gets activated again at $t = 2$ and schedules j_1 and later, at $t = 4$, j_2 to complete the execution of the scheduling table.

4.3.3 Hierarchic Scheduling Model

In this section, we assume that there exist multiple partitions in the system and the job-shifting algorithm is enabled in partition $p \in P$, i.e., $V_p \neq \emptyset$. For hierarchic scheduling model, we need to define two operators; the Blocking operator $B(p, q)$ and the Adjust operator $A(u)$. The blocking operator $B(p, q)$ returns the sum of the partition blocking duration between the interval $(p,q]$. The operator $A(u)$ modifies the time instant u such that it does not lie within the partition blocking time V. When the time instant u denotes the job finishing time f_i, the operator $A(u)$ also makes sure that the duration between the job activation time a_i and the job finish time f_i is enough to complete the job, i.e., $f_i = a_i + B(a_i, f_i) + C_i$. Unlike the flat scheduling model, the finish time f_i for a job j_i in the hierarchic scheduling model is also defined by the scheduling table S_p. Moreover, it is assumed that the parameters a_i and f_i are adjusted offline using the adjust operator $A(u)$.

Offline Phase

Similar to the offline phase of flat scheduling model, the flexibility coefficient x_i for each job $\mathtt{j}_i \in S_p$ is calculated during the offline phase of hierarchic scheduling model starting from the job $\mathtt{j}_l$ (where l represents the index of the last job in S_p) and ending at the job $\mathtt{j}_f$ (where f represents the index of the first job in S_p). Assuming $a_{l+1} = d_l$ and $x_{l+1} = 0$, for each job $\mathtt{j}_i$ the following steps are performed;

i) Calculate the overlap parameter O_i using the following equation;

$$O_i = \max(0, d_i - a_{i+1} - B(a_{i+1}, d_i)) \tag{4.5}$$

ii) Calculate the flexibility coefficient x_i using the following equation;

$$x_i = d_i - a_i - C_i - B(a_i, d_i) - O_i + \min(x_{i+1}, O_i) \tag{4.6}$$

The validity of equation 4.6 can be proven similarly to Lemma 4.3.1.

Example: Assume two jobs j_1 and j_2 with release time r, activation time a, finish time f, deadline d and worst-case execution time C as shown in Figure 4.6 and defined in Table 4.4. Also assume that the partition blocking set V is defined to be $\{\langle 5, 8\rangle\}$. During the offline phase of job-shifting algorithm, the flexibility coefficient x is calculated starting from job $\mathtt{j}_2$ and ending at job $\mathtt{j}_1$. The calculation is provided as below;

For job $\mathtt{j}_2$, $a_3 = d_2$ and $x_3 = 0$.

$$\begin{aligned} O_2 &= \max(0, d_2 - d_2 - B(d_2, d_2)) \\ &= 0 \\ x_2 &= d_2 - a_2 - C_2 - B(a_2, d_2) - O_2 \\ &\quad + \min(x_3, O_2) \\ &= 10 - 2 - 2 - 3 - 0 + \min(0, 0) \\ &= 3 \end{aligned}$$

For job $\mathtt{j}_1$, $a_2 = 2$ and $x_2 = 3$.

$$\begin{aligned} O_1 &= \max(0, d_1 - a_2 - B(a_2, d_1)) \\ O_1 &= \max(0, 8 - 2 - 3) = 3 \\ x_1 &= d_1 - a_1 - C_1 - B(a_1, d_1) - O_1 \\ &\quad + \min(x_2, O_1) \\ &= 8 - 0 - 2 - 3 - 6 + \min(6, 6) \\ &= 3 \end{aligned}$$

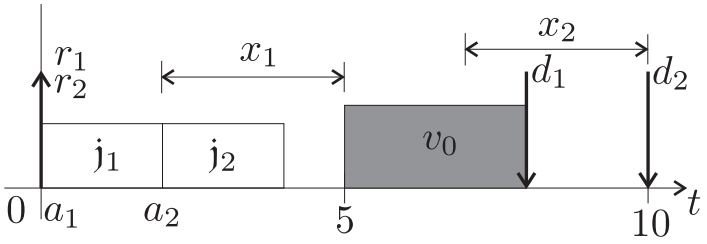


Figure 4.6: *Example for offline phase of hierarchic scheduling model*

j	r	a	f	d	C	x
1	0	0	2	8	2	**3**
2	0	2	4	10	2	**3**

Table 4.4: *Example parameters for offline phase*

Online Phase

After finishing the offline phase, all the parameters for each job and the partition blockings V_p are passed to the online scheduler. The online scheduler is activated at each job activation time a, finish time f, and when the partition is active *and* the processor is idle at the release of a new aperiodic job.

When the scheduler is activated at time t and an aperiodic job release is detected, the guarantee test is performed. Similar to the flat scheduling model, the job $\mathtt{j}_n$ is defined as the next job to be activated from the scheduling table S_p, while the job $\mathtt{j}_l$ is defined as the first job activated after the deadline of the released aperiodic job $\mathtt{j}_{ap}$. During the guarantee test, the aperiodic room R_i for each job i from $\mathtt{j}_n$ to $\mathtt{j}_l$ is calculated using the following set of equations;

$$\begin{aligned} s_i &= \begin{cases} f_{i-1}, & n < i < l \\ t, & i = n \end{cases} \\ R_i &= a_i - s_i - B(s_i, a_i) + \min(d_{ap} - a_i - B(a_i, d_{ap}), x_i) \end{aligned} \tag{4.7}$$

The validity of equation 4.7 can be proven similarly to Lemma 4.3.2.

The guarantee test passes if, for any job $\mathtt{j}_i | n \leq i < l$, the aperiodic room R_i is larger than or equal to C_{ap}. Once the guarantee test passes, the guarantee procedure is activated. The guarantee procedure requires the following steps to be performed in the defined order:

i) Insert the released aperiodic job $\mathtt{j}_{ap}$ in the schedule S_p before job $\mathtt{j}_i$, i.e., $a_i = a_{ap} = s_{i+1}$ and $f_i = f_{ap} = A(a_{ap} + C_{ap})$.

ii) For each job $\mathtt{j}_k$, such that $k > i \wedge a'_k > a_k$, perform the following steps starting from $\mathtt{j}_{i+1}$;
   a) $a'_k = A(a_k + \max(f_{k-1} - a_k, 0))$
   b) $f_k = A(f_k + (a'_k - a_k - B(a_k, a'_k)))$
   c) $x_k = x_k - (a'_k - a_k - B(a_k, a'_k))$
   d) $a_k = a'_k$

iii) For each job $\mathtt{j}_p$, such that $p \leq i$, calculate x_p similar to the offline phase starting from $\mathtt{j}_i$ and ending at $\mathtt{j}_n$. The process of modifying x_p stops when the old x_p is equal to the new x_p.

During run-time, the jobs are scheduled similar to the flat scheduling model.

Example: Assume an aperiodic job $\mathtt{j}_{ap}$ with $C_{ap} = 2$ and $d_{ap} = 3$ is released at time $t = 0$ in the example shown in Figure 4.6. The online scheduler performs the guarantee procedure with $n = 1$ and $l = 3$ (i.e., the first job of the next SC). For job $\mathtt{j}_i$ with $i = 1$, $s_1 = t = 0$. The aperiodic room R_1 is calculated as under;

$$\begin{aligned} R_1 &= a_1 - s_1 + \min(d_{ap} - a_1 - B(a_1, d_{ap}), x_1) - B(a_1, s_1) \\ &= 0 - 0 + \min(3 - 0 - 0, 3) - 0 \\ &= 3 \end{aligned}$$

Since $R_1 > C_{ap}$, the newly released job $\mathtt{j}_{ap}$ can be guaranteed to finish before its deadline d_{ap}. Therefore, guarantee procedure is triggered. The steps are shown below;

1. Job $\mathtt{j}_{ap}$ is inserted at location 0, i.e., $a_0 = a_{ap} = s_1 = 0$, $f_0 = f_{ap} = A(a_{ap} + C_{ap}) = 2$, $C_0 = C_{ap} = 2$ and $d_0 = d_{ap} = 3$.

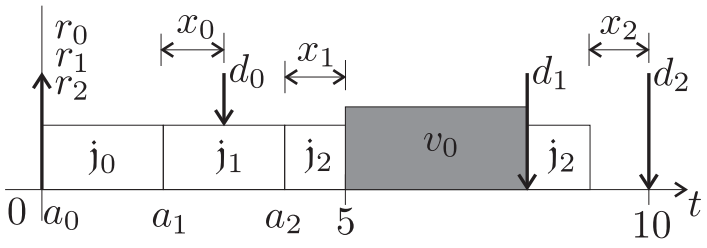


Figure 4.7: *Example for online phase of hierarchic scheduling model*

j	r	a	f	d	C	x
0	0	**0**	**2**	3	2	**1**
1	0	**2**	**4**	8	2	**1**
2	0	**4**	**9**	10	2	**1**

Table 4.5: *Example parameters for online phase*

2. For job j_1:

$$\begin{aligned} a_1' &= A(a_1 + \max(f_0 - a_1, 0)) = A(0 + \max(2 - 0, 0)) = 2 \\ f_1 &= A(f_1 + (a_1' - a_1 - B(a_1, a_1')) = A(2 + (2 - 0 - B(0, 2))) = 4 \\ x_1 &= x_1 - (a_1' - a_1 - B(a_1, a_1')) = 3 - (2 - 0 - B(0, 2)) = 1 \\ a_1 &= a_1' = 2 \end{aligned}$$

For job j_2:

$$\begin{aligned} a_2' &= A(a_2 + \max(f_1 - a_2, 0)) = A(2 + \max(4 - 2, 0)) = 4 \\ f_2 &= A(f_2 + (a_2' - a_2 - B(a_2, a_2')) = A(4 + (4 - 2 - B(2, 4))) = 9 \\ x_2 &= x_2 - (a_2' - a_2 - B(a_2, a_2')) = 3 - (4 - 2 - B(2, 4)) = 1 \\ a_2 &= a_2' = 4 \end{aligned}$$

3. For job j_0:

$$\begin{aligned} O_0 &= \max(0, d_0 - a_1 - B(a_1, d_0)) = \max(0, 3 - 2 - B(2, 3)) = 1 \\ x_0 &= d_0 - a_0 - C_0 - B(a_0, d_0) - O_0 + \min(x_1, O_0) \\ &= 3 - 0 - 2 - B(0, 3) - 1 + \min(1, 1) = 1 \end{aligned}$$

The modified scheduling table is shown in Figure 4.7, while the modified parameters are shown in Table 4.5. After performing the guarantee test, the job j_0 is selected for execution since $a_0 = t = 0$ and the best-effort queue is empty. When the job j_0 completes, the scheduler gets activated again at $t = 2$ and schedules j_1 and later, at $t = 4$, j_2 to complete the execution of the scheduling table.

4.4 Discussion

This section provides a detailed description of which factors are considered by job-shifting algorithm, which factors affect the overheads incurred by job-shifting algorithm and how this algorithm can be optimized for a system under consideration.

4.4.1 Mixed-Critical Tasks

It is important to note in Section 4.3 that neither flat nor hierarchical scheduling models take the task criticality Y into account. The reason for such a deliberate elimination is the use of the industrial mixed-criticality model defined by the standards IEC61508 [IEC10], DO-178C [DC11] and ISO26262 [ISO11]. As discussed in Section 1.1.2, the task criticality designation Y can only be exploited by the scheduler when a partition contains both critical and non-critical tasks. However, executing critical and non-critical tasks in a single partition are discouraged by the standards [ENNT15] in order to provide strong spatial and temporal isolation between tasks from different criticalities.

In cases where critical and non-critical tasks *must* execute inside the same partition, the non-critical tasks can be dropped in favor of servicing critical aperiodic tasks [ENNT15]. To handle such a situation with job-shifting algorithm, a flag J to indicate task criticality (i.e., either critical or non-critical) needs to be added to each job in the partition scheduling table S_p. In order to drop non-critical tasks in favor of critical tasks, the guarantee test can be implemented based on one of the two heuristics: optimistic or pessimistic guarantee test heuristics.

To implement optimistic guarantee test heuristic, at first the guarantee test (i.e., $R_i \geq C_{ap}$) is performed without considering the job criticality flag J. If the test fails and the next job $i|n \leq i < l$ is a non-critical job, the guarantee test is performed after merging the flexibility of jobs j_i and j_{i-1} (by defining flexibility $x_{i-1} = \max[x_{i-1} + C_i + x_i - O_i, d_{i-1} - a_{i-1}]$ and ignoring job i). If the test passes, the job i is dropped and the guarantee procedure is performed.

To implement pessimistic guarantee test heuristic, in the first stage, the guarantee test assumes all non-critical jobs to be dropped (by merging the flexibilities x similar to the optimistic guarantee test heuristic), but only drops the non-critical job(s) when the test passes. Once the test passes, it can be checked (when required) if the non-critical job can still be executed.

The problem with optimistic and pessimistic heuristics is large (implementation and run-time) complexity and memory overheads. Therefore, it is discouraged to use such optimizations in safety critical systems and, thus, are not considered an integral part of the job-shifting algorithm.

4.4.2 Offline Guarantee Analysis

In order to design a robust and responsive real-time system, a usual practice is to guarantee the correct behaviour of a sub-system or task offline. To guarantee some service to aperiodic activities, a certain amount of processor bandwidth is usually reserved for aperiodic tasks (or servers). Due to the limited system resources and large number of constraints, there is always a limit to the bandwidth which is considered a "safe reservation" for handling aperiodic activities. In industry, such reservations are considered mandatory to guarantee system safety. However, reserving a specific bandwidth for such a case still limits the service for aperiodic execution. Moreover, the response-time of aperiodic tasks in such reservations also has a strong dependency on the employed

reservation technique. In this work, we conjecture that, instead of reserving a limited amount of processor bandwidth, all free partition/processor bandwidth can be utilized to service aperiodic tasks. The following theorem provides the base for our conjecture:

Theorem 4.4.1. For a given TT schedule S, if there exists a reservation such that a number of non-preemptive aperiodic jobs with a defined job release pattern can be executed feasibly, the job-shifting algorithm can also execute them without a reservation.

Proof. Consider a TT scheduling table S with defined activation times a_i and finish times f_i for each periodic job $\mathfrak{j}_i$. A reservation window $\Lambda_{[t_1,t_2)}$ with window start time t_1 and end time t_2 is defined as contiguous *reserved* processing time for the execution of non-preemptive aperiodic job(s). According to Lemma 4.3.2, the aperiodic room R_k provides the *maximum* room for executing non-preemptive aperiodic job, where $k \in \mathbb{N} | f_{k-1} \leq t_1 \wedge a_k \geq t_2$. When a non-preemptive aperiodic job $\mathfrak{j}_{ap}$ with $C_{ap} = t_2 - t_1$ is released at $r_{ap} \leq t_1$, then by definition, the condition $\Lambda_{[t_1,t_2)} \leq R_k$ always holds. If the non-preemptive aperiodic job $\mathfrak{j}_{ap}$ can be feasibly executed by $\Lambda_{[t_1,t_2)}$, R_k can also feasibly execute it without an offline reservation. The proof can be iteratively applied to multiple reservation windows. Hence, the theorem is proven. □

In other words, the same practice of "safe reservation" can be utilized to give guarantees for aperiodic execution in job-shifting without any *online* reservation. However with job-shifting, the bandwidth loss due to partitioning [LWVH12] can be reduced (further discussed in Section 4.5), the aperiodic response-times can be significantly improved (discussed in Section 4.4.4) and the task jitter can be controlled (discussed in Section 4.4.9).

Being a non-preemptive algorithm, a necessary condition for aperiodic guarantee can also be checked offline in job-shifting. Independent of the release time of the aperiodic job, there must exist enough aperiodic room R_i in the schedule S_p to accommodate complete aperiodic job. For an aperiodic task, if there exists no aperiodic room $R_i | R_i \geq C_{ap}$ for any $\mathfrak{j}_i \in S_p$, the aperiodic job will never be able to execute while keeping feasibility of other tasks in S_p. This necessary condition is checked between the offline and the online phases of the job-shifting algorithm.

4.4.3 Hypervisors and Clocks

There exists a multitude of real-time hypervisors in the market today. Based on the type of interface they provide to the partition, the online complexity of the job-shifting algorithm can be modified. Some hypervisors, e.g., XtratuM [CRM+09], provide a partition local clock which ticks only when the partition is active. In such hypervisors, the equations for the online phase of the flat scheduling model can be used in a hierarchic design by defining a_i, r_i and d_i in terms of the partition local clock. For such an implementation, the adjust operator $A(u)$ can be completely eliminated and the blocking operator $B(p, q)$ is only required for the last step of guarantee procedure (where the flexibility coefficient is calculated). However, there exist hypervisors which do not provide a partition local clock, e.g., PikeOS [SYS17], and therefore need to implement both operators $A(u)$ and $B(p, q)$.

4.4.4 Feasibility & Complexity

When the scheduler needs to adjust a number of aperiodic jobs, the problem of guaranteeing all the aperiodic jobs becomes a variant of bin-packing problem. The decision version of the bin-packing problem is known to be NP-complete [CGJ78]. Moreover, a non-clairvoyant online scheduler, by definition, cannot guarantee that all randomly released aperiodic jobs can be accepted.

For a single aperiodic job, the *maximum* number of checks (i.e., $R_i \geq C_{ap}$) performed by the guarantee test are equal to the number of guaranteed jobs activated during the interval $[t, d_{ap})$, which is not different from the slot shifting algorithm [Foh94]. Depending on the required performance metric, the guarantee test can utilize any bin-packing heuristic, e.g., first-fit, best-fit or worst-fit. The first-fit heuristic can be used when least aperiodic response-time and online overheads are required. In such a heuristic, the guarantee test starts from job $\mathfrak{j}_n$ and inserts the newly released aperiodic job before the first job with enough aperiodic room R_i. On the contrary, when larger online overheads can be tolerated but better distribution of aperiodic jobs is required, the best- or worst-fit heuristics can be utilized. Moreover, the order in which multiple aperiodic jobs are guaranteed also impacts the overall feasibility of jobs. Lupu et al. [LCGG10] recommend worst-fit decreasing utilization heuristic in order to maximize the number of guaranteed tasks.

For the flat scheduling model, the guarantee test and the guarantee procedure have a complexity of $O(m)$, where m is the number of jobs activated during the interval $[t, d_{ap})$ (or $l - n$). For the hierarchic scheduling model, we introduced two operators $A(u)$ and $B(p, q)$. A naïve implementation of these operators has a complexity of $O(r)$, where r is the cardinality of the blocking set V. When partition local clock is available, the complexity of the guarantee test does not change. However, the complexity of the guarantee procedure changes to $O(mr)$. On the contrary, when the partition local clock is not available, the complexity of the guarantee test becomes $O(mr)$, while the complexity of the guarantee procedure changes to $O(mr^3)$.

4.4.5 Optimizations

In Section 4.2, it was assumed that all the tasks execute for their complete worst-case execution time C. However, this assumption seldom holds during system operation. A task executing for more than C can lead to the violation of the temporal isolation between tasks of the same application. In the worst case, such violations may never lead to a processor yield for other tasks to execute. To protect from such a situation, a hardware timer can be programmed to generate an overrun interrupt which terminates the misbehaving job and returns the control to the scheduler. When the job executes less than C time, the overrun interrupt can be terminated (or rescheduled for the next job). In such a situation, the free processing time can be utilized by using a simple approach. At the observed job finish time t, if for the next job $a_n > r_n$, the flexibility coefficient x_n can be updated to $x_n + (a_n - \max(t, r_n))$ and the job can be preponed to activate at $\max(t, r_n)$. If for the next job $a_n = r_n$, the free resources can be accounted for by the aperiodic room R_n and therefore, require no further action.

When a large number of aperiodic jobs are released before the scheduler activation, performing guarantee procedure for all of them may lead to large scheduler overheads. In the worst case, the reserved processing time to execute the next job might be reduced, leading to a job incompletion. To avoid such a scenario, Schorr [Sch15] proposed a heuristic where at most one aperiodic job is guaranteed for each activation of the scheduler.

When the guarantee test passes or when the schedule repeats after SC, an insert operation is required on the scheduling table S_p. Therefore, the scheduling table is recommended to be stored as a double linked list, due to its $O(1)$ insert operation complexity. However, the blocking set V can be stored as a contiguous vector as random access may be required. All the hypervisors supporting cyclic scheduling policy for partition scheduling store the blocking set V or the partition availability set V' but may not expose it to the partition. A simple API can be added to open-source hypervisor, e.g., XtratuM [CRM+09], to expose this information. In order to save the overheads related to the hyper-calls, the partition blocking set V can also be stored in partition local memory.

4.4.6 Intra-Partition Scheduling

In the job-shifting algorithm, the degree of flexibility of a job is defined by the flexibility coefficient. If the flexibility coefficient for all the jobs is zero, the job-shifting algorithm reduces to the background processing approach. Therefore, it can be said that the flexibility coefficient is the direct measure of the adaptability of a schedule. Depending on the scheduler used prior to the offline phase of job-shifting, the flexibility coefficient can vary significantly. The EDF approach results in the largest flexibility coefficients, while the latest-deadline-first (LDF) results in the least.

In Section 4.2, the input for the offline phase of job-shifting was defined to be a feasible TT partition schedule S_p. However, this restriction can be relaxed when (i) the online scheduling strategy is known *and* (ii) it can be made sure that the estimated order of execution of jobs will not be violated online. For such an online scheduler, the flexibility coefficient for all the jobs can be calculated offline. Nonetheless, the online scheduling of non-preemptive periodic tasks is an NP-hard problem [JSM91] even with harmonic periods [CK96] or with arbitrary period ratios [NF16]. Due to the complexity of the problem, a feasible TT partition schedule S_p seems a better choice as an offline exhaustive search can be done (perhaps with large search time) with minor efforts to generate such a schedule.

4.4.7 Free Resources Management

After generating the offline scheduling table S for the system, there might exist scheduling holes (i.e., time duration which is not used by any partition). Note that in our system model (defined in Section 4.2), the hypervisor is not aware of the released aperiodic jobs *and* the partition is not aware of the scheduling holes as they reside outside of the partition scheduling table S_p. If an online scheme for management of free resources

is required, either hypervisor needs to be aware of the aperiodic tasks or the partition needs to be aware of the scheduling holes, so that the partition slots can be stretched or shrunk online (similar to flattening method by Lackorzyński et al. [LWVH12]). However in this case, it is very difficult to provide offline guarantees and the cyclic executive strategy (recommended by certification authorities [ENNT15]) needs to be modified. Therefore, it is recommended that these scheduling holes are managed offline.

Depending on the hypervisor interface and the application specifications, there exists a number of offline schemes for the management of free resources. An easier and viable scheme for offline management is to enlarge the existing partition slots (where possible) to consume the scheduling holes or to make new partition slots (where enlargement is not possible) for the preferred partition. This method can help avoid modifications to the cyclic executive scheduler and provide flexibility in selecting which partition needs to have more free time. If the hypervisor switches to an idle partition during the scheduling holes, another solution is to use the idle partition to service aperiodic jobs using job-shifting algorithm as defined in Section 4.3. However, this solution might require hypervisor modifications and, therefore, it is not recommended. On the contrary, if the hypervisor stays in the kernel/privileged mode during the scheduling holes, the aperiodic tasks cannot execute in the context of the hypervisor and, hence, such a solution cannot be implemented.

4.4.8 Memory Requirement

As the amount of memory required by an algorithm is strongly dependent on the processor clock frequency, the word size (e.g., 32-bit) and the required resolution of timely parameters (e.g., a_i, d_i), we will measure the memory overheads in terms of variables, instead of bytes. Moreover, for the two implementations, i.e., Cyclic executive [ARI03] intra-partition scheduler with bandwidth reservation for aperiodic tasks (CEIPS-BR) and CEIPS with job-shifting (CEIPS-JS), only the structures which vary in size will be discussed.

Implementation: CEIPS-BR

For the simple case of CEIPS-BR, there exist only one intra-partition scheduling table per partition, containing jobs for periodic as well as aperiodic (as periodic server/ reservation) tasks. The scheduling table is defined as a static array of entries, each defining a job using the tuple $\langle a, *\tau \rangle$ (where $*\tau$ represents the pointer to the task code to which the job belongs to). Note that there is no need to store the partition blockings V for this implementation.

Implementation: CEIPS-JS

For a naïve implementation of CEIPS-JS, two intra-partition scheduling tables per partition are required, one for reloading the new table and one for job-shifting. Note that the first scheduling table used for reloading in CEIPS-JS only contains jobs for periodic tasks. The scheduling tables are defined either as a static array or as a static double

linked list of entries, each defining a job using the tuple $\langle a, C, d, x, *\tau \rangle$. It is important to mention here that when the schedule is implemented as a double linked list, two pointers for left and right elements of the list are also needed. When the hierarchic scheduling model is used, the job entry in the scheduling tables also includes the job finishing time f. Moreover, for the hierarchic scheduling model the partition blockings V are stored as a static array of entries each defined by the tuple $\langle b, m \rangle$. Other than the two scheduling tables (for a naïve implementation), a third table is required to store the information about the aperiodic tasks. The entries in this table are stored as the tuple $\langle D, C, \$, *\tau \rangle$ (where \$ represents the source of aperiodic task activation, e.g., interrupt flag).

In summary, even for the best-case, CEIPS-JS requires 3 more variables per periodic job (with two copies of the tables for a naïve implementation) and 2 more variables per aperiodic task compared to CEIPS-BR. However, note that the scheduling table in CEIPS-BR stores entries for all aperiodic jobs, which is not the case with CEIPS-JS.

4.4.9 Jitter Control

In control systems, for instance, the task jitter has a strong impact on the overall performance of the system. Similarly, a task executing too early or too late (within its execution window, i.e., $d - r$) can lead to large variations in end-to-end latencies for large task chains. Although the worst-case jitter for a task can be estimated offline for non-preemptive priority-based scheduling approaches [NBFK14] (for instance npRM), the task jitter cannot be controlled without modifying the priority of a task. Moreover, modification of the priority of a single task may affect all tasks. On the contrary, offline TT scheduling approaches can be used to have a fine control over task jitter.

As mentioned in the previous section, the flexibility coefficient x for a job in job-shifting algorithm provides a direct measure of the execution freedom the job has in the offline schedule. By limiting the flexibility coefficients of all the jobs of a task, the jitter can be controlled. In other words, the following condition must be satisfied either by construction of the partition schedule S_p or by decreasing the job flexibility coefficients x.

$$\forall i | \mathfrak{j}_i \in \tau_j : a_i - r_i + x_i \leq e_j \tag{4.8}$$

where e_j is the desired activation jitter of the periodic task τ_j. Moreover, the activation jitter of an aperiodic task can be controlled online during the guarantee test by verifying the condition: $s_i - r_i + x_i \leq e_j$ (see equations 4.4 and 4.7). Note that limiting jitter for a task in job-shifting might potentially lead to reduced jitter of jobs executed before the task under consideration.

4.5 Efficiency Evaluation

In this section, we present the results of the experiments performed to evaluate efficiency of the job-shifting algorithm. In Section 4.5.1, we provide the description of the

experimental setup. While in Section 4.5.2, we present and briefly discuss the obtained results.

4.5.1 Experimental Setup

In order to provide a reference point for the job execution guarantees provided by the job-shifting (JS) algorithm, we implemented a non-preemptive hierarchic background scheduler (NP-BG). The NP-BG scheduler services the aperiodic jobs during the idle time available in the partition scheduling table, i.e., a non-preemptive aperiodic job is executed only when it can be guaranteed that the job will finish within the idle time. Note that any existing algorithm (e.g., non-preemptive version of slot shifting [Sch15], constant bandwidth server [AB98], sporadic server [SSL89], etc.) cannot be used to provide such a reference point for comparison due to varying system models (e.g., preemptive execution environment, only flat scheduling model, etc.).

For the evaluation of job-shifting algorithm, 1000 task-sets were generated. The parameter ranges for generating task-sets were selected as defined by Schorr [Sch15]. Each generated task-set consisted of at most one processor and one partition. The partition blockings V are generated with a strict period of 10 and a relative beginning time $b = 6$ of each blocking $v \in V$ until the end of the scheduling cycle SC. As the number of parameters and their ranges for generating task-sets are very large, the specific periods and sizes of the partition blockings V do not lead to a biased result.

Inside the partition, [1, 3] periodic tasks were generated with phases $\phi = 0$, WCET C in the interval [1, 15] and the period T in the interval [15, 30] (with implicit deadlines). The periodic task parameters were generated using UUniFast [BB05] algorithm to get uniform tasks distribution. The aperiodic tasks were generated with release time r in the interval [0, SC), the WCET C in the interval [5, 10] and the deadline d = DLX $\times C$, where DLX is the deadline extension factor. The factor DLX defines the tightness of the deadline compared to C, i.e., DLX $= (d - r)/C$. The generation process of the aperiodic tasks is stopped when the aperiodic utilization reaches the target utilization.

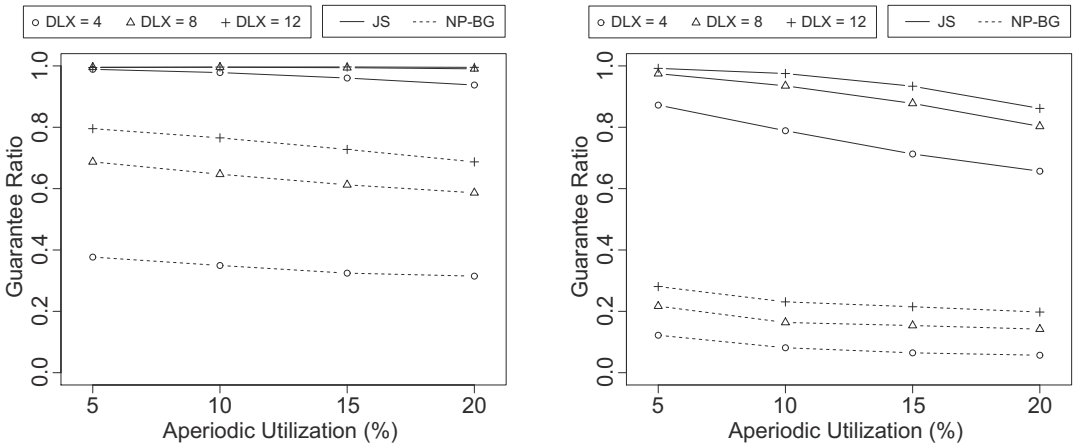
To distinguish the effects of varying urgencies, task-sets were generated for each different DLX factor in the list {4, 8, 12} and the aperiodic tasks utilization in the list {5%, 10%, 15%, 20%}. The guarantee ratios (i.e., the ratio of the number of guaranteed aperiodic jobs to the total number of aperiodic jobs) of the algorithms are also affected by the partition supply and the total utilization of the periodic tasks. Therefore, all the task-sets were generated for two different partition supplies in the set {70%, 50%} and two different periodic task demands in the set {25%, 35%}.

As mentioned in Section 4.2, the job-shifting (JS) algorithm requires a cyclic executive schedule for the partition. Therefore during the pre-processing stage, the partition schedule S_p is generated using the EDF scheduler. The task-sets which resulted in SC lengths outside of the interval [500, 5000) were rejected since the real-world tables are smaller [Sch15]. Moreover, the task-sets were also filtered using the offline guarantee analysis method mentioned in Section 4.4.2.

4.5.2 Results and Discussion

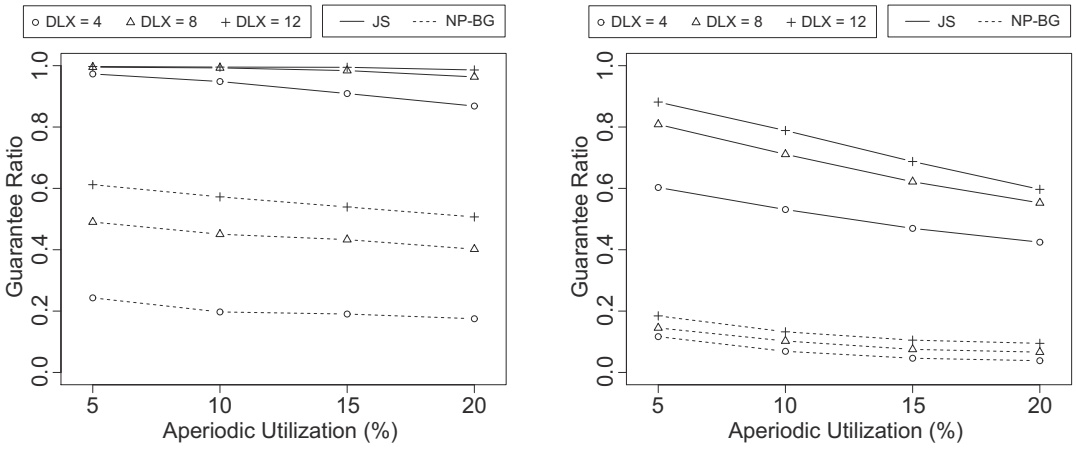
The figures 4.8 and 4.9 show the average guarantee ratios for the generated task-sets (i.e., each point in the figures represents the average guarantee ratio of 1000 task-sets) for varying task utilizations. In the figures, the DLX factor is represented with different pointer types and the schedulers (JS and NP-BG) with different line types. For the non-preemptive background scheduler (NP-BG), the guarantee ratio defines the ratio of the number of aperiodic jobs which finished before their deadlines to the total number of aperiodic jobs. Note that the NP-BG scheduler provides the least average guarantee ratio which can be achieved by a simple background queue.

The figures 4.8 and 4.9 show that, for the background scheduling, the number of



(a) *70% Partition Supply* **(b)** *50% Partition Supply*

Figure 4.8: *25% Periodic Task Utilization*



(a) *70% Partition Supply* **(b)** *50% Partition Supply*

Figure 4.9: *35% Periodic Task Utilization*

finished aperiodic jobs decreases to a great extent (e.g., from 0.7 to 0.2 for DLX=12, aperiodic task utilization $U_{ap} = 20\%$ and periodic task utilization $U_p = 25\%$) due to just 20% decrement in the partition supply. However, for the job-shifting algorithm, the number of finished aperiodic jobs did not suffer as much (e.g., from 1.0 to 0.9 for DLX=12, aperiodic task utilization $U_{ap} = 20\%$ and periodic task utilization $U_p = 25\%$). When the periodic task demand is increased from 25% (in Figure 4.8) to 35% (in Figure 4.9), the guarantee ratios of all the schedulers are decreased. However, the JS scheduler provides way better guarantee ratio than the NP-BG scheduler (e.g., ≈0.1 for NP-BG vs ≈0.5 for JS for $U_{ap} = 20\%$, $U_p = 35\%$ and partition supply = 50%). Notice that for the extreme case of $U_{ap} = 20\%$, $U_p = 35\%$ and partition supply = 50% in Figure 4.9b, a guarantee ratio of lower than 1 is inevitable as the partition supply is less than partition demand.

In summary, the average guarantee ratio of the job-shifting algorithm is quite large compared to the background processing. Therefore, we conclude that the job-shifting algorithm is more suitable compared to the background processing for online aperiodic job admission in hierarchic non-preemptive systems.

4.6 Overheads Evaluation

In this section, we present the results of the experiments performed to evaluate overheads incurred by the job-shifting algorithm. In Section 4.6.1, we provide the description of the experimental setup. While in Section 4.6.2, we present and briefly discuss the obtained results.

4.6.1 Experimental Setup

In order to evaluate the job-shifting overheads, we implemented the job-shifting algorithm in the DREAMS project[5]. An introduction to the hardware and software platforms and the toolchain used in the DREAMS project was provided in Section 1.1.6. In the following, we provide a description of the safety critical demonstrator (developed as part of the DREAMS project), the job-shifting integration strategy and the system deployment specification we used for job-shifting evaluation in the DREAMS project.

Safety Critical Demonstrator [DFG+16, KGGP+16]

In order to assess the technologies developed within the DREAMS project, a demonstrator of a safety critical system was developed by Thales[6]. The demonstrator can be seen as a distributed system composed of three platforms (2 NXP QorIQ T4240s and a DHP) interconnected via a TTEthernet switch [DFG+16]. The safety critical demonstrator is composed of four different applications (APP1–APP4) communicating with each other as shown in Figure 4.10. To carry out different operations, each application is composed of different periodic and aperiodic tasks as shown in Table 4.6. Apart from

[5] `http://dreams-project.eu/`
[6] Thales Group - `https://www.thalesgroup.com/en`

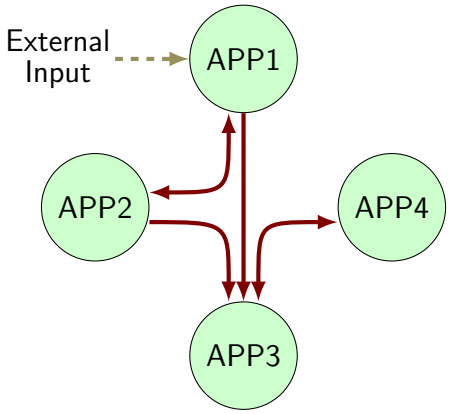


Figure 4.10: *Demonstrator communications (solid arrows) and external inputs (dashed arrow)*

Table 4.6: *Tasks for the safety critical demonstrator in DREAMS project*

Application	# Periodic Tasks	U_p	# Aperiodic Tasks	
			Comm	Ext. Input
APP1	6	0.021	1	17
APP2	10	0.545	1	0
APP3	5	0.055	3	0
APP4	1	0.015	0	0
Total	22	0.636	5	17

the communicating tasks, there exist a number of tasks which are activated by external events (shown as dashed arrow in Figure 4.10), e.g., push of a button.

Job-Shifting Integration

In order to generate the input required to execute the job-shifting algorithm for partition p, the web-based tool Xoncrete [BMR+10] is used to generate the inter- and intra-partition schedules for XtratuM [CRM+09] hypervisor. Afterwards, a command-line tool called MCOSF (abbreviation of 'Mixed-Criticality Offline Scheduling Framework') is used, which calculates the flexibility coefficients x for each job of partition p until the end of the scheduling cycle SC (defined in Section 4.2). The generated information is then forwarded to the Local Resource Scheduler (LRS) [KGGP+16] of partition p via Partition Configuration File (PCF) (see Figure 1.4).

In this work, the partition LRS is modified to implement the job-shifting algorithm with dense time-base [Kop92] (see Section 4.2) on top of cyclic executive intra-partition scheduler (CEIPS) [ARI03]. As CEIPS does not make use of a ready-queue, a simple job dispatch table for each partition slot [CRM+09] is specified in the Partition Configuration File (PCF). To detect the aperiodic job activation, the sources of the aperiodic task activation are also specified in the PCF (i.e., virtual network ports for message activated tasks, and interrupt sources for tasks activated due to external inputs). In order

to avoid the implementation of dynamic memory management, the memory required for accommodating newly released aperiodic job(s) is statically reserved. Moreover, similar to the heuristic proposed by Schorr [Sch15], at most one aperiodic job is guaranteed per LRS activation in order to bound the scheduler overheads.

Besides the constraints mentioned in Section 4.2, the scheduler CEIPS puts forward another constraint; A job started in a partition slot s_p [CRM+09] must finish before the end of s_p. In other words, a task cannot be paused at the partition boundary. To accommodate this constraint in the job-shifting algorithm, each partition blocking $v \in V$ (i.e., the opposite of partition slots in XtratuM [CRM+09]) can be considered a job for the guarantee test/procedure (with $a = b, C = m - b, f = m$ and $x = 0$). The advantage of such an assumption enables the use of flat scheduling model for the implementation in the LRS, in turn reducing the run-time complexity of the job-shifting algorithm (See Section 4.4.4).

System Deployment

As mentioned, each application of the safety critical demonstrator consists of a number of periodic and aperiodic tasks as defined in Table 4.6. The tasks sending a message to other task(s) are executed as periodic tasks, during which they decide if data needs to be communicated; while the tasks receiving messages from other task(s) are executed as aperiodic tasks[7] when data is received on a defined virtual network port. Moreover, the tasks activated by external inputs (shown with dashed arrow in Figure 4.10) are implemented as aperiodic tasks.

Even though the DREAMS Harmonized Platform (DHP, see Section 1.1.6) provides multiple cores, only one of the ARM cores running the XtratuM hypervisor was used for the overhead evaluation. This approach enables evaluation with large enough core utilization, further exhibiting benefits of the job-shifting algorithm over reservation based or background approaches. Moreover, other DREAMS services (e.g., global resource management) were disabled as their focus is orthogonal to the overheads evaluation of this study.

For the overheads evaluation of the job-shifting algorithm, the four demonstrator applications were each deployed in their own user partitions, while the resource management applications, i.e., the LRMs and MONs (see Section 1.1.6), were deployed in system partitions on a single ARM core of the DHP. Moreover, the external inputs to activate aperiodic tasks were triggered with a defined probability to simulate event occurrence. This strategy enables more frequent aperiodic activations compared to generating the events manually or through measurements in the environment.

Due to large range of the task periods, the generated hypervisor schedule (i.e., inter-partition schedule) resulted in a scheduling cycle length SC of 40s. To provide a rough idea of the inter-partition schedule, the initial part of the schedule is shown in Figure 4.11, while the range of flexibility coefficients and the activation probabilities for

[7] With the exception of the communication from APP3 to APP4, to evaluate overheads for partitions without any aperiodic task.

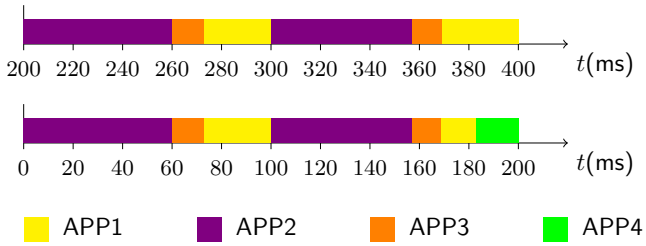


Figure 4.11: *Hypervisor schedule for safety critical demonstrator*

Table 4.7: *Application deployment parameters*

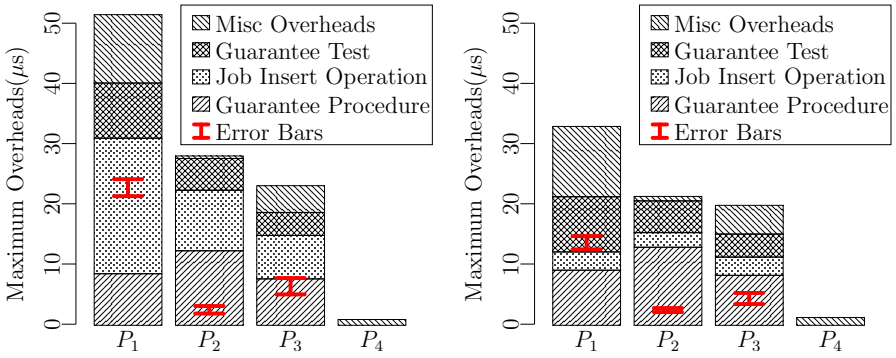
Application	Flexibility Coefficient (x)		Activation Probability	
	min (ms)	max (ms)	min	max
APP1	13	30	0.008	0.35
APP2	0	4	NA	NA
APP3	0	7	NA	NA
APP4	0	2	NA	NA

tasks activated by external events are provided in Table 4.7. In the table, the activation probability of 1 means that a job of the aperiodic task is released every second.

4.6.2 Results and Discussion

Figure 4.12 shows the bar plots of the measured maximum overheads for each partition utilizing job-shifting scheduler within 1000 scheduling cycles SC (See Section 4.2). Figure 4.12a shows the maximum overheads for the case where the schedule is implemented as a static C array, while Figure 4.12b shows the same for the case where the schedule is implemented as a double linked list. As mentioned earlier, the dynamic memory management scheme is not implemented, due to which the size of the double linked list is statically defined but still permits O(1) add job operation for the guarantee procedure. In order to exhibit the difference between the two implementations, the job insert operation (the first step in guarantee procedure) is shown separately from the guarantee procedure.

The error bars in Figure 4.12 represent the 98% confidence interval for the cumulative overheads. The legends in Figure 4.12 represent the overheads distribution, where 'Job Insert Operation' overheads represent the time elapsed in inserting a newly released aperiodic job in the job dispatch table and the 'Misc Overheads' represents the rest of the scheduling overheads (e.g., next job selection, looping, updating pointers). It is important to mention here that the graph in Figure 4.12 includes the logging overheads and excludes the overheads due to the detection of aperiodic task arrival, since these overheads are dependent on the aperiodic activation source, the hyper-calls implementation, and the platform architecture. For the aperiodic tasks activated by the reception of a message on a virtual network port, the overheads were measured to be approximately 13μs per aperiodic task. Whereas the detection of the aperiodic tasks triggered by the



(a) *Schedule implemented as a C array.*

(b) *Schedule implemented as a double linked list.*

Figure 4.12: *Job-Shifting overheads on Zynq board (Error bars represent 98% confidence interval for cumulative overhead).*

external events required approximately 2μs per aperiodic task.

Figure 4.12 shows that the overheads incurred by the job-shifting algorithm strongly depend on the number of tasks inside a partition. Moreover, the overheads of the guarantee test and the guarantee procedure depend on the number of periodic tasks, the number of aperiodic tasks and the flexibility of guaranteed tasks inside a partition. Note for the double linked list implementation of the schedule (see Figure 4.12b), the job insert operation took merely 3 to 4μs. Although with this implementation the rest of the overheads are slightly increased, the cumulative overhead is decreased.

It was also observed that, on average, 22.6 aperiodic jobs were released per second during the execution of 1000 scheduling cycles SC. Furthermore, none of the released aperiodic jobs missed its deadline and all the aperiodic jobs were guaranteed for the first invocation of Equation 4.4 due to relatively smaller utilizations of periodic tasks and good enough aperiodic room R.

In 2015, Schorr [Sch15] measured the overheads of the slot shifting algorithm on a cycle accurate MPARM simulator running at 200MHz. He mentioned that, for his synthetic task-set, the overheads for slot shifting acceptance test ranged from 8.695μs to 23.7μs, while the overheads for the guarantee procedure ranged from 9.82μs to 42.99μs (excluding overheads due to job insert operation, see experiment 3 by Schorr [Sch15]). For our safety critical demonstrator application, Figure 4.12 shows that the overheads of job-shifting algorithm are comparatively smaller (i.e., 3.825μs to 9.17μs for guarantee test and 7.71μs to 12.3775μs for guarantee procedure). The smaller overheads for job-shifting on the Zynq ZC706 platform were partly expected as the operating frequency of the platform is twice compared to the MPARM. However, the overheads with job-shifting algorithm are smaller than half of the overheads with slot shifting, which manifests that the overheads incurred by the job-shifting algorithm are quite *comparable* to preemptive algorithms like slot shifting [Foh94, Sch15]. Note that this comparison does not indicate superiority of job-shifting algorithm over slot shifting algorithm, due to varying system models and test scenarios.

Chapter 5

Mode-Changes and Fault-Tolerance

Real-time systems require accuracy and timeliness to control and manage physical processes. As the working environment of such systems undergoes major changes, they need to support adaptivity in order to adjust to the new operating conditions. Similarly, when the system resources change, e.g., component failure or dynamic plug-in of components, a real-time system must take adequate action to accommodate the change without casting a major impact on the system performance.

In order to support adaptivity in systems with priority-based scheduling, tasks can be created or destroyed based on (often) mutually exclusive phases of operations. However, for such systems it is difficult to guarantee schedulability during the change of operation phases [Foh94]. The reason for this difficulty is intrinsic pessimism and uncertainty in determining the system state at the phase- or mode-change instant.

For offline scheduled systems, the task of determining the system state becomes easy, as everything is (assumed to be) known before system run-time. After constructing an offline schedule, the system state at a given point in time is defined by the scheduling table. However, due to the (relative) ease of scheduling, offline scheduling is often utilized for systems with complex constraints. Due to these constraints, it is not trivial to see if a mode-change at a given time will lead to feasible task execution.

Over the last few decades, several approaches have been proposed to support mode-changes in offline scheduled real-time systems. In 1994, Fohler [Foh94] defined an approach to generate offline time-triggered (TT) schedules and to estimate the worst-case mode-change delay by defining safe points in time. However, the approach defined by Fohler is limited to a single partition, a single processor and a single criticality level. Recently, Real et al. [RSC16] defined an approach for adaptivity in hierarchic real-time systems. They embellished the offline schedule with mode-change partition slots, which are responsible for executing a mode-change upon a mode-change request. Due to the generation of special mode-change partition slots, the mode-change instants for this approach are limited. Moreover, they used Vestal's model [Ves07] for mixed-criticality applications, which is considered impractical in industry [ENNT15].

In this chapter, we present a novel approach for generating scheduling tables for transition modes [Foh94] and defining mode-change instants in hierarchic industrial mixed-critical environment. Our approach guarantees task-set schedulability during a mode-change and reduces mode-change delays. In this work, we focus on offline

scheduled *service* modes responsible for system survivability in order to provide fault-tolerance against core failures.

The rest of the chapter is organized as follows; In Section 5.1, we describe the state-of-the-art approaches to handle mode-changes and present their limitations. In Section 5.2, we provide details of the concepts which can help understand the developed approach. In Section 5.3, we provide a description of the used notations and present the system model. In Section 5.4, we present our approach for generating transition mode schedules and for defining safe points for executing a mode-change. And in Section 5.5, we provide an overview on how our approach is used in the DREAMS project.

5.1 Related Work

As the classification of modes is only dependent on the way they are used (see Section 1.1.2), all mode types can be scheduled in similar ways irrespective of their classification. However, some scheduling approaches are more suitable for specific mode types. For instance, Fohler [Foh93] used *operation* modes in MARS project and defined a scheduling approach to maintain strict temporal and causal ordering of events by using sparse time-base [Kop92]. In order to guarantee that the system remains schedulable during a mode-change, Fohler defined blackout slots (where a slot is the scheduling granule as defined by the sparse time-base [Kop92]). The blackout annotation for a slot is calculated offline, where a slot is marked as blackout when a mode-change before the start of the slot under consideration leads to a deadline miss. In order to reduce the number of blackout slots, Fohler proposed a task scheduling approach, where a task is scheduled at the same point in time in all modes (where it is supposed to be executed). Note that the approach by Fohler did not consider mixed-criticality task model or hierarchic scheduling model. Moreover, his approach is limited to a single processor/core.

Recently, Real et al. [RSC16] proposed an approach to combine time-triggered schedule and priority-based tasks in a hierarchic schedule in order to reduce task jitter and temporal isolation of mixed-criticality tasks (based on Vestal's mixed-criticality model [Ves07]). The authors created a number of mode-change partition slots (which is different from the scheduling granule defined by Fohler [Foh94]) offline, which define if changing the mode during this partition slot is feasible. During the online execution, the scheduler checks if the mode-change request is pending at the start of each mode-change slot. Upon detection of a pending request, the mode-change is executed. After the mode-change, the new mode schedule is executed from the first slot (i.e., immediate mode-change as defined by Kopetz [KG94], see Section 1.1.2). To reduce the analysis complexity, the approach defined by Real et al. limited the number of tasks inside each partition to one.

Our approach is similar to Real et al. [RSC16], however, we focus on (i) mode-changes between *service* modes (see Section 1.1.2), (ii) industrial mixed-criticality task model, (iii) guaranteeing schedulability during a mode-change, (iv) estimation of safe points for mode-changes and (v) multiple time-triggered tasks per partition slot. Moreover, in our approach we only consider time-triggered tasks as they provide strict guarantees for easy validation and certification [ENNT15]. Furthermore, instead of restricting the point of

mode-change (as by Real et al. [RSC16]), we define mode-change blackout intervals based on the partition schedule by extending the concept of mode-change blackout slots defined by Fohler [Foh93].

5.2 Background

In this section, we provide a brief introduction of common terms used in mode-change analysis and discuss their relationships. Furthermore, we also discuss how these terms are used in literature and what functionality is required in order to support mode-changes.

5.2.1 Reconfiguration Graphs

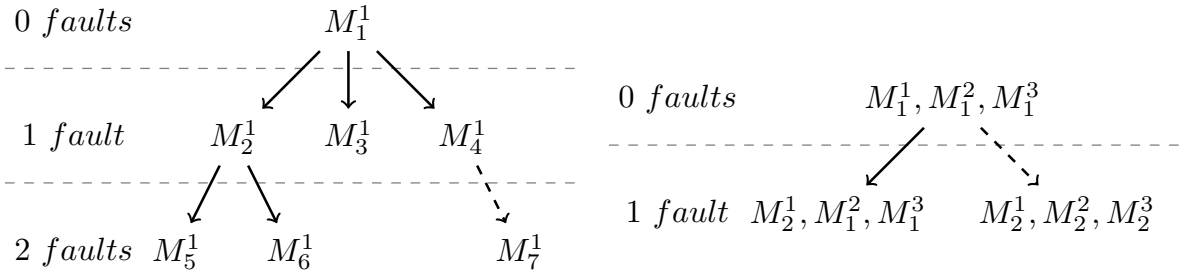
In order to construct an adaptive offline scheduled real-time system, a large number of mode schedules might be required to accommodate for different changes. Based on the occurred event, the currently executing mode schedule is switched to a new mode schedule. In order to give a global view of the changes required for a particular fault, a reconfiguration graph is often used [BV15, DFG+16]. This graph defines an order for switching between modes and, therefore, guarantees deterministic system state at a given point in time. In the following, we discuss how different hypervisors implement reconfiguration graphs and which approach is utilized by the DREAMS project.

Reconfiguration in Hypervisors The PikeOS [SYS17] hypervisor provides support for multiple modes by defining a time partition *scheme* for each mode of operation. Although PikeOS does not define an explicit reconfiguration graph or execution order of modes, the `SCHED_BOOT` scheme is executed at system start-up and the new scheme is either started immediately [KG94] or at the end of the hyper-period or the Major Application Frame (MAF) depending on the request by the user application. Contrary to PikeOS, the Xen [BDF+03] hypervisor is only capable of executing cyclic executive [ARI03] for one core [TLF17] and only supports a single mode of operation.

The XtratuM [CRM+09] hypervisor defines a *plan* as an implementation of a mode that allows reallocation of tasks on platform cores in a controlled way. In XtratuM, a plan defines how the partitions and the tasks inside them are scheduled. A plan is defined for a fixed length of time (i.e., MAF), and is repeated periodically. The XtratuM plans may be established immediately or at the end of the MAF depending on the event, e.g., a fault or a user request. In XtratuM, plan 0 and plan 1 are reserved for system services, while the later plans are defined as user plans. Plan 0 is termed as the initialization plan (similar to the `SCHED_BOOT` scheme in PikeOS), where partitions are brought to operation. Upon a user request, a plan $p_i \in P$ can be changed to another plan $p_j \in P$, such that $j \neq 1$. In case of an error detected by the XtratuM health monitor, an immediate mode-change is executed to switch to plan 1 (or p_1).

DREAMS Resource Management: Revisited In the DREAMS project, a reconfiguration graph defines an order for switching between modes and can be of two types; local and

global. A local reconfiguration graph (LRG) defines the mode-change order for a single resource, e.g., processor or a network switch. A global reconfiguration graph (GRG) provides the system wide notion of a mode and, therefore, a GRG is a combination of all LRGs in the system. Figure 5.1 shows examples of LRG and GRG [DFG+16] as seen by the DREAMS resource management services, where graph nodes represent modes, solid arrows represent local reconfigurations while dashed arrows represent global reconfigurations. In the figure, M_i^j represents the mode with identifier i for processing node or switch j.



(a) *Local reconfiguration graph (LRG)* **(b)** *Global reconfiguration graph (GRG)*

Figure 5.1: *Reconfiguration graphs in DREAMS project [DFG+16]*

5.2.2 Transition Modes

Fohler [Foh94] pointed out that a system, which is schedulable in individual modes, might be unschedulable during a mode-change. The reason for this unschedulability might be transition overheads or sudden release of new workload from the new mode, possibly with urgent deadlines. To tackle this issue, Fohler created mode-change blackout slots to prevent mode-changes during the defined slot duration. Moreover, he considered reconfigurations between modes as separate temporary modes, and therefore termed them transition modes. Using transition modes, the worst-case mode-change delays can be reduced by breaking down long event chains (further in the next section). Similarly, Real et al. [RSC16] pointed out that it might be infeasible to start a new task from the destination mode immediately at the point of mode-change (as memory needs to be reserved for the new task). In such a situation, a transition mode can be used to prepare the platform for the destination mode.

From the perspective of scheduling, a transition mode might provide benefits and, therefore, the online scheduler needs to distinguish between two types of modes. A *continuous* mode is executed cyclically until there is a mode-change request. This type of mode is defined by the system designer or the application. On the contrary to a continuous mode, a *transition* mode is executed at most once [Foh94] or long enough such that the platform becomes ready to execute the new mode. As soon as the required condition for switching to the new mode is fulfilled, a transition mode switches to a continuous mode when there is a safe point to switch. A transition mode can be defined

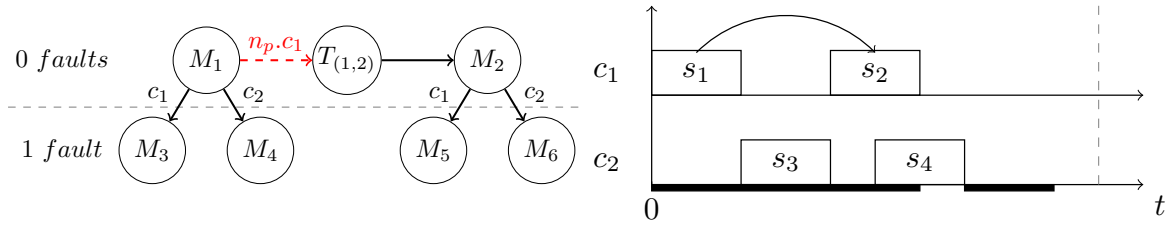
by the scheduler to decrease the mode-change delay in the presence of long event chains and/or by the system/application designer to initiate a service before the new continuous mode is established.

5.3 System Model

In this chapter, we focus on the DREAMS project and, therefore, assume that the system is subjected to (temporal and spatial) partitioning enforced by the XtratuM hypervisor. We utilize the industrial mixed-criticality task model [IEC10, DC11, ARI03, ISO11] and therefore assume the time-triggered (TT) scheduling approach with dense time-base [Kop92] for inter-partition scheduling and cyclic executive [ARI03] for intra-partition scheduling. We execute the tasks on a multi-core platform. However, we do not consider the interferences due to shared resources since, in DREAMS, the safety-critical workload is assumed to be executed in isolation, for instance using a periodic memory server. Moreover, we focus only on the task schedule and assume that the network schedule is generated using an all-in-one [Foh94] approach.

In this work, we assume that the intended purpose of modes is limited to *service modes* [GB13] with fault-tolerance only against core failures. The mode-changes are assumed to be enforced in two ways [Foh94, KG94]; immediate and deferred. For the explanation of our approach, we will assume an immediate mode-change implementation as defined by Fohler [Foh94] (i.e., the destination mode schedule is executed with an offset equal to the current position in the source mode schedule), as opposed to the one defined by Kopetz [KG94] (i.e., the destination mode schedule is executed from the start). However, we will generalize these two implementations and relax this assumption in Section 5.5. All mode-changes are assumed to be executed by the hypervisor and the respective mode-change information is assumed to be conveyed to the resource management services.

In Section 1.1.6, an introduction to the components local resource manager (LRM) and global resource manager (GRM) was presented. Moreover, a brief description of the local reconfiguration graphs (LRG) and the global reconfiguration graph (GRG) was presented in Section 5.2.1. In the following, a reconfiguration graph is modeled using a directed graph $\langle \mathfrak{M}, E \rangle$, where graph nodes $\mathfrak{M}$ represent modes and the edges E between them represent the transitions/reconfigurations. A mode transition can be one of two types; global transition (activated by the GRM) and local transition (activated by the LRM). A continuous mode with identifier i is represented with $M_i \in \mathfrak{M}$, while a transition mode is represented with $T_{(i,j)} \in \mathfrak{M}$, where i is the source mode identifier for mode M_i, and j is the destination mode identifier for mode M_j. Figure 5.2a shows an example of a local reconfiguration graph for a processing node n_k, where solid edges represent a local reconfiguration (activated by the LRM), dashed edges represent a global reconfiguration (activated by the GRM), while the number of local faults is represented by the level/depth of the graph node. Note that different edges of the graph node represent different failures. For instance, in Figure 5.2a, M_1 is changed to M_3 when core c_1 of node n_k fails, M_1 is changed to $T_{(1,2)}$ when core c_1 of node n_p fails, and $T_{(1,2)}$ is changed to M_2 either at the end of the Major Application Frame (MAF) or at the



(a) *An example of a local reconfiguration graph* **(b)** *An example of a scheduling table*

Figure 5.2: *Notations and examples*

end of the blackout interval. It is important to mention here that, in this chapter, we only focus on LRGs since a GRG is a projection of multiple LRGs. Furthermore, we assume that the LRGs and their respective schedules are generated before mode-change analysis.

In this chapter, a partition slot s represents a single temporal allocation of budget for a specific partition (and its tasks) on a core and is defined using a slot start instant and an end instant. The interval between the start and the end of a partition slot defines when the processor is available to service the tasks belonging to this partition. Note that this slot does not refer to the task schedule granularity (as defined by Fohler [Foh94]). In this chapter, we make no distinction between the slots of different partitions, i.e., all slots are considered separate entities by the scheduler irrespective of their association with a particular partition.

Similarly to the blackout slots defined by Fohler [Foh94], we create mode-change blackout intervals, each defined by an interval start time instant and an end time instant. The mode-change blackout interval defines if changing a mode during this interval leads to consistency problems. Note that, due to the dense time-base, our blackout intervals are different from the blackout slots defined by Fohler [Foh94]. In this work, a blackout interval does not represent a simple boolean property of the scheduling granule. Instead, it represents the start and end of the mode-change blackout for the complete platform (i.e., for all cores C) and is only defined when necessary. Note that the mode-change blackout intervals are only required for reconfigurable modes i.e., non-leaf nodes in the local reconfiguration graphs. An example of a schedule is provided in Figure 5.2b where

Table 5.1: *Summary of used notations and symbols*

Symbol	Sub-element	Description
$\mathfrak{M}$	M	Modes in a reconfiguration graph
n		Processing node
C	c	Cores on a processing node n
$T_{(i,j)}$		Transition mode between modes M_i and M_j
s		Partition slot

the thick black lines along the time axis represent mode-change blackout intervals. In the figure, the cores on the platform are represented with $c \in C$, while a communication or an event chain [BDM+17] between two partition slots is shown by a connection over the slots. In the context of the DREAMS project, an event chain [DRE15] or an end-to-end flow (ETEF [BMR+10]) defines a relationship between events, e.g., precedence relationship between jobs or preemption and resumption of a job at a partition slot boundary. A summary of notations defined in this section is provided in Table 5.1.

5.4 Fault-Tolerance and Schedule Generation

In this section, we present our approach for transition mode schedule generation and mode-change delay analysis. In Section 5.4.1, we discuss how the local reconfiguration graphs (LRG), defined by the DREAMS resource management services, grow with respect to the number of tolerable faults and processor cores. In Section 5.4.2 and 5.4.3, we describe how mode-change blackout intervals can be generated and, based on that, how to estimate worst-case mode-change delay. In Section 5.4.4 and 5.4.5, we explain where and how to add new transition modes to reduce mode-change delays, respectively.

5.4.1 Growth of Local Reconfiguration Graphs

Assuming that all the cores are utilized to service tasks and a new mode is required for each core failure, Figure 5.3 shows the local reconfiguration graph (LRG) for a single processing node n_1 with 4 cores, i.e., $|C| = 4$. In the figure, the edge labels, indicating the failed cores, are removed to avoid confusion. Note that the LRG shown in Figure 5.3 does not support any global reconfiguration. Also notice that after 4 core failures, the complete node n_1 is failed and, therefore, no more modes of operation are required.

For supporting fault-tolerance against all core failures, the number of tolerable faults f for a processing node needs not be equal to $|C|-1$. Any number smaller than this value

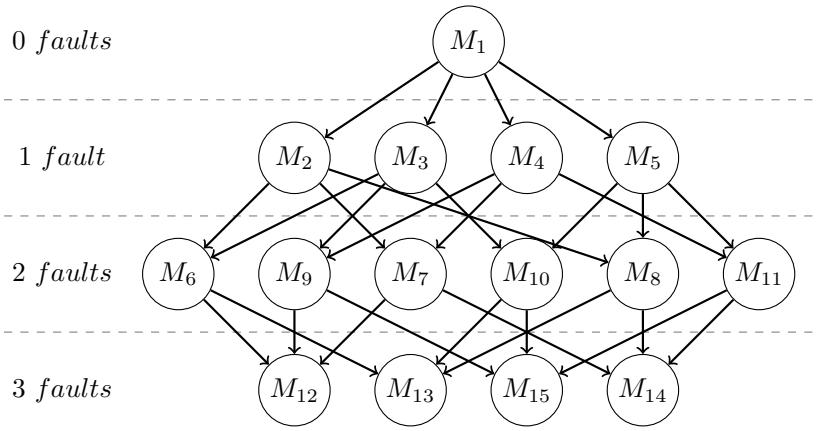


Figure 5.3: *Local reconfiguration graph (LRG) of a 4-core node without global reconfigurations*

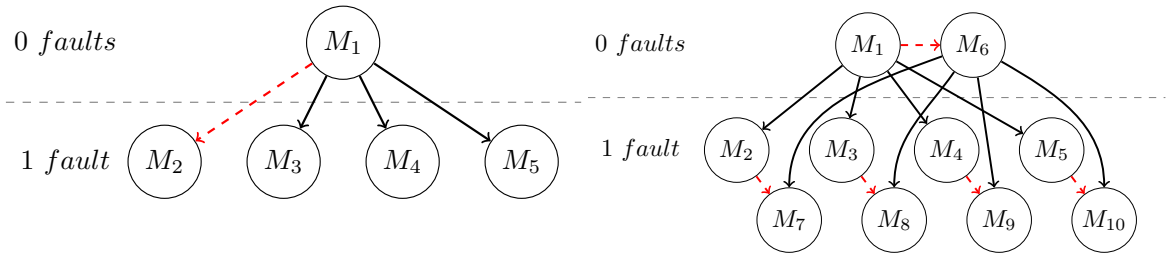
might affect proper handling of critical applications, since after $f+1$ faults (such that $f < |C|-2$) the node might be deemed overloaded or dead even when there exist working cores with idle time. Note that the minimum number of required local reconfigurations or modes L_i^{min} for node n_i depends on both the number of used cores and the number of tolerable faults. The following equation system can be used to calculate the value for L_i^{min} assuming no global reconfigurations.

$$\begin{aligned} P(j) &= \min(|C|, k+j-1) \\ L_i^{min} &= \sum_{0 \le j < f} \binom{P(j)}{j} = \sum_{0 \le j < f} \frac{P(j)!}{j! \times (P(j)-j)!} \end{aligned} \tag{5.1}$$

where k represents the number of used cores on processing node n_i and $P(j)$ represents an intermediate function which is used for the simplification of the equation. Note that the minimum number of reconfigurations L_i^{min} for an LRG can be very large, even for a small number of tolerable faults.

In order to provide an estimate of the total number of local modes after global reconfigurations, Figure 5.4 shows examples of the LRGs for two nodes with one tolerable core failure on each node and one global reconfiguration on node n_1. The figure shows that a failure of core c_1 on node n_1 leads to an infeasible schedule (see Figure 5.4a), due to which a number of applications need to be allocated to node n_2 using a global reconfiguration. In order to add a new global reconfiguration on node n_2, a new local mode has to be created from each local mode since node n_2 might be executing in any of its local modes at the time of global reconfiguration. Therefore, the LRG for node n_2 (after one global reconfiguration on node n_1) is shown in Figure 5.4b.

The example above shows that the total number of local reconfigurations or modes L_i^* on node n_i depends on (a) the minimum number of required local reconfigurations L_i^{min}, and (b) the number of global reconfigurations which lead to reallocation of new tasks on node n_i. The following equation provides the relationship between the minimum and the total number of local reconfigurations for a processing node after g global reconfigurations (which require new tasks to be reallocated to this node).



(a) *LRG for node n_1 with 1 global reconfiguration*

(b) *LRG for node n_2 with one global reconfiguration on node n_1*

Figure 5.4: *Local reconfiguration graphs (LRG) for two processing nodes with one tolerable core failure f on each node and one global reconfiguration on n_1*

$$L_i^* = L_i^{min} \times g$$

The above equation demonstrates a linear but significant rise in the number of local modes upon addition of even small number of global reconfigurations, since the value of L_i^{min} is usually large. Therefore, careful design techniques are required to reduce the number of transition modes, since they can further increase the size of LRGs.

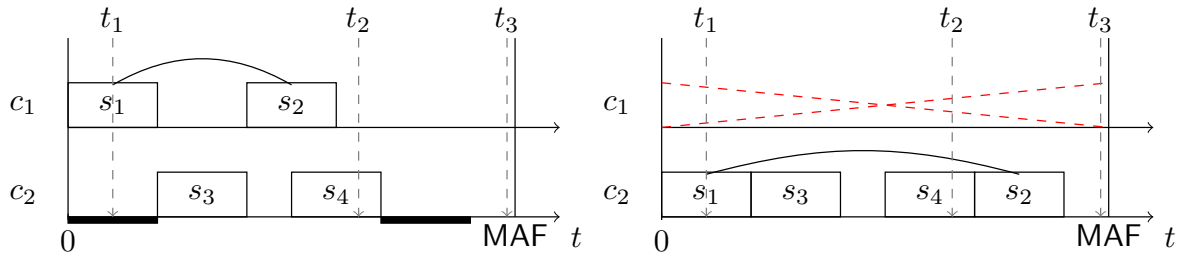
In DREAMS, the tool *GRec* (further explained in Section 5.5.1) is responsible for generating LRGs for all the local resource managers and GRG for the global resource manager. It performs a number of optimizations to reduce the number of modes. For instance, a new mode after a core failure is only added when the core, which is failed in the destination mode, executes one or more tasks in the source mode.

5.4.2 Blackout Interval Generation

In this work, the mode-change blackout intervals are defined based on the presence of partition slots in the schedule, since (i) a mode-change might require task migration, which incurs significant overheads [Sar12] and (ii) the job execution times range from a few microseconds to several milliseconds [Sar12, PSG+14]. Such a blackout annotation is course-grained, as individual job executions cannot be controlled based on the source and the destination modes. However, the strategy defined in this chapter can also be used to define mode-change blackout intervals based on the state of individual jobs, if (i) the worst-case mode-change execution duration is negligible, and (ii) the jobs are not allowed online to execute before their scheduled time.

In the light of above discussion, we create a set of mode-change blackout intervals for each edge $e \in E$ of each LRG, which is stored in the hypervisor memory space before system start-up. When a mode-change request (released by a health-monitor or an application) is detected online by the hypervisor, it chooses the respective set of mode-change blackout intervals (depending on the failed core) and makes the decision about the mode-change execution. If the hypervisor finds out that the current time instant in the source schedule does not lie within a mode-change blackout interval, the mode-change is executed. If the current time instant *does* lie within a blackout interval, the mode-change is deferred to a point in time before the end of MAF. This deferrable point in time is marked by the end of a mode-change blackout interval.

For a source mode, the mode-change blackout intervals are defined for the size of all old and new partition slots, compared to the destination mode, on all cores. Figure 5.5 shows example schedules for the source and the destination modes of a dual-core platform, where the core c_1 in the destination mode has failed. Notice in the figure that mode-change blackout interval is defined in the source mode for all time instants where (i) there is no change in the core-local ready-queue for the same time instant between the source and the destination modes *and* an unserviced event is pending on the core other than the failed core, and (ii) there is a change in the core-local ready-queue on a core other than the failed core, e.g., t_1 in Figure 5.5. Note that the blackout intervals are generated assuming that there is no danger of executing tasks more frequently than



(a) *Source mode schedule* **(b)** *Destination mode schedule with c_1 failed*

Figure 5.5: *Generation of mode-change blackout intervals*

required. For instance, when the core c_1 fails just after slot s_2 and the mode is changed at time instant t_2 in Figure 5.5, the slot s_2 might be executed twice.

On modern platforms, the task migration takes considerable amount of time and causes WCET dilation [Sar12]. Although, the WCET dilation problem does not cast a great impact on tasks in DREAMS due to their non-preemptive nature, the task migration (simulated or otherwise [Bra11]) might consume time budget allocated for a specific partition slot. For example, when a mode-change is executed at time instant t_3 in Figure 5.5a, core c_2 might not be ready to execute the tasks from slot s_1 right away. In order to avoid these situations, the start of a mode-change blackout interval is preponed by a small amount δ when a partition is migrated (and executed right away) from one core to the other core of the same platform. This makes sure that the migration delays do not affect the timing behaviour of the tasks. In the context of this chapter, we refer to δ as *safety margin*. Note that the value of the safety margin δ depends on the platform and the migrated partition. Other than the task migration delay, the mode-change execution delay by the processor (which is usually a few microseconds [BMR+10]) can also be considered a part of δ.

After the generation of mode-change blackout intervals, a number of optimizations can be made. For instance, it can be made sure that the gaps between the blackout intervals is larger than δ, otherwise the overlapping intervals are merged. Note that, in the worst-case, the defined blackout generation approach might lead to a single blackout interval of length equal to MAF. The reason might be the execution of at least one partition slot on a core at any given time in the schedule. In order to avoid this situation and reduce the length of the mode-change blackout intervals, non-critical partition slots can be ignored during the blackout interval generation process.

5.4.3 Worst-Case Mode-Change Delay

The worst-case mode-change delay (WCMCD) is defined as the maximum time duration between the release of a mode-change request and the time instant when the new mode is established. Once the mode-change blackout intervals are generated offline as defined in the previous section, the WCMCD can be calculated for each edge $e \in E$ of each LRG. Since task-migration and mode-change execution delays by the processor are already taken into account, the WCMCD for an edge $e \in E$ is simply defined as the

length of the largest blackout interval plus a small amount of time γ required by the hypervisor to detect the mode-change request. The value of γ depends on the hypervisor implementation and the platform.

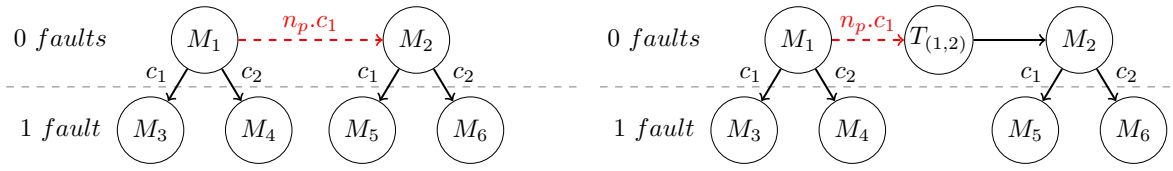
5.4.4 Reconfiguration Graph Enrichment

In the context of DREAMS, defining transition modes between GRG nodes might lead to complications as GRG is a merged graph of all LRGs. Therefore, transition modes in our approach are limited to the LRGs. Moreover, the local reconfigurations in LRGs (e.g., solid edges in Figure 5.4) are not required to have transition modes because in DREAMS, (i) all local edges have the same set of tasks/partitions but different number of resources (i.e., the continuous modes are service modes, see Section 1.1.2), (ii) there exists no intermediate service task which needs to be executed before establishing the destination mode, and (iii) individual tasks cannot be controlled based on the destination mode due to hierarchical scheduling. Therefore, the local edges in the LRGs are not embellished with transition modes. Note that, for the local mode-changes in LRGs, mode-change blackout intervals can still provide a possibility to reduce the mode-change delays.

In the light of above discussion, a transition mode is only added on global reconfigurations in LRGs. Figure 5.6 shows an example of the modification required in the LRGs. Note that the mode $T_{(1,2)}$ in the figure represents a transition mode, i.e., the entry in this mode is activated by the application or the health monitor but the exit is activated by the hypervisor at the end of the mode-change blackout interval or at the end of MAF, whichever is earlier.

5.4.5 Scheduling Transition Mode

Before starting the schedule generation process for the transition mode, the worst-case mode-change delay (WCMCD) between the source and destination modes, without the transition mode, is calculated as mentioned in Section 5.4.3. In order to generate transition mode schedule, (i) all the common slots (i.e., slots which start and end at the same time on the same core) between the source and the destination modes are copied in the transition mode schedule, (ii) long overlapping event chains in the source mode are identified, and (iii) the event chains from the source mode are added iteratively to the transition mode schedule such that the overlap is minimized (e.g., no other chain

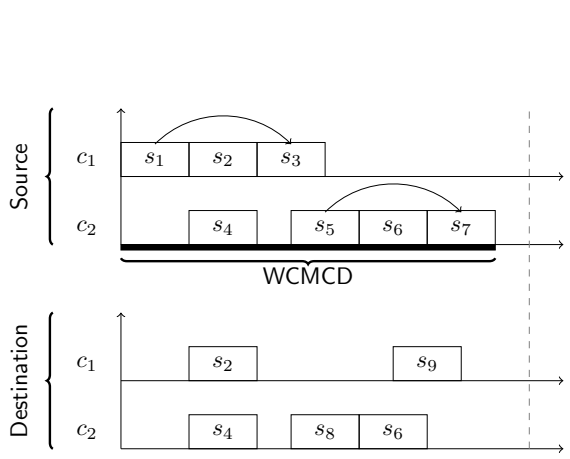


(a) *Before enrichment* **(b)** *After enrichment*

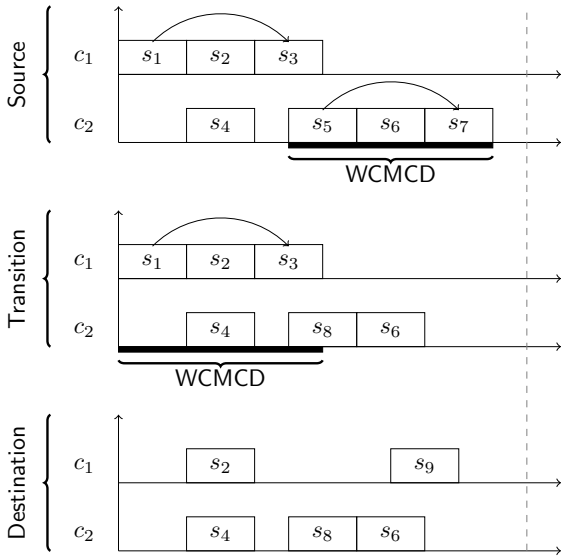
Figure 5.6: *Enrichment of the local reconfiguration graphs (LRG) with transition modes*

is started before the chain under consideration is finished). After the schedule generation, the mode-change blackout intervals between the source, transition and destination modes are temporarily recalculated and merged in a forward merge fashion (i.e., merging intervals only if the first interval in the overlapping sequence starts in the source mode). The WCMCD, based on the forward merged blackout intervals, is then recalculated similar to the process mentioned in Section 5.4.3. At the end of this process, if there is no improvement in the WCMCD, the newly added transition mode can be discarded.

An example for the WCMCD with and without a transition mode is shown in Figure 5.7. In the figure, consider a situation where the mode-change request arrives just after starting the slot s_1 in the source mode. In the absence of a transition mode (i.e., Figure 5.7a), the mode-change has to be postponed until the end of slot s_7, otherwise the schedule will result in either an incomplete event chain or an incomplete job. In such a situation, the slot s_8 in the destination mode will execute during the next MAF, i.e., the destination mode is established one MAF later. However with transition mode included (i.e., Figure 5.7b), the slot s_8 can be started earlier. Therefore, with the help of a transition mode, the new activities from the destination mode can be started even before the destination mode is completely established. Note in Figure 5.7b that by using a simple merge between the blackout intervals from the source and the transition modes, no change in the WCMCD will be observed. However, by using a forward merge, the WCMCD can be reduced.



(a) *WCMCD without transition mode*



(b) *WCMCD with transition mode*

Figure 5.7: *Transition modes and the worst-case mode-change delay (WCMCD)*

5.5 Mode-Changes & DREAMS

In this section, we describe the implementation of our approach, defined in Section 5.4, in the DREAMS project, its impact on the results and the learned lessons. In Section 5.5.1, we list sources of bottlenecks from the perspective of the DREAMS architecture. In Section 5.5.2, we describe how blackout intervals and transition mode schedules are generated using the DREAMS toolchain. In Section 5.5.3, we present a case study which indicates the performance gain due to the presented approach. And in Section 5.5.4, we discuss a number of optimization techniques which might provide benefits for future system design.

5.5.1 Implementation

In DREAMS, a mode-change is executed in the context of a local resource manager (LRM) partition. Although there can be more than one partition slot for each LRM per MAF, this strategy limits the applicability of our approach to LRM partition slots (similarly to the approach developed by Real et al. [RSC16]). In order to fully exploit the strategy defined in the previous section, the resource management services must not limit the instants at which a mode-change may be executed. Therefore, the hypervisor should be responsible for executing a mode-change and informing the resource management services about the new mode (as considered in Section 5.3). As the implementation of the mode-change blackout intervals and transition modes on the physical platform requires a number of functionalities from different domains (i.e., the hypervisor, the resource management services), the modification of the platform services is left up to the hypervisor designers and resource management service providers.

From the perspective of XtratuM, the mode-change blackout information for each edge in the LRGs needs to be stored and processed at the start and at the end of each partition slot. At each partition slot boundary, the hypervisor must execute a mode-change when (a) there is a pending mode-change request (e.g., generated by the DREAMS local resource monitor MON) from source mode s to destination mode d, *and* (b) currently executing position t in the scheduling table of mode s does not lie within a mode-change blackout interval (for the edge between modes s and d in the LRG). In the hypervisor defined by the DREAMS architecture (i.e., XtratuM [CRM+09]), the immediate mode-change refers to a change from mode s to mode d such that after the change, the destination mode d starts from the beginning (i.e., immediate mode-change as defined by Kopetz [KG94]). Although our mode-change approach was developed assuming an immediate mode-change implementation as defined by Fohler [Foh94] (see Section 5.3), the generated mode-change blackout intervals are still valid. Since mode-change blackout intervals merely indicate unsafe time points for executing immediate mode-changes, all safe time points for executing immediate mode-changes are equivalent. Therefore, during the online execution phase, any implementation for executing immediate mode-changes can be used.

5.5.2 Development Flow

A graphical representation of the DREAMS toolchain and a brief introduction to the tools inside the DREAMS toolchain were presented in Section 1.1.6. Here we only define the development flow in order to design a fault-tolerant system using the DREAMS architecture. The development process is started by model creation for the application, the middle-ware (i.e., the local and global resource managers, schedulers and monitors), the platform and the hypervisor using the tool `af3` [BDM+17]. The generated models are then forwarded to the tool `Xoncrete` [BMR+10], which generates a mapping between the application/hypervisor models and the platform model. The Xoncrete tool also generates the schedule for the first continuous mode (see Section 5.2.2) for each processing node. The mapping between the models and the schedules are then imported back as models in `af3`. Then, all the generated models are forwarded to the tool `GRec` [DFG+16] to generate different fault-tolerant modes, their schedules and a corresponding local reconfiguration graph (LRG) for each processing node and a global reconfiguration graph (GRG) for the global resource manager (GRM). The generated artifacts are imported back in `af3` and then forwarded to the tool `MCOSF` (see Section 1.1.6). The `MCOSF` tool calculates the mode-change blackout intervals and transition mode schedules, which are then imported back as models in `af3`. The `MCOSF` tool also reports the worst-case mode-change delay (WCMCD), which can then be checked against the application requirements. At the end, the platform configuration files for all processing nodes are exported using the tool `af3`.

5.5.3 Case Study

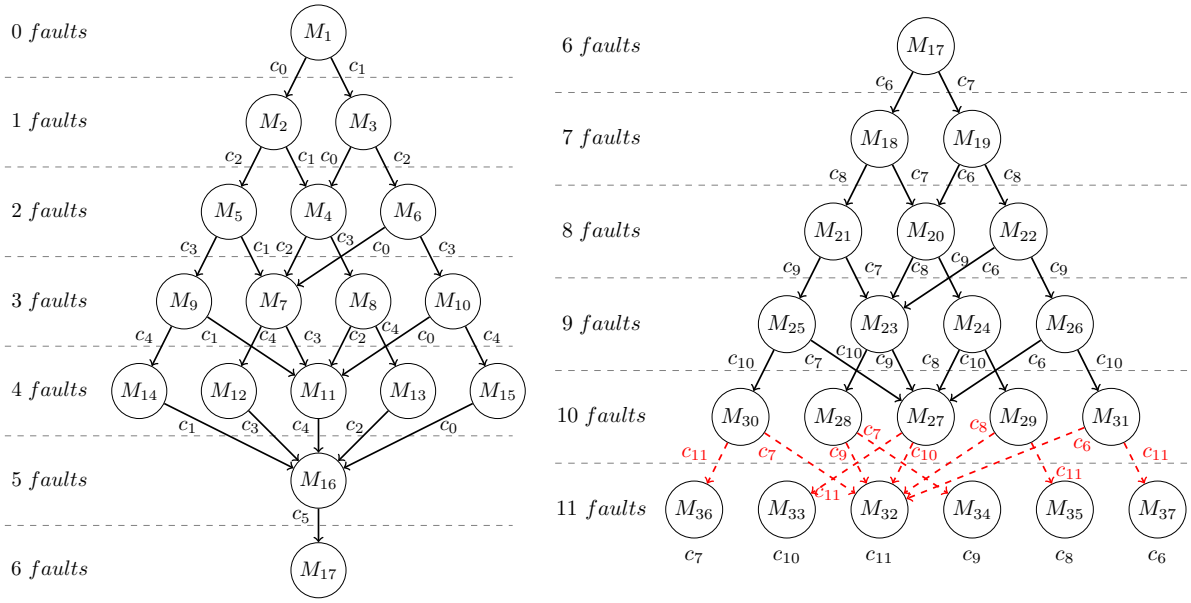
As the existing approaches do not fulfil the requirements of modern distributed real-time systems (e.g., industrial mixed-criticality model, hierarchical scheduling, etc.), a fair comparison between mode-change delays due to different approaches cannot be made. Therefore, a case study has been designed in order to assess the developed approach which consists of three applications [BDM+17]; (i) Research Open-Source Avionics and Control Engineering (ROSACE [PSG+14]), (ii) Moving Picture Experts Group (MPEG) server [ARSF11], and (iii) Control set-point generator (CSPG) for ROSACE. The ROSACE application consists of 12 tasks, while the rest of the applications consists of only a single task. In this case study, we consider two event chains from the ROSACE application; (i) `aircraft_dynamics` → `h_filter` → `Vz_control` → `elevator` [PSG+14], and (ii) `Va_control` → `engine`. Each of the applications are allocated to one application partition, which are then mapped to two processing nodes, i.e., (i) n_1 having Zynq board with GRM & CSPG on one of the two cores, and (ii) n_2 having NXP QorIQ T4240 with ROSACE & MPEG-Server on two of the 12 cores. It is important to mention here that all the applications are single core applications and the applications ROSACE & MPEG-Server are mapped on different cores. For the case study, the tolerable number of core failures f_1 for node n_1 is defined to be 0 (i.e., no expected core failure), while f_2 is defined to be 11. Moreover, to avoid *strict* [PSG+14] scheduling of jobs, the periods of certain tasks in ROSACE application were reduced [PSG+14]. In other words, some tasks were executed more often than

required by the event chains.

Observations After following the DREAMS development flow [BDM+17] mentioned in Section 5.5.1, it was observed that valid mode schedules and their corresponding reconfiguration graphs were generated with reduced mode-change delays. Furthermore, the MAF for all mode schedules was observed to be one second.

Figure 5.8 shows the local reconfiguration graph (LRG) for node n_2, of the case study, generated using the `GRec` tool. In the figure, the last remaining cores for the last modes (i.e., the leaf nodes in the LRG) are indicated below the graph nodes. Note that for node n_2, any two cores on the platform can be used (i.e., using other 10 cores as backup) at a time. Due to the LRG generation algorithm utilized by the `GRec` tool, the number of reconfigurations for node n_2 is small (i.e., 37 modes as shown in Figure 5.8, instead of 66 modes as estimated using Equation 5.1). Note that the LRG generated by `GRec` for node n_2 is incomplete, e.g., no mode exists when all but core c_2 has failed.

An example schedule with mode-change blackout intervals for the reconfiguration $M_1 \xrightarrow{c_0} M_2$ on node n_2 is shown in Figure 5.9, where thick rectangles represent the partition slots and the dashed lines inside the partition slots represent job start/finish. In the figure, the schedule for only the first $30ms$ from core c_0 of mode M_1 and core c_1 of mode M_2 is shown. It is worth mentioning that the temporal difference between two consecutive dashed lines in the figure does not represent the WCET of the job, instead these lines are merely used in the figure to separate one job from the other. In the figure, the two event chains are shown as A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ D and K $\rightarrow$ L. Notice that, due to the period relaxation [PSG+14], all the jobs of the tasks A, B, C, D, K and L



(a) *First half of LRG* **(b)** *Second half of LRG*

Figure 5.8: *Local reconfiguration graph (LRG) for T4240 node from the case study*

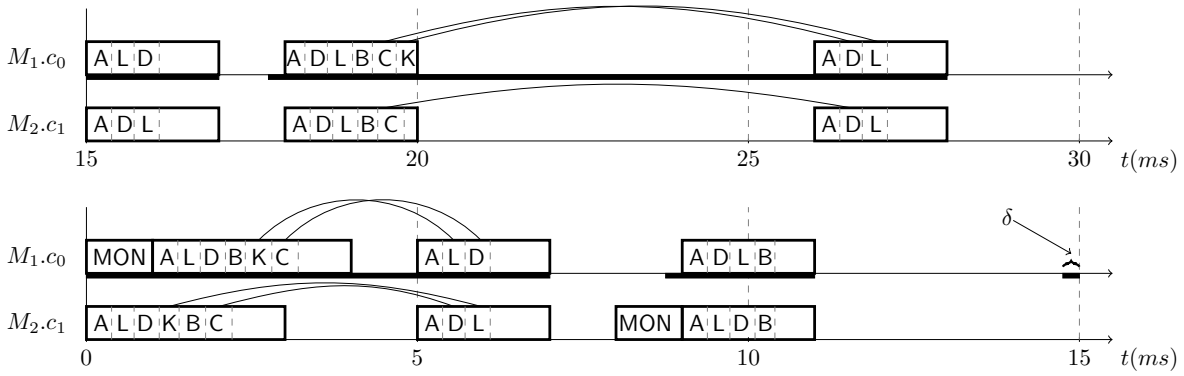


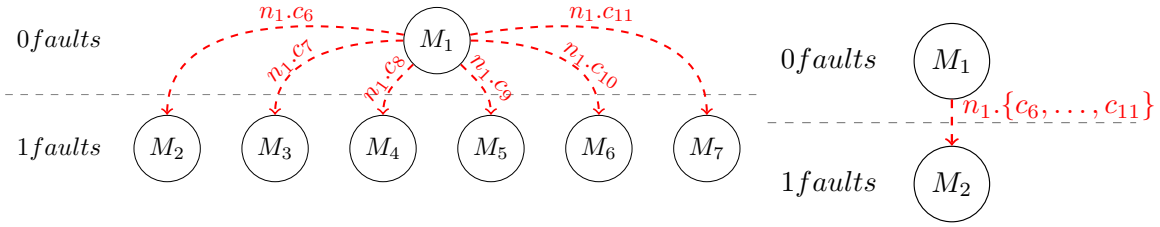
Figure 5.9: *Scheduling table sample from the case study*

do not contribute to the event chain. The figure shows that, upon failure of core c_0, the application partition slots are moved to core c_1. In the figure, observe that the blackout intervals exist for almost all partition slots since these slots execute on a different core in the destination mode. Moreover, there exists no blackout interval for the health monitor (MON) slot on core c_1 since this slot execute on the same core at the same time in both modes.

For this case study, the tool `MCOSF` was run with a safety margin δ (see Section 5.4.2) of $0.5ms$. After the execution of the tool `MCOSF`, it was observed that the worst-case mode-change delay (WCMCD) for all the transitions in the LRG was decreased from 1s (i.e., the hyperperiod, for deferred mode-changes) to $10.5ms$ (for immediate mode-changes) for node n_1 and from 1s to $6.5ms$ for node n_2. It is important to mention here that the resulting WCMCDs are as large as the longest event chain plus δ. Since the event chains are not modified due to a reconfiguration, all the WCMCDs for a platform were observed to be equivalent. Note that there exist only two application partitions per processing node, i.e., CSPG + MPEG-Server for node n_1 (after global reconfigurations) and ROSACE + MPEG-Server for node n_2 (before global reconfigurations). Due to this reason, no transition mode resulted in better mode-change delays and, therefore, the generated transition modes were discarded.

5.5.4 Optimizations

Considering that the `MCOSF` tool was not given full control over the generation process of the mode schedules and the reconfiguration graphs, a number of optimizations can be made for the future implementations of the DREAMS architecture. For instance, if the same partition is moved from node n_1 to node n_2 for a given set of global reconfigurations from node n_1, all these global reconfigurations from node n_1 can be mapped to one global reconfiguration on node n_2. For example, the LRG for the Zynq board from the case study, presented in Section 5.5.3, can be reduced to only two n_1 local modes as shown in Figure 5.10. Notice that this approach also leads to a smaller GRG for the GRM. Furthermore, it was observed that each core was responsible for executing only one



(a) *LRG before optimization* **(b)** *LRG after optimization*

Figure 5.10: *Local reconfiguration graph optimization for Zynq node from the case study*

partition. However, by mapping multiple partitions to a single core, the number of used cores can be minimized which, in turn, leads to less number of node-local modes (see Equation 5.1). Similarly, by creating a global reconfiguration only when the load in the source mode is at all not schedulable on the same platform, the number of global reconfigurations and the reconfiguration graph sizes can be greatly reduced. Note that this strategy is a heuristic approach since (i) the problem of mapping partitions to minimum number of cores is a variant of bin-packing problem, which is an NP-complete problem [CGJ78], and (ii) it might not be always practical to execute different partitions on the same core due to different criticality levels defined for the tasks inside the partitions.

From the job execution perspective, it can be observed that the tasks may not always execute for the complete WCET. Therefore, another approach to further reduce the WCMCD is to yield the processor/core to the hypervisor upon finishing the jobs from the current partition slot. Upon detection of a mode-change request and an early completion of the partition slot, the hypervisor can shorten the mode-change blackout interval and change the mode before the offline defined point in time.

As described in Section 5.4.1, a large number of modes are required in the DREAMS architecture to support fault-tolerance against core failures. Since the processing platforms utilized in DREAMS are homogeneous, if the reconfiguration graph generation strategy and the hypervisor kernel are allowed to be modified, a large number of reconfigurations can be saved by defining a single schedule for a mode with n core failures and dynamically moving the core schedules online depending on the failed core. In other words, instead of creating a new reconfiguration for each failed core c_x, one can create a new reconfiguration for n number of core failures, i.e., if any n number of cores fail, this new mode schedule is established.

Chapter 6

AUTOSAR Toolchain

The complexity of the embedded systems have increased tremendously over the last few decades. Moreover, the computing systems utilized in Cyber-Physical Systems (CPS) require deterministic and time bounded behaviour. In order to guarantee safe operation of CPS upon deployment, the hardware and software components of CPS are certified based on their respective safety constraints. As the certification process is expensive in terms of time and money, the industry is reluctant to make modifications to developed components. In order to support and ease certification process, new design technologies are being developed which provide modularity and scalability. These design technologies enable modular certification and provide uniform and deterministic component behaviour across multiple platforms.

In the automotive domain, an initiative to enable modular certification and to encourage industrial collaboration was taken by a number of car manufacturing companies and automotive original equipment manufacturers (OEMs). This initiative, termed as AUTomotive Open System ARchitecture (AUTOSAR) [AUT17b], defined an open standard for automotive software design. On top of an OSEK/VDX [OSE05] compliant Real-Time Operating System (RTOS), AUTOSAR uses model-based design and development processes by defining software components with standard interfaces. Due to the well defined interfaces and layered software architecture, AUTOSAR allows relocatability of software components and permits component reuse, modularity and scalability.

Despite being an open standard, the design and development tools for AUTOSAR are proprietary, e.g., Arctic Studio [Arc17]. Since AUTOSAR (meta-)models are large, the analysis and development of AUTOSAR based components is difficult without the use of proper tools [Gra17]. However, such tools are often expensive for small-scale third-parties (including academia). More importantly, the available tools have less or no potential for extensions and innovations by third-parties. These tools are designed solely for application deployment and not for new developments as the extension capabilities are industry secrets.

There are two major drawbacks of this *closed* nature of AUTOSAR tools. Firstly, new algorithms which may lead to improved system performance cannot be integrated in the available tools by third-parties. Secondly, the hardware modules and platforms which can be used for system development are tied to specific set of proprietary tools. Therefore, the support for new hardware modules/platforms can only be provided by

the tool manufacturers.

Recently, a number of attempts were made to provide extensions in OSEK/VDX compliant RTOSes. For instance, Apuzzo et al. [ABB16] added support for adaptive variable rate (AVR) task model in tools from ERIKA Enterprise [ERI17]. These OSEK/VDX implementations have strong component binding, low scalability and lack modularity. The component reuse in such implementations is either difficult or impossible. Similar to other OSEK/VDX implementations, AUTOSAR itself was subjected to extensions, e.g., Ni et al. [NKA14], Frühwirth et al. [FSS15]. These extensions focused on a very specific part of the AUTOSAR standard and imposed significant impact on the existing AUTOSAR architecture. Moreover, these extensions, although, required extensive tool support (for instance, new module support and code generators), these authors provided very minimal or no tool support.

In this chapter, we present an open-source cross-platform toolchain for the extension of AUTOSAR (meta-)models, code-generators and run-time environment. The presented toolchain, termed as AUTO-XTEND, can be used to provide support for new modules and real-time algorithms. Along with other hardware platforms, AUTO-XTEND also provides support for a low-cost System-On-Module (SOM) to give an easy-to-acquire evaluation platform for academic researchers. Moreover, we propose an extension of the AUTOSAR meta-model which allows AUTO-XTEND to directly connect with third-party toolchains. In order to exhibit the capabilities of the toolchain, we present two use cases: (i) Inverted pendulum control system using Local Interconnect Network (LIN) bus interface and (ii) Software-based Time-Triggered Ethernet (TTEthernet) end-system.

The toolchain AUTO-XTEND consists of an open-source proprietary OSEK/VDX core[1] and two GUI tools: *AUTOSAR App Studio* for creating software component models and *AUTOSAR Design Studio* for system configuration and automatic code generation. The GUI tools in AUTO-XTEND utilize AUTOSAR meta-models form ArcCore AB [Arc17]. These meta-models make AUTO-XTEND models compatible with the industrial tools, in turn, allowing model portability. Moreover, the tool *AUTOSAR Design Studio* hosts a framework for code generation of hardware (i.e., Micro-Controller Abstraction Layer aka MCAL) and software modules. This enables easy extensions in the capabilities of the toolchain by third-parties. With this work, we intend to bridge the gap between automotive industry and third-parties.

The reminder of this chapter is organized as follows; In Section 6.1, we present the related work and their limitations. Section 6.2 provides an overview of the AUTOSAR architecture, its design methodology and its toolchains as provided by the AUTOSAR tool vendors. In Section 6.3, we present an overview of AUTO-XTEND and its features. In Section 6.4, we present an extension of the AUTOSAR meta-model which enables it to connect to the third-party toolchains. Finally, Section 6.5 provides a brief description of the two case studies developed to exhibit the capabilities of AUTO-XTEND.

[1] ArcCore 12.0 [Arc17]- AUTOSAR 4.2.2, classic platform, licensed under GNU GPL.

6.1 Related Work

Various open-source OSEK/VDX implementations are available today, e.g., ERIKA Enterprise [ERI17], Toppers ATK2 [TOP17][2] and Trampoline [Tra17][3]. Similarly, the commercial products like Elektrobit tresos OsekCore [Ele17b] and Vector's osCAN [Vec17c] provide OSEK/VDX kernels and their respective tools. Common problems with these (open and commercial) OSEK/VDX implementations include strong component coupling, inflexibility and low scalability compared to AUTOSAR. Moreover, the extension capabilities of the commercial tools are not open, making them difficult to extend by third-parties.

Commercial AUTOSAR tools and libraries (e.g., Arctic Studio [Arc17], Elektrobit tresos AutoCore [Ele17a], Artop [AUT17c], etc.) overcome the problems of above mentioned OSEK/VDX implementations, but these tools have no support for extensions and innovations. They are solely designed for application deployment and the extension capabilities (if any) are industry secrets. Moreover, these tools are expensive and are tied to specific hardware platforms, i.e., a third-party cannot add support for a new hardware module/platform.

There exist a few open-source AUTOSAR tools and libraries. For instance, Parai [Par18] developed a collection of AUTOSAR COM modules based on an open-source AUTOSAR core from ArcCore [Arc17] to provide software simulation support on Windows and Linux. He also developed an open-source tool called OpenSAR [Par17] in order to configure Base SoftWare (BSW) modules. These solutions lack necessary documentation and utilize old AUTOSAR standard 3.1. Moreover, his solutions do not support Run-Time Environment (RTE) code generation, which is necessary for code reuse, modularity and scalability. Although these tools support simulation of AUTOSAR communication modules on a variety of OSEK/VDX implementations, their applicability on automotive hardware platforms is doubtful.

A number of researchers also provided extensions to the AUTOSAR meta-model and its architecture either by compromising the architectural integrity (e.g., [AKN+14], [NKA14]) or by ignoring the AUTOSAR design methodology (e.g., [FSS15]). In order to keep brevity of the chapter, in the following we will mention only the major contributions and extensions in AUTOSAR meta-model/architecture.

In 2014, Axelsson et al. [AKN+14][4] ported the ArcCore [Arc17] AUTOSAR core to Raspberry Pi [RPi] and later ported Java virtual machine [NKA14] on ArcCore to enable support for dynamic plug-in components. The core provided by Axelsson et al. is open-source and freely available. However, they only provide a port of the AUTOSAR Core. Therefore, the application designer has to manually write the code for the runtime environment and the modules (i.e., the glue-code to connect different interfaces, manage events, etc.). Moreover, this port of AUTOSAR Core can only be used with Raspberry Pi.

In 2015, Frühwirth et al. [FSS15] presented the architecture of a TTEthernet software-

[2]Not modular, no tool support, documentation only in Japanese.
[3]Not modular, no tool support.
[4]Extension of ArcCore 8.0, mostly spaghetti code.

based end-system for AUTOSAR. To achieve synchronization over Ethernet, they created a new AUTOSAR Base SoftWare (BSW) module called *TtePl* (short for 'TTEthernet Protocol Layer'). However, they did not define toolchain integration strategy (aka model-driven design methodology) or the source of required parameters.

In this chapter, we address the problem of the extension of AUTOSAR meta-models and tools. We bridge the gap between automotive industry and third-parties (including academia) by presenting an open-source cross-platform toolchain for AUTOSAR, which we call AUTO-XTEND. The toolchain AUTO-XTEND is licensed under GNU GPL, which makes it free to use and modify. The AUTOSAR core, provided as part of AUTO-XTEND, supports Raspberry Pi along with other hardware platforms. In order to demonstrate the capabilities of the presented toolchain, we extend the AUTOSAR meta-model such that proprietary modules and their specialized third-party toolchains can be connected with AUTO-XTEND. In order to provide the proof-of-concept, we present two use cases including consolidation of AUTO-XTEND and TTEthernet toolchain.

6.2 Background

In this section, we first introduce the AUTOSAR architecture in Section 6.2.1. Section 6.2.2 provides an overview of the AUTOSAR design methodology, while Section 6.2.3 summarizes the functionality of the main tools provided as part of the AUTOSAR toolchain by the AUTOSAR tool vendors. To ease the process of understanding, a list of abbreviations is provided at the start of this dissertation.

6.2.1 Introduction to AUTOSAR

AUTOSAR is an open automotive standard for model-driven design of an application based on an RTOS. It describes the architecture of standard modules with well defined interfaces which can be deployed based on the application requirements. AUTOSAR defines a layered software architecture as shown in Figure 6.1. The SoftWare Component (SWC) layer defines the application and service SWCs with their interfaces. The SWC interfaces are used to interact with other SWCs and with modules in the lower layers. The Run-Time Environment (RTE) layer manages different events and platform resources. The Base SoftWare (BSW) layer provides modules for different system services, e.g., scheduling services, communication services, etc. The lowest software layer, i.e., the MicroController Abstraction Layer (MCAL), abstracts the hardware details and presents a generic interface of the hardware modules to the upper layers.

In AUTOSAR, an application is modeled as a composition of interconnected SWCs. A SWC implements complete or partial functionality of the application and is independent of the communication mechanisms by which it interacts with other SWCs or with the underlying hardware. This design approach allows application portability and provides deterministic behaviour across multiple platforms.

The so called Virtual Function Bus (VFB) concept allows a strict separation between the applications and the infrastructure. It abstracts the interfaces between the individual

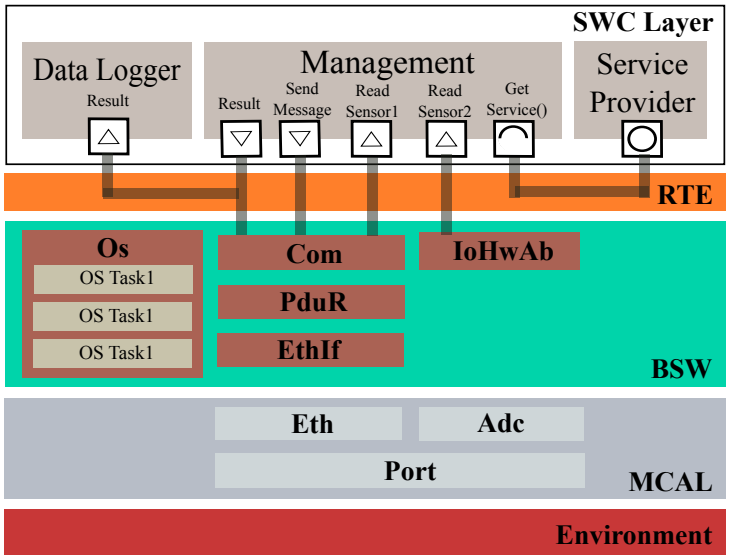


Figure 6.1: *AUTOSAR layered software architecture [AUT17b]*

software components and the BSW modules. The VFB for a specific Electronic Control Unit (ECU) is concretely implemented in the RTE layer. The RTE layer serves as the glue-code and is generated based on the ECU configuration (further discussed in the next section).

The BSW layer consists of different modules set up based on the requirements of the ECU. For example, if a SWC uses a communication service, only then the communication modules are instantiated. The BSW modules include operating system services, communication services, diagnostic services, memory management services and complex device drivers.

6.2.2 AUTOSAR Design Methodology

In order to successfully generate an executable binary, the AUTOSAR standard defines a model-based design methodology which employs Extensible Markup Language (XML) files. A summary of the AUTOSAR design methodology is shown in Figure 6.2, where crosshatched and striped thick arrows identify design process, thin arrows identify the process input/output, while the boxes represent the files. In the figure, crosshatched thick arrows represent a process accomplished with the help of the AUTOSAR configuration tool (discussed in the next section) and the stripped thick arrow represents a platform dependent binary generation process.

The system development starts with the creation of the system configuration input file by the system designer. This file provides information about the software components (including their ports and interfaces), ECU resources (including sensors and actuators) and system constraints (including mapping and timing constraints). The process `Configure System` maps the SWCs from the system configuration input file to ECUs, based on the resource and timing constraints. In turn, a system configuration

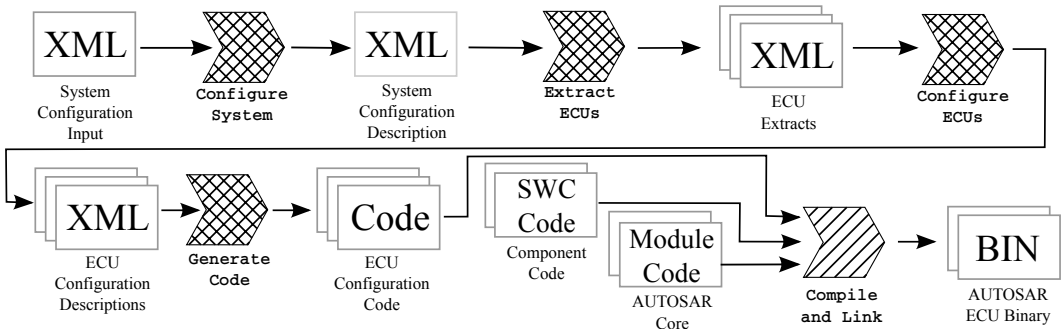


Figure 6.2: *AUTOSAR design methodology [AUT16b]*

description file is generated. From this file, the process `Extract ECUs` generates ECU specific information for each ECU in the system. Each of the ECU specific information file is termed as an ECU extract. For each ECU extract, the process `Configure ECU` adds all necessary information for ECU implementation including BSW modules and their configurations, assignment of runnable entities to tasks, etc. As a result, an ECU configuration description file is generated for each ECU. This file stores all information that is local to a specific ECU and is used by the process `Generate Code` to automatically generate ECU configuration code, including module configurations and the run-time environment.

The application developer is responsible for writing code for the SWCs defined in the system configuration input file or the ECU extract. During the SWC development process, the component APIs are created and the SWC functionality is implemented. Once the system integrator acquires the code for the SWCs, the generated ECU configuration code and the AUTOSAR core (discussed in the next section), the platform dependent `Compile and Link` process generates an executable binary file, which can then be loaded on the ECU hardware.

6.2.3 AUTOSAR Toolchain

A typical AUTOSAR toolchain, as provided by the AUTOSAR tool vendors, consists of two essential tools; the *AUTOSAR Configuration Tool* and the *AUTOSAR Core*. The *AUTOSAR Configuration Tool* provides an easy way to create system configuration, to generate ECU extracts and to generate ECU configuration code. In other words, the *AUTOSAR Configuration Tool* is responsible for executing the crosshatched processes shown in Figure 6.2. In contrast to the *AUTOSAR Configuration Tool*, the *AUTOSAR Core* provides the embedded operating system support for an ECU. It consists of different configurable software modules for performing different system services, e.g., scheduling, hardware abstraction, communication, debugging, etc. The *AUTOSAR Core* makes cross-compilation of code possible and provides support for a variety of hardware platforms. Note that these two tools are provided as part of a single toolchain, since the code generation process (shown in Figure 6.2) used by the *AUTOSAR Configuration Tool* must generate ECU configuration code compatible to the *AUTOSAR Core*.

6.3 AUTO-XTEND

In this section, we present an open-source toolchain[5] for AUTOSAR, which we call AUTO-XTEND. The AUTO-XTEND toolchain provides an embedded software development environment based on the AUTOSAR standard 4.2. The toolchain provides support at different stages in the ECU development process, e.g., application development and embedded platform development. The toolchain is designed with special focus on AUTOSAR design methodology, i.e., the application timing and scheduling analysis is not the target of the toolchain.

A summary of the differences between a typical AUTOSAR toolchain and AUTO-XTEND is shown in Figure 6.3, where boxes represent tools (or part of a tool). In the figure, the gray boxes highlight our contribution, the bullets inside tools highlight the development processes which can be achieved with the tools, while the bullets inside the core highlight its components. As shown in the figure, in AUTO-XTEND, the work of the *AUTOSAR Configuration Tool* (see Section 6.2.3) is broken down into two applications, namely *AUTOSAR App Studio* and *AUTOSAR Design Studio*. In the following, we provide details of the three components which fall under the umbrella of AUTO-XTEND.

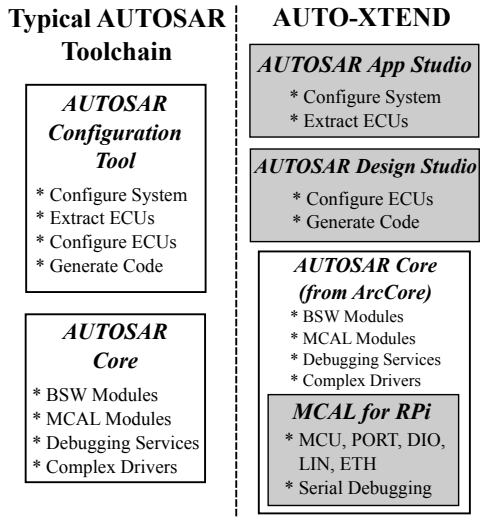


Figure 6.3: *Differences between a typical AUTOSAR toolchain and AUTO-XTEND*

6.3.1 AUTOSAR App Studio

The tool *AUTOSAR App Studio* is used to define automotive system configuration, including application SWCs, their behaviours (with runnables) and system/timing constraints. Using this tool, one can generate an extract for an ECU in the system as an AUTOSAR XML (ARXML) file. The tool *AUTOSAR App Studio* generates an ECU

[5]Source link: `https://bitbucket.org/syed_ali_abbas_jaffari/autoxtend`

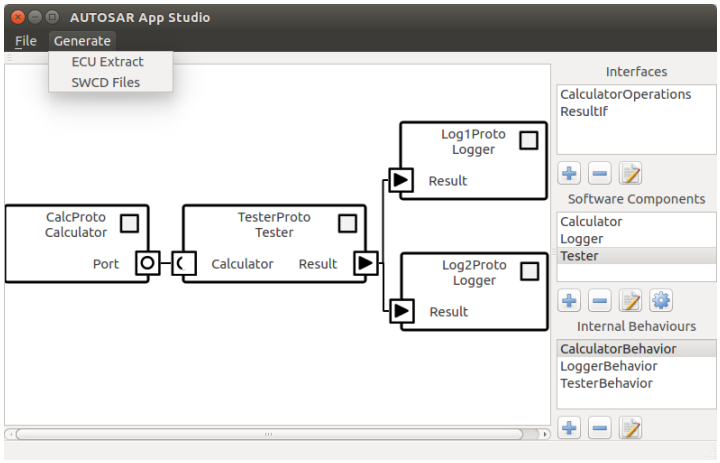


Figure 6.4: *Main design window of AUTOSAR App Studio*

extract compliant to the AUTOSAR standard 4.2, which enables direct import of this file in any commercial AUTOSAR tool, in turn, supporting portability and modularity. It is a GUI tool which complements system development using ARText [AUT17a] (i.e., AUTOSAR textual language modeling framework). If required, the tool can directly import and export SoftWare Component Description (SWCD) files written in ARText languages.

The tool *AUTOSAR App Studio* is written with Qt5/C++ to enable cross-platform development of the tool. Figure 6.4 provides a view of the main designer window. The main design area provides a visual representation of the SWCs, their ports and their connections. New software components can be instantiated and dragged/dropped on the design area to make connections. The section in the right column shows settings for the available interfaces, SWC types and their internal behaviours. When the application designer is satisfied with the design, the extract for an ECU can be generated. It is important to mention here that only one composition can be generated when using *AUTOSAR App Studio*. In future, we intend to extend the functionality by multiple compositions and service components.

6.3.2 AUTOSAR Design Studio

The tool *AUTOSAR Design Studio*[6] enables the user to configure ECU (i.e., define module configurations, map runnables to tasks, configure run-time environment) and automatically generate ECU configuration code (see Section 6.2.2). The tool provides command-line operation enabling bulk code generation to diversify evaluation test-cases.

Similar to the *AUTOSAR App Studio*, this tool is written with Qt5/C++. This design choice enables reduced execution times compared to the equivalent available Eclipse based tools[7]. Moreover, the tool makes use of the meta-models from *Arctic Core*

[6]Demo @ `https://www.youtube.com/watch?v=C6ldHiPJhN4`.

[7]Note that there exist only Eclipse [EF18b] based commercial tools, since the Artop consor-

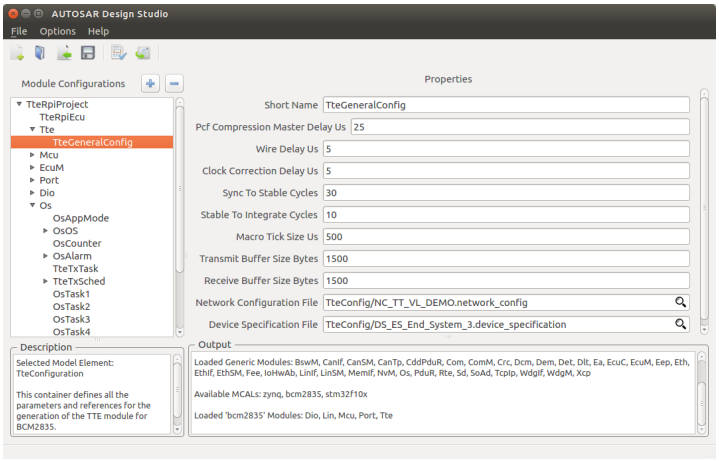


Figure 6.5: *Main design window of AUTOSAR Design Studio*

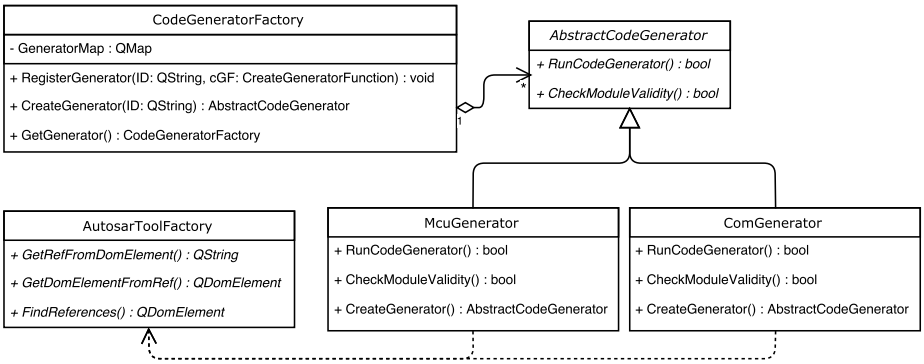


Figure 6.6: *UML class diagram for code generation classes*

to provide the same flexibility and functionality as by any equivalent commercial Eclipse based tool[7].

Figure 6.5 shows a view of the main configuration window. The object tree on the left provides a module configuration explorer. The "Properties" area enables modification of the configuration parameters and references, while the "Output" window provides a description of the errors/warnings/updates.

AUTOSAR Design Studio provides code generators for several generic modules, including Operating System (Os), Communication Manager (ComM), Protocol Data-Unit Router (PduR) and Run-Time Environment (RTE). The tool also contains several code generators for MCAL modules (for Raspberry Pi and STM32 micro-controller family). By hosting a code generation framework, the tool also enables extensions to allow creation of new code generators for AUTOSAR modules. Note that this extension capability is not available in any commercial AUTOSAR tool.

tium [AUT17c] (from where the tool vendors acquire plugins) decided to use Eclipse modeling framework [EF18a].

The code generation framework is designed such that the developer is capable of using any available code generation method, e.g., writing code on a file stream, model to text frameworks like Xpand [EF13], etc. An abridged version of the UML class diagram for the code generation classes is shown in Figure 6.6. The `CodeGeneratorFactory` provides methods for accessing and registering a code generator. Once registered with the factory, a code generator is called automatically depending on the ECU configuration description file (see Figure 6.2). The abstract class `AbstractCodeGenerator` defines the interface of the code generator for an AUTOSAR module. From this interface, concrete code generators can be implemented. For instance in Figure 6.6, the `AbstractCodeGenerator` class is shown as a generalization of MCU and COM module code generators. Furthermore, the static class `AutosarToolFactory` provides helpers for exploring AUTOSAR (meta-)models using Document Object Model (DOM) APIs.

6.3.3 AUTOSAR Core

The *AUTOSAR Core* in AUTO-XTEND is an extension of the open-source *Arctic Core*[8], which is distributed under the GNU General Public License by ArcCore AB [Arc17]. The *Arctic Core* provides an OSEK/VDX compliant RTOS and supports several platform architectures including ARM, PPC and Renesas. It provides MCALs for more than 25 boards and ECUs. Board specific debugging services (i.e., T32, UDE, WinIdea and SCI) along with CLib port are also available as part of the *Arctic Core*.

To support a low-cost hardware platform, we ported the *Arctic Core* to Raspberry Pi[9] and provided a number of MCAL modules, including Digital I/O (DIO) and LIN modules. Moreover, a new board specific debugging service based on UART is added for Raspberry Pi. In the next sections, we will use the term *AUTOSAR Core* and *Arctic Core* interchangeably to refer to the core capable of being executed on Raspberry Pi.

6.4 3rd-Party Toolchain Integration

In this section, we describe how the AUTOSAR meta-model is modified to enable third-party toolchain integration. Moreover, we present how AUTOSAR and TTEthernet toolchains can be intertwined together based on the proposed meta-model modification.

6.4.1 Extension of AUTOSAR Meta-Model

AUTOSAR meta-model defines a large number of model elements in terms of an XML Schema Definition (XSD). These model elements are used to define the models in Extensible Markup Language (XML) files. An example of the XSD group defined by the AUTOSAR meta-model is shown in Figure 6.7. This group defines several parameter types (including integer and string) and several reference types. For instance, an

[8] Version 12.0 [Arc17], AUTOSAR 4.2.2, classic platform.
[9] Tested on Raspberry Pi 2 Model B and Zero.

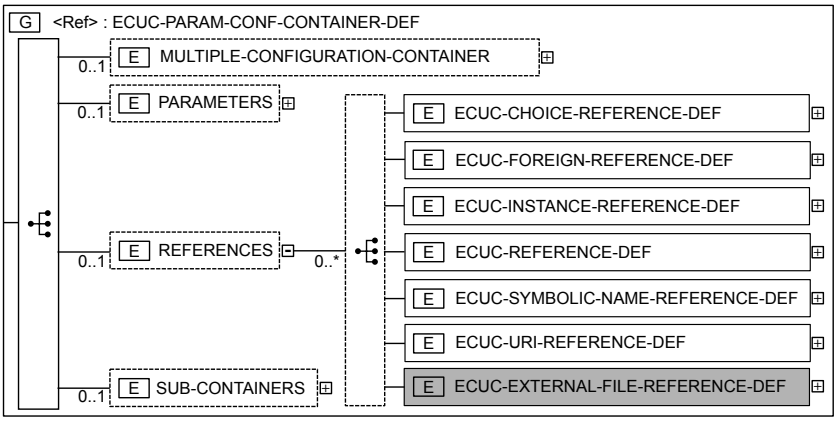


Figure 6.7: *Overview of an XSD group in AUTOSAR 4.2 schema [AUT17b]*

ECUC-FOREIGN-REFERENCE-DEF element specifies a reference to an XML entity described in another AUTOSAR XML (e.g., ECU extract).

In order to support third-party toolchain integration, we add a new type of reference element which we call ECUC-EXTERNAL-FILE-REFERENCE-DEF (highlighted with gray color in Figure 6.7). This reference type specifies a reference to a third-party model file. The referenced model file can then be used by the code generator (in AUTOSAR Design Studio) to extract the information from a third-party model and (when required) invoke the third-party tool. This enables automating the artifact generation process, not only for the AUTOSAR modules, but also for the external modules.

6.4.2 Integrating Toolchains: AUTOSAR & TTEthernet

In Section 1.1.5, we provided a brief introduction to the TTEthernet design methodology, its model files and its toolchain. In this section, we show how the design methodologies of AUTOSAR and TTEthernet can be merged using the meta-model extension defined in the previous section.

An abridged version of the extended design methodology to intertwine AUTOSAR and TTEthernet toolchains is shown in Figure 6.8, where thick arrows represent processes, thin arrows indicate process input/output and the boxes represent the files (or collection of files). In the figure, the crosshatched processes are executed by *AUTOSAR App Studio*, the stripped processes are executed using *AUTOSAR Design Studio*, while the checkered processes are executed by the TTEthernet toolchain. The figure shows that, from the AUTOSAR system configuration description files and the ECU extracts, the tool *AUTOSAR App Studio* generates a network description (ND) file for the TTEthernet network. The tool *TTPlan* processes this file and generates a number of network configuration description files (see Section 1.1.5 for more details). After the generation of network description files, the tool *AUTOSAR Design Studio* is used to generate code for the software-based TTEthernet end-system for AUTOSAR and *TTPlan/TTBuild* to generate and build the HEX files for other devices (i.e., non-AUTOSAR end-systems and TTEthernet switches).

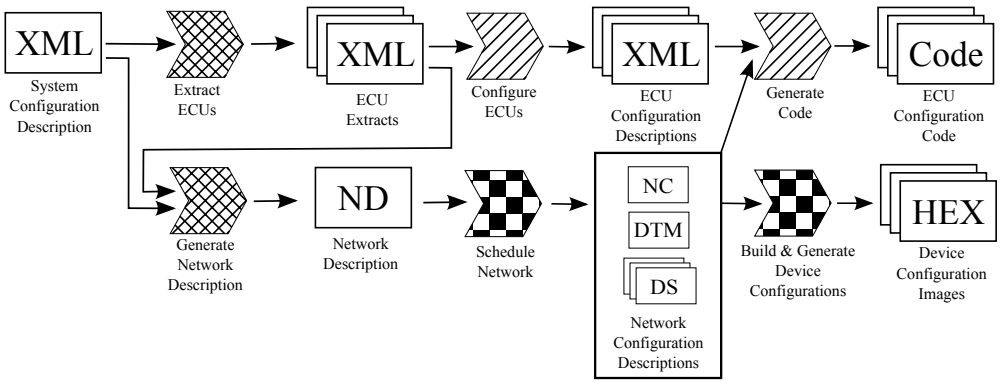


Figure 6.8: *AUTOSAR Design Methodology with TTEthernet*

Note that the AUTOSAR meta-model extension (mentioned in the previous section) and a code generator in *AUTOSAR Design Studio* are not sufficient to complete this development process. Either a TTEthernet ND file needs to be manually created using the TTEthernet toolchain, or an ND file generator needs to be written in *AUTOSAR App Studio*. With adequate understanding of TTEthernet and AUTOSAR meta-models, creation of such a model generator is possible, given the open-source nature of AUTO-XTEND.

6.5 Case Studies

As quantitative analysis of the efficiency of a design tool is impractical, usually some case studies are performed in order to exhibit the behaviour of the tool in different scenarios, e.g., [BDM+17]. In this section, we define two case studies based on real-time applications: inverted pendulum control via LIN bus and software-based TTEthernet end-system. Using these case studies, we evaluate the AUTO-XTEND toolchain based on several factors, e.g., functional correctness, extension capability, etc. By means of these case studies, our goal is to test the following hypotheses:

Hypothesis 6.1. The operations performed by the toolchain generate valid artifacts, e.g., the generated code is compatible with the *AUTOSAR Core*.

Hypothesis 6.2. The extension capabilities of the toolchain support the application design, i.e., no dirty workarounds in the toolchain or the core are required to accomplish the task at hand and the toolchain does not hinder the development of the application.

Hypothesis 6.3. The system behaves in correspondence with the application design, i.e., the observation is the same as the expectation.

Since there exist two major contributions in this chapter (i.e., the toolchain and the meta-model extension), two use cases are necessary. The target of the inverted pendulum use case, presented in Section 6.5.1, is to exhibit the correctness of the generated code. While the TTEthernet use case, presented in Section 6.5.2, exhibits the AUTO-XTEND toolchain integration capabilities by working in collaboration with the TTEthernet toolchain (overview in Section 1.1.5).

6.5.1 Case Study 1: Inverted Pendulum Control via LIN Bus

An inverted pendulum is a mechanical system where a body of mass is balanced over a pivot point on a controlled cart. Due to the inherent instability of the system, a closed loop control system is needed to keep the mass balanced.

The goal of this case study is to make use of as many AUTOSAR modules as possible. This gives an estimate if the code generators create correct code based on the project configuration. Therefore, we decided to use an inverted pendulum system (IPS) [YFR] and constructed a distributed system consisting of two processing nodes, i.e., Raspberry Pi connected via LIN bus to an STM32F103 microcontroller board [YFR]. For the case study, we chose LIN bus since it uses a simple protocol and does not (strictly) require special hardware modules on the platform. Due to low baud-rates of LIN bus (i.e., 20000bps maximum), the complete control loop - for keeping the pendulum balanced - is implemented on the STM32 board, while the set-point generator - for defining where the cart should move - is implemented on Raspberry Pi (further in the next section). Since Arctic Core 12.0 (see Section 6.3.3) does not yet support binary generation for STM32F103 microcontroller[10], we wrote bare-metal code for STM32F103, while we used AUTOSAR on Raspberry Pi (further in the next section).

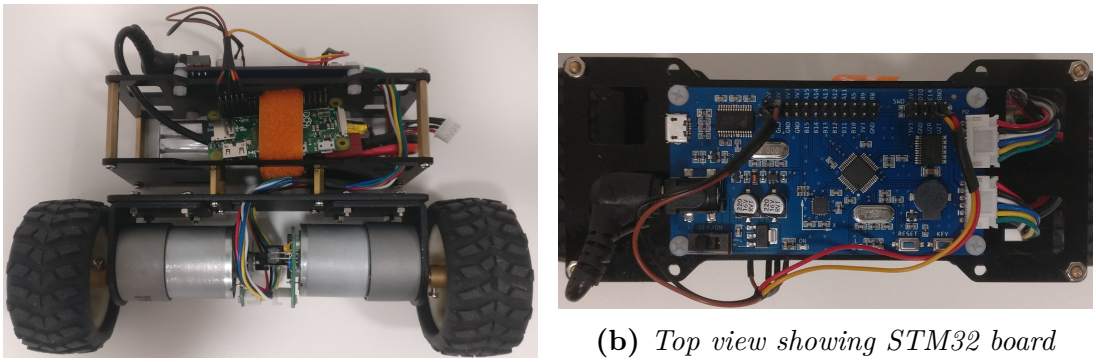
Note that the complete inverted pendulum system could be implemented on a single board, i.e., either on Raspberry Pi (with gyro and motor driver) or on STM32 board. However, the goal of the use case is to show the code generation process of the tool, instead of the demonstrated application. The code generation process of the communication modules would demonstrate that the tools perform all the required operations, while the validity and (functional) correctness of the generated code will be shown by the motion of the IPS.

In the following, we first provide a brief overview of the final inverted pendulum system (IPS). Then, we describe how the software modules on the IPS interact with each other. It is followed by a description of the observations from the perspective of the case study. Finally, from these observations, we infer the capabilities of AUTOXTEND and correctness of the code generators.

Inverted Pendulum System (IPS)

Figure 6.9 shows the inverted pendulum system (IPS) employed for the use case containing RPi Zero and STM32 boards. The STM32 board contains an STM32F103 [STM17] microcontroller, an MPU6050 [IS13] gyroscope and a TB6612 [Tos07] motor driver. The erection angle of the IPS is measured using the MPU6050 gyroscope, while the IPS is controlled via geared DC motors using PWM signals generated via TB6612 driver. Due to different PWM signals to the left and right motors, the IPS is controlled in a differential-drive configuration (e.g., SegWay). In order to reduce the hardware, we did not use the physical layer of LIN bus while designing the inverted pendulum system. However, the IPS design can be altered to use the physical layer of LIN bus using

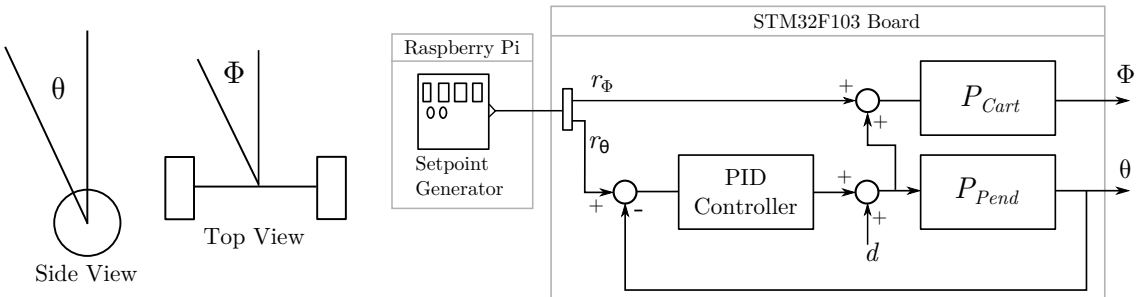
[10]In future, we intend to extend the functionality of Arctic Core 12.0 to enable binary generation for STM32F103 microcontroller.



(a) *Side view showing Raspberry Pi Zero*

(b) *Top view showing STM32 board*

Figure 6.9: *Hardware design of the inverted pendulum system*



(a) *Angular representation*

(b) *Control System*

Figure 6.10: *Control System Design*

LIN transceivers (e.g., SN65HVDA100 [TI15], TJA1020 [Phi02], LIN board [Mik17], NCV7425 [Sem11]) without any modification in the software.

Control System Design

Figure 6.10 shows the control system design which we used for the inverted pendulum case study. In Figure 6.10, θ and Φ represent the erection and rotation angle (shown in Figure 6.10a), respectively. In Figure 6.10b, the symbols r_θ and r_Φ represent the reference points for erection and rotation, respectively. Moreover, the blocks P_{Cart} and P_{Pend} represent the transfer function blocks for the cart rotation and erection, respectively. Furthermore, the disturbance d represent the external factors affecting the IPS erection, e.g., wind gusts, external shaking and vibrations, surface variations.

The control system in Figure 6.10b shows that the set-points for defining the erection and rotation angle are sent to the STM32 board from RPi. After reception of the signal, the STM32 board decouples/demuxes the signal into erection and rotation signals and diverts to their respective controller/plant. Note that the control system shown in Figure 6.10b does not control the position of the inverted pendulum on the floor. Due to the gravitational pull, the IPS moves forward/backward when the erection angle

reference r_θ is positive/negative. Moreover, the rotation angle Φ is controlled in open-loop fashion by modifying duty cycle of PWMs for left/right motors.

Software Modules and their Interactions

In Figure 6.11, we show how the software modules on the platforms interact with each other. As mentioned earlier, the figure shows two different ECUs, Raspberry Pi (RPi) implementing a set-point generator on AUTOSAR and an STM32 board [YFR] implementing a bare-metal PID controller. The two ECUs communicate via the LIN bus, where RPi is defined as the LIN master while the STM32 board is defined as the LIN slave. Due to low baud-rates of the LIN bus (i.e., 20000bps maximum), only set-points to control the heading of the IPS are sent by RPi. Moreover, in order to avoid the design complexity, the RTE layer on RPi is not used and the SWC-Layer is employed as a virtual layer (i.e., no sub-division of the application in runnables).

From RPi, the commands for heading and rotation are sent by the only SWC (mapped to the *Generator* OS task in BSW) through the communication module (i.e., Com) in a single Protocol Data Unit (PDU). The PDU is routed to the LIN interface modules (LinIf) via PDU Router (PduR), and then sent to the MCAL LIN module of RPi. The data from RPi is sent to the STM32 board at predetermined points in time as defined by the LIN schedule. In the LIN schedule, three *schedule expiry points* (see AUTOSAR OS specification [AUT16a]) are defined: for standing still (0s), moving forward (3s) and turning right (4s). After all these entries are executed, the LIN schedule restarts from the first entry.

On the STM32 board, we used a PID controller to control the IPS erection angle (i.e., not position on the floor) with respect to the ground. The PID controller was tuned such that the IPS balances. We wrote a bare-metal driver[11] for LIN bus for the STM32 board which enables the reception of the heading and rotation commands from

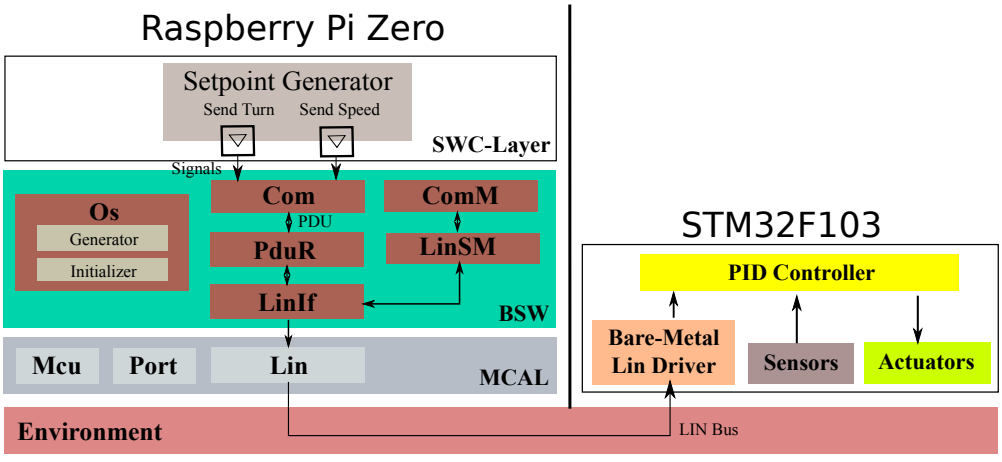


Figure 6.11: *System architecture for inverted pendulum control using LIN bus*

[11]Similar to the one by Macchina - `https://github.com/macchina/LIN`.

RPi. Upon receiving the heading command the STM32 board forwards it to the PID controller to move the cart forward or backward, while the rotation command changes the duty cycle of the PWMs for motors to rotate the IPS clockwise or counter-clockwise (see Φ in Figure 6.10a).

Observation

In order to get an executable binary for RPi, code generators for MCAL modules (e.g., Mcu, Port and Lin) were written using the code generation framework presented in Section 6.3.2. These generators create APIs for accessing hardware modules on RPi based on the project configuration. Moreover, code generators for generic modules (e.g., Os, Com, ComM, PduR, etc.) were utilized to generate OS structures (including schedule for LIN bus), PDU identifiers and their routes. The generated code and the developed application code were then compile along with the AUTOSAR Core in order to generate an executable binary for RPi. When RPi is powered on, the generic modules make use of APIs defined by the MCAL modules to control the board in accordance with the application code.

The code generation process and the IPS balancing demonstration is shown in the video[12]. The video demonstrates that *AUTOSAR Design Studio* allows easy configuration of the modules and enables quick code generation for the configured modules. Moreover, it shows that the IPS balances and moves as anticipated considering the defined LIN schedule in RPi. This indicates that the IPS heading and rotation commands were sent by RPi at correct times and received by the STM32 board where they were fed to the IPS controllers.

From the perspective of AUTO-XTEND, in the following we present the observations for each hypothesis defined in Section 6.5:

- Hypothesis 6.1: The generation of software artifacts (i.e., OS configurations, communication ports, routing paths, LIN schedule) lead to code compatible with the *AUTOSAR Core*. The code compiled without errors and executed without problems on RPi.

- Hypothesis 6.2: No dirty workarounds were required in the toolchain. The code generators supported the artifact generation.

- Hypothesis 6.3: The inverted pendulum system was expected to move forward for 1 second and turn for 1 second (as defined by the LIN schedule). This behaviour was indeed observed.

In summary, we observed that the AUTOSAR core successfully executed on RPi and the tool AUTOSAR Design Studio generated correct code. Therefore, we can assert that the hypotheses are confirmed.

[12]Code generation and balancing demonstration - `https://www.youtube.com/watch?v=C6ldHiPJhN4`.

6.5.2 Case Study 2: Software-Based TTEthernet End-System

In this use case, we provide a software-based implementation of a TTEthernet end-system (ES) using AUTOSAR[13]. The use case is designed such that it exhibits the AUTO-XTEND extension capability. In this section, we present only a brief description of the TTEthernet use case, while the implementation and evaluation details for the use case can be found in Appendix A. In order to keep a reminder of the terminology, a list of abbreviations is provided at the start of this dissertation.

Terminology

TTEthernet networks consist of TTEthernet End-Systems (ES) or processing nodes connected via TTEthernet switches through physical links. Similar to other Ethernet based networks, TTEthernet utilizes Ethernet frames to transport the data payload. Although TTEthernet supports three network traffic classes, in this section, we will only focus on the time-triggered (TT) traffic. In TTEthernet, the network bandwidth for a TT frame is reserved using the concept of a Virtual Link (VL). A VL is defined by a period, message size, a source ES, a set of destination ESs and a route through the physical links.

In TTEthernet, the TT frame transmission takes place based on an offline generated TT scheduling table. For each VL, the scheduling table defines scheduling windows, i.e., time interval during which the TT frame belonging to a VL uses a network physical link. Based on these scheduling windows, a TTEthernet device (i.e., an ES or switch) dispatches to-be-transmitted TT frames and filters the received TT frames, i.e., discards if the frame is not received during the defined time duration. In order to make sure that the TT messages are sent and received at correct point in time as defined by the TT scheduling tables, all the end-systems and the switches in the network must have the same notion of time. This is achieved using a fault-tolerant clock synchronization protocol.

The clock synchronization protocol requires periodic exchange of Protocol Control Frames (PCF) at well defined points in time. These PCFs contain timing information which is utilized by the receiving device to estimate the global time. During the synchronization process, PCFs are transmitted from ESs to the switches. The switches then compute the fault-tolerant average of the relative arrival times of received PCFs to generate global time. This time or clock value is then sent as a PCF to all ESs, where each ES modifies its local clock in accordance with the global clock.

Hardware Design

In order to create software-based TTEthernet end-system, the Raspberry Pi (RPi) board was used as one of the TTEthernet end-systems. However, RPi does not generate interrupt based on Ethernet frame reception, which is necessary for clock synchronization of the network. Therefore, we used ENC624J600 [MTI09] Ethernet controller, which allows raw frame mode which is necessary for the reception of PCF. The ENC624J600

[13]Note that the existing commercial AUTOSAR tools do not yet support TTEthernet networks.

controller provides transfer speeds of 14Mbps using Serial Peripheral Interface (SPI). Although the SPI speed of 14Mbps presents a throughput bottleneck (compared to the typical Ethernet link throughput), it serves the purpose of providing a proof-of-concept for this use case.

Software Design

In order to support TT traffic in AUTOSAR, the BSW Ethernet Interface (EthIf) module and the MCAL Ethernet (Eth) module need to disjoint frame buffering from frame dispatching/reception [FSS15]. For TT frame transmission, this enables frame dispatch on the physical link based on the transmission windows defined in the TT scheduling table (as opposed to immediately after buffering). While for the TT frame reception, this modification allows frame filtering based on the TT reception windows.

In RPi based TTEthernet ES, an OSEK Interrupt Service Routine (ISR) *TteRxIsr* performs the synchronization service upon reception of a PCF. The ISR *TteRxIsr* directly affects the counter values responsible for keeping a notion of time (i.e., implicit synchronization, see AUTOSAR OS specification [AUT16a]). If the received frame is a TT frame (other than the PCF), *TteRxIsr* performs the TT frame filtering based on the scheduling table *TteRxSched*. If the frame is valid, it forwards the received frame to the TT Reception (RX) buffer. The TT traffic is transmitted by a highest priority OSEK task *TteTxTask*. The task *TteTxTask* checks the TT transmission (TX) buffer. If there exists a TT frame to be sent at the current activation time, it dispatches the frame to the network link. The task *TteTxTask* is activated at predefined points in time defined by the AUTOSAR scheduling table *TteTxSched*. The scheduling table *TteTxSched* is a mirror image of the TT scheduling table generated by TTPlan (see Section 1.1.5), however, without the TT frame reception windows.

In order to generate necessary artifacts (i.e., *TteRxIsr*, *TteRxSched*, etc.), a code generator is implemented in *AUTOSAR Design Studio*. The code generator takes the static parameters from the user, configured OS module and TTEthernet model files (defined in Section 1.1.5) as input and generates TTEthernet ES configuration files for AUTOSAR.

Experimental Setup

Figure 6.12 shows the network setup for the evaluation of this use case. All physical links shown in the figure are 100Mbps links. The devices ES0 and ES1 are used for providing synchronized network time and to transmit and receive network traffic. The end-systems ES0 and ES1 are standard TTEthernet ESs [TTT18b] without AUTOSAR, while ES2 is TTEthernet software-based ES with AUTOSAR.

In order to evaluate the TT communication capability, we generated 100 task-sets with varying parameters. The parameters for the synthetic task-sets were chosen in accordance with the industrial applications defined by Craciunas et al. [CO16]. The task and the virtual link (VL) periods were chosen randomly from the list of 10ms, 30ms and 100ms. The frame sizes were chosen from 100 bytes to 500 bytes and the TT

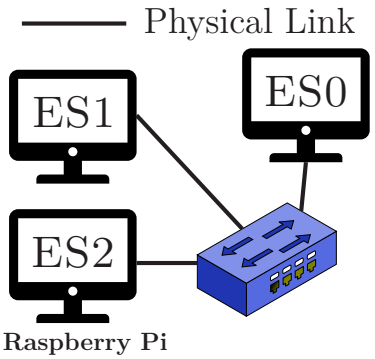


Figure 6.12: *Experimental Setup for TTEthernet End-system with AUTOSAR*

window sizes were fixed to 124μs while the PCF window sizes were fixed to 24μs. For each task-set, ES2 served 16 periodic OSEK tasks, 8 of which had a dedicated VL for sending or receiving a TT frame through the physical link. As this use case targets TT communication, these communicating tasks only sent and received TT frames before termination, while the free tasks performed busy-waiting for a bounded random time and terminated. Moreover, each generated task-set ran for at most one hour.

The OSEK application tasks were scheduled offline corresponding to their communication schedule, in order to reduce transmission latency. The TT schedule was generated using TTPlan (see Section 1.1.5) and the ECU configuration code for ES2 was generated using *AUTOSAR Design Studio*. The TTEthernet device configuration (DC) files and their corresponding HEX files for the non-AUTOSAR TTEthernet devices were generated and uploaded to them using TTLoad or Eclipse based TTEthernet plugin (see Section 1.1.5).

Observations

A detailed analysis of the software-based TTEthernet end-system is presented in Appendix A. Here, we summarize the major observations. The presented system design lead to the synchronization precision of $\pm$8μs, frame reception jitter of $\pm$7μs and successful frame transfer when the frame is dispatched 8μs before the end of transmission window. Moreover, the effective Ethernet throughput was observed to be $\approx$7.5Mbps due to the SPI bottleneck.

From the perspective of AUTO-XTEND, in the following we summarize the observations for each hypothesis defined in Section 6.5:

- Hypothesis 6.1: The generation of end-system local artifacts (i.e., tasks and schedules) lead to valid code, compatible with the *AUTOSAR Core*. The merging process of the toolchains lead to valid binaries for the end-systems and the switches.

- Hypothesis 6.2: No dirty workarounds were required in the toolchain. The code generation framework supported the artifact generation. A modification to decouple frame buffering from frame dispatching was required in EthIf and Eth

modules. However, this modification is inevitable [FSS15] in order to support TT traffic transmission. No dirty workaround was required in the *AUTOSAR Core*.

- Hypothesis 6.3: The expectation from RPi included long lasting synchronization, successful transmission and low jitter. All of these qualities were observed.

It was also observed that AUTO-XTEND greatly decreased the programming efforts (due to code generation of tasks and scheduling tables for each different task-set) and provided a possibility to perform thorough evaluation of TTEthernet end-system for AUTOSAR. Therefore, we can assert that the hypotheses are confirmed.

Chapter 7

Real-Time Scheduling & Analysis Framework

The complexity of modern real-time systems is increasing day by day, due to ever increasing demand for functionality and scarcity of system resources. The development of modern multi-processors and variety of network protocols and architectures have made such a leap in functionality possible. Albeit, the software development process needs improvements in order to support rapid-prototyping for ever changing system designs. Since real-time systems are often required to handle complex constraints, rapid-development of offline scheduling and analysis tools is a challenge. One way to meet this challenge is to design a modular yet simple scheduling and analysis framework.

There exist a variety of open-source and commercial scheduling and analysis tools, e.g., RTaW-Timing [RTa17], Xoncrete [BMR+10]. The open-source tools for scheduling and analysis have strong component binding which prevents modularity and code reusability. On the contrary, the commercial tools are closed-source and offer no extension capability to build a new offline scheduler or analysis algorithm by third-parties. Therefore, a new scheduler or analysis technique either requires tool creation from scratch or entails substantial changes in existing open-source tools. To alleviate this problem, a flexible framework is consequently required in order to support code reuse and rapid-development. However, there exist no such framework (to the best of our knowledge) for offline scheduling and analysis of real-time systems.

A scheduling and analysis framework is expected to have a number of qualities to ascertain its acceptance in (industrial and academic) research and development. These qualities include modularity, concern separation, speed, ease-of-use and powerful debugging tools. However, existing tools lack some (if not all) of these qualities, which hinders their deployment in industrial and academic settings.

In this chapter, we present a modular cross-platform framework for scheduling and analysis of real-time systems; we call this framework GAIA (from Greek mythology) as it enables creation of variety of schedulers. The tool GAIA enables rapid-development[1] of new scheduling and analysis algorithms for the next generation of real-time systems. It provides scalable search algorithms and analysis tools for offline scheduling of distributed, multi- and/or many-core systems. It requires less resources compared to available tools (e.g., combinatorial optimizations [CO16], Python or Java based

[1]By *rapid-development*, we mean rapid compared to other C++ tools, since a number of scheduling and analysis components are already available. See Section 7.4 for more details.

tools [BFG02]) and allows verification and validation of timely constraints for large complex systems.

The tool GAIA enables creation of different types of offline schedulers and analyzers. It allows code reuse, modularity, flexibility, concern-separation and (schedule synthesis and analysis) speed. Moreover, this framework provides features like scheduler specific compilation & cross-compilation to enable portability and release generation to boost distribution process.

The rest of the chapter is organized as follows; In Section 7.1, we present a brief overview of the state-of-the-art scheduling and analysis tools and present their limitations. In Section 7.2, we provide a description of the software architecture and available components in GAIA. In Section 7.3, we discuss different benchmarks and task-set generators, which can be used for evaluation of scheduling and analysis algorithms. Finally in Section 7.4, we discuss software design issues and provide recommendations for designing a scheduling and/or analysis algorithm.

7.1 Related Work

There exists a large number of tools for simulation and analysis of real-time systems, most of which come from industry and are specifically designed for a certain domain, e.g., automotive. In this section, we will provide an overview of the available tools and analyze their offline scheduling capabilities. In the following, we will classify available tools in two categories; (i) freely available, and (ii) commercial tools.

Among freely available tools, Noulard et al. [NPC12] developed an open-source programming and analysis tool for periodic tasks defined in formal languages UPPAAL and PRELUDE. They called their tool SchedMCore. Although their tool supports offline scheduling, albeit only for simple priority-based schedulers. Moreover, the tool SchedMCore is written in C (instead of C++), which complicates its design, and provides almost no scheduling or analysis component for future developments.

Similarly, Brocal et al. [BMR+10] presented a web-based offline scheduling tool for partitioned multi-core platforms called Xoncrete. The tool Xoncrete generates output schedules compatible to the XtratuM hypervisor [CRM+09]. It can allocate and schedule tasks on a multi-core platform, however it employs a fixed set of constraints and does not support scheduler extension by third-parties. Moreover, Xoncrete cannot be used to schedule network and does not scale well for large applications.

Similarly, Chéramy et al. [CHD14] developed an open-source tool, called SimSo, for simulation of event-triggered schedulers for multi-processors. The tool supports many event-triggered schedulers and bin-packing heuristics for task allocation on a multi-processor platform. Moreover, the tool also provides task-set generators to acquire periodic workloads but lack support for offline, hierarchical and network scheduling.

Recently, Becker et al. [BDM+16] presented a GUI based tool for the analysis of multi-rate effect chains on an automotive platform. They called their tool MECHAniSer. The tool is capable of analyzing effect chain latencies and data age. However, the tool does not allow offline scheduling and is not open-source. Therefore, any extensions by third-parties in the task model or the analysis algorithms is not possible.

A large variety of commercial tools also exist for scheduling and analysis of real-time systems. For instance, CANoe [Vec17a] by Vector is a comprehensive tool for development, testing and analysis of automotive networks. Moreover, RTaW-Timing [RTa17] by RTaW enables cross-domain timing verification for distributed real-time systems. A common problem with these commercial tools is lack of extension support for third-parties and limited (or no) offline scheduling support for real-time systems.

In this work, we present a tool called GAIA, which resembles in architecture to that of SchedMCore [NPC12], however it differs in its capabilities. There exist a number of features which are unique to GAIA, e.g., template-based thread-safe library for common search algorithms, analysis components, solution verification tools, etc. Although GAIA does not focus on multi-processors or event-triggered schedulers, it can be used for such purposes. For instance, priority assignment can be done similar to SchedMCore [NPC12] and latency analysis can be done similar to MECHAniSer [BDM+16]. GAIA focuses on offline scheduling of real-time systems, and therefore provides many tools for generation and validation of schedules.

7.2 GAIA: Framework for Scheduling & Analysis

In this section, we present a framework for offline scheduling and analysis of real-time systems. We call this framework/tool GAIA. The tool GAIA enables development, testing and evaluation of new real-time algorithms for modern distributed, multi- and many-core systems. GAIA is a command-line tool, written in C++, which provides a wide variety of development components (e.g., search algorithms, parsers, file generators, logger, visualizers, etc.) to facilitate rapid-development of scheduling and analysis algorithms. It enables code reuse, and provides cross-compilation and library compilation (for generation of the release version) using CMake [HM03] build manager.

In order to give an idea about the target and focus of GAIA, we provide a brief description of what this tool is *not* good for. GAIA is not a worst-case execution time analysis tool (e.g., RTMemController [LAG14]) and not a control task simulation tool (e.g., TrueTime [CHL+03]). Moreover, GAIA is not designed for task profiling or scheduler benchmarking (e.g., rt-muse [Mag15]). Furthermore, it is not a formal verification tool (e.g., SchedMCore [NPC12]) and cannot be used as a benchmark or task-set generation tool (further details in Section 7.3).

In the following, we will use the term *development component* to refer to a class or function which provides a feature and can be used at the discretion of the algorithm developer. Moreover, we will use the term *module* to refer to a scheduling/analysis class.

In this dissertation, we used the framework GAIA to design and evaluate (i) pruning based scheduler SAS for time-triggered systems presented in Chapter 2, (ii) pruning based TTEthernet scheduler SAS-TTE presented in Chapter 3, (iii) offline analysis and online simulation of Job-Shifting algorithm presented in Chapter 4, and (iv) mode-change delay analysis and scheduling for DREAMS project presented in Chapter 5 (where the tool `MCOSF` represented a release version of the DREAMS *module*).

In the following, we provide an overview of the software architecture of GAIA in Section 7.2.1. Furthermore in Section 7.2.2, we present a description of development

components available in the proposed offline scheduling and analysis framework.

7.2.1 Software Architecture

A typical implementation of a scheduling/analysis module in GAIA is shown in Figure 7.1 using UML notation. The figure shows the relationships between different classes and their types. In the figure, the main `Application` class contains other classes and controls their creation and destruction. The control flow is started by the `Application` class with validation of command-line parameters, followed by parsing and creation of an empty `SystemDescription` class (which will hold all the global data structures required by the scheduling/analysis module). The main class then creates an instance of `InputParser` which generates global data structures and stores them in the `SystemDescription` class. Once all the necessary information is available in the `SystemDescription` class, the `Scheduler/Analyzer` class is instantiated which goes through three phases of execution; pre-processing, algorithm-activation and post-processing. The pre-processing phase is used for preparing data structures local to the class and for model transformation, e.g., periodic task unrolling, deadline shortening [CSB90], etc. The algorithm-activation phase performs scheduling/analysis and the post-processing phase generates (optimized) output artifacts from the result generated by the algorithm-activation phase, e.g., generation of schedule from the search path, decreasing number of preemptions, etc. When the data-structure for the final result is created, the main class instantiates the `OutputGenerator` class which writes the required data structures to the disk (for instance, using `XMLGenerator` in Figure 7.1).

During scheduling/analysis or output generation phases, a number of development components can be utilized to visualize the state of the algorithm (e.g., `GraphicGenerator` in Figure 7.1) or to make use of common scheduling and analysis algorithms (e.g., `SimulatedAnnealing` and `ProcessorDemandTest` in Figure 7.1).

It is worthwhile to mention that the performance of an algorithm does not only depend on the underlying methodology (e.g., `SimulatedAnnealing`) but also on the data structures defined in the `SystemDescription` class. For these data structures, there exist two extremes; size optimized and speed optimized. In size optimized implementation of `SystemDescription`, parameters of an imaginary child class are estimated/calculated when required using an existing parent object, e.g., storing uni-directional references to necessary objects, calculating absolute deadline of a job from period of a task object, etc.

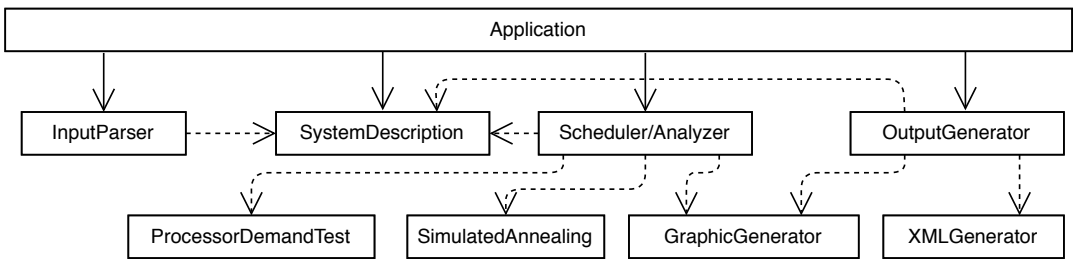


Figure 7.1: *UML class diagram for a typical scheduling/analysis module in GAIA*

As a result, less number of object are required in memory but a derived parameter needs to be calculated every time it is used. On the contrary, in speed optimized implementation of `SystemDescription`, parameters are estimated/calculated in the pre-processing phase and stored as a data structure for later retrieval, e.g., storing bi-directional references between all related objects, creating an object for each job of a task object, etc. This topic will further be discussed in Section 7.4.

7.2.2 Development Components

In the following, we provide a description of the development components available in GAIA. These development components enable rapid-development of scheduling or analysis modules and can be used at the discretion of the algorithm developer.

XML Parsing and Generation

Over the last few decades, Extensible Markup Language (XML) has proven to be an efficient way to store structured data. Nowadays, a large variety of tools are available in market which can be utilized to support parsing and generation of files based on XML specification. In order to parse an XML file, two flavors of parsers are available; Document Object Model (DOM) and Simple API for XML (SAX). The XML parsers based on DOM read the complete XML file into buffers and create object-trees corresponding to the structure of the XML input. The object-tree is provided to the user for exploration using simple APIs. However, when the size of input XML exceeds a certain limit, DOM parsers start to lag since they read complete XML file at once and store every element and attribute of XML file in the memory.

On the contrary to the DOM parsers, SAX parsers read XML file line-by-line and store only a part of it in memory. As a result, the memory requirement of SAX parsers is very low. However, SAX parsers require event-triggered parsing which is difficult to use and require more lines of code compared to DOM parsers.

A common shortcoming of these DOM and SAX parsers is that the user needs to write code in order to convert the input XML object-tree into the required data structure. This process can be automated by creating an XML schema (XSD) file and using XML data binding tools (e.g., CodeSynthesis XSD [CS18], Eclipse modeling framework (EMF) [EF18a]) to generate code for parsers and data-structures. Using XML data binding tools, the user can generate data structures which can directly be mapped to the input/output XML elements. The direct mapping approach for data binding is fast, but it requires all objects in the tool to work on the data structure as defined by the XSD file. Moreover by using a direct mapping approach, algorithm designer becomes dependent on the internal implementation of XSD data types defined by the data binding tool. If required, one can write converters to and from the generated data structures and their types to the required implementation with size or speed optimized variants (see Section 7.2.1). However, this results in a development overhead. Another downside of direct mapping approach is that when a modification in the model is required, the scheduler designer has to go through the XSD file and generate new/modified classes and modify the code respectively. Although some of these processes can be automated

by modern XML data binding tools (e.g., EMF), however this process requires adequate experience.

In the light of aforementioned issues, XML data binding capability is not made an integral part of GAIA, however such tools can be utilized at the discretion of the developer. The framework design is kept simple and modular such that the extension of a module to use such data binding tools is trivial. Since the goal of GAIA is to provide easy-to-use framework, only light-weight but powerful DOM parsers/generators called PugiXML [Kap17] and FXML are integrated in GAIA. PugiXML is a C++ XML processing library, which provides DOM APIs with rich traversal/modification capabilities. It is a fast XML parser [Kap17] which provides features like XPath implementation and search queries for large XML files. On the contrary, FXML provides simple API to parse and generate XML elements/attributes. The focus of PugiXML is speed and functionality, while FXML focuses on simplicity.

Logging

Logging is an important part of writing correct and efficient code and is utilized for several purposes, e.g., identifying state of the program, disclosure of variables' values, revealing potential error points, etc. A good logging library can help identify bugs and problems early in the development phase. Often, logging requires different severity levels, multiple output streams and conditional logging support. However for simple implementations, pre-processor directives are often employed.

A simple `Logger` class with pre-processor directive support is provided as part of GAIA. The `Logger` class provides several features, e.g., 8 logging severity levels (i.e., ERROR, WARNING, INFO and 5 debugging levels), compile-time inclusion or exclusion support, output stream redirection, etc. These features enable rapid-prototyping and provide good textual logging support for development of an algorithm.

Graph Generation

Every now and then, search and analysis problems are mapped on graphs (e.g., Scheduler and task graph by Fohler [Foh94], MECHAniSer [BDM+16], RTaW-Timing [RTa17]). When the sizes of these graphs become large, it is often convenient to visualize them for troubleshooting or analysis purposes. In GAIA, the task of `GraphGenerator` class is to enable graph creation using simple APIs. This class uses graph description language DOT to create and store vector images of graphs. Note that generating huge graphs consumes a large amount of memory, slows down the algorithm (because of frequent I/O operations) and takes a long time to render. Although the generated graph files can be rendered using conventional viewers, these viewers fail when rendering large graph files (i.e., typically graphs with more than 1000 nodes). In such situations, a viewer called ZGRViewer [Pie05] is preferred.

Graphic Generation

Time and again, it is required to visualize the state of the algorithm or to generate a graphical trace for the generated schedule. Since GAIA does not provide a Graphical User Interface (GUI), a class called `GraphicGenerator` is provided which can be used for visualization using simple APIs. The `GraphicGenerator` class uses Scalable Vector Graphics (SVG) specification for generation of graphic files. When required, the generated SVG files can be converted to any required format using conventional graphic editing tools.

Search Algorithms

Search algorithms have been used throughout the research history of real-time systems to deal with complex scheduling problems, e.g., Simulated Annealing by Tindell et al. [TBW92], Tabu Search by Tamas-Selicean et al. [TSPS12]. Implementations of such algorithms enable code reuse and decrease programming efforts. Although any open-source search algorithm library can be utilized off-the-shelf with GAIA (e.g., Tabu-search by Maischberger [Mai17], k Nearest Neighbour (kNN) by Fu and Cai [FC16]), the tool provides two search algorithm implementations; Parallel IDA* (PIDA*) [RKR87] and Simulated Annealing (SA) [KGV83].

In GAIA, PIDA* algorithm is implemented using multi-threaded template-based approach. The PIDA* class creates defined number of threads each executing on a separate core. For multi-core host platforms with core-local L1 and L2 caches, executing single *worker* thread [RKR87] per core enables linear speed-up of search [SF14]. Moreover, the PIDA* implementation in GAIA allows *spawning* of a search-tree node to reduce exploration run-time (see Chapter 2 for details). Similar to PIDA*, GAIA also provides a class for Simulated Annealing (SA) which is implemented using C++ templates. This implementation of SA can only be used on a single host core, however the implementation can be extended for multiple cores.

Scheduling and Analysis Aid

In real-time systems literature, a number of algorithms are used time and again to generate scheduling tables (e.g., deadline-shortening), or to analyze constraint satisfaction (e.g., processor demand test). These algorithms have often helped in achieving optimal results for certain scheduling problems, e.g., task-set transformation defined by Chetto et al. [CSB90] used by Jonsson [Jon99]. In order to facilitate algorithm development, a number of such algorithms are provided as part of GAIA. For instance, processor demand test (PDT) by Baruah et al. [BRH90] and task-set transformation by Chetto et al. [CSB90].

In order to check if the final solution generated by a developed module in GAIA indeed satisfies all constraints, a schedule verification tool is also integrated with GAIA which can be used for analysis like estimation of end-to-end latencies, jitter, data age, etc. Note that the implementation of this verification tool strongly depends on how the data structures are defined and what constraints are handled by the designed algo-

rithm. Therefore, the user must define how system constraints can be validated. The architecture of the verification tool is similar to MECHAniSer [BDM+16] and RTaW-Timing [RTa17]. However, the verification tool does not provide an interactive GUI. Therefore, post-generation schedule modification, similar to MECHAniSer, can be done either in the post-processing phase of the scheduler/analyzer or specified as constraint for scheduling.

In order to further increase the development speed, pruning techniques (defined in Chapter 2) can be used to design a task scheduler. These techniques can be extended to create schedulers for complex networks (similar to Chapter 3). Moreover, for scheduling strictly periodic tasks/messages, the phase generation algorithm defined by Marouf and Sorel [MS10] is also implemented using Interval-Tree data structures in GAIA.

7.3 Real-time Benchmarks and Task-set Generators

In order to evaluate the performance of a scheduling algorithm, a number of benchmarks or task-sets are required. Although this topic is orthogonal to the focus of the chapter, but it is an important part of algorithm design. Therefore, in the following, we enumerate benchmarks and tools which can be used to evaluate GAIA modules.

From the perspective of real-time systems, a handful of case studies and benchmarks are available in literature. For instance, Shan et al. [SWF+14] provided Chinese Lunar Rover Control Software, Pagetti et al. [PSG+14] defined flight controller case study which they called Research Open-Source Avionics and Control Engineering (ROSACE). Note that the creation of such benchmarks is a tedious task since it might be subjected to a bias or may represent over-simplified designs [Hei10]. As a result, a number of real-time communities are working on providing realistic unbiased case studies. For instance, Euromicro Technical Committee on Real-Time Systems (ECRTS) provides an industrial challenge every year in Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) by working in close collaboration with industrial practitioners. These case studies provide good starting point for evaluation of scheduling and analysis algorithms.

Considering that the available benchmarks and case studies use only a limited set of real-time constraints, another option for evaluation of the developed scheduling algorithm is to use task-set generators. In 1998, a tool called Task Graph For Free (TGFF) was developed by Dick et al. [DRW98] which is capable of generating task graphs (with or without communication) and hardware specifications. The tool TGFF provides two graph generation modes but allows only random distribution of parameters. Almost a decade later, Cordeiro et al. [CMP+10] developed a graph generator called GGen. GGen provides various graph generation methods and parameter distributions. However, GGen does not generate information related to hardware platforms, which is required for categorical evaluation of real-time scheduling algorithms. Similarly, Neukirchner et al. [NSE11] developed an open-source tool called System Model For Free (SMFF). The tool SMFF provides more flexibility in parameter generation compared to TGFF, however its development is discontinued and does not provide sufficient documentation. Note that none of these tools provide any guarantee if the generated task-set

will indeed be schedulable, since the scheduling problem (dealt by these tools) is complex. To check schedulability of a generated task-set, third-party toolchains can be utilized. For instance, Neukirchner et al. [NSE11] used SymTA/S [HHR+04] for timing analysis of task-sets generated with SMFF.

7.4 Discussion

Throughout the research history of real-time systems, a large number of design choices have been made for the development of an algorithm. For instance, Venugopalan and Sinnen [VS15] chose ILP solver and proposed an improved ILP formulation for optimal allocation of tasks, Pozo et al. [PRNSH16] proposed different constructions of SMT constraints leading to different performance, Craciunas et al. [CO16] used different SMT solvers (Yices, Z3, Gurobi) leading to varying schedulability within the same amount of time. Based on the results of these design choices, it is trivial to see that the performance of an algorithm depends on both the design choice *and* the way this design choice is implemented.

Similarly, a widely known problem in computer science is the selection of data structures for providing solution to a specific problem. Nearly all programming languages provide several linear (e.g., vector, list) and non-linear (e.g., trees and hashes) containers and data structures. Although all of these containers can be used to solve a specific problem, the complexity and scalability of an algorithm is strongly dependent on the utilized data structures. For instance, Andersson and Ekelin [AE07] used balanced-trees to reduce the complexity of the processor demand test from $O(n)$ to $O(\log n)$.

In algorithm development, a trade-off exists between algorithm development time and algorithm run-time. When an algorithm needs to be developed in a small amount of time, general purpose tools for mathematical optimizations (e.g., constraints solvers, integer programming) are viable solutions since these tools can be used to *test* the approach [Mon16] by writing equations or constraints. However as demonstrated by many studies (e.g., [CO16, PRNHS17]), such general purpose tools require exponentially large amount of time for large systems. Furthermore, in order to implement problem specific optimizations with such tools, two approaches are used in literature; (i) new constraint/equation formulation, and (ii) divide-and-conquer. The former approach leads to increase in number of constraints/variables (in turn, leading to substantial increase in run-time), while the latter approach leads to solution composition problems (e.g., [PRNSH16, PRNHS17]). Therefore for large systems or for final products, we do not recommend mathematical optimizations. Instead, we recommend search algorithms since they can be used to define how the artifacts can be generated and they have been demonstrated ([Jon99], also see chapters 2 and 3) to scale well for large systems.

Other than the tool related factors (enumerated above), choice of programming language and source code implementation also have a large impact on algorithm development and run-time [BFG02]. For instance, virtual machine based languages (e.g., Java, .NET) are known to be slower than languages running code natively on the platform (e.g., C, C++) [BFG02]. However, virtual machine based languages require less lines of code compared to natively executing languages. Since the GAIA framework is designed

using C++, the algorithm development time is not as little as with virtual machine based languages or scripting languages (e.g., Python) but the run-time is indeed smaller.

Chapter 8

Conclusion

Future applications are expected to be complex with immense networks interconnecting multiple factories and/or geographical locations [McK13]. These applications pose several challenges in achieving the goal of collaborative system operation along with efficiency. The avionics and automotive industries, for example, have already moved towards mixed network types to exploit the best features of each network type where required and to provide flexibility where beneficial. Moreover, the application environment of such large mixed networks poses stringent constraints on the design, e.g., timeliness, multiple modes of operation, etc.

In order to make progress towards future real-time systems and to cope with existing issues, in this dissertation, we presented solutions to many problems and highlighted the advantages and disadvantages of our proposed approaches. With experimental evaluation and case studies, we demonstrated that our solutions are efficient and provide portability and scalability. We assert that solutions presented in this dissertation can be used to develop future cyber-physical systems with mixed network types and complex constraints.

In the following, we provide a summary of our contributions in Section 8.1. In Section 8.2, we present intended future work for the extension of the contributions presented in this dissertation. Finally, we provide disclaimer statements for the proprietary and open-source modules mentioned in this dissertation in Section 8.3.

8.1 Overview of Conclusions

This dissertation started with an introduction to the terms and their meanings in the context of real-time systems. Later in this dissertation, we contributed by proposing solutions to six problems in the real-time domain. The following sections summarize our contributions.

8.1.1 Time-Triggered Scheduling and Search-Tree Pruning

In Chapter 2, we introduced a novel search-tree pruning technique based on job response-times for scheduling large distributed time-triggered (TT) systems. Our response-time based pruning technique efficiently reduces the apparent search-space for a problem

instance and helps find feasible TT scheduling tables for complex task-sets. In order to implement our pruning technique, we proposed a modular offline scheduler design based on the PIDA* search algorithm. To reduce memory requirements and speed-up the scheduling process without compromising the search-tree determinism, we presented a search-tree node spawning technique. We evaluated the schedulability and run-times of our scheduler with varying system sizes and configurations. The results showed that our scheduler is capable of efficiently exploring the search-space and that the response-time based pruning technique is very effective in finding TT schedules for complex hard real-time tasks.

8.1.2 TTEthernet Scheduler

In Chapter 3, we provided an extension of our offline scheduler for large distributed networks based on TTEthernet. To generate TT scheduling tables for each mode of operation, we employed an efficient implementation of phase generation technique defined by Marouf and Sorel [MS10] in order to schedule TT virtual links. We also defined how to reduce number of backtracks when the network is subjected to strictly periodic constraints. Based on our single mode schedule generation technique, we defined how TT windows can be stacked to generate scheduling tables for multiple modes of operation. These generated schedules can be utilized to provide multi-mode support in TTEthernet based devices (which do not explicitly support it). With thorough experimental evaluation, we demonstrate that our scheduler is highly scalable (compared to the state-of-the-art) and quickly finds a feasible TT schedule for both tasks and network.

8.1.3 Job-Shifting Algorithm

In Chapter 4, we presented an algorithm for online admission of non-preemptive aperiodic tasks in hierarchical systems. We provided the basis of a necessary test for the admission of non-preemptive aperiodic tasks and presented an approach to provide guarantees similar to the bandwidth reservation technique. This approach enables use of existing verification and validation techniques to guarantee aperiodic tasks offline. Moreover, we provided a detailed discussion on the implementation of our algorithm and highlighted different factors which affect its performance. We evaluated the efficiency of our scheduler on synthetic but practical task-sets and demonstrated that our approach efficiently utilizes available system resources. Furthermore, we also implemented our algorithm on an ARM based platform. The experiments manifested that the overheads incurred by our scheduler are very small and practical for safety critical applications.

8.1.4 Mode-Changes and Fault-Tolerance

In Chapter 5, we presented a novel approach for defining safe time points for mode-change execution in partitioned industrial mixed-criticality environment. Our approach reduces the mode-change delays by generating mode-change blackout intervals, for each mode-change between *service* modes (see Section 1.1.2). Furthermore, we created intermediate scheduling tables, termed as transition mode schedules [Foh94], between

different service modes (where possible). Based on the generated blackout intervals and transition mode schedules, our technique guarantees task-set feasibility during a mode-change and calculates the worst-case mode-change delays. In order to demonstrate the performance of our mode-change scheduling approach, we presented a case study for a safety critical application in the DREAMS project, which demonstrated that the developed approach leads to significant improvements in mode-change delays.

8.1.5 AUTOSAR Toolchain

In Chapter 6, we presented AUTO-XTEND which is an open-source toolchain for AUTOSAR, consisting of (i) an open-source *AUTOSAR Core* which supports Raspberry Pi and other hardware platforms, (ii) a GUI based tool *AUTOSAR App Studio* for application design and (iii) a GUI based tool *AUTOSAR Design Studio* for ECU configuration and code generation. We also introduced an easy-to-use and -extend code generation framework for AUTOSAR modules. Furthermore, we proposed an extension of the AUTOSAR meta-model which allows AUTO-XTEND to integrate with third-party toolchains. To evaluate the effectiveness and the integration capabilities of AUTO-XTEND, we presented two case studies: an inverted pendulum controlled via LIN bus interface and an AUTOSAR TTEthernet software-based end-system. Our evaluation demonstrated that AUTO-XTEND provides extension capabilities and flexibility for rapid development of modern automotive software.

8.1.6 Real-Time Scheduling and Analysis Framework

In Chapter 7, we presented a modular cross-platform framework for design and development of real-time scheduling and analysis tools; we call this framework GAIA. GAIA enables rapid-prototyping of scheduling and analysis algorithms for modern and the next generation of real-time systems. GAIA provides a rich set of development components (e.g., parsers, file generators, logger, visualizers, etc.) to facilitate rapid-prototyping and -development of scheduling and analysis algorithms. It also provides scalable search algorithms and analysis tools for offline scheduling of distributed, multi- and/or many-core systems. GAIA efficiently utilizes platform resources and allows verification and validation of timely constraints for large complex systems. The framework GAIA was used in this dissertation to implement and evaluate contributions of chapters 2, 3, 4 and 5.

8.2 Future Work

In future, we plan to exploit the knowledge of symmetries in the system to generate solutions for the next generation of mixed networks (e.g., CAN + LIN in automobiles) with varying requirements and goals. We also plan to extend our symmetry based scheduler to incorporate other constraints introduced by the application environment, e.g., mixed-criticality, multi-rate tasks, etc. Moreover, we intend to improve our scheduler for TTEthernet by utilizing a look-ahead technique similar to the one proposed by Coelho [Coe17].

For our mode-change analysis technique, we intend to evaluate the mode-change execution overheads δ for XtratuM implementation on common platforms and include these estimates in determining a tighter bound for worst-case mode-change delays. We also intend to handle the mode-change execution for DREAMS platforms in the XtratuM hypervisor. This enables mode-change execution at the end of any partition slot (instead of only defined partition slots).

In future, we also intend to improve AUTO-XTEND by supporting AUTOSAR service components and multiple compositions in *AUTOSAR App Studio*. We also plan to embellish *AUTOSAR Core* with more MCAL modules (e.g., I2C) for Raspberry Pi and enable support for more processors and development boards. Moreover, we intend to broaden the coverage of *AUTOSAR Design Studio* by writing more code generators for generic modules (e.g., CanIf, Nvm) and MCAL modules for other platforms.

In the context of our scheduling and analysis framework GAIA, we intend to extend the `ProcessorDemandTest` component to cover all cases of task-set constraints, e.g., constrained or arbitrary deadlines, arbitrary phases. Moreover, we are considering expansion of the framework by provide support for other common search algorithms, e.g., Branch & Bound [LD60], graph coloring, clique finding. For the existing search algorithms, we intend to provide support for execution on multiple processing nodes in collaboration using Message Passing Interface (MPI). Furthermore, we intend to provide a library for common heuristics, e.g., bin-packing, EDF, RM.

8.3 Disclaimer

The copyrights for *Arctic Core* and its meta-model provided as part of AUTO-XTEND presented in this dissertation are reserved by ArcCore AB [Arc17], Sweden. Moreover, the copyrights for TTEthernet toolchain used in this dissertation are reserved by TTTech GmbH [TTT18a].

The AUTOSAR toolchain presented in this dissertation is not intended to replace proprietary tools and is provided AS IS without warranties. The authors of this toolchain assume no responsibility or liability for the use of the software. See GNU General Public License for more details.

Appendix A

AUTO-XTEND Case Study 2: Software-based TTEthernet End-System

In Section 1.1.5, we provided an introduction to TTEthernet, its design methodology and its toolchain. Moreover in Section 6.5.2, we presented a brief description of the 2^{nd} case study for AUTO-XTEND, exhibiting its third-party toolchain integration capability. In this appendix, we provide details of this case study including thorough evaluation of our software-based implementation of TTEthernet end-system (ES).

We start the use case description with an overview of the used terminology in Section A.1. Then, we present the details of the hardware and software design of our software-based TTEthernet end-system in sections A.2 and A.3, respectively. In Section A.4, we describe the experimental setup which we used for the analysis of our implementation. In Section A.5, we present a thorough evaluation of RPi based TTEthernet end-system. And finally in Section A.6, we provide a description of the observations from the perspective of AUTO-XTEND and compare our implementation of software-based TTEthernet end-system with the state-of-the-art. In order to keep a reminder of the defined terms, a list of abbreviations is provided at the start of this dissertation.

A.1 Background and Terminology

Time-Triggered Ethernet (TTEthernet) [TTT18b] is a switched Ethernet network which is employed in safety critical systems to support mixed-criticality traffic, i.e., Time-Triggered (TT), Rate-Constrained (RC) and Best-Effort (BE). To support TT traffic, all devices are synchronized by a fault-tolerant clock synchronization protocol, according to which Protocol Control Frames (PCF) are periodically exchanged at well defined points in time. These PCFs contain a field called transparent clock which is utilized by the receiving device to estimate the global time. The PCF transmission period is called Integration Cycle (IC), while the LCM of all periods in the network is termed as Cluster Cycle (CC). In order to achieve synchronization, an end-system (ES) can have one of two synchronization roles, Synchronization Master (SM) or Synchronization Client (SC). However, a switch can have one of three synchronization roles, SM, SC or Compression Master (CM). When the network is powered up, the devices communicate their synchronization roles with each other using PCFs. The synchronization

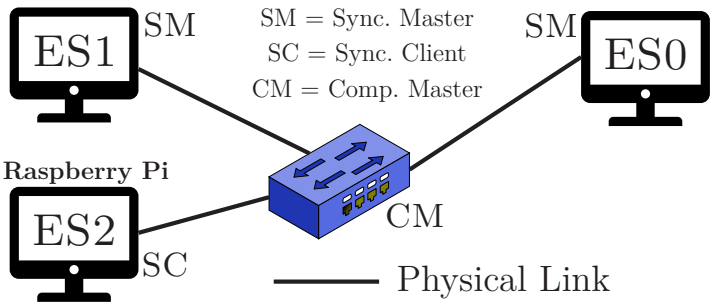


Figure A.1: *TTEthernet network setup for use case 2*

process proceeds by the transmission of PCFs from synchronization masters (SMs) to compression masters (CMs) at the start of an integration cycle (IC). It is followed by the reception of PCFs by CMs from SMs. CMs then compute the fault-tolerant average of the relative arrival times to generate a new PCF, which is then sent to all SMs and SCs to synchronize their local clocks. An example of a TTEthernet network is shown in Figure A.1.

In TTEthernet, the network bandwidth for a TT or an RC frame is reserved using the concept of a Virtual Link (VL). A VL is defined by a period (or bandwidth allocation gap), message size, a source end-system (ES), a set of destination ESs and a route through the physical links. A TT schedule defines TT windows for each VL. A TT window defines an interval in which a frame of a specific VL can be transferred through the physical link. A TT window acts as a filtering policy for the received TT traffic, while for the transmission TT traffic, the TT windows define the frame dispatch time.

Contrary to the TT traffic, RC traffic does not require synchronization but requires offline analysis methods, e.g., Network Calculus, to verify the timing behaviour and frame latency constraints. Furthermore, non-critical BE traffic is transmitted when physical link is not used by TT or RC traffic.

Since a physical link can be used to transmit different frames belonging to different traffic classes, an intelligent approach is needed to integrate/isolate these classes such that the timely properties of each class are maintained. The SAE standard [SAE11] for TTEthernet has defined two different traffic integration policies, shuffling and preemption. There exist other traffic integration policies provided by TTEthernet OEMs, for example, Timely Block. Using shuffling, the transmission of a TT frame can be postponed until the end of currently transmitting frame. With preemption, the currently transmitting frame is preempted and re-transmitted after the transmission of a TT frame. While, timely block prevents the transmission of a frame if it can interfere with a TT frame.

A.2 Hardware Design of Raspberry Pi Based End-System

Figure A.1 shows the network setup for the evaluation of this use case, where all physical links are 100Mbps links. The devices ES0 and ES1 are used for providing synchronized

network time and to transmit and receive network traffic. The end-systems ES0 and ES1 are standard TTEthernet ESs [TTT18b] without AUTOSAR (i.e., computers with TTEthernet PCI cards), while ES2 is software-based TTEthernet ES with AUTOSAR executing on top of Raspberry Pi (RPi).

In this use case, we do not use the Ethernet controller built in RPi, since in RPi 1 and 2 [1] the ARM processor is connected to the Ethernet Controller LAN9514 [SMS12] via a USB 2.0 hub. USB standard provides four transfer types; control, bulk, interrupt and isochronous. According to the standard, the control and bulk transfers types are used for best effort transfers, the interrupt transfers require polling by the host [BL17], while the isochronous transfers provide no guarantees of data delivery. Moreover, no interrupt pin exists between the Ethernet controller and the ARM core to perceive a frame reception. In order to achieve synchronization (and TT traffic handling) frame time-stamping is required, which cannot be achieved by a USB interface to the Ethernet controller.

In order to provide time-stamps to the TT frames, a separate Ethernet controller is needed to transfer frames in and out of RPi. For this purpose, we chose the ENC624J600 [MTI09] chip. ENC624J600 is a stand-alone 10/100 Ethernet controller with 24KB SRAM send/receive buffers. The controller allows raw frame mode, which is necessary for the reception of PCFs. It offers transfer speeds ranging from 14Mbps with Serial Peripheral Interface (SPI) to 160Mbps with parallel interfaces. For parallel interface, the network controller ENC624J600 requires at least 19 GPIO pins. In order to use these many GPIO pins from RPi GPIO header, the data to/from RPi needs some software transformation to conform to the connections, which might require extra efforts and incur overheads. Therefore, we chose the SPI interface with 14Mbps speed. Although the SPI speed of 14Mbps presents a throughput bottleneck (compared to a typical Ethernet link), it serves the purpose of providing a proof-of-concept for the use case. The effective Ethernet throughput is evaluated and discussed in Section A.5.

A.3 Software Design of Raspberry Pi Based End-System

TTEthernet software-based end-system designed with RPi considers only TT and BE traffic types, each with separate buffers. Moreover in this use case, we limited the synchronization role of RPi ES to synchronization client and used shuffling as the traffic integration policy, since the Ethernet controller ENC624J600 does not support frame preemption and timely block can introduce extra overheads [FSS15].

In order to support TT traffic in AUTOSAR, the BSW Ethernet Interface (EthIf) module and the MCAL Ethernet (Eth) module need to disjoin frame buffering from frame dispatching/reception [FSS15]. For TT frame transmission, this enables frame dispatch on the physical link based on the transmission windows defined in the TT scheduling table (as opposed to immediately after buffering). While for the TT frame reception, this modification allows frame filtering based on the TT reception windows.

The AUTOSAR standard 4.2 defines a number of modules which help keep the net-

[1] In RPi zero, there exist no Ethernet port. Moreover, peripheral datasheet for RPi 3 is not available.

work node synchronized based on the network global time, e.g., Synchronized Time Base Manager (StbM), Ethernet Time Synchronization (EthTSyn). Considering that the StbM and EthTSyn modules do not exist in *Arctic Core* 12.0, an OSEK Interrupt Service Routine (ISR) *TteRxIsr* performs the synchronization service in our RPi based TTEthernet ES upon reception of a PCF from the compression master (CM). For this purpose, the *TteRxIsr* directly affects the counter values responsible for keeping local time (i.e., implicit synchronization, see AUTOSAR OS specification [AUT16a]). When the modules StbM and EthTSyn are available, the implementation can easily be modified to utilize their services.

The ISR *TteRxIsr* is responsible for receiving TT (including PCF) and BE frames. If the received frame is a TT frame, *TteRxIsr* performs the TT frame filtering based on the scheduling table *TteRxSched*. If the frame is valid, *TteRxIsr* forwards the received frame to the TT Reception (RX) buffer. If the received frame is a BE frame, it is forwarded to the BE RX buffer.

The traffic transmission is controlled by an OSEK task *TteTxTask*, which has highest priority in the system. In every activation, *TteTxTask* first checks if a BE frame needs to be transmitted (from BE transmission buffer). If there exists a frame in the buffer, it is dispatched on the physical link. Then, *TteTxTask* checks the TT transmission (TX) buffer for a frame. If there exists one and the current activation time lies within the TT transmission window, the TT frame is dispatched.

Considering that the task *TteTxTask* handles both BE and TT transmission, it is activated using two approaches. To handle BE frame transmission, an application task is supposed to write the BE frame to the BE TX buffer and activate the task *TteTxTask* manually. To handle TT frame transmissions, the task *TteTxTask* is activated at predefined points in time defined by the AUTOSAR scheduling table *TteTxSched*. The scheduling table *TteTxSched* is a mirror image of the TT scheduling table generated by TTPlan (see Section 1.1.5), however, without the TT frame reception windows. In order to reduce the TT transmission latencies, the scheduling table *TteTxSched* can be embellished with application OSEK tasks.

To generate code for the TTEthernet ES, a code generator is implemented in *AUTOSAR Design Studio*. The code generator takes the static parameters from the user, configured OS module and TTEthernet model files (defined in Section 1.1.5) as input and generates TTEthernet ES configuration files for AUTOSAR. The generated files contain the scheduling tables *TteRxSched*/*TteTxSched* and code for *TteRxIsr* and *TteTxTask*.

A.4 Experimental Setup

In this section, we provide details of the experiment which we performed in order to evaluate the software-based TTEthernet end-system and the capabilities of AUTOXTEND. The following details includes (i) generation procedure of synthetic task-sets, (ii) generation of binary for the software-based TTEthernet end-system with AUTOSAR for RPi using AUTO-XTEND, and (iii) generation of HEX files for the rest of the network using TTEthernet toolchain (see Section 1.1.5).

In order to evaluate the TT communication capability, we generated 100 task-sets

with varying parameters. The parameters for the synthetic task-sets were chosen in accordance with the industrial applications defined by Craciunas et al. [CO16]. The task and the VL periods were chosen randomly from the list of 10ms, 30ms and 100ms. The frame sizes were chosen from 100 bytes to 500 bytes and the TT window sizes were fixed to 124μs while the PCF window sizes were fixed to 24μs. The integration cycle (IC) length was randomly selected from the list of 1ms, 2ms, 5ms and 10ms, while the cluster cycle (CC) length is the least common multiple of all the message periods. The TT TX/RX buffers for ES2 were fixed to 1500 bytes. For each task-set, ES2 served 16 periodic OSEK tasks, 8 of which had a dedicated VL for sending or receiving a TT frame through the physical link. As this use case targets TT communication, these communicating tasks only sent and received TT frames before termination, while the free tasks performed busy-waiting for a bounded random time and terminated. Moreover, each generated task-set ran for at most one hour.

The OSEK application tasks were scheduled offline corresponding to their communication schedule, in order to reduce transmission latency. The TT schedule was generated using TTPlan (see Section 1.1.5) and the ECU configuration code for ES2 was generated using *AUTOSAR Design Studio*. To simplify the evaluation process, the TCP/IP stack and the BE cross-traffic were not used for the RPi. The DC files and their corresponding HEX files for the non-AUTOSAR TTEthernet devices were generated and uploaded to them using TTLoad or Eclipse [EF18b] based TTEthernet plugin.

A.5 Evaluation of Raspberry Pi Based End-System

In this section, we present the evaluation of our software-based TTEthernet end-system (ES) based on RPi. The goal of these evaluations is to test (i) if the RPi ES synchronizes with the network, (ii) if the application frame transmission/reception is successful, and (iii) what is the effective Ethernet throughput.

For the sake of determining the synchronization precision, we record the clock corrections made by the RPi ES at the reception of each PCF from the Compression Master

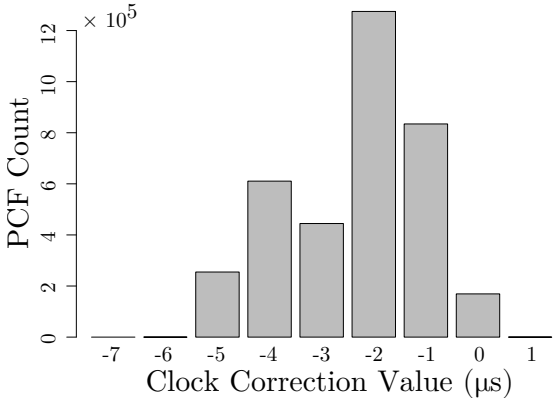


Figure A.2: *Raspberry Pi clock correction for TTEthernet synchronization*

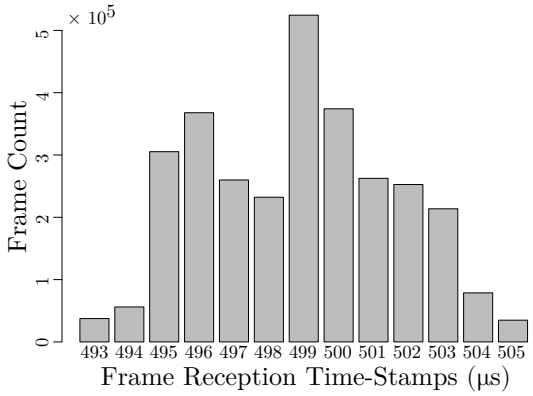


Figure A.3: *Time-stamps for TT frames received by Raspberry Pi End-System*

(CM, see Section A.1). Figure A.2 shows the histogram of clock correction values of RPi for one of the task-sets with 1ms integration cycle (IC) length. The figure shows that the RPi clock is faster than the global network clock and hence, it needs to shift its local clock backwards upon reception of almost every PCF.

The TTEthernet toolchain uses 6μs as the default clock precision value. When the local clock of an ES in the network deviates more than this defined clock precision, the ES is deemed out-of-sync by the clock synchronization protocol [SAE11]. For the RPi ES, we used the same interval, i.e., 6μs, to change the system synchronization state to out-of-sync, after which the synchronization process was restarted. As the IC length is increased, the value of required clock corrections is also increased due to longer duration between reception of two consecutive PCFs. This lead to frequent loss of synchronization by the RPi ES for longer IC lengths. Therefore, the IC length of 1ms was chosen for the later experiments.

In RPi ES, each received TT frame is assigned a time-stamp, which identifies the synchronized RPi clock value at the time instant when the last bit of the frame is received. In order to measure the TT frame reception jitter, we logged the time-stamps for each TT frame received by RPi ES (sent from ES0 or ES1, see Figure A.1). Figure A.3 shows the time-stamp histogram for one of the task-sets. It shows the time-stamps for a specific TT message with 10ms VL period, 100 bytes message size and a TT window of [400, 524]μs. In Figure A.3, the x-axis represents the frame reception time-stamp after the start of the VL period. The figure shows that all the TT frames of this VL are received approximately 100μs after the window starts and the frame reception jitter is $\pm$6μs. The standard deviation of all the time-stamps of the received frames of all the task-sets was measured to be 2.94μs. Note that the median reception time of a TT frame depends on the link speed and the sending device.

The transmission of TT frames from RPi ES is evaluated by shifting the frame dispatch time in RPi around the TT transmission window and checking if the messages are received by ES0/ES1. The reception of TT frames in ES0/ES1 is recorded using WireShark. Figure A.4 shows the resulting success ratio for a specific TT message from one of the task-sets with 10ms VL period, 100 bytes message size and a TT window of

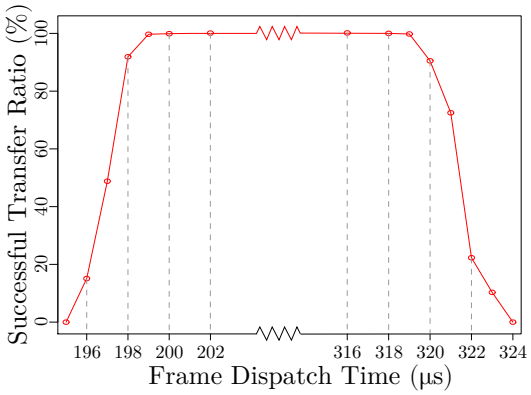


Figure A.4: *TT frame transfer ratio with shifted dispatch time*

[200, 324]μs. Each point in the figure represents one run of the task-set for 10 minutes with a message dispatch time shown on the x-axis. Note that the figure shows only the start and end of the TT transmission window, since the frame dispatch time between the interval [202, 316]μs lead to 100% transfer ratios. The figure shows that all the frames sent by RPi ES are received by ES0/ES1 when the frame dispatch time lies within an interval of [199, 319]μs.

In order to give a rough idea about the Ethernet throughput, Figure A.5 shows the box-plot of SPI transfer times (between RPi and the ENC624J600 chip) of all the task-sets for different Ethernet frame sizes. In the figure, the dashed line shows the ideal linear transfer time. The figure shows that the SPI transfer time for a given frame size varies only by a few microseconds (indicated by very small box sizes). Due to different SPI frame padding for different Ethernet frame sizes and 64 bytes buffer size in RPi hardware SPI module, the SPI transfer time for different Ethernet frame sizes deviates from the ideal linear transfer time.

Although Figure A.5 provides a rough estimate of the Ethernet throughput (i.e., 8Mbps), it fails to provide an exact value. To get the effective Ethernet throughput, an ECU configuration with Ethernet traffic of more than 10Mbps was generated and

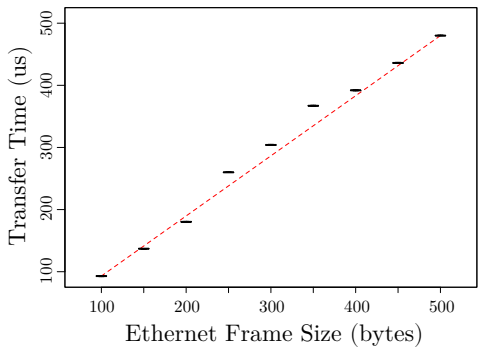


Figure A.5: *SPI transfer duration between Raspberry Pi and ENC624J600*

the frames received by ES0/ES1 were counted. It was observed that beyond ≈7.5Mbps throughput, the Ethernet buffer in ENC624J600 chip overflows and renders RPi out-of-sync. The difference between the estimated and the effective throughput is due to the inter-frame gaps on the SPI interface and overheads introduced due to the OS and MCAL modules.

A number of conclusions can be drawn from the above results, which are summarized in the following:

1. The clock precision is $\pm 8\mu$s for 1ms IC length (see Figure A.2).

2. The Ethernet frame reception jitter is $\pm 7\mu$s (see Figure A.3), while a frame transfer is successful when the frame is delayed as much as the clock precision (see Figure A.4).

3. The effective throughput for all traffic types is low (i.e., ≈7.5Mbps) due to SPI interface bottleneck (see Figure A.5).

4. With proper parameters selection (i.e., IC length in the range of 1ms to 2ms, frame dispatch shift equal to clock precision), every TT frame can be received by their respective devices.

A.6 Observation and Discussion

For all generated task-sets, the code generators for Microcontroller Unit (MCU), Electronic Control Unit Manager (ECUM), MCU Port (PORT), Digital I/O (DIO), Operating System (OS), Ethernet for MCU (ETH) and Ethernet Interface (ETHIF) modules were used. Moreover, a code generator for a virtual TTEthernet module was written, which generated TTEthernet related code for OS tasks, ISR and schedules (defined in Section A.3). All the generated code compiled without any error. The generated code was found compatible with the AUTOSAR Core, as indicated by the observations in Section A.5. Although minor inevitable modifications in the ETH and ETHIF modules were required (defined in Section A.3), no dirty workaround was required to achieve the required task. The artifact generation process lead to long lasting synchronization, successful TT traffic transfer and low reception jitter.

Although the idea of this use case is to exhibit the extension capabilities of AUTOXTEND, it is not a novel idea, instead only a novel approach. In 2015, Frühwirth et al. [FSS15] designed a software-based TTEthernet end-system for AUTOSAR. They provided support for mixed-criticality traffic with TCP/IP stack. To achieve synchronization over Ethernet, they created a new AUTOSAR BSW module *TtePl* without declaring the toolchain integration design (aka AUTOSAR design methodology) or the source of required parameters. Frühwirth et al. provided support for Time-Triggered (TT) and Best-effort (BE) traffic types. However, they used Timely Block (see Section A.1) as their traffic integration policy, which complicates the process of frame transmission, incurring overheads. Due to the same reason, they required a safety margin of 15μs before sending every BE frame.

Our software-based TTEthernet ES implementation is different from the one provided by Frühwirth et al. [FSS15] in a number of ways. In this work, we use shuffling as our traffic integration policy to avoid software overheads and use existing AUTOSAR services (i.e., OSEK tasks, interrupt service routines and scheduling tables) which do not cast any impact on the AUTOSAR architecture. We also provide the source of parameters for these services and follow the AUTOSAR design methodology. However compared to our Ethernet throughput (i.e., ≈7.5Mbps due to SPI bottleneck), the implementation provided by Frühwirth et al. allows an estimated throughput ranging from 30.9Mbps (for smallest Ethernet frame size and 15μs safety margin) to 89Mbps (for largest Ethernet frame size and 15μs safety margin) on a 100Mbps Ethernet link.

Bibliography

[AB98] L. Abeni and G. Buttazzo. Integrating Multimedia Applications in Hard Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, RTSS, Washington, DC, USA, 1998. IEEE Computer Society.

[AB10] Vallabh Anwikar and Purandar Bhaduri. Timing analysis of real-time embedded systems using model checking. In *18th International Conference on Real-Time and Network Systems*, pages 119–128, 2010.

[ABB16] V. Apuzzo, A. Biondi, and G. Buttazzo. OSEK-Like Kernel Support for Engine Control Applications under EDF Scheduling. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 1–11, April 2016.

[AE07] Björn Andersson and Cecilia Ekelin. Exact admission-control for integrated aperiodic and periodic tasks. *Journal of Computer and System Sciences*, 73(2):225–241, 2007.

[AFD09] Aircraft Data Network Part 7X. Avionics Full-Duplex Switched Ethernet Network. Standard, ARINC Specification 664 P7-1, sept 2009.

[AK98] Ishfaq Ahmad and Yu-Kwong Kwok. Optimal and Near-Optimal Allocation of Precedence-Constrained Tasks to Parallel Processors: Defying the High Complexity Using Effective Search Techniques. In *International Conference on Parallel Processing*, pages 424–431. IEEE, 1998.

[AKN+14] Jakob Axelsson, Avenir Kobetski, Ze Ni, Shuzhou Zhang, and Eilert Johansson. MOPED: A Mobile Open Platform for Experimental Design of Cyber-Physical Systems. In *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, SEAA, pages 423–430, Washington, DC, USA, 2014. IEEE Computer Society.

[Arc17] ArcCore. AUTOSAR tools and software for the automotive industry. `https://www.arccore.com`, 2017. Retrieved: 18th Sept.

[ARI03] ARINC-653. Avionics Application Software Standard Interface. Standard, ARINC Inc., 2003.

[ARSF11] Karl-Erik Arzen, Vanessa Romero, Stefan Schorr, and Gerhard Fohler. Adaptive Resource Management Made Real. In *Proceedings of the 3rd Workshop on Adaptive and Reconfigurable Embedded Systems (APRES)*, Chicago, April 2011.

[AS99] Tarek F. Abdelzaher and Kang G. Shin. Combined Task and Message Scheduling in Distributed Real-Time Systems. *IEEE Transaction on Parallel Distributed Systems*, 10(11):1179–1191, November 1999.

[AS00] Tarek F. Abdelzaher and Kang G. Shin. Period-Based Load Partitioning and Assignment for Large Real-Time Applications. *IEEE Transaction on Computers*, 49(1):81–87, January 2000.

[AUT16a] AUTOSAR. Standard 4.2.2 - Specification of Operating System. Document ID: 034, 2016.

[AUT16b] AUTOSAR. Standard 4.2.2 - Virtual Function Bus. Document ID: 056, 2016.

[AUT17a] AUTOSAR. ARText - An AUTOSAR Textual Language Framework. `https://www.artop.org/artext/`, 2017. Retrieved: 18th Sept.

[AUT17b] AUTOSAR. AUTomotive Open System ARchitecture. `http://www.autosar.org/`, 2017. Retrieved: 18th Sept.

[AUT17c] AUTOSAR. AUTOSAR Tool Platform (ARTOP). `https://www.artop.org/`, 2017. Retrieved: 18th Sept.

[AVT+15] Vincent Aravantinos, Sebastian Voss, Sabine Teufl, Florian Hölzl, and Bernhard Schätz. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. In *ACES-MB&WUCOR@ MoDELS*, pages 19–26, 2015.

[Bar03] Sanjoy K. Baruah. Dynamic- and Static-priority Scheduling of Recurring Real-time Tasks. *Real-Time Systems Journal*, 24(1):93–128, 2003.

[Bar12] Sanjoy Baruah. Certification-cognizant scheduling of tasks with pessimistic frequency specification. In *Industrial Embedded Systems (SIES), 2012 7th IEEE International Symposium on*, pages 31–38. IEEE, 2012.

[BB05] Enrico Bini and Giorgio C Buttazzo. Measuring the performance of schedulability tests. *Real-Time Systems Journal*, 30(1-2):129–154, 2005.

[BB11] Alan Burns and Sanjoy Baruah. Timing faults and mixed criticality systems. In *Dependable and Historic Computing*, pages 147–166. Springer, 2011.

[BDF+03] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *ACM SIGOPS operating systems review*, volume 37, pages 164–177. ACM, 2003.

[BDM+16] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. Synthesizing Job-Level Dependencies for Automotive Multi-Rate Effect Chains. In *The 22th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, August 2016.

[BDM+17] Simon Barner, Alexander Diewald, Jörn Migge, Ali Abbas Jaffari Syed, Gerhard Fohler, Faugére Madeleine, and Daniel Gracia Pérez. DREAMS Toolchain: Model-Driven Engineering of Mixed-Criticality Systems. In *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems*, September 2017.

[BDNM16] Marc Boyer, Hugo Daigmorte, Nicolas Navet, and Jörn Migge. Performance impact of the interactions between time-triggered and rate-constrained transmissions in TTEthernet. 2016.

[BFG02] Doug Bagley, Brent Fulgham, and Isaac Gouy. The Computer Language Benchmarks Game. `https://benchmarksgame.alioth.debian.org/`, 2002. Retrieved: 18th Jan, 2018.

[BFR75] Paul Bratley, Michael Florian, and Pierre Robillard. Scheduling with Earliest Start and Due Date Constraints on Multiple Machines. *Naval Research Logistics Quarterly*, 22(1):165–173, 1975.

[BL17] Beyond Logic USB in a NutShell - Chapter 4 - Endpoint Type. `http://www.beyondlogic.org/usbnutshell/usb4.shtml`, 2017. Retrieved: 18th Sept.

[BMR+10] Vicent Brocal, Miguel Masmano, Ismael Ripoll, Alfons Crespo, Patricia Balbastre, and Jean-Jacques Metge. Xoncrete: a scheduling tool for partitioned real-time systems. *Embedded Real-Time Software and Systems*, 2010.

[Bra11] Björn B. Brandenburg. *Scheduling and Locking in Multiprocessor Real-Time Operating Systems*. PhD thesis, The University of North Carolina at Chapel Hill, 2011.

[BRH90] Sanjoy K Baruah, Louis E Rosier, and Rodney R Howell. Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor. *Real-time systems Journal*, 2(4):301–324, 1990.

[But11] Giorgio Buttazzo. *Hard real-time computing systems: predictable scheduling algorithms and applications*, volume 24. Springer Science & Business Media, 2011.

[BV15] K. Becker and S. Voss. Analyzing Graceful Degradation for Mixed Critical Fault-Tolerant Real-Time Systems. In *18th IEEE International Symposium on Real-Time Distributed Computing*, pages 110–118, April 2015.

[CGJ78] Edward G Coffman, Jr, Michael R Garey, and David S Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[CHD14] Maxime Chéramy, Pierre-Emmanuel Hladik, and Anne-Marie Déplanche. SimSo: A Simulation Tool to Evaluate Real-Time Multiprocessor Scheduling Algorithms. In *Proc. of the 5th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*, WATERS, 2014.

[CHL+03] Anton Cervin, Dan Henriksson, Bo Lincoln, Johan Eker, and Karl-Erik Årzén. How Does Control Timing Affect Performance? Analysis and Simulation of Timing Using Jitterbug and TrueTime. *IEEE Control Systems Magazine*, 23:16–30, June 2003.

[CK96] Yang Cai and MC Kong. Nonpreemptive scheduling of periodic tasks in uni- and multiprocessor systems. *Algorithmica Journal*, 15(6):572–599, 1996.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2001.

[CMP+10] Daniel Cordeiro, Grégory Mounié, Swann Perarnau, Denis Trystram, Jean-Marc Vincent, and Frédéric Wagner. Random Graph Generation for Scheduling Simulations. In *International ICST Conference on Simulation Tools and Techniques (SIMUTools)*, 2010.

[CO16] Silviu S. Craciunas and Ramon Serna Oliver. Combined task- and network-level scheduling for distributed time-triggered systems. *Real-Time Systems Journal*, 52(2):161–200, 2016.

[COCS16] Silviu S Craciunas, Ramon Serna Oliver, Martin Chmelík, and Wilfried Steiner. Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 183–192. ACM, 2016.

[Coe17] Rodrigo Ferreira Coelho. *Buffer Analysis and Message Scheduling for Real-Time Networks*. PhD thesis, Nov 2017.

[CRM+09] A Crespo, I Ripoll, M Masmano, P Arberet, and JJ Metge. XtratuM an Open Source Hypervisor for TSP Embedded Systems in Aerospace. *Data Systems In Aerospace DASIA, Istanbul, Turkey*, 2009.

[CS18] Code Synthesis CodeSynthesis XSD - XML Data Binding for C++, 2018. `https://www.codesynthesis.com/products/xsd/`.

[CSB90] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic Scheduling of Real-time Tasks Under Precedence Constraints. *Real-Time Systems Journal*, 2(3):181–194, September 1990.

[DC11] DO-178C. Software Considerations in Airborne Systems and Equipment Certification. RTCA. Standard, Inc., 2011.

[DFG+16] Guy Durrieu, Gerhard Fohler, Gautam Gala, Sylvain Girbal, Daniel Gracia Pérez, Eric Noulard, Claire Pagetti, and Simara Pérez. DREAMS about reconfiguration and adaptation in avionics. In *ERTS*, 2016.

[DRE14] DREAMS consortium. Architectural Style of DREAMS . D1.2.1, 2014.

[DRE15] DREAMS consortium. Meta-models for Application and Platform. D1.4.1, March 2015.

[DRW98] Robert P. Dick, David L. Rhodes, and Wayne Wolf. TGFF: Task Graphs for Free. In *Proceedings of the 6th International Workshop on Hardware/Software Codesign*, CODES/CASHE, pages 97–101, Washington, DC, USA, 1998. IEEE Computer Society.

[EF13] Eclipse Foundation Xpand - M2T Code Generation Language. June 2013. `https://wiki.eclipse.org/Xpand`.

[EF18a] Eclipse Foundation Eclipse Modeling Project, 2018. `https://www.eclipse.org/modeling/emf/`.

[EF18b] Eclipse Foundation The Eclipse Foundation open source community website., 2018. `https://www.eclipse.org/`.

[EJ01] Cecilia Ekelin and Jan Jonsson. Evaluation of Search Heuristics for Embedded System Scheduling Problems. In *Principles and Practice of Constraint Programming*, volume 2239 of *Lecture Notes in Computer Science*, pages 640–654. Springer Berlin Heidelberg, 2001.

[Ele17a] Elektrobit. tresos AutoCore. `https://www.elektrobit.com/products/ecu/eb-tresos/autocore/`, 2017. Retrieved: 18th Sept.

[Ele17b] Elektrobit. tresos OsekCore. `https://www.elektrobit.com/products/ecu/eb-tresos/osekcore/`, 2017. Retrieved: 18th Sept.

[EN16] R. Ernst and M. Di Natale. Mixed Criticality Systems - A History of Misconceptions? *IEEE Design Test*, 33(5):65–74, Oct 2016.

[ENNT15] Alexandre Esper, Geoffrey Nelissen, Vincent Nélis, and Eduardo Tovar. How realistic is the mixed-criticality real-time system model? In *Proceedings of the 23rd International Conference on Real Time and Networks Systems*, pages 139–148. ACM, 2015.

[ERI17] ERIKA Enterprise. Open Source RTOS OSEK/VDX Kernel. `http://erika.tuxfamily.org/drupal/`, 2017. Retrieved: 18th Sept, 2017.

[FC16] Cong Fu and Deng Cai. EFANNA : An extremely fast approximate nearest neighbor search algorithm based on knn graph. *CoRR*, abs/1609.07228, 2016.

[FKPY07] Elena Fersman, Pavel Krcal, Paul Pettersson, and Wang Yi. Task automata: Schedulability, decidability and undecidability. *Information and Computation*, 205(8):1149–1172, 2007.

[Foh93] Gerhard Fohler. *Realizing Changes of Operational Modes with a Pre Run-Time Scheduled Hard Real-Time System*, pages 287–300. Springer Vienna, Vienna, 1993.

[Foh94] Gerhard Fohler. *Flexibility in Statically Scheduled Real-Time Systems*. PhD thesis, TNF, Wien, Österreich, April 1994.

[FSS15] T. Frühwirth, W. Steiner, and B. Stangl. TTEthernet SW-based end system for AUTOSAR. In *10th IEEE International Symposium on Industrial Embedded Systems (SIES)*, pages 1–8, June 2015.

[GB13] Patrick Graydon and Iain Bate. Safety assurance driven problem formulation for mixed-criticality scheduling. In *Proceedings of WMC (RTSS)*, 2013.

[GBC16] Ana Guasque, Patricia Balbastre, and Alfons Crespo. Real-time hierarchical systems with arbitrary scheduling at global level. *Journal of Systems and Software*, 119:70 – 86, 2016.

[GD99] Joël Goossens and Raymond Devillers. Feasibility Intervals for the Deadline Driven Scheduler with Arbitrary Deadlines. In *Proceedings of the Sixth International Conference on Real-Time Computing Systems and Applications*, RTCSA, Washington, DC, USA, 1999. IEEE Computer Society.

[GK03] F.W. Glover and G.A. Kochenberger. *Handbook of Metaheuristics*. International Series in Operations Research & Management Science. Springer US, 2003.

[Gra17] Andreas Graf. How to configure your free AUTOSAR model viewer. October 2017. `https://blogs.itemis.com/en/how-to-configure-your-free-autosar-model-viewer`.

[GRS96] Laurent George, Nicolas Rivierre, and Marco Spuri. Preemptive and Non-Preemptive Real-Time UniProcessor Scheduling. Research Report RR-2966, INRIA, 1996. Projet REFLECS.

[HBS10] Z. Hanzalek, P. Burget, and P. Sucha. Profinet IO IRT Message Scheduling With Temporal Constraints. *IEEE Transactions on Industrial Informatics*, 6(3):369–380, Aug 2010.

[Hea63] B. R. Heap. Permutations by Interchanges. *The Computer Journal*, 6(3):293–298, 1963.

[Hei10] Gernot Heiser. Benchmarking Crimes. `http://gernot-heiser.org/benchmarking-crimes.html`, January 2010. Retrieved: 12th Dec, 2017.

[HHR+04] Arne Hamann, Rafik Henia, Razvan Racu, Marek Jersak, Kai Richter, and Rolf Ernst. SymTA/S - Symbolic Timing Analysis for Systems. In *WIP Proc. Euromicro Conference on Real-Time Systems 2004 (ECRTS'04)*, pages 17–20, 2004.

[HM03] William Hoffman and Ken Martin. The CMake Build Manager - Cross platform and open source, January 2003. `http://www.drdobbs.com/cpp/the-cmake-build-manager/184405251`.

[HNR68] P.E. Hart, N.J. Nilsson, and B. Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transaction on Systems Science and Cybernetics*, 4(2):100–107, July 1968.

[HS04] Udo Hönig and Wolfram Schiffmann. Fast Optimal Task Graph Scheduling by Means of an Optimized Parallel A*-Algorithm. In *PDPTA*, pages 842–848, 2004.

[HSF16] Florian Heilmann, Ali Abbas Jaffari Syed, and Gerhard Fohler. Mode-Changes in COTS Time-Triggered Network Hardware without Online Reconfiguration. In *14th Workshop on Real-Time Networks (RTN) in conjuction with 28th Euromicro International Conference on Real-time Systems (ECRTS)*, July 2016.

[IEC10] IEC61508. Functional Safety of Electrical/electronic/programmable electronic safety-related systems. Standard, IEC, 2010.

[IF00] Damir Isović and Gerhard Fohler. Efficient Scheduling of Sporadic, Aperiodic, and Periodic Tasks with Complex Constraints. In *Proceedings of the 21st IEEE Conference on Real-time Systems Symposium*, RTSS, pages 207–216, Washington, DC, USA, 2000. IEEE Computer Society.

[IS13] InvenSense 6-Axis MotionTracking Device. Datasheet MPU6050, `https://store.invensense.com/datasheets/invensense/MPU-6050_DataSheet_V3%204.pdf`, 2013.

[ISO11] ISO26262. Road vehicles - Functional Safety. Standard, International Organization for Standardization, 2011.

[Iti07] Jean-Bernard Itier. A380 integrated modular avionics. In *Proceedings of the ARTIST2 meeting on integrated modular avionics*, volume 1, pages 72–75, 2007.

[JLM88] Farnam Jahanian, Raymond Lee, and Aloysisu K Mok. Semantics of modechart in real time logic. In *System Sciences, 1988. Vol. II. Software Track, Proceedings of the Twenty-First Annual Hawaii International Conference on*, volume 2, pages 479–489. IEEE, 1988.

[JLT85] E Douglas Jensen, C Douglas Locke, and Hideyuki Tokuda. A time-driven scheduling model for real-time operating systems. In *RTSS*, volume 85, pages 112–122, 1985.

[Jon99] Jan Jonsson. Effective Complexity Reduction for Optimal Scheduling of Distributed Real-Time Applications. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 360–369, 1999.

[JSM91] Kevin Jeffay, Donald F Stanat, and Charles U Martel. On non-preemptive scheduling of period and sporadic tasks. In *Proceedings of Real-Time Systems Symposium (RTSS)*, pages 129–139. IEEE, 1991.

[KALW91] Jan Korst, Emile Aarts, Jan Karel Lenstra, and Jaap Wessels. Periodic Multiprocessor Scheduling. In *Parallel Architectures and Languages Europe (PARLE)*, pages 166–178. Springer, 1991.

[Kap17] Arseny Kapoulkine. PugiXML: Light-weight, simple and fast XML parser for C++ with XPath support. `https://github.com/zeux/pugixml`, 2017. Retrieved: 12th Dec.

[KDMK10] K. Klobedanz, G. B. Defo, W. Mueller, and T. Kerstan. Distributed coordination of task migration for fault-tolerant FlexRay networks. In *International Symposium on Industrial Embedded System (SIES)*, 2010.

[KG94] H. Kopetz and G. Grunsteidl. TTP - A Protocol for Fault-Tolerant Real-Time Systems. *Journal of Computer*, 1994.

[KGGP+16] Thomas Koller, Gautam Gala, Daniel Gracia Pérez, Christoph Ruland, and Gerhard Fohler. DREAMS: Secure Communication Between Resource Management Components in Networked Multi-Core Systems. IEEE Conference on Open Systems, 2016.

[KGV83] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[KKM11] Kay Klobedanz, Andreas Koenig, and Wolfgang Mueller. A reconfiguration approach for fault-tolerant FlexRay networks. *Design, Automation & Test in Europe*, 2011.

[KKMR11] K. Klobedanz, A. Koenig, W. Mueller, and A. Rettberg. Self-Reconfiguration for Fault-Tolerant FlexRay Networks. In *14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW)*, 2011.

[KNH+98] H Kopetz, R Nossal, R Hexel, A Krueger, D Millinger, R Pallierer, C Temple, and M Krug. Mode handling in the Time-Triggered Architecture. *Control Engineering Practice*, 1998.

[Kop91] H. Kopetz. Event-triggered versus time-triggered real-time systems. In Arthur Karshmer and Jürgen Nehmer, editors, *Operating Systems of the 90s and Beyond*, pages 86–101, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[Kop92] Hermann Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In *Proceedings of the 12th International Conference on Distributed Computing Systems*, pages 460–467, Jun 1992.

[Kor85] Richard E. Korf. Depth-First Iterative-Deepening: An Optimal Admissible Tree Search. *Artificial Intelligence*, 27(1):97 – 109, 1985.

[Kur09] Stan Kurkovsky. Experimenting with IDA* Search Algorithm in Heterogeneous Pervasive Environments. *Artificial Intelligence Review*, 29(3):277–286, 2009.

[LAG14] Y Li, B Akesson, and K Goossens. RTMemController: Open-source WCET and ACET analysis tool for real-time memory controllers. `http://www.es.ele.tue.nl/rtmemcontroller/`, 2014.

[Law97] Wolfhard Lawrenz. Can system engineering. *From theory to practical applications, New York*, 1997.

[LCGG10] Irina Lupu, Pierre Courbin, Laurent George, and Joël Goossens. Multi-criteria evaluation of partitioning schemes for real-time systems. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2010.

[LD60] A. H. Land and A. G. Doig. An automatic method for solving discrete programming problems. *ECONOMETRICA*, 28(3):497–520, 1960.

[Leh92] J.P. Lehoczky. An optimal algorithm for scheduling soft-aperiodic tasks in fixed-priority preemptive systems. In *Real-Time Systems Symposium*, pages 110–123, Dec 1992.

[LL73] C. L. Liu and James W. Layland. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *Journal of the ACM (JACM)*, 20(1):46–61, January 1973.

[LM80] Joseph Y-T Leung and ML Merrill. A note on preemptive scheduling of periodic, real-time tasks. *Information processing letters*, 11(3):115–118, 1980.

[LRSF04] P. Li, Binoy Ravindran, S. Suhaib, and S. Feizabadi. A formally verified application-level framework for real-time scheduling on POSIX real-time operating systems. *IEEE Transactions on Software Engineering*, 30(9):613–629, Sept 2004.

[LSST02] Chenyang Lu, John A. Stankovic, Sang H. Son, and Gang Tao. Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms*. *Real-Time Systems Journal*, 23(1):85–126, Jul 2002.

[LWVH12] Adam Lackorzyński, Alexander Warg, Marcus Völp, and Hermann Härtig. Flattening Hierarchical Scheduling. In *Proceedings of the Tenth ACM International Conference on Embedded Software*, EMSOFT, pages 93–102, New York, NY, USA, 2012. ACM.

[Mag15] Martina Maggio. rt-muse - A Linux Scheduler Benchmarking Tool. `https://github.com/martinamaggio/rt-muse`, 2015.

[Mai17] Mirko Maischberger. An Open Source Tabu Search Metaheuristic framework in modern C++ (cpp). `https://projects.coin-or.org/metslib`, 2017. Retrieved: 22nd Feb, 2018.

[Man17] Renato Mancuso. *Next-generation safety-critical systems on multi-core platforms*. PhD thesis, University of Illinois at Urbana-Champaign, 2017.

[McK13] McKinsey. The Internet of Things and the Future of Manufacturing. `http://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/the-internet-of-things-and-the-future-of-manufacturing`, 2013. Retrieved: 10th Nov, 2016.

[MGUU11] Stefan Metzlaff, Irakli Guliashvili, Sascha Uhrig, and Theo Ungerer. A dynamic instruction scratchpad memory for embedded processors managed by hardware. In *International Conference on Architecture of Computing Systems*, pages 122–134. Springer, 2011.

[Mik17] Mikroe. LIN Board - Add-on board with MCP201 LIN BUS physical interface. `https://shop.mikroe.com/lin-board`, 2017. Retrieved: 18th Oct.

[Mok83] A. K. Mok. Fundamental Design Problems of Distributed Systems for the Hard-Real-Time Environment. Technical report, Cambridge, MA, USA, 1983.

[Mon16] David Monniaux. A survey of satisfiability modulo theory. In *International Workshop on Computer Algebra in Scientific Computing*, pages 401–425. Springer, 2016.

[MS10] Mohamed Marouf and Yves Sorel. Schedulability conditions for non-preemptive hard real-time tasks with strict period. In *18th International Conference on Real-Time and Network Systems RTNS*, Proceedings of 18th International Conference on Real-Time and Network Systems (RTNS), Toulouse, France, November 2010.

[MS11] Mohamed Marouf and Yves Sorel. Scheduling non-preemptive hard real-time tasks with strict periods. In *16th IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8. IEEE, 2011.

[MTI09] Microchip Technology Inc.. Stand-Alone 10/100 Ethernet Controller with SPI or Parallel Interface. Datasheet ENC424J600/624J600, `http://ww1.microchip.com/downloads/en/DeviceDoc/39935b.pdf`, 2009.

[NBFK14] Mitra Nasri, Sanjoy Baruah, Gerhard Fohler, and Mehdi Kargahi. On the optimality of RM and EDF for non-preemptive real-time harmonic tasks. In *Proceedings of the 22nd International Conference on Real-Time Networks and Systems*, page 331. ACM, 2014.

[NF16] Mitra Nasri and Gerhard Fohler. Non-Work-Conserving Non-Preemptive Scheduling: Motivations, Challenges, and Potential Solutions. In *Euromicro Conference on Real-Time Systems (ECRTS)*, July 2016.

[NKA14] Ze Ni, Avenir Kobetski, and Jakob Axelsson. Design and Implementation of a Dynamic Component Model for Federated AUTOSAR Systems. In *Proceedings of the 51st Annual Design Automation Conference*, DAC, pages 94:1–94:6, New York, NY, USA, 2014. ACM.

[NNF16] Mitra Nasri, Geoffrey Nelissen, and Gerhard Fohler. A new approach for limited preemptive scheduling in systems with preemption overhead. In *Real-Time Systems (ECRTS), 2016 28th Euromicro Conference on*, pages 25–35. IEEE, 2016.

[NPC12] Eric Noulard, Claire Pagetti, and Mikel Cordovilla. SchedMCore: A scheduler software framework designed for research purpose targeting multi-core architecture. `http://sites.onera.fr/schedmcore/node/1`, 2012.

[NSE11] Moritz Neukirchner, Steffen Stein, and Rolf Ernst. SMFF: System Models For Free. In *2nd International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2011)*, page 6, 2011.

[OSE05] OSEK. OSEK/VDX Specification. February 2005. `http://paginapessoal.utfpr.edu.br/lalucas/disciplinas/sistemas-operacionais-em-tempo-real/osek/os223%20-comentado.pdf/at_download/file`.

[Par17] Parai. OpenSAR. `https://github.com/parai/OpenSAR`, 2017. Retrieved: 18th Sept.

[Par18] Parai. Automotive Software and its Toolchain. `https://coding.net/u/parai/p/as/git`, 2018. Retrieved: 18th Jan.

[Phi02] NXP Philips. LIN Transceiver. Datasheet TJA1020, `https://www.nxp.com/docs/en/data-sheet/TJA1020.pdf`, 2002.

[Pie05] Emmanuel Pietriga. A Toolkit for Addressing HCI Issues in Visual Language Environments. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 00:145–152, 2005.

[PP07] Isabelle Puaut and Christophe Pais. Scratchpad memories vs locked caches in hard real-time systems: a quantitative comparison. In *Design, Automation & Test in Europe Conference & Exhibition, 2007. DATE'07*, pages 1–6. IEEE, 2007.

[PRNHS17] Francisco Pozo, Guillermo Rodriguez-Navas, Hans Hansson, and Wilfried Steiner. Schedule synthesis for next generation time-triggered networks. Technical report, February 2017.

[PRNSH16] Francisco Pozo, Guillermo Rodriguez-Navas, Wilfried Steiner, and Hans Hansson. Period-aware segmented synthesis of schedules for multi-hop time-triggered networks. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2016 IEEE 22nd International Conference on*, pages 170–175. IEEE, 2016.

[PS89] D.-T. Peng and K.G. Shin. Static Allocation of Periodic Tasks with Precedence Constraints in Distributed Real-Time Systems. In *9th International Conference on Distributed Computing Systems*, pages 190–198, Jun 1989.

[PS93] D.-T. Peng and K.G. Shin. Optimal Scheduling of Cooperative Tasks in a Distributed System Using an Enumerative Method. *IEEE Transaction on Software Engineering*, 19(3):253–267, Mar 1993.

[PSG+14] Claire Pagetti, D. Saussié, R. Gratia, Erik Noulard, and P. Siron. The ROSACE case study: From Simulink specification to multi/many-core execution. In *19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 309–318, April 2014.

[PSRNH15] Francisco Pozo, Wilfried Steiner, Guillermo Rodriguez-Navas, and Hans Hansson. A decomposition approach for smt-based schedule synthesis for time-triggered networks. In *Emerging Technologies & Factory Automation (ETFA), 2015 IEEE 20th Conference on*, pages 1–8. IEEE, 2015.

[RKR87] V. Nageshwara Rao, Vipin Kumar, and K. Ramesh. A Parallel Implementation of Iterative-Deepening-A*. In *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 1*, pages 178–182, 1987.

[RM08] H. Ramaprasad and F. Mueller. Bounding Worst-Case Response Time for Tasks with Non-Preemptive Regions. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 58–67, April 2008.

[RPi] Raspberry Pi Foundation. Raspberry Pi. `https://www.raspberrypi.org/`. Retrieved: 18th Sept, 2017.

[RS95] S. Ronngren and B. Shirazi. Static Multiprocessor Scheduling of Periodic Real-Time Tasks with Precedence Constraints and Communication Costs. In *Proceedings of the Twenty-Eighth Hawaii International Conference on System Sciences*, volume 2, pages 143–152, Jan 1995.

[RSC16] Jorge Real, Sergio Saez, and Alfons Crespo. Combining Time-Triggered Plans with Priority Scheduled Task Sets. In *21st Ada-Europe International Conference on Reliable Software Technologies, Pisa, Italy, June 13-17*, pages 195–212, 2016.

[RTa17] RealTime-at-Work. Timing Analysis and Design Tools. `http://www.realtimeatwork.com/`, 2017. Retrieved: 22nd Sept.

[SAE11] Society of Automotive Engineers. Time-Triggered Ethernet. SAE Standard AS6802, 2011.

[Sar12] Abhik Sarkar. *Predictable Task Migration Support and Static Task Partitioning for Scalable Multicore Real-Time Systems*. PhD thesis, North Carolina State University, 2012.

[Sch15] Stefan Schorr. *Adaptive Real-Time Scheduling and Resource Management on Multicore Architectures*. PhD thesis, March 2015.

[SCM+14] Lui Sha, Marco Caccamo, Renato Mancuso, Jung-Eun Kim, Man-Ki Yoon, Rodolfo Pellizzoni, Heechul Yun, Russel Kegley, Dennis Perlman, Greg Arundale, et al. Single core equivalent virtual machines for hard real-time computing on multicore processors. Technical report, 2014.

[Sed77] Robert Sedgewick. Permutation Generation Methods. *ACM Computing Surveys (CSUR)*, 9(2):137–164, June 1977.

[Sem11] ON Semiconductor. LIN Transceiver. Datasheet NCV7425EVB, `http://www.onsemi.com/pub/Collateral/EVBUM2045-D.PDF`, 2011.

[SF14] Ali Abbas Jaffari Syed and Gerhard Fohler. Search-Tree Exploration For Scheduling using PIDA*. Technical report, August 2014.

[SKF+11] Stefan Schorr, Anand Kotra, Gerhard Fohler, Johan Eker, Arzen Karl-Erik, and Vanessa Romero. Adaptive resource management in the actors framework - a live dvb-t/webcam demo. In *RTSS@Work Demo Session of the IEEE Real-Time Systems Symposium*, November 2011.

[SMS12] SMSC. USB 2.0 Hub and 10/100 Ethernet Controller. Datasheet LAN9514/LAN9514i, http://ww1.microchip.com/downloads/en/DeviceDoc/9514.pdf, 2012.

[SP98] William Stallings and Goutam Kumar Paul. *Operating systems: internals and design principles*, volume 3. prentice hall Upper Saddle River, NJ, 1998.

[SSL89] Brinkley Sprunt, Lui Sha, and John Lehoczky. Aperiodic task scheduling for Hard-Real-Time systems. *Real-Time Systems Journal*, 1(1):27–60, 1989.

[Ste10] W. Steiner. An Evaluation of SMT-Based Schedule Synthesis for Time-Triggered Multi-hop Networks. In *Proceedings of Real-Time Systems Symposium (RTSS)*, pages 375–384, Nov 2010.

[STM17] STMicroelectronics. STM32F10X Reference Manual. Reference Manual STM32F10X, http://www.st.com/content/ccc/resource/technical/document/reference_manual/59/b9/ba/7f/11/af/43/d5/CD00171190.pdf/files/CD00171190.pdf/jcr:content/translations/en.CD00171190.pdf, 2017.

[SWF+14] Lijun Shan, Yuying Wang, Ning Fu, Xingshe Zhou, Lei Zhao, Lijng Wan, Lei Qiao, and Jianxin Chen. *Formal Verification of Lunar Rover Control Software Using UPPAAL*, pages 718–732. Springer International Publishing, 2014.

[SYS17] SYSGO. PikeOS Hypervisor - Embedding Innovations. https://www.sysgo.com/products/pikeos-hypervisor/, 2017. Retrieved: 18th Sept, 2017.

[TBW92] K.W. Tindell, A. Burns, and A.J. Wellings. Allocating Hard Real Time Tasks: An NP-Hard Problem Made Easy. *Real-Time Systems Journal*, 4(2):145–165, 1992.

[TFB13] Jens Theis, Gerhard Fohler, and Sanjoy Baruah. Schedule Table Generation for Time-Triggered Mixed Criticality Systems. In *1st Workshop on Mixed Criticality Systems, IEEE Real-Time Systems Symposium*, December 2013.

[The15] Jens Theis. *Certification-Cognizant Mixed-Criticality Scheduling in Time-Triggered Systems*. PhD thesis, March 2015.

[TI15] Texas Instruments LIN Physical Interface. Technical Report SN65HVDA100, http://www.ti.com/lit/ds/symlink/sn65hvda100-q1.pdf, 2015.

[TL94] S. R. Thuel and J. P. Lehoczky. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proceedings of Real-Time Systems Symposium (RTSS)*, pages 22–33, Dec 1994.

[TLF17] The Linux Foundation. What is the ARINC653 Scheduler? https://blog.xenproject.org/2013/12/16/what-is-the-arinc653-scheduler/, 2017. Retrieved: 18th Sept.

[TLS96] Too-Seng Tia, Jane W. S. Liu, and Mallikarjun Shankar. Algorithms and optimality of scheduling soft aperiodic requests in fixed-priority preemptive systems. *Real-Time Systems Journal*, 1996.

[TOP17] TOPPERS Project. ATK2 - Automotive Kernel. https://www.toppers.jp/en/atk2.html, 2017. Retrieved: 18th Sept.

[Tos07] Toshiba. Dual DC Motor Driver. Datasheet TB6612, https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf, 2007.

[Tra17] TrampolineRTOS. Trampoline. https://github.com/TrampolineRTOS/trampoline, 2017. Retrieved: 18th Sept.

[TSPS12] Domitian Tamas-Selicean, Paul Pop, and Wilfried Steiner. Synthesis of Communication Schedules for TTEthernet-based Mixed-criticality Systems. In *Proceedings of the Eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, pages 473–482, New York, NY, USA, 2012. ACM.

[TTT17] TTTech. NASA Orion Multi-Purpose Crew Vehicle. https://www.tttech.com/markets/space/projects-references/nasa-orion/, 2017. Retrieved: 18th Sept.

[TTT18a] TTTech. Robust Networked Safety Controls - TTTech. https://www.tttech.com/, 2018. Retrieved: 10th Jan.

[TTT18b] TTTech. Time-Triggered Ethernet. https://www.tttech.com/technologies/deterministic-ethernet/time-triggered-ethernet/, 2018. Retrieved: 10th Jan.

[Tuc06] Alan Tucker. *Applied Combinatorics*. John Wiley & Sons, Inc., New York, NY, USA, 2006.

[TVS07] Andrew S Tanenbaum and Maarten Van Steen. *Distributed systems: principles and paradigms*. Prentice-Hall, 2007.

[Vec17a] Vector. CANoe. https://vector.com/vi_canoe_en.html, 2017. Retrieved: 18th Sept.

[Vec17b] Vector. Example of an AUTOSAR Webinar. https://vector.com/portal/medien/cmc/events/Webinars/2015/Vector_Webinar_AUTOSAR_Introduction_20150505_EN.pdf, 2017. Retrieved: 18th Sept.

[Vec17c] Vector. osCAN. https://vector.com/vi_oscan_en.html, 2017. Retrieved: 18th Sept.

[Ves07] Steve Vestal. Preemptive scheduling of multi-criticality systems with varying degrees of execution time assurance. In *28th Proceedings of Real-Time Systems Symposium (RTSS)*, pages 239–243. IEEE, 2007.

[VS15] S. Venugopalan and O. Sinnen. ILP Formulations for Optimal Task Scheduling with Communication Delays on Parallel Systems. *IEEE Transactions on Parallel and Distributed Systems*, 26(1):142–151, Jan 2015.

[WDR11] Md Tawhid Bin Waez, Juergen Dingel, and Karen Rudie. Timed automata for the development of real-time systems. *Research Report 2011-579, Queen's University-School of Computing*, 2011.

[WW07] C. B. Watkins and R. Walter. Transitioning from federated avionics architectures to integrated modular avionics. In *2007 IEEE/AIAA 26th Digital Avionics Systems Conference*, pages 2.A.1–1–2.A.1–10, Oct 2007.

[XHL14] L. D. Xu, W. He, and S. Li. Internet of Things in Industries: A Survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, Nov 2014.

[YFR] YFRobot. BalanceRobot - Inverted Pendulum Platform based on STM32F103 Microcontroller. `http://yfrobot.com/forum.php?mod=viewthread&tid=11896`, Retrieved: 22nd Sept, 2017.

[YG09] Jin-yong Yin and Guo-chang Guo. An algorithm for scheduling aperiodic real-time tasks on a static schedule. In *Second International Conference on Information and Computing Science (ICIC)*, volume 1, pages 70–74. IEEE, 2009.

[ZGSC14] Licong Zhang, D. Goswami, R. Schneider, and S. Chakraborty. Task- and Network-level Schedule Co-Synthesis of Ethernet-based Time-triggered Systems. In *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 119–124, Jan 2014.

Summary

In real-time systems, the time of delivery of a result is often just as important as its value. Such systems focus on providing guarantees for delivery of certain services before their deployment. The extent to which these guarantees are ensured depends on the application, since every fault in the system might not lead to a safety hazard. Therefore, a statistical study of system related faults and hazards is performed, which is used to identify verification and validation techniques to evaluate guarantees for a system service, as defined by (often application dependent) industrial standards.

Due to widespread use of real-time systems in safety critical applications, advanced and efficient methods are required to provide guarantees for a safety critical service. Often, distributed systems are used to partition functionality or services provided by a complex system on simple processing nodes connected together via a deterministic network. For such systems, there exist two major issues, i.e., valid generation of system configurations during development and adaptivity to changing system states or environmental conditions after deployment. In this dissertation, we focus on these two issues and contribute by proposing solutions to six problems which are faced during the development and deployment of real-time systems.

In the following, we summarize the contents of each chapter of this dissertation. We also provide an introduction to the problems and their solutions which we propose in this dissertation.

Chapter 1 - Introduction

We start this chapter by an introduction to the terminology used in real-time and cyber-physical systems. We proceed with an explanation of the problems which are faced during the development and deployment of such systems. After that we provide a summary of our contributions and present the outline for this dissertation.

Chapter 2 - Time-Triggered Scheduling and Pruning Techniques

Distributed real-time time-triggered (TT) systems are difficult to design, since they require solving an NP-hard scheduling problem. As the size of a modern distributed system increases linearly, the time required to schedule the system increases exponentially. Consequently, a task-set scheduling process might require years or centuries to

find if a feasible TT schedule exists. Since it might be impractical to spend this much time (perhaps, only to discover that the task-set is infeasible), the scheduler run-time is often assigned a maximum limit. This limit might lead to unschedulability of a task-set by the scheduler within the allowed duration. Therefore, the schedulability of a task-set by a scheduler is directly related to the scheduler run-time and, hence, to its scalability. Over the last few decades, scheduling algorithms have been proposed to reduce TT scheduler run-times. However, these solutions fail to schedule modern large distributed systems. This indicates that the run-time and scalability problem for TT scheduling needs to be addressed, such that the current and future large distributed networks can be scheduled within an acceptable time duration.

In this chapter, we introduce a novel search-tree pruning technique by identifying symmetric search-tree paths which feature same set of response-times for all ready jobs. By implementing an algorithm with avoids these symmetries (by pruning symmetric paths), the run-time of a TT scheduler can be reduced drastically. Moreover, we propose a search optimization technique, which we call *node spawning*, for search-tree based schedulers. In this optimization, instead of generating all immediate nodes of a search-tree node during the traversal of a search-tree, we generate only a sub-set of immediate nodes in increasing or decreasing cost. This optimization results in low overheads when the first few immediate nodes lead to a solution. This can be achieved by designing a good scheduling heuristic.

Based on the pruning and spawning techniques, we propose an implementation of a TT scheduler which is capable of scheduling large task-sets. With thorough evaluation, we demonstrate that our approach is capable of exploring the search-space and leads to feasible TT schedules for large distributed systems within reasonable time.

Chapter 3 - Time-Triggered Ethernet and Scheduling

Time-Triggered Ethernet (TTEthernet) is an extension of Ethernet 802.3, which enables Best-Effort (BE), Rate-Constrained (RC) and Time-Triggered (TT) traffic to coexist in the same network. Distributed systems based on TTEthernet can provide a large variety of services to the network traffic, e.g., low jitter, guaranteed bandwidth, etc. However, this requires TT traffic on each physical link to be scheduled strictly periodic, i.e., each consecutive TT frame belonging to a particular message on a physical link must be separated in time by an exact duration. This problem is known to be NP-complete and, therefore, requires adequate attention, especially for large distributed networks. Another problem with TTEthernet based networks is that the TT traffic is inflexibility, i.e., once a TT schedule for TT traffic is generated, no more changes can be made online to adapt to variations in the environment or the system itself. Such an adaptation is often required in complex or large distributed networks and is often implemented using TT modes. However, neither the TTEthernet standard (SAE AS6802 [SAE11]) nor the hardware developed by the TTEthernet OEMs provide explicit support for this adaptivity, which limits its applicability.

In this chapter, we present a multi-mode scheduler for TT traffic in TTEthernet based distributed networks. Our scheduler is capable of scheduling large systems in a small amount of time. To generate TT schedule for each mode, our approach utilizes (a)

the pruning technique defined in Chapter 2 to schedule tasks on each processing node, and (b) an optimized implementation of the phase generation approach proposed by Marouf and Sorel [MS10] to generate schedule for TT network traffic. In order to enable adaptivity using multiple modes, our scheduler generates a combined scheduling table for all modes by stacking TT scheduling windows. Such scheduling tables are feasible implementations since stacked scheduling tables can be executed on processing nodes by a simple software implementation, while COTS TTEthernet devices can service all stacked TT windows as long as they are a few nanoseconds apart.

We evaluate the scheduling capabilities of our scheduler for single mode with varying network sizes and topologies. The results demonstrate that our scheduler generates TT scheduling tables even for huge network sizes within a few seconds. To evaluate multi-mode scheduling capability, we compare capabilities of our stacking scheduler with a non-stacking approach. The results clearly indicate dominance of our stacking approach in terms of schedulability and run-time.

Chapter 4 - Online Admission of Aperiodic tasks

In safety critical systems, time-triggered model of computation is often used (due to ease of certification), where event-triggered activities are modeled using aperiodic tasks. However, the approaches used to guarantee aperiodic task execution in such systems (i.e., aperiodic servers or periodic tasks) often lead to utilization loss, high jitter and large response-times. Recently, Lackorzyński et al. [LWVH12] demonstrated that aperiodic servers or periodic tasks for servicing aperiodic load might aggravate utilization loss issue in hypervisor based partitioned systems. These problems can be solved by utilizing online aperiodic job admission algorithms, however, these algorithms have problems of their own, e.g., no offline guarantees for job execution, require change in the verification and validation process, no support for non-preemptive execution, partitioning or industrial mixed-criticality task model.

In this chapter, we propose an algorithm for online admission of aperiodic jobs which provide offline guarantees similarly to aperiodic servers or periodic tasks such that the certification process need not be modified. Moreover, our algorithm reduces the utilization loss, can be implemented on non-preemptive industrial mixed-criticality systems, supports a wide variety of hypervisors, provides control over task jitter and reduces aperiodic task response-times. We evaluate the efficiency of our algorithm on synthetic but practical task-sets and demonstrate that our algorithm schedules more aperiodic jobs compared to best-effort approaches. Furthermore, we evaluate scheduling overheads incurred by our algorithm for a flight control system built on an ARM based platform. These evaluations demonstrate that the online overheads are sufficiently small for implementation of our algorithm in safety critical systems.

Chapter 5 - Mode-Changes and Fault-Tolerance

For a multi-mode distributed system, TT scheduling tables are often large (due to varying task/message periods). When a mode-change request is released, waiting for the end of the remaining scheduling table, before executing a mode-change, might lead

to longer mode-change delays. In order to reduce mode-change delays and maintain data consistency, mode-changes can be executed before the end of the scheduling table at safe points in time. For an independent constrained deadline task-set, safe points for executing a mode-change can be trivially defined at time points when no task results in partial execution upon mode-change (e.g., at the end of the hyper-period). However, in the case of mixed-criticality communicating tasks inside resource partitions and on- and off-chip networks, defining this point in time is not as trivial.

In this chapter, we present a novel approach for defining safe time points for mode-change execution in partitioned industrial mixed-criticality environment. Our approach generates mode-change blackout intervals for each mode-change between TT modes and generates intermediate scheduling tables, termed as transition mode schedules [Foh94], between these TT modes. Based on the generated blackout intervals and transition mode schedules, our technique guarantees task-set feasibility during a mode-change and calculates the worst-case mode-change delays. We implement our mode-change analysis and reduction technique in the DREAMS project. For a case study of a safety-critical system, we demonstrate that the developed approach leads to significant improvements in mode-change delays.

Chapter 6 - AUTOSAR Toolchain

Due to ever increasing demand for functionality and complexity of software architectures, real-time applications require new design and development technologies. Other than fulfilling the technical requirements, the main features required from such technologies include scalability, re-usability and modularity. These features allow low cost of certification in terms of both time and money. AUTOSAR is one of these technologies in automotive industry, which defines an open standard for software architecture of a real-time operating system. However, being an industrial standard, the available proprietary tools do not support model extensions and/or new developments. These limitations prevent the development, evaluation and integration of new algorithms and approaches by third-parties and therefore hinder the software evolution. A solution to this problem is to develop an open-source AUTOSAR toolchain which supports application development and code generation for common modules.

In this chapter, we present an open-source toolchain for AUTOSAR, which encourages automotive software evolution. The toolchain consists of a real-time operating system core with a low cost System-On-Module (SOM) support, a tool based on a graphical user interface (GUI) for automotive application development and a GUI tool for system configuration and code generation. Our toolchain provides code generators for a large number of modules including run-time environment (RTE), communication manager (ComM) and protocol data router (PduR). To enable easy code generation, our toolchain also provides a code generation framework and an extension of the AUTOSAR meta-model which enables it to connect with third-party toolchains. To exhibit the capabilities of the toolchain, we present two case studies (i.e., inverted pendulum control via LIN bus and software-based TTEthernet end-system) and demonstrate that our toolchain generates valid artifacts and supports automotive application design.

Chapter 7 - Real-Time Scheduling & Analysis Framework

Modern real-time systems are evolving day by day in order to cope with complex constraints and, at the same time, reducing the design, development and running cost. Moreover, the hardware platforms are also improving in order to increase determinism for real-time application development. Such diverse and evolving system designs require scheduling and analysis tools for verification and validation of their performance. Since a scheduling or analysis technique is usually tailored for a specific system (responsible for handling a specific set of constraints), rapid-prototyping of scheduling and analysis tools is required. In order to solve this issue, a number of scheduling and analysis frameworks were proposed over the last few decades. However, these frameworks focus on a specific class of applications and, therefore, only consider a sub-set of real-time task constraints.

In this chapter, we present a modular cross-platform framework for design and development of real-time scheduling and analysis tools; we call this framework GAIA. GAIA enables rapid-development of new scheduling and analysis algorithms for modern and the next generation of real-time systems. GAIA provides a rich set of development components (e.g., parsers, file generators, logger, visualizers, etc.) to facilitate rapid-prototyping and -development of scheduling and analysis algorithms. It also provides scalable search algorithms and analysis tools for offline scheduling of distributed, multi- and/or many-core systems. GAIA efficiently utilizes platform resources and allows verification and validation of timely constraints for large complex systems.

Chapter 8 - Conclusion

In this chapter, we summarize the main contribution of this dissertation and present future work.

Appendix A

In this appendix, we present detailed setup and analysis for the case study, which was developed for demonstrating extension capabilities of our AUTOSAR toolchain presented in Chapter 6.

Zusammenfassung

Modellbasiertes Design und adaptives Scheduling von verteilten Echtzeitsystemen

In Echtzeitsystemen ist der Zeitpunkt der Lieferung eines Ergebnisses oft genauso wichtig wie dessen Wert. Solche Systeme konzentrieren sich auf die Bereitstellung von Garantien für die Lieferung bestimmter Dienstleistungen vor dem Einsatz des Systems. Das Ausmaß, in dem diese Garantien gewährleistet sind, hängt von der Anwendung ab, weil nicht jeder Fehler im System zu einem Sicherheitsrisiko führen muss. Daher wird eine statistische Untersuchung von systembedingten Fehlern und Gefahren durchgeführt, die zur Identifizierung von Verifizierungs- und Validierungstechniken zur Bewertung von Garantien für einen Systemdienst verwendet wird, wie durch (häufig anwendungsabhängige) Industriestandards definiert.

Aufgrund der weit verbreiteten Verwendung von Echtzeitsystemen in sicherheitskritischen Anwendungen sind fortschrittliche und effiziente Verfahren erforderlich, um die Funktionalität eines sicherheitskritischen Diensts zu garantieren. Verteilte Systeme werden häufig dazu verwendet, Funktionen oder Dienste zu partitionieren, die von einem komplexen System auf einfachen Verarbeitungsknoten bereitgestellt werden, die über ein deterministisches Netzwerk miteinander verbunden sind. Für solche Systeme gibt es zwei Hauptprobleme: erstens, eine gültige Erzeugung von Systemkonfigurationen während der Entwicklungsphase, und zweitens, eine Anpassung an sich ändernde Systemzustände oder Umgebungsbedingungen während der Laufzeit. In dieser Dissertation konzentrieren wir uns auf diese beiden Themen und tragen dazu bei Lösungen für sechs Probleme vorzuschlagen, die bei der Entwicklung und dem Einsatz von Echtzeitsystemen auftreten.

Im Folgenden fassen wir den Inhalt jedes Kapitels dieser Dissertation zusammen. Wir geben außerdem eine Einführung in die Probleme und deren Lösungen, die wir in dieser Dissertation vorschlagen.

Kapitel 1 - Einleitung

Wir beginnen dieses Kapitel mit einer Einführung in die Terminologie von Echtzeit- und Cyber-Physical-Systemen. Wir stellen die Probleme vor, die bei der Entwicklung und

dem Einsatz solcher Systeme auftreten. Danach geben wir einen Überblick über unsere Beiträge und stellen den Gliederung für diese Dissertation vor.

Kapitel 2 - Zeitgesteuerte Scheduling und Beschneidungstechnik

Verteilte Time-Triggered (TT) Echtzeitsysteme sind schwierig zu entwerfen, da ein NP-Schweres Schedulingproblem gelöst werden müss. Wenn die Größe eines modernen verteilten Systems linear zunimmt, nimmt die für die Scheduling des Systems benötigte Zeit exponentiell zu. Folglich kann die Scheduling eines bestimmtes Task-Set Jahre oder Jahrhunderte benötigen, um herauszufinden, ob ein realisierbarer TT-Schedule existiert. Da es unpraktisch ist so viel Zeit aufzuwenden (evtl. nur um festzustellen, dass das Task-Set nicht ausführbar ist), wird der Laufzeit des Schedulers oft begrenzt. Diese zeitliche Begrenzung kann dazu führen, dass ein Task-Set vom Scheduler innerhalb der zulässigen Dauer nicht mehr Schedulable ist. Daher steht die Schedulability eines Task-Sets durch einen Scheduler in direktem Zusammenhang mit der Laufzeit des Schedulers und somit mit seiner Skalierbarkeit. In den letzten Jahren wurden algorithmen vorgeschlagen, um die Laufzeit des TT-Schedulers zu reduzieren. Diese Lösungen schedulen jedoch keine modernen großen verteilten Systeme. Dies zeigt, dass das Laufzeit- und Skalierbarkeitsproblem für TT-Scheduling angegangen werden muss, sodass die aktuellen und zukünftigen großen verteilten Netzwerke innerhalb einer akzeptablen Zeitdauer geplant werden können.

In diesem Kapitel stellen wir eine neuartige Suchbaumbeschneidungstechnik vor, bei der symmetrische Suchbaumpfade identifiziert werden, bei denen alle ausführungsbereiten Jobs die gleichen Antwortzeiten aufweisen. Durch Implementieren eines Algorithmus mit Vermeidung dieser Symmetrien (durch Beschneiden symmetrischer Pfade) kann die Laufzeit eines TT-Schedulers drastisch reduziert werden. Darüber hinaus schlagen wir für Suchbaumbasierte Scheduler eine Suchoptimierungstechnik vor, die wir *node spawning* nennen. Bei dieser Optimierung erzeugen wir nicht alle unmittelbaren Knoten eines Suchbaumknotens während der Durchquerung eines Suchbaums, sondern erzeugen nur eine Teilmenge von unmittelbaren Knoten mit steigenden oder fallenden Gesamtkosten. Diese Optimierung führt zu geringen Verwaltungsdaten, wenn die ersten Knoten zu einer Lösung führen. Dies kann erreicht werden, indem eine gute Schedulingsheuristik entworfen wird.

Basierend auf den Beschneidungs- und Spawntechniken präsentieren wir eine Implementierung eines TT-Schedulers, der in der Lage ist, TT-Schedules für große Task-Sets zu erstellen. Mit gründlicher Evaluierung demonstrieren wir, dass unser Ansatz in der Lage ist, den Suchraum zu erkunden und innerhalb einer angemessenen Zeit zu brauchbaren TT-Schedules für große verteilte Systeme führt.

Kapitel 3 - Zeitgesteuerte Ethernet und Scheduling

Time-Triggered Ethernet (TTEthernet) ist eine Erweiterung von Ethernet 802.3, das Best-Effort (BE), Rate-Constrained (RC) und Time-Triggered (TT) Datenverkehr im selben Netzwerk koexistieren lässt. Verteilte Systeme, die auf TTEthernet basieren, können dem Netzwerkverkehr eine große Vielfalt von Diensten bereitstellen, z.B. gerin-

gen Jitter, garantierte Bandbreite, usw. Dies erfordert jedoch, dass der TT-Verkehr auf jeder physikalischen Verbindung streng periodisch geplant wird, d.h. zu jedem TT-Frame (zugehört zu einer bestimmten Nachricht) auf einer physikalischen Verbindung muss zeitlich genau getrennt sein. Dieses Problem ist bekanntermaßen NP-vollständig und erfordert daher eine angemessene Beachtung, insbesondere für große verteilte Netzwerke. Ein weiteres Problem bei TTEthernet-basierenden Netzwerken besteht darin, dass der TT-Verkehr unflexibel ist, d.h. sobald ein TT-Schedule für TT-Verkehr erzeugt wurde, können keine weiteren Änderungen während der Laufzeit vorgenommen werden, um sich an Variationen in der Umgebung oder dem System selbst anzupassen. Eine solche Anpassung wird oft in komplexen oder großen verteilten Netzwerken benötigt und wird unter Verwendung von TT-Modi implementiert. Weder der TTEthernet-Standard (SAE AS6802 [SAE11]) noch die von den TTEthernet-OEMs entwickelte Hardware bieten jedoch explizite Unterstützung für diese Adaptivität, was ihre Anwendbarkeit einschränkt.

In diesem Kapitel stellen wir einen Multi-Mode-Scheduler für TT-Verkehr in TTEthernet-basierten, verteilten Netzwerken vor. Unser Scheduler ist in der Lage, große Systeme in kurzer Zeit zu schedulen. Um eine TT-Schedule für jeden Modus zu erzeugen, verwendet unser Ansatz (a) die in Kapitel 2 definierte Bereinigungstechnik, um Aufgaben auf jedem Verarbeitungsknoten zu schedulen, und (b) die optimierte Implementierung des Phasenerzeugungsansatzes von Marouf und Sorel [MS10], um eine Schedule für den TT-Netzwerkverkehr zu erzeugen. Um die Adaptivität über mehrere Modi zu ermöglichen, generiert unser Scheduler eine kombinierte Schedule für alle Modi, indem TT-Schedulingwindows gestapelt werden. Solche Schedules sind plausibele Implementierungen, da gestapelte Schedules auf Verarbeitungsknoten mithilfe einer einfachen Softwareimplementierung ausgeführt werden können, während COTS TTEthernet-Geräte alle gestapelten TT-Fenster bedienen können, solange sie einige Nanosekunden voneinander entfernt sind.

Wir evaluieren die Fähigkeit unseres Schedulers für den Einzelmodus mit unterschiedlichen Netzwerkgrößen und Topologien. Die Ergebnisse zeigen, dass unser Scheduler innerhalb weniger Sekunden TT-Schedules, auch für große Netzwerkgrößen, generiert. Um das Multi-Modus-Scheduling zu evaluieren, vergleichen wir die Fähigkeiten unseres Stacking-Schedulers mit einem Non-Stacking-Ansatz. Die Ergebnisse zeigen deutlich die Dominanz unseres Stacking-Ansatzes in Bezug auf Schedulability und Laufzeit.

Kapitel 4 - Online-Zulassung von aperiodischen Tasks

In sicherheitskritischen Systemen wird häufig ein Time-Triggered Berechnungsmodell verwendet (aufgrund der einfachen Zertifizierung), bei dem Event-Triggered Aktivitäten mit aperiodischen Tasks modelliert werden. Die Ansätze, die verwendet werden, um eine aperiodische Taskausführung in solchen Systemen zu gewährleisten (d.h. aperiodische Server oder periodische Tasks), führen jedoch häufig zu Nutzungsverlusten, hohem Jitter und großen Antwortzeiten. Lackorzyński et al. [LWVH12] haben gezeigt, dass aperiodische Server oder periodische Tasks zur Bedienung aperiodischer Tasks das Problem des Nutzungsverlusts in Hypervisor-basierten partitionierten Systemen verstärken können. Diese Probleme können durch die Verwendung online-aperiodischer Jobzulassungsalgo-

rithmen gelöst werden. Diese Algorithmen haben jedoch eigene Probleme, z.B. keine Offline-Garantien für die Jobausführung, notwendige Modifikationen im Verifikations- und Validierungsprozess, keine Unterstützung für nicht-präemptive Ausführung, Partitionierung oder das industrielles Mixed-Criticality Taskmodell.

In diesem Kapitel schlagen wir einen Algorithmus für die Online-Zulassung von aperiodischen Jobs vor, der Offline-Garantien ähnlich wie aperiodische Server oder periodische Tasks bereitstellt, sodass der Zertifizierungsprozess nicht modifiziert werden muss. Darüber hinaus reduziert unser Algorithmus den Nutzungsverlust, kann auf nicht-präemptiven industriellen Mixed-Criticality-Systemen implementiert werden, unterstützt eine Vielzahl von Hypervisoren, bietet Kontrolle über Task-Jitter und reduziert Antwortzeiten des aperiodisches Tasks. Wir evaluieren die Effizienz unseres Algorithmus für synthetische, aber praktische Task-Sets und demonstrieren, dass unser Algorithmus im Vergleich zu Best-Effort-Ansätzen mehr aperiodische Jobs zuläst. Darüber hinaus evaluieren wir Ausführungskosten, die durch unseren Algorithmus für ein Flugsteuerungssystem entstehen, das auf einer ARM-basierten Plattform basiert. Diese Auswertungen zeigen, dass die Online-Verwaltungsdaten für die Implementierung unseres Algorithmus in sicherheitskritischen Systemen ausreichend gering sind.

Kapitel 5 - Modusänderung und Fehlertoleranz

Für ein verteiltes Multimodesystem sind TT-Schedules häufig groß (aufgrund unterschiedlicher Task-/Nachrichtenperioden). Wenn eine Moduswechselanforderung erzeugt, kann das Warten auf das Ende der verbleibenden Schedule vor der Durchführung einer Modusänderung zu längeren Verzögerungen führen. Um Moduswechselverzögerungen zu reduzieren und die Datenkonsistenz beizubehalten, können Modusänderungen vor dem Ende der Schedule zu sicheren Zeitpunkten ausgeführt werden. Für ein unabhängiges Task-set mit implizit Frist können sichere Punkte zum Ausführen einer Modusänderung zu Zeitpunkten trivial definiert werden, wenn kein Task zu einer teilweisen Ausführung bei einer Modusänderung führt (z.B. am Ende der Hyperperiode). Im Fall von Kommunikationstasks mit Mixed-Criticality innerhalb von Ressourcenpartitionen und On- und Off-Chip-Netzwerken ist die Identifizierung dieses Zeitpunkts jedoch nicht trivial.

In diesem Kapitel stellen wir einen neuartigen Ansatz vor, um sichere Zeitpunkte für die Modenumwandlung in einer partitionierten industriellen Umgebung mit Mixed-Criticality zu definieren. Unser Ansatz erzeugt Modusänderungs-Blackout-Intervalle für jeden Moduswechsel zwischen TT-Modi und erzeugt zwischen diesen TT-Modi Zwischenplanungstabellen, die als Transition-Mode-Schedule bezeichnet werden [Foh94]. Basierend auf den generierten Blackout-Intervallen und den Transition-Mode-Schedule garantiert unsere Technik ein Feasibility während einer Modusänderung und berechnet die Worstcasemodusänderungsverzögerungszeiten. Wir implementieren unsere Modusänderungsanalyse- und -reduzierungstechnik im DREAMS-Projekt. Für eine Fallstudie eines sicherheitskritischen Systems zeigen wir, dass der entwickelte Ansatz zu signifikanten Verbesserungen der Modusänderungsverzögerungen führt.

Kapitel 6 - AUTOSAR Toolchain

Aufgrund der ständig steigenden Nachfrage nach Funktionalität und Komplexität von Softwarearchitekturen erfordern Echtzeitanwendungen neue Design- und Entwicklungstechnologien. Abgesehen von der Erfüllung der technischen Anforderungen umfassen die Hauptmerkmale, die von solchen Technologien gefordert werden, Skalierbarkeit, Wiederverwendbarkeit und Modularität. Diese Merkmale ermöglichen niedrige Kosten für die Zertifizierung in Bezug auf Zeit und Geld. AUTOSAR ist eine dieser Technologien in der Automobilindustrie, die einen offenen Standard für die Softwarearchitektur eines Echtzeitbetriebssystems definiert. Da es sich jedoch um einen Industriestandard handelt, unterstützen die verfügbaren proprietären Tools keine Modellerweiterungen und/oder Neuentwicklungen. Diese Einschränkungen verhindern die Entwicklung, Bewertung und Integration neuer Algorithmen und Ansätze durch Dritte und behindern somit die Softwareentwicklung. Eine Lösung für dieses Problem ist die Entwicklung einer Open-Source-AUTOSAR-Toolchain, die die Anwendungsentwicklung und Codegenerierung für gängige Module unterstützt.

In diesem Kapitel stellen wir eine Open-Source-Toolchain für AUTOSAR vor, die die Entwicklung von Automobilsoftware fördert. Die Toolchain besteht aus (i) einem Echtzeitbetriebssystemkernels mit günstigen System-On-Module (SOM), (ii) ein Tool auf der Basis eine grafische Benutzeroberfläche (GUI) für die Unterstützung des Automobil-Anwendungsentwicklungs und (iii) ein GUI-Tool für die Systemkonfiguration und Codeerzeugung. Unsere Toolchain bietet Code-Generatoren für eine große Anzahl von Modulen, einschließlich des Run-Time Environment (RTE), Communication Manager (ComM) und Protocol Data Unit Router (PduR). Um eine einfache Code-Generierung zu ermöglichen, bietet unsere Toolchain auch ein Code-Generierungs-Framework und eine Erweiterung des AUTOSAR-Metamodells, das ermöglicht, sich mit Toolchains von Drittanbietern zu verbinden. Um die Fähigkeiten der Toolchain darzulegen, stellen wir zwei Fallbeispiele vor (d.h. Umkehrpendelsteuerung über LIN-Bus und ein softwarebasiertes TTEthernet Endsystem) vor und zeigen, dass unsere Toolchain gültige Artefakte erzeugt und Automobilanwendungsdesign unterstützt.

Kapitel 7 - Echtzeitscheduling und Analyse Framework

Moderne Echtzeitsysteme entwickeln sich Tag für Tag weiter, um komplexe Einschränkungen zu bewältigen und gleichzeitig die Kosten für Design, Entwicklung und Betrieb zu reduzieren. Darüber hinaus verbessern sich auch die Hardware-Plattformen, um den Determinismus für die Echtzeit-Anwendungsentwicklung zu erhöhen. Solche unterschiedlichen und sich entwickelnden Systemdesigns erfordern Schedulings- und Analysetools zur Verifizierung und Validierung ihrer Leistung. Da eine Schedulings- oder Analysetechnik normalerweise auf ein bestimmtes System zugeschnitten ist (verantwortlich für die Handhabung bestimmter Randbedingungen), ist ein Rapid-Prototyping von Schedulings- und Analysetools erforderlich. Um dieses Problem zu lösen, wurde in den letzten Jahrzehnten eine Reihe von Scheduling- und Analyse-Frameworks vorgeschlagen. Diese Frameworks konzentrieren sich jedoch auf eine bestimmte Klasse von Anwendungen und berücksichtigen daher nur eine Untergruppe von Echtzeit-Task-Constraints.

In diesem Kapitel stellen wir ein modulares plattformübergreifendes Framework für Design und Entwicklung von Echtzeit-Scheduling- und Analysetools vor. Wir nennen dieses Framework GAIA. GAIA ermöglicht die schnelle Entwicklung neuer Scheduling- und Analysealgorithmen für moderne und für zukünftige Generation von Echtzeitsystemen. GAIA bietet eine große Auswahl an Entwicklungskomponenten (z.B. Parser, Dateigeneratoren, Logger, Visualisierer, usw.), um das Rapid-Prototyping und die Entwicklung von Scheduling- und Analysealgorithmen zu erleichtern. Es bietet auch skalierbare Suchalgorithmen und Analysetools für das Offline-Scheduling von verteilten, Mehr- und/oder Vielcoresystemen. GAIA nutzt Plattformressourcen effizient und ermöglicht die Überprüfung und Validierung von Zeiteinschränkungen für große, komplexe Systeme.

Kapitel 8 - Fazit

In diesem Kapitel fassen wir den Hauptbeitrag dieser Dissertation zusammen und geben einen Ausblick auf zukünftige Arbeiten.

Anhang A

In diesem Anhang zeigen wir eine detaillierte Analyse der Fallstudie, die entwickelt wurde, um die Erweiterungsfähigkeiten unserer AUTOSAR-Toolchain zu demonstrieren, welche im Kapitel 6 vorgestellt wurden.

Ali Abbas Jaffari **Syed**

Real-Time Systems Engineer and Researcher

Ali Syed (https://www.linkedin.com/in/ali-syed-532634138/)
syed.ali.abbas.jaffari AT gmail DOT com (Private)
+49 (0)631 205 3129 (Office)
Erwin-Schrödinger-Str. 12-519, 67663, Kaiserslautern (Office)

Work Experience

July 2014 – Now
Wissenschaftlicher Mitarbeiter, PhD Candidate.
Chair of Real-time Systems, Technische Universität Kaiserslautern, Germany
- Engaged in research in the field of Real-Time Systems
- Lead scheduling task in the EU FP7 project DREAMS (Distributed Real-time Architecture for Mixed Criticality Systems) and collaborated with Thales (TRT, France), ONERA (France), fortiss GmbH (Germany) and RTaW (France)
- Managed cooperative research activities with industrial research partners from SYSGO AG (Germany), TTTech Computertechnik AG (Austria), RISE SICS AB (Sweden)

Nov 2011 – May 2014
Studentischer Hilfswissenschaftler.
Chair of Real-time Systems, Technische Universität Kaiserslautern, Germany
- Collaborated with Lunds Universitet (Sweden) for development and implementation of time-triggered scheduling, slot shifting and gravitation task model in TrueTime simulator
- Developed scheduling and analysis framework for real-time systems
- Developed task-set generator for creating synthetic real-time workloads

Nov 2012 – May 2013
Studentischer Hilfswissenschaftler.
Chair of Integrated Sensor Systems, Technische Universität Kaiserslautern, Germany
- Developed gesture recognition system using artificial neural networks in SmartKitchen project

Jul 2010 – May 2011
Assistant Manager and Researcher, *Department of Research and Development*.
Alsons Auto-Parts Private Limited, Karachi, Pakistan
- Developed non-destructive testing system for automotive parts
- Designed computer aided parameter evaluation system for speedometers

Nov 2008 – Jul 2010
Control & Automation Engineer, *Department of Development and Automation*.
Flame Engineering Private Limited, Karachi, Pakistan
- Automation of fabric dyeing machine @ Fazal Sardar Textile Mill, Karachi
- Automation of gas chromatography machine @ PCSIR laboratories, Karachi
- Automation & instrumentation of impact testing machine @ Mehran University, Hyderabad

Education

Jul 2014 – May 2018
PhD in Electrical Engineering.
Research Interests: Real-Time Scheduling, Adaptive Real-Time Algorithms, Model-based Engineering, Automotive Software Architecture
Thesis Title: Model-Based Design and Adaptive Scheduling of Distributed Real-Time Systems
Chair of Real-Time Systems, Technische Universität Kaiserslautern, Kaiserslautern, Germany

Aug 2011 **Master of Science in Electrical & Computer Engineering**.
–May 2014 Track: Embedded Systems, Note 1.3,
Thesis Title: Framework for Offline Scheduling of Real-time Systems
Technische Universität Kaiserslautern, Kaiserslautern, Germany

Jan 2006 **Bachelor of Engineering**.
–Feb 2010 Track: Industrial Electronics, 81.73%,
Thesis Title: Process Control System using Fieldbus Interface
IIEE, NED University of Engineering & Technology, Karachi, Pakistan

Teaching Experience

Jul 2014 **Technische Universität Kaiserslautern, Kaiserslautern, Germany**.
–now Supervised and designed experiments for real-time systems laboratory,
Supervised master theses and student Hilfswissenschaftler,
Delivered exercises for courses Real-Time Systems I & II and Operating Systems

2007–2009 **City of Knowledge, Institute of Engineering Studies, Karachi, Pakistan**.
Delivered lectures on Microprocessor Architecture, Programmable Logic Controllers and Programming Principles

2008–2009 **Malir Polytechnic Institute, Karachi, Pakistan**.
Delivered lectures on Microcontroller Programming and Microprocessor Principles