Modeling and Analysis of Automotive Cyber-physical Systems

Formal Approaches to Latency Analysis in Practice

Max Jonas Friese

Dissertation
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)
der Technischen Fakultät
der Christian-Albrechts-Universität zu Kiel
eingereicht im Jahr 2020

Kiel Computer Science Series (KCSS) 2021/02 dated 2021-03-30

URN:NBN urn:nbn:de:gbv:8:1-zs-00000373-a5
ISSN 2193-6781 (print version)
ISSN 2194-6639 (electronic version)

Electronic version, updates, errata available via https://www.informatik.uni-kiel.de/kcss

The author can be contacted via maxfriese@posteo.de

Published by the Department of Computer Science, Kiel University

Dependable Systems Group

Please cite as:

▻ Max J. Friese. *Modeling and Analysis of Automotive Cyber-physical Systems* Number 2021/02 in Kiel Computer Science Series. Department of Computer Science, 2021. Dissertation, Faculty of Engineering, Kiel University.

```
@book{Friese21,
  author    = {Max J. Friese},
  title     = {Modeling and Analysis of Automotive Cyber-physical Systems},
  publisher = {Department of Computer Science, Kiel University},
  year      = {2021},
  number    = {2021/02},
  doi       = {10.21941/kcss/2021/2},
  series    = {Kiel Computer Science Series},
  note      = {Dissertation, Faculty of Engineering,
               Kiel University.}
}
```

© 2021 by Max J. Friese

About this Series

The Kiel Computer Science Series (KCSS) covers dissertations, habilitation theses, lecture notes, textbooks, surveys, collections, handbooks, etc. written at the Department of Computer Science at Kiel University. It was initiated in 2011 to support authors in the dissemination of their work in electronic and printed form, without restricting their rights to their work. The series provides a unified appearance and aims at high-quality typography. The KCSS is an open access series; all series titles are electronically available free of charge at the department's website. In addition, authors are encouraged to make printed copies available at a reasonable price, typically with a print-on-demand service.

Please visit `http://www.informatik.uni-kiel.de/kcss` for more information, for instructions how to publish in the KCSS, and for access to all existing publications.

1. Gutachter:	Prof. Dr. Dirk Nowotka Christian-Albrechts-Universität Kiel
2. Gutachter:	Prof. Dr. Martin Leucker Universität zu Lübeck

Datum der mündlichen Prüfung: 18.12.2020

Zusammenfassung

Basierend auf neuen Erkenntnissen aus der Scheduling-Analyse entwickelte sich in den 1970er Jahren mit der formalen Ende-zu-Ende Latenzanalyse in Echtzeitsystemen ein neues Forschgungsgebiet. Obgleich verschiedene Ansätze erfolgreich in der Praxis Anwendungen gefunden haben, ist eine Lücke zwischen den Möglichkeiten, die die formale Analyse bietet und dem Bedarf der Automobilindustrie entstanden, da hier cyber-physische Systeme die klassischen eingebetteten Systeme zunehmend abgelöst haben. Diese fußen auf der Vernetzung eingebetteter Systeme, die zudem heutzutage meist mit Mehrkernprozessoren ausgestattet sind. Der Einsatz cyber-physischer Systeme hat zur Folge, dass etablierte wissenschaftliche Modelle und Methoden nicht mehr mächtig genug sind. Weiterhin wurden wichtige Ende-zu-Ende Latenzen lange nicht präzise genug abgeschätzt. Aus diesem Grund wird in dieser Arbeit ein umfassendes formales Modell vorgestellt, das die Basis für präzise formale Abschätzungen von Ende-zu-Ende Latenzen in modernen automobilen cyber-physischen Systemen ist. Ansätze für die Abschätzung von Ende-zu-Ende Latenzen werden in Kapitel 4 und Kapitel 5 dieser Arbeit vorgestellt. Die vorgestellten Ansätze unterstützen entsprechend eine Vielzahl relevanter Systemkonfigurationen. Der für die Abschätzung der Latenzzeiten von Task-Ketten vorgeschlagene Ansatz weist dabei eine höhere Präzision auf als bisher vorgestelle Verfahren. Im letzten Kapitel findet ein Exkurs in die Messdatenauswertung statt, da Messungen und Simulationen in der heutigen Praxis wichtige Verifikationswerkzeuge sind.

Abstract

Based on advances in scheduling analysis in the 1970s, a whole area of research has evolved: formal end-to-end latency analysis in real-time systems. Although multiple approaches from the scientific community have successfully been applied in industrial practice, a gap is emerging between the means provided by formally backed approaches and the need of the automotive industry where cyber-physical systems have taken over from classic embedded systems. They are accompanied by a shift to heterogeneous platforms build upon multicore architectures. Scientific techniques are often still based on too simple system models and estimations on important end-to-end latencies have only been tightened recently. To this end, we present an expressive system model and formally describe the problem of end-to-end latency analysis in modern automotive cyber-physical systems. Based on this we examine approaches to formally estimate tight end-to-end latencies in Chapter 4 and Chapter 5. The developed approaches include a wide range of relevant systems. We show that our approach for the estimation of latencies of task chains dominates existing approaches in terms of tightness of the results. In the last chapter we make a brief digression to measurement analysis since measuring and simulation is an important part of verification in current industrial practice.

Acknowledgements

I would like to thank my supervisor, Dirk Nowotka, for his guidance and practical advice, as well as for the encouraging words.

Furthermore, I thank the members of my examining committee, Reinhard von Hanxleden, Reinhard Koch, and Martin Leucker for their efforts and the positive atmosphere during the oral defense. Additional thanks to Martin Leucker for providing the second assessment.

I want to thank my colleagues of team *PLE MBOS@PT& E/E Plattform-Technologie* and former team *E/E Topologien und Wirkketten* at Mercedes-Benz AG for creating a great working environment and the fruitful discussions. Especially I want to thank Hannes Walz for his pragmatic support.

Thanks to my former and current colleagues of the *Dependable Systems Group* in Kiel. It was always nice to visit because I always felt welcome. Thanks to Pamela for all the cake.

I thank my family: my parents for raising me and giving me the chance to enjoy a decent education, and my siblings Marie and Felix for being the wonderful persons they are. I would like to thank Sarah's family for the support and for being genuinely interested in my work.

Finally, I come to my personal acknowledgements: I would like to thank Mitja — without him I would not have started this work, Thorsten — without him I would not have continued this work successfully, and Sarah — without her I would not have finished this work. Her heartfelt and loving support kept me going through the tough times, especially for the final sprint towards the finish line while moving and living in a pandemic. Thanks for taking this 4 years, long work hours, up- and downhill journey through my tiny part of the world of computer science with me although it had this cumbersome, 760km long approach way almost all over of Germany. ♡

Contents

List of Figures

List of Tables

List of Algorithms

Listings

List of Acronyms

AC	Alternating current
ADAS	Advanced driver-assistance system
API	Application programming interface
ASW	Application software
AUTOSAR	Automotive Open System Architecture
BCET	Best-case execution time
BCRT	Best-case response time
BET	Bounded execution time
BSW	Basic software
CAN	Controller area network
CAN-FD	CAN with Flexible Data-rate
CET	Core execution time
COM	Communication
CP	Constraint programming
CPA	Compositional performance analysis
CPCM	Central powertrain control module
CPS	Cyber-physical system
CSE	Chain Segment Expression
CSMA	Carrier-sense multiple access

List of Acronyms

CSP	Constraint satisfaction problem
DC	Direct current
DSE	Design space exploration
E2EE	End-to-end encryption
ECM	Egine control module
ECU	Electronic control unit
E/E	Electrical/electronic
EMI	Electromagnetic interference
FM	Formal methods
GUI	Graphical user interface
HIL	Hardware in the loop
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IL	Interaction layer
IP	Internet Protocol
I-PDU	Interaction layer package data unit
IpduM	I-PDU multiplexer
ISO	International Organization for Standardization
LET	Logical execution time
LIDAR	Light detection and ranging
LIN	Local interconnect network
MDA	Measurement data analysis

MOST	Media oriented systems transport
NCD	Network communication design
OEM	Original equipment manufacturer
OS	Operating system
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
OSI	Open Systems Interconnection
PDU	Package data unit
POSIX	Portable Operating System Interface
RADAR	Radio detection and ranging
RTC	Real-time Calculus
RTE	Runtime environment
RTOS	Real-time operating system
SDU	Sevice data unit
SecOC	Secure onboard communication
SIL	Software in the loop
SOA	Service-oriented architecture
STL	Signal temporal logic
TCM	Transmission control module
TCP	Transmission Control Protocol
TMDA	Time-division multiple access
TeSSLa	Temporal stream-based specification language
TIMEX	Timing Extensions

TRE	Timed regular expressions
TTCAN	Time-tiggered CAN
UDP	User Datagram Protocol
VDX	Vehicle Distributed eXecutive
VFB	Virtual functional bus
VLAN	Virtual local area network
WCET	Worst-case execution time
WCRT	Worst-case response time

Introduction

A wide range of automotive and industrial systems have strong requirements on correct behavior in order to avoid annoying flaws or even accidents. Nowadays, such systems are classified as distributed cyber-physical systems (CPSs). Being an extension of classic embedded computing, CPSs stand for the integration of computing and physical processes in networks of heterogeneous components [86, 73]. Consequently, functionalities provided by the system require the collaboration of different parts of the system. There are two issues which need to be addressed to assess the correct behavior of a CPS with regard to a distributed system function: *how* does the system react and *when* does the system react. The first issue is concerned with the so-called functional correctness of the system: do all possible inputs lead to a functionally correct response? The second question is all about timing and although sometimes treated as being of secondary relevance, for the outcome in many situations it is equally important whether the system reacts in time. If the sensors of an automated car correctly recognize an obstacle on the road but the command to brake does not reach the braking system fast enough, the situation will have the same outcome as if the obstacle got not detected correctly: the car will not be able to avoid a collision. That is why end-to-end latency analysis is an inherent step in building dependable CPS.

Due to the distributed nature of many of today's systems the question whether end-to-end latencies are fulfilled can not be answered by conducting local analyses only. Only looking at how the different parts of the system interact results in a meaningful analysis with tight estimations. Given the importance of these problems, a lot of approaches to tackle this problem have been proposed from the scientific community. Although some approaches have been successfully applied on industrial use cases, a gap exists between industry demands and available solutions. It must also be pointed out that tool support for the task of latency estimation in the development of today's complex automotive systems is indispens-

able. Even the most experienced engineer can not possibly understand the complex interactions of huge distributed systems as it simply exceeds the scope of human understanding. At the same time, electrical/electronic (E/E) architectures of automotive CPSs have made a long evolution [95] and this trend is currently reinforced due to the electrification of powertrains. In addition to that, an increasing digitalization of the automotive industry demands for faster processes once again [134]. Estimations for latencies have to be available as early as possible to reduce development costs and develop products faster. Therefore it is important to increase the capabilities of formal approaches to provide solutions for this demand.

The aim of this work is to make a huge step towards closing this gap. First, we will define a formal model allowing to encode the problem instances relevant in the automotive industry. Secondly, two approaches are presented to obtain end-to-end latencies. Hence, this work is structured as follows. In Chapter 1 an introduction on how the systems we consider here looked like yesterday, how they look today and how they might look tomorrow, as well as an overview on how they are developed is given. In Chapter 2 the mathematical basis for the following chapters is introduced. Chapter 3 brings together the systems to encode and the formal structures of the previous chapters to create a comprehensive model suiting industrial applications while being completely formal. In Chapter 4 and Chapter 5 instances of this model are taken as an input for an approach to obtain safe and tight upper bounds for different end-to-end latency semantics. Chapter 4 is based on the author's work published in [49] and [50] and the work [75] which is submitted and currently under review. Chapter 5 is based on the work published in [52]. Finally, in Chapter 6 we look at the problem from a different perspective once again: how can end-to-end latencies be determined from data obtained by measurements. The presented specification language was first published in [53].

Chapter 1

Automotive CPS - A biased Overview

In this chapter, the most important characteristics automotive CPS as well as some general industrial practices for their development are described. The requirements imposed on the formal methodology can be derived the characteristics and development practices. The verification process has to cope with different levels of concreteness as implementation details change through the development process. The chapter starts with an overview about the V-model, a common development life cycle in the automotive industry. Subsequently in Section 1.2, two examples for automotive E/E topologies are described and a general outline of the communication design and both hardware and software of ECUs is given. Although some of the topics are worth a book on their own some key aspects are described and put into a historical context.

1.1 Development of Automotive CPS

In this section we take a look at the different stages of automotive development especially with an eye on the role of end-to-end timing analysis through the process. The development of automotive CPS can be divided into three major phases: a concept phase, a serial development phase and a production phase. Each phase poses different challenges for when it comes to the evaluation of the systems timing behavior. In the concept phase, different system designs have to be compared. A comparison requires methods of measuring to obtain indicators for the expected performance. The systematic analysis of such indicators is called design space explo-

ration (DSE) [68]. The complexity of this task tremendously increases when multiprocessor systems and dynamic behavior are involved. To cope with the resulting design space, scenario-based DSE was introduced [101]. An inherent part however is and remains a method to determine fitness values for the different design candidates. Some fitness values, e.g. power consumption or heat dissipation, only require knowledge about local factors of one ECU and can be evaluated individually. End-to-end latencies on the other hand require the well-orchestrated interplay of multiple ECUs. From an overall-system´s point of view, it is therefore necessary to integrate design candidates with different degrees of implementation when it comes to evaluating different system designs. In Chapter 3 we discuss an approach for this challenge which allows to combine different levels of implementation details.

At the end of the concept phase, an initial choice for the hardware is made and the network topology of the overall system is outlined. In the subsequent phase of series development, the network communication design and software-side implementation is put into more concrete terms. Different process models exists to support the development, e.g. the waterfall model, the V-model, the V-model-XT, and the W-Model [66, 117]. The V-model-XT, which was introduced as an extension of the V-model in 2005, is currently the most common process model in the automotive industry [110, 94, 104, 69]. In all V-model-like process models, the development process is subdivided into two phases: a top-down specification, and a bottom up integration process. At the transition of these two phases the implementation of system elements, e.g. software components, is realized. In practice, the overall process is repeated iteratively [104]. The results of the previous iteration are the baseline for the next one. After specifying the requirements for newly added features, the overall set of requirements is amended and broken down to individual requirements.

A visual representation of the process tailored for the development of automotive CPS based on timing requirements is depicted in Figure 1.1. In this tailoring, an iteration starts with the breakdown of end-to-end timing constraints into individual timing constraints for network transmission and data processing within ECUs. Different types of timing constraints need to be considered on system- and component-level. System-level timing constraints apply to end-to-end latencies while component-level

timing constraints refer to the worst-case execution time (WCET) of software components. Accordingly, the verification on the left-hand side starts with checking whether local deadlines are met. The interaction of different components regarding end-to-end timing is examined subsequently. This results in two stages in the process where the estimation of end-to-end latencies needs to be carried out: (1) for checking the specification whether all timing constraints can be met, and (2) for verification of the implementation. Both points are marked with red dots in Figure 1.1.

With the start of production, series development ends and development efforts are focused on the next product. Only if defects occur in the field, development continues for a comprehensive fault analysis and to develop a fault removal. However, these scenarios are obviously not foreseen in a standard development process and methodologies strongly depend on the circumstances.

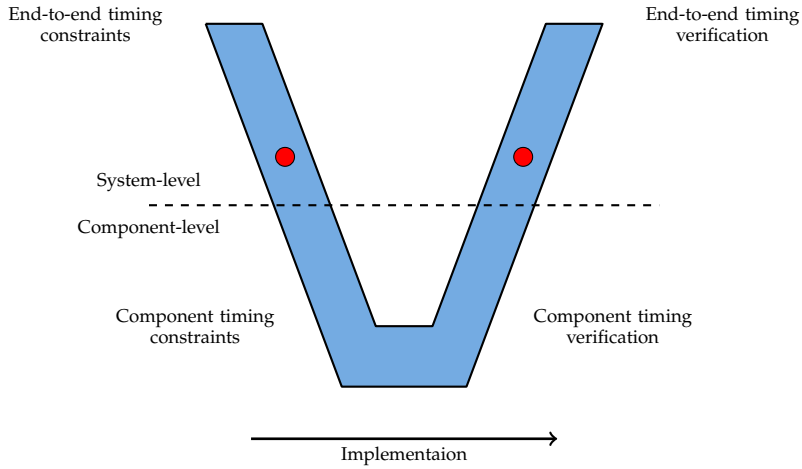


Figure 1.1. End-to-end timing analysis whilst system development based on ISO 26262 (Cf. [122])

1.2 Properties of Automotive CPS

In this section we examine the general properties of automotive CPS. First, we take a broader perspective and look at network topologies usually found in cars and how they will possibly look like in the future. Subsequently, two elements of the communication clusters described by these topologies are examined in more detail: fieldbuses and ECUs. For both of these elements a plethora of different properties need to be considered for a comprehensive analysis of the temporal behavior. The most important aspects for the analysis of networks are the topology of the network and its communication design. Some current and expected future topologies and communication designs are described in Section 1.2.2 and Section 1.2.1 respectively. For ECUs hard- and software properties need to be considered. On the hardware side microcontroller and microprocessors are distinguished. The differences are discussed in Section 1.2.3. On the software side different properties of the real-time operating system (RTOS) as well as the software architecture need to be considered as described in Section 1.2.4.

1.2.1 Communication Design

Similar to the emerging E/E architectures, network communication is changing to cope with the expected complexity and to meet demands in flexibility. Classically, the networks of embedded systems in cars were characterized by signal-based communication which emerged from the switching signals computed by controllers in early systems. In early systems, communication between ECUs was realized with discrete point-to-point wiring [87]. With an increasing amount E/E functionalities however, the sheer weight of the wiring harness limited possibilities. Thus, discrete wiring got replaced by serial fieldbuses [99]. The best known representative of classical fieldbuses is the controller area network (CAN) bus which was introduced in 1986 and developed by Bosch and Intel [84]. as The next major changes in the evolution process of automotive communication came as part of the requirements opposed by the so-called *drive-by-wire* technologies in the late 1990s. The term subsumes the replacement of mechanical or hydraulic components by electrical or electro-mechanical

substitutes [88]. However, since many components where this replacements were implemented are safety critical, e.g. steering and braking, new requirements for the networks in use arose. This particularly concerned communication latencies and jitter which are hard to predict for the original CAN bus due to its event-triggered nature and the priority-based access mechanism [4]. To improve predictability, time-triggered alternatives were developed. Two approaches were followed. Firstly, time-tiggered CAN (TTCAN) defines a session layer extension (see page 8) for CAN which is based on a static schedule [89]. Secondly, the FlexRay protocol was introduced with a time-division multiple access (TMDA) mechanism. In contrast to carrier-sense multiple access (CSMA) where a transmitting node checks for other transmission processes before initiating a transmission attempt, in TMDA, the channel is divided into multiple time slots. Some details and a comparison to other fieldbuses is given in Section 1.2.2. The time-triggered communication systems allow to implement so-called *fail-silent* systems because message losses can be detected if a message is not received in the expected time slot [84, 38, 39]. Alongside reduced jitter and therefore more deterministic communication latencies, the improved detection for message losses laid the foundations to create reliable systems with less mechanical components as proposed in the drive-by-wire concepts.

With an increasing number of functionalities now being implemented in the *cyber* part of the system, the demand for communication capacities continued to increase. In order to make better use of existing communication channels, package data unit (PDU) multiplexing was introduced in the mid 2000s with the AUTOSAR I-PDU multiplexer (IpduM) [12]. This was another step in the direction of more flexible systems as it diluted the statically configured communication in favor of mappings which are decided at run-time. On the downside, less static configuration is inseparably tied to less predictable behavior. New approaches to cope with the arising challenges for end-to-end latency estimation are discussed in Chapter 5.

The next step towards even more dynamic systems is currently made with the implementation of so-called service-oriented architectures (SOAs). However, this time the transformation is imposed by external factors. Technology companies which are testing out opportunities of the automotive market as well as new original equipment manufacturers (OEMs) force traditional car makers to take initiative on digital transformation [55, 133].

One consequence is, that OEMs need to break with architectures grown with the organizational structure [79]. SOAs is expected to help coping with some of the arising challenges. Accordingly, efforts are made to formalize the concepts and improve tool support for architectural tasks [28, 71].

However due to cost and safety constraints in the hard real-time parts of CPSs and because methods to calculate response time guarantees SOAs are still in development, the next generation's architectures will contain a mixture of both worlds. In mechatronic parts of the systems, classic signal-based communication will be used. The backbone of the car on the other hand will be built upon services. Consequently, the main ECUs of each architectural domain need to meet the requirements from both worlds. This is further described in Section 1.2.2.

OSI model

The OSI reference model is a model to describe the abstract structure of communication systems. It divides such systems in several layers and was first published in [114]. Later it was standardized by the ISO in the International Electrotechnical Commission (IEC) standard 7498-1 [123].

In the OSI reference model each layer only communicates with its direct vertical predecessor and successor. More precisely, a layer submits data to be send to and accepts received data from the next lower layer. The data of layer N is the so-called sevice data unit (SDU) of the layer $N + 1$. There it is placed together with the protocol information of layer $N + 1$ to the PDU of that layer.

The classic automotive fieldbuses are concerned with the Physical Layer up to the Data Link Layer [138]. Application software takes over the role of Network Layer and above in this cases. Only recently, in the 2010s, Transport Layer protocols for CAN and FlexRay have been published, the ISO 15765-2 (cf. [124]) for CAN-like buses and the ISO 10681-2 (cf. [125]) for FlexRay to be precise. Ethernet brings the TCP and UDP stacks into the car. Together with the IP protocol these stacks span over Physical, Data Link, Network and Transport Layer.

Table 1.1. Layers of the OSI Reference Model

Layer		Function
7 6 5	Application Layer Presentation Layer Session Layer	Encryption, data compression, session management, and high-level APIs
4	Transport Layer	Segmentation of data possibly state-full and with acknowledgments
3	Network Layer	Addressing of communication nodes and routing of variable length packets
2	Data Link Layer	Transmission of data frames with correction of errors from physical layer
1	Physical Layer	Symbol encoding on the physical medium, e.g. voltages

1.2.2 Network Topologies

The topology of a network is closely linked to the communication design. The complexity of the topologies is one aspect making it harder to conduct a verification of the temporal behavior. Main driver for the complexity is the amount of connected ECUs. Different numbers are circulating: the numbers reach from 100 (cf. [108]) over 150 (EMCC presentation [67]) to around 200 (cf. [126]). Usually, these numbers include all possible variants, meaning that no configuration with all ECUs exists. Furthermore, this number also varies with the market segment of the considered car. The network of a full-size luxury sedan comprises more ECUs than the network of a compact car. However, one thing is uncontested: topologies are steadily evolving and the number of nodes has risen constantly up to the complexity found in today's cars.

In this section, the network interconnecting the nodes automotive communication clusters are described. In the previous section we already

learned about CAN. Local interconnect network (LIN) and media oriented systems transport (MOST) are completing the list of more traditional standards. They were additionally introduced as a cost-saving alternative in the case of LIN and as a multimedia-oriented alternative in the case of MOST.

LIN is primarily used in less performance critical application areas, like e.g. seat control or door locks. A LIN network can have at most 16 nodes. One of the nodes is configured to be the master and works through a static schedule by sending the message identifier for the next message to be send. One node is then responsible to send this message which is received by all other nodes.

MOST was developed to be optimized for applications where the focus lies on bandwidth rather than real-time properties. Typical areas of application therefore include the audio and video players or the navigation system [58]. MOST uses synchronous transmission for efficient transfer rates. Newer versions of MOST also use packet-based communication for control data when the bus is free. MOST is mostly used in ring-topologies where data is transmitted from one device to another.

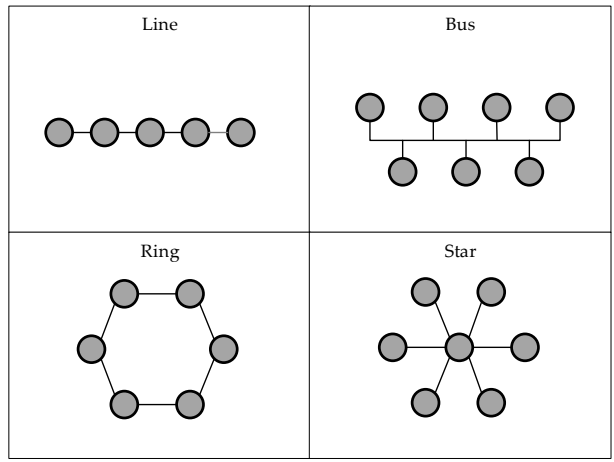


Figure 1.2. Examples of Network Topologies

As indicated in the previous section, the data transfer rates as well as

synchronization mechanisms of the classical fieldbuses did not meet the requirements for implementing advanced electronic control systems in the late 1990s. Different technologies were developed to cope with these challenges. FlexRay and CAN with Flexible Data-rate (CAN-FD) are two of these which prevailed in the automotive industry.

FlexRay was developed between 2000 and 2006 by a partnership of automotive OEMs and two telecommunications companies. The first use in series production was in 2006 [115]. FlexRay was developed aiming for a CAN-alternative with time-triggered communication, high dependability, and an increased data transmission rate. The FlexRay specification therefore allows data rates up to 10 Mbps. Furthermore, the dependability was increased by adding a second physical channel including a synchronization mechanism to support redundancy. On Data Link Layer, FlexRay uses TMDA as a channel access method. In TMDA, transmission is organized in time slots. Each slot has a dedicated message assigned and only one ECU is allowed to send within one slot. A characteristic feature of FlexRay is, that it combines static TMDA and dynamic TMDA. The so-called *Macroperiod* is divided into two segments: a static segment with a fixed number of so-called time slots and a dynamic segment with so-called *Minislots*. After the static and dynamic segment are over, a new Macroperiod starts. Minislots are shorter than the time slots of the static segment, conversely, a message can span over multiple Minislots. In the dynamic segment communication is priority based on the slot count. Each ECU has the right to send at its slot count or increase directly to pass on the next ECU. The slot counts may vary for the different channels but the maximum length of the dynamic segment is determined at design-time. Therefore, low-priority messages of the dynamic segment might postponed to the next Macroperiod. In terms of the network topology, FlexRay supports line and star topologies. ECUs do not need to be connected to both channels, allowing for a different logical topology for each channel.

The CAN-FD technology is newer than FlexRay. It was released in 2012 and has been developed with a strong focus on backward-compatibility due to the prevalence of classic CAN [62]. With CAN-FD a data transfer rate about six times higher can be achieved while also being able to take part in classic CAN communication. To the downside of compatibility, the CAN-FD specification is only for Physical and Data Link Layer and a mechanism to

achieve synchronization is therefore not included.

Alongside the further development of proprietary fieldbuses, Ethernet has found its way into the car. It promises higher bandwidth and less development cost for dedicated hardware as Ethernet is widely used together with the IP and hardware costs already have reduced sharply. The great success of Ethernet in computer networking lets the question arise why it was not adopted for in-car usage before 2008. Not only the cost for hardware capable of participating in Ethernet communication in terms of processing power and available memory was a problem. The physical properties of Ethernet did not meet automotive needs. Two issues existed. Firstly, standard Ethernet was not designed to work within the harsh in-car conditions. Besides temperature and humidity, especially requirements on electromagnetic interference (EMI) disturbance immunity have to be mentioned. These issues have been addressed in the *BroadR-Reach* Physical Layer standard for Ethernet [32]. Secondly, Ethernet follows the so-called *best-effort* approach. This means, the network service does not give any guarantee for bit rate, latency or even packet loss. Therefore, some additional features for the application in real-time environments had to be added. These are in particular: (1) guarantees on latencies, (2) guarantees on free bandwidth, and (3) mechanisms for clock synchronization. Support for these issues has been developed within the IEEE audio/video bridging standard and the *time-triggered* Ethernet technology [22]. However, the real-time application also entails some curtailing of Ethernet, particularly when it comes to network expansion. Although scalability is often listed as an advantage, adding devices to safety-critical networks has to be done with great care [91]. Nevertheless, Ethernet allows for new topologies. Although classical fieldbuses still take a major role, it is likely that Ethernet will play a stronger role in the future. The following sections contain a not purely scientific outlook of how topologies might develop.

Physical Layer technologies which are currently popular in the automotive industry are compared regarding some key metrics in Table 1.2. The information given is based on [127, 62, 138]).

In the following, the general form of current and next generation, as well as expected future E/E architectures are described to give the reader an idea of the systems being the subject to mathematical modeling.

Table 1.2. Automotive fieldbuses in comparison

Bus Technology	Topology	Max. Bit Rate	Frame size	Application (exemplary)
LIN 2.0	Bus with Master/ Slave	20 Kbps	88 bits	Connecting ECUs to sensors/actuators
CAN 2.0	Bus	1 Mbps	109 bits	Chassis electronics
MOST 150	Ring	10 Mbps	384 bits	Transmission of multimedia contents
CAN FD	Bus	8 Mbps	512 bits	Powertrain electronics
FlexRay	Line/Star	10 Mbps	2096 bits	Brake-by-wire, Steer-by-wire
Ethernet	various	100 Mbps	1530 **bytes**	Backbone connecting domain controllers

Automotive CPS - Current Architecture

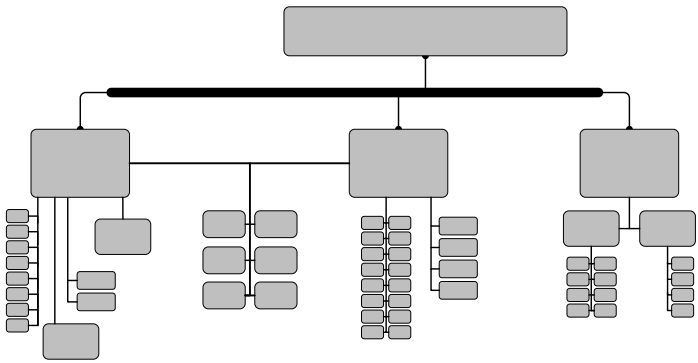


Figure 1.3. Current Automotive E/E Architecture (exemplary)

A general form of an architecture likely to be found in today's middle and upper class cars is depicted in Figure 1.3. It features a gateway handling the connections between central control units of vaguely outline domains (sometimes also called zones). Gateways might be directly connected to multiple other communication clusters where tight interaction is demanded. Some cross-domain connections might be found in around the advanced driver assistance systems which are closely connected to chassis

and powertrain functions.

Automotive CPS - Upcoming Architecture

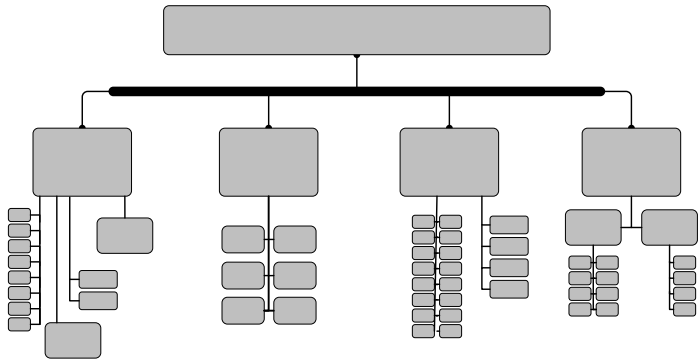


Figure 1.4. Next Generation E/E Architectures (exemplary)

The E/E architectures of cars hitting the market around 2020 are likely to feature stronger outlined domains with a dedicated domain controller each. Presumably, an Ethernet backbone continues to be the choice to connect the domain controllers with the central vehicle controller as depicted in Figure 1.4. Inter-domain communication is presumably implemented with the help of services provided by the domain controllers. For network communication with security requirements, the backbone is separated in logical sub-nets, so-called VLANs. The increased demand for computing power owing to the increased bandwidth of Ethernet and service provision additionally brings new hardware and therefore inevitably new software for the basis system. It is expected that some domain controllers will be switched from microcontrollers to microprocessors to satisfy new performance demands [105]. However, semiconductor companies may not be able to justify huge research efforts for dedicated automotive processors.

This means, general purpose processors witch a wider range of application must be used. For the ECUs within the different domains with their hard real-time requirements, e.g in mechatronic parts of the system, the use of microcontrollers is however still more likely due to their more

predictable behavior. Consequently, the domain controllers decouple the dynamic part of the system from the static parts with high requirements on functional safety.

Automotive CPS - Future Architecture

The exact shape of future architectures is hard to predict. It is however probable that they will contain less dedicated hardware [6, 15]. Instead, some of the functionalities currently implemented with the help of specialized microcontrollers might be virtualized and computed on microprocessor-based platforms. The processing nodes are possibly connected via Ethernet, but more integrated than today, meaning that component sub-systems are combined to one functional system. Some experts even predict full centralized architectures [139]. It is clear, however, that sensors and actuators still need to be connected to the processing hardware and that key mechatronic systems, e.g. anti-lock braking systems, will remain.

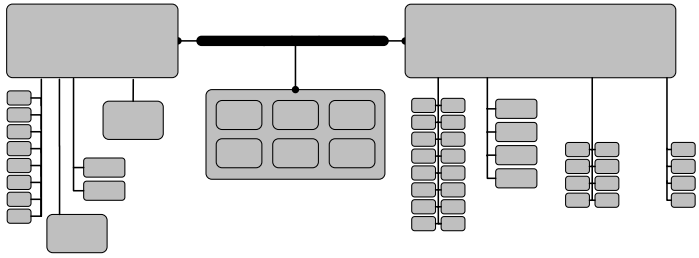


Figure 1.5. Future E/E Architectures (exemplary)

1.2.3 ECU Hardware

The beginning of the widespread use of electronic fuel injection in the 1970s marks the starting point of the successful use of ECUs in cars. Initially, the new possibilities of computing were used to calculate the optimal amount of fuel and achieve better engine performance based on different sensors. The general structure of an ECU is depicted in Figure 1.6.

The applications of ECUs expanded steadily after their introduction. Transmission control modules and telematic units followed shortly after

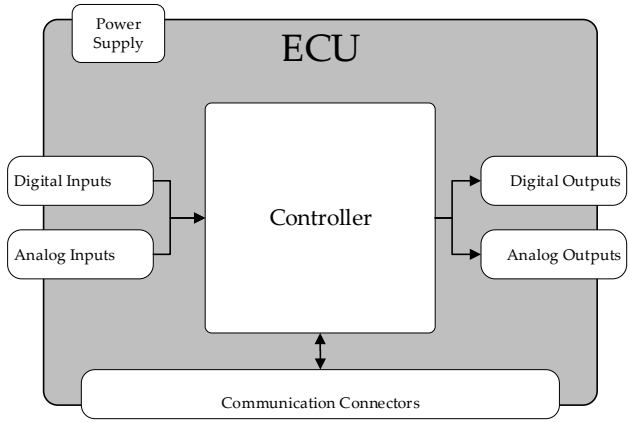


Figure 1.6. Structure of an ECU

the engine control module. The introduction of displays showing basic travel information like average fuel consumption or estimated time of arrival soon raised the demand for distributed computing. Different information needed to be delivered inside the vehicle's cabin from the motor compartment. At the same time, the first driver assistance systems raised questions about dependability of hard- and software. From here on, driver assistance has undergone a continuous development to this day. In parallel, the term *connected car* has been coined, which stands for the connection of the car and devices inside the car to the internet and other capable devices [31]. The connected car has its origin in a safety feature, which automatically calls for help in the case of an emergency. From this starting-point, various features from commerce through the navigation system over well-being and driver comfort to breakdown prevention were added. As a consequence, telematic control units need even more computing power. Microcontrollers integrating a microprocessor with peripheral devices on one system are used less often for this task. Instead, microprocessors are deployed. The difference can be seen when comparing Figure 1.7 and Figure 1.8. Besides the general higher clock speed, microprocessor-based implementations can be better scaled with regards to connected peripherals due to their less dedicated nature. On the

other hand, microcontroller-based solutions are less costly and generally have a lower power consumption. Furthermore, most microcontroller solution feature a dedicated hardware watchdog to detect malfunctions [26]. Another difference is, that microcontrollers are often build following the *Harvard Architecture* while microprocessor-based solutions feature a *Von Neumann* architecture. One major difference is the separation of memory for data and programs in Harvard Archictectures as depicted in Figure 1.7. In accordance with the dedicated function, microcontrollers are used with more specialized RTOSs. Microprocessors on the other hand can be used with embedded operating systems or even lightweight versions of desktop operating systems like known from personal computers.

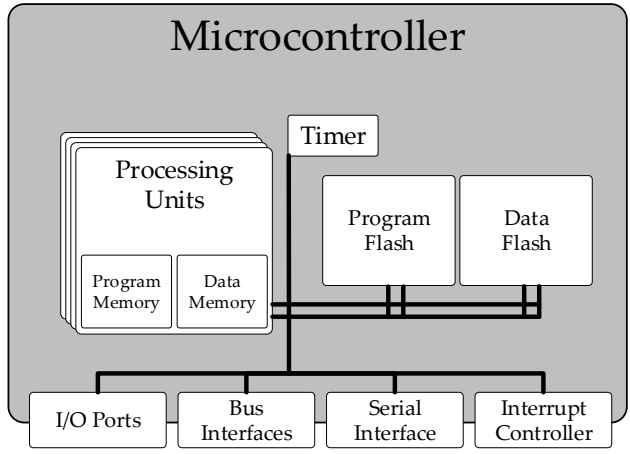


Figure 1.7. Structure of a Microcontroller

The latest and once again significant trend affecting the demand for processing resources in cars are advanced driver-assistance systems (ADASs) [105]. In the context of ADASs, classic microcontroller-based ECUs are replaced by microprocessor-based systems with high computing power and even systems build on manycore processors.

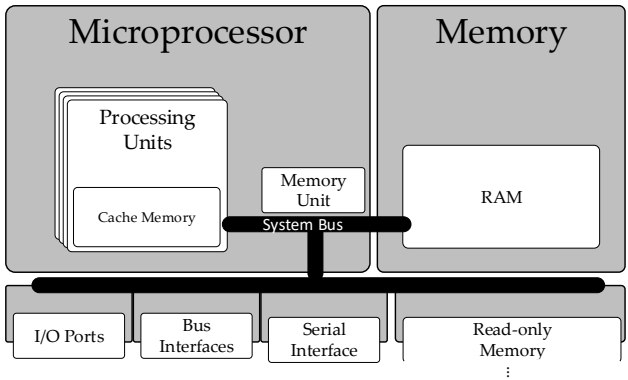


Figure 1.8. Structure of a Microprocessor

Sensors

Sensors translate physical measurands into electrical values. Measured variables come from a wide spectrum of physical values ranging from temperature and pressure over rotational speeds to the strength of magnetic fields and even information about chemical compositions. Lately, for advanced driver assistance functionalities different new sensors entered the car, especially sensors for environment recognition, like cameras ultrasonic sensors, laser scanners and different-range light detection and ranging (LIDAR) and radio detection and ranging (RADAR) detectors.

Two aspects of sensors are important to consider when looking at the dynamic behavior of the system. Firstly, the question whether the sensor samples value periodically or captures a sporadic event, e.g. the press of a button. The second aspect is the the amount of data to be processed. This particularly concerns LIDAR and image data. Generating a three-dimensional model of the environment from point clouds and image data requires the processing of complex models with huge amounts of data [137].

Actuators

Actuators are used to control physical processes, e.g. a throttle valve for regulating the air intake of an engine. Actuators can be differentiated based on two properties: (1) control signal type and (2) source of power. The control signal type can be *digital*, only allowing two points of operation or *analog* allowing the actuator to function at a range, e.g. opening a throttle valve halfway. In the latter case the discrete states of the ECUs controlling signal need to be mapped to a range of voltages. This can either be achieved with the help of *pulse-width modulation* [1] or with the help of a dedicated component called *digital-to-analog converter*.

The most common sources of power are hydraulic or pneumatic pressure and electric current. In modern cars a wide range of electric motors can be found. The area of application starts with small motors used for power windows or sunroofs and extends to propulsion technology in electric vehicles. In Table 1.3 some examples of the different actuator types are listed.

Table 1.3. Examples of different actuator types

Actuator	Control	Power	Area of Application
Directional control valve	Digital	Hydraulic pressure	Automatic transmission
Control valve	Analog	Pneumatic pressure	Brake booster
Electric motor (AC)	Analog	Electric current	Asynchronous motor (driving)
Electric motor (DC)	Digital	Electric current	Power windows

1.2.4 ECU Software

For a long time the development of automotive software was highly interdependent with the development of the hardware it was processed on. As the use of electronics was not widespread in the car, and control algorithms were focused on very specific tasks, software was usually developed in independent monoliths [126]. However, with the connection of multiple ECUs the question arises which functionality is located on which ECU.

[1] The interested reader is referred to [17] for more details.

Another perspective on the software processed on ECUs is looking at the dynamic aspects. This perspective is concerned with the behavior of the system at run-time. In the early days of automotive software no distinction was made between the static and dynamic behavior architecture. As ECUs had single-core processing units, only the frequency of processing and the processing order of the functions had to be decided. Hard real-time processes were able to be implemented with embedded solutions. Over time, activation rates spread more broadly and multi-core ECUs were introduced. As a result, it is practically impossible to directly understand the behavior at run-time anymore. Consequently, scheduling and end-to-end analyses are important tools for engineers to verify correct dynamic behavior.

A typical automotive task set in a time-critical application consists of periodic tasks and sporadic tasks. Sporadic tasks are activated when different events occur, e.g. at sensor readings. A special type of sporadic tasks are angle-synchronous tasks which are activated synchronously to the rotation of the crankshaft. The activation rates of periodic tasks range from one second to one millisecond [78].

Scheduling of such task sets is either done using fixed-priority preemptive scheduling or fixed-priority non-preemptive scheduling. The static priorities of periodic tasks are assigned rate monotonic, meaning that frequent task have a higher priority. Sporadic tasks usually have the highest priorities for fast handling of incoming events.

Lately, a new paradigm for real-time software emerged: LET. The idea of LET was introduced with the time triggered programming language *Giotto* in 2000 [65] and is driven by the observation, that the key aspects of the dynamic behavior of real-time systems are the points time when input is read and output is written. So when these points of time are fix relatively to the start of the schedule, the actual processing in between is not relevant for the temporal behavior as long as deadlines are met. Although the LET idea is discussed contrary, it is gaining more acceptance [74] and has also found its way into automotive systems [64].

Opposing the increased determinism achieved by deploying LET, the above-mentioned shift to more dynamic systems negatively impacts predictability of the systems behavior at run-time. If more decisions are made at run-time the dynamic perspective on the system needs to consider more

possibilities. This aggravates system verification. In short, one could say: more dynamic at run-time often also means less predictability at design-time. In addition to this trend POSIX-based systems with *Automotive Linux* hit the market in 2018 [8]. The targeted use-case in a first step are however in the are of audio and video applications, and rear seat entertainment. In this work we focus on the evolving real-time systems.

1.2.5 OSEK/VDX

The *Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug* (OSEK) [2] is a standards body founded 1993 by different German automotive OEMs and the University of Karlsruhe. In 1994 two French automotive OEMs, which previously grouped together to the VDX, joined the consortium. The OSEK/VDX consortium created a operating system (OS) and communication standard of the same name. Moreover, the OSEK-OS and OSEK-communication (COM) has been standardized by the ISO in the standards [118, 119, 120, 121].

We want to go into more detail on two parts of the standard: (1) the task concept and (2) the interaction layer (IL). The concepts for the tasks as well as the ideas around the IL have been included and further developed in the AUTOSAR standards. As a consequence, they still have a huge relevance in today's automotive embedded systems and will accompany us throughout this work.

The task concept differentiates two types of tasks: *basic* tasks and *extended* tasks which can either be *active* or in *idle*-state. The difference between basic tasks and extended tasks is, that the latter have and additional waiting state which allows to preempt a extended task without actually terminating it. The possible states and state transitions for a extended task are shown in Figure 1.9. The states and transitions for basic tasks are a proper subset of the states and transitions as the *Waiting*-state is omitted. The state automaton for basic tasks is depicted in Figure 1.10. Additionally, mechanism for task activation, switching and scheduling policies are defined which found wide application in the automotive industry.

[2] German for "open systems and their interfaces for the electronics in motor vehicles"

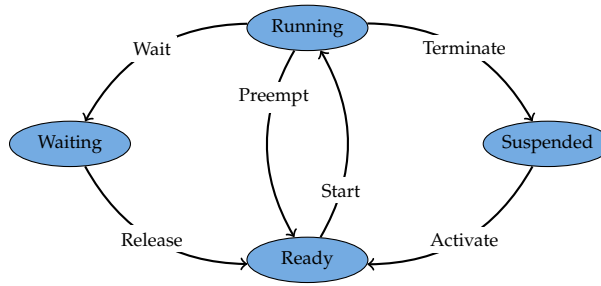


Figure 1.9. States of OSEK/VDX Extended Tasks (Cf. [120])

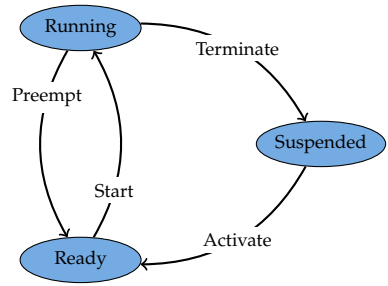


Figure 1.10. States of OSEK/VDX Basic Tasks (Cf. [120])

Just like the task concept, the concept of the IL is the basis for many implementations in the automotive domain. It is message-based and provides a common interface for communication between different applications to abstract from the communication protocol used for the actual transmission. Two kinds of communication are distinguished: (1) *internal* communication and (2) *external* communication. Messages which are sent to internal receivers are directly routed through the IL. Messages which are sent to external receivers are packed into interaction layer package data units (I-PDUs). Furthermore, an external message has a transfer property which is either *triggered* or *pending*. If the transfer property of a message is set to be triggered, the content in the assigned I-PDU is updated and a request for transmission is made. If the transfer property of an I-PDU is set to pending, the message is updated without transmission request. Furthermore, it is

specified that an I-PDU can have one of the following transmission modes: *direct*, *periodic*, or *mixed*. Providing, the transmission mode of an I-PDU is set to direct, the I-PDU is only sent due to a transmission request by a contained message. On the contrary, if a PDU is configured to have a periodic transmission mode, it is sent periodically with a predefined frequency. Finally, if the transmission mode is set to mixed periodic and on-demand, transmissions are combined.

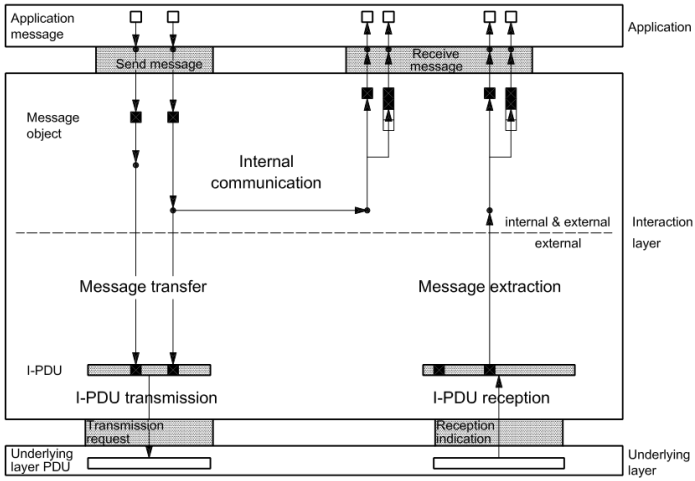


Figure 1.11. Message Passing via OSEK/VDX IL (Source: [121])

1.2.6 AUTOSAR

Automotive Open System Architecture (AUTOSAR) is a consortium similar to the OSEK/VDX consortium. It was founded in 2003 by German OEMs and suppliers. In the course of the year, international OEMs joined the partnership. The self-defined goal of AUTOSARs consortium is to create standards for development methodologies, basic system functionalities and their interfaces. Because there is a clear overlap in the technical field and in the stakeholders involved, some ideas from the OSEK/VDX standards were adopted and continued in the standards around the AUTOSAR *classic platform*. Responding to the increased demand for computing performance

specifications for a *adaptive platform* have been added around 2017 [10].

Several ideas of the OSEK/VDX standards are continued and further developed in AUTOSAR. The IL for example has strong similarities to the virtual functional bus (VFB) of AUTOSAR [9]. In some parts the standars of OSEK/VDX are even referenced in the AUTOSAR standards, e.g. in the case of *OsTask*s in [13].

A further important concept which AUTOSAR is the *Layered Software Architecture* introduced in the year 2005 (cf. [11]). It aims at providing an interface to abstract from the underlying hardware. Consequently, the two most important layers of the architecture are the basic software (BSW) layer and the application software (ASW) layer. Between these two layers a runtime environment (RTE) layer is generated. It completely decouples both layers as depicted in Figure 1.12. To achieve this, it implements the communication paths according to the VFB specifications. This approach allows to develop software in the ASW layer independently from the underlying hardware and improves portability of the of the application software components. Furthermore, the encapsulation of memory accesses allows to partition memory into regions to prevent software components to interfere wrongfully.

Non-functional requirements however have not been part of the methodology right at the outset. Standardized methods to define timing requirements have been added in 2009 with the initial version of the TIMEX [14]. How the AUTOSAR TIMEX can be used to define so-called *event chains* and their timing requirements is described in more detail in Section 3.1.4.

1.3 Scope of the Thesis

The analysis of various performance metrics for real-time systems is subject to academic research for over five decades now [35] and although some work has successfully been applied in industry, a gap exists between industry needs and solutions provided by academia [103]. Precision and scalability are important criteria for acceptance of an approach, but a question which is often observed to be the main difficulty is how to encode the real system in a formal model. At the same time, systems are still getting more complex and there is no doubt that even the most

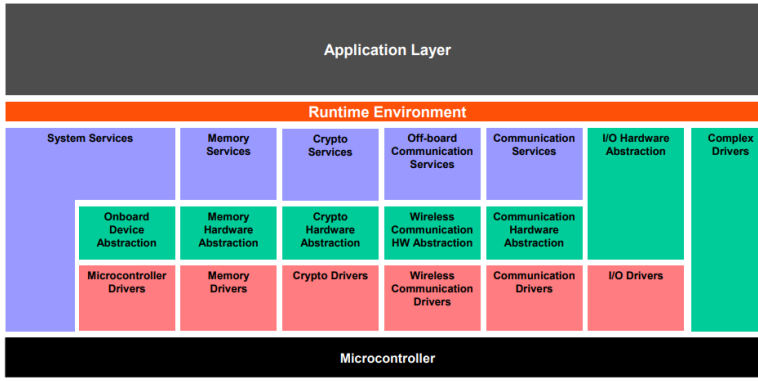


Figure 1.12. Components and Interfaces View of AUTOSAR *Layered Software Architecture* (Source: [11])

experienced engineers cannot possibly keep the overview of the behavior of the complete system, especially when it comes to end-to-end latencies. As a consequence, tool support for latency analysis is more important than ever. It is therefore the objective of the present work to close the above-mentioned gap for automotive systems and present precise estimation approaches for end-to-end latencies scaling for relevant systems. To compromise between theorists and practitioners and to support easy comprehensibility the modeling process is divided into multiple steps.

Our main focus for all presented formal approaches is expressiveness and precision. These are, in the author's opinion, the most important criteria for the acceptance of such approaches. Nonetheless, scalability for the relevant systems is a necessary requirement for the actual application of any approach. For this reason, we test the proposed approaches with actually implemented as well as synthetically generated systems. An example which will accompany us through this work is part of the communication cluster of automotive powertrains. As an example for an actual system, we study the formal model of the central powertrain control module (CPCM) as well as the network communication of the CPCM with the egine control module (ECM) and transmission control module (TCM). The CPCM is particularly suitable because of its domain controller role. The blurring borders

between static and dynamic architectures can be observed here. Some hard real-time properties are an inherent part of the powertrain but the CPCM also needs to participate in dynamic back-end communication. The size of the task set and the amount and type of network communication are consequently a profound benchmark.

A different challenge in the area of latency analysis is taken up in the last chapter: the work is no viable solution with the growing amount of measurement data produced. An increased amount must be expected since systems are getting bigger and data logging gets potent, e.g. with FPGA-based live tracing [51]. In the next sections we discuss the problem of end-to-end latency analysis and why a combination of both formal verification and measurement data evaluation is the most solid option for reliable estimations.

1.3.1 End-to-end Latency Analysis

The objective of end-to-end latency analysis is to estimate the system-level performance of network of heterogeneous ECUs to verify that end-to-end latency constraints are met. Such constraints exists for different functionalities of the car, e.g. braking. Multiple so-called cause-effect chains (sometimes called event chains) are assigned to one system function to describe the sequences of processing steps made by the system to implement the functionality. Usually, these chains start with an user action or at one of the above-mentioned sensors and ends at one of the above-mentioned actuators. An instance of the chain starts with the triggering event and ends at actuation. Depending on the possible relative offsets between the events within the course of the cause-effect chain, different instances with different end-to-end latencies are possible. The set of possible instances therefore determines the set of possible end-to-end latencies. The events in the course of the chain refer to tasks reading and producing signals while processing the chain and the transmission of signals via network.

Predicting end-to-end latencies in early design phases and verifying them for the final product are important steps in building dependable systems and with the current developments as outlined above, these task are in strong demand for computer-aided solutions. Data acquisition for this can either be done with the help of formal methods or by measurement

as discussed in the next section.

End-to-end latency constraints exist for different latency semantics. Age of data and time to first response are the two important latency semantics for automotive systems [45]. The age of data, or *data age*, is the time span between the start an instance of the event chain and the latest possible impact the generated signal value might have in the course of the effect chain. The time to first response or *response time* on the other hand is the time between the start of an instance of the chain an the first reaction at the end of the chain.

Different measures of data age and response time are of interest. The most important one is the maximal possible value. If it can be shown that the maximum latencies meet the constraints on the end-to-end latency for all chains in ay situation, the system meets all end-to-end constraint in a worst-case scenario. Additionally, the minimum and average value are interesting to get an idea about the range of possible behaviors. A deviation from the expected value is also referred to as *jitter*. Jitter is especially challenging for calibration engineers as they need to find calibration parameters working for a broad range of possible latencies.

1.3.2 Formal Methods vs. Measurement Data Evaluation

The challenge in end-to-end latency analysis for a system function is, that *all* possible instances the cause-effect chains must be considered. Basically, two approaches exist to identify the set of possible chain instances: (1) analyzing simulation or measurement data and (2) conducting a formal analysis.

The challenge for simulation and measurement data is test coverage. Since all instances of cause-effect chains spanning over multiple ECUs need to be covered, potentially a lot of stimuli need to be tested.

Figure 1.13 depicts the problem potentially exiting for non-exhaustive analyses like simulation or measure-data analysis. In general, not all input combinations can be possibly tested. Non-tested input combinations however potentially include a situation with a worse case than found thus far.

Covering all possible situations is also the main challenge for formal approaches. However, in this case it leads to a trade-off between computa-

tional complexity and precision.

In summary, from a economic efficiency point of view, both formal methods (FM) and measurement data analysis (MDA) need to be pursued. Only a combination gives the best validation results with an eye on cost efficiency, because although overestimation is in contrast to profit motives since they lead to too pessimistic system designs, undetected timing defects in products possibly result in product callbacks with costs very difficult to assess but definitely high.

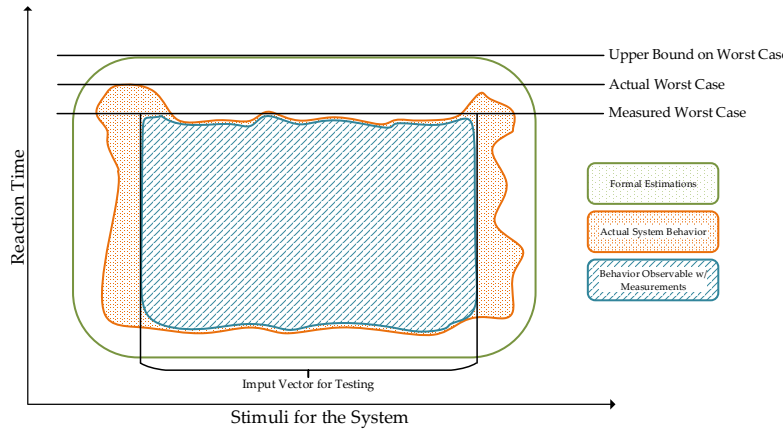


Figure 1.13. Test Coverage vs. Over Estimation

Chapter 2

Mathematical Preliminaries

In the previous chapter a broad outline of the systems which are subject to the analyses developed in this work was given. The following chapters are concerned with the most important two steps of performance analysis: encoding the system in a mathematical model and proving or measuring properties of the system. Before we look at the systems and their formal representation in more detail, the basis for the formal models are introduced. The models used for measuring or proving properties of the system often have a clear focus, e.g. the temporal properties of the system. A model for the analyses of data dependencies, on the other hand, is not concerned with this perspective at all. However, for the comprehensive analysis, first the latter and secondly the former perspective might be needed. In other words, the different models need different but possibly overlapping input data. Therefore, a more general and exhaustive system model should be used for data acquisition prior to and between the different analyses. For measuring or reasoning with specialized models, a projection or a transformation from the more general model can be used.

In this chapter, the basic mathematical concepts and the language used throughout this work are introduced. First, notations and constants are defined. Secondly, some extended structures are introduced. In this work, tuples are used to specify the system model. This approach has two advantages. On the one hand, a database model can be constructed directly to support the development of computer-aided tools. On the other hand, describing translations of data to more specialized models, i.e. transformation algorithms, can be described more comfortable in a formal way.

The subsequent sections are concerned with records, graphs, a formal representation of time, and constraint programming. Their purposes are:

▷ **Records**
to formally describe the input data of the models used to encode the real-word systems.

▷ **Graphs**
to describe different relationships between sets of objects. For this purpose, nodes and vertices are labeled, e.g. a node can be labeled with a tuple representing an artifact in the system model.

▷ **Time**
for both, reasoning about temporal properties and evaluation of measurement data, a formal representation of time is needed.

▷ **Constraint Programming**
for automated reasoning. The properties of the system are encoded in more specialized models. Constraint programming allows for declaratively stating the systems behavior in terms of constraints. Feasible solutions for the variables of these constraints represent possible system's behavior.

2.1 Basic notations

Let A be a set. The cardinality A is denoted by $|A|$, e.g. $|\{5,2,7\}| = 3$. Let B be a second set. The Cartesian product, or product set $A \times B$ is the set of all ordered pairs (a,b) where $a \in A$ and $b \in B$, that is:

$$A \times B = \{ (a,b) \mid a \in A \text{ and } b \in B \} \ . \tag{2.1.1}$$

The power set of a set A is the set of all subsets of A and is denoted by $\mathrm{P}(A)$.

The inline restriction of a set A is denoted $A_{|f}$ where f is a function mapping elements from A to the set $\{\text{true}, \text{false}\}$ and is defined as the set $\{ a \in A | f(a) = \text{true} \}$.

2.1.2 Definition (Kleene closure)**.** An alphabet is a set A containing symbols (or characters). Set $A^0 = \{ \epsilon \}$ where ϵ is a special symbol representing the empty string, $A^1 = A$, and $A^i = \{ uv \mid u \in A^{i-1} \text{ and } v \in A \}$ for $i \in \mathbb{N}$

and $i > 1$. The string uv denotes the concatenation of $u \in A^i$ and $v \in A$. The definition of the Kleene closure of A is:

$$A^* = \bigcup_{i \in \mathbb{N}} A_i \,. \tag{2.1.3}$$

Throughout this work the following symbols are used to refer the respective set:

$\mathbb{N}$ Denotes the set of *Natural* numbers, $\{0, 1, 2, \dots\}$. The number is included 0 unless stated otherwise.

$\mathbb{Z}$ Denotes the set of *Integers*, $\{\dots, -2, -1, 0, 1, 2, \dots\}$, i.e. the set of natural numbers and their additive inverses.

$\mathbb{Q}$ Denotes the set of *Rationals*. A rational number is any number that can be expressed as the fraction of two integers $\frac{p}{q}$, $p, q \in \mathbb{N}$ with $q \neq 0$.

$\mathbb{R}$ The set of *Reals*. The real numbers include the rational numbers and the irrational numbers, i.e. the numbers which can not be represented as fractions of two integers (excluding imaginary numbers).

$\mathbb{S}$ The set of *Strings*. Let $\Sigma = \{\mathtt{a}, \dots, \mathtt{z}, \mathtt{A}, \dots, \mathtt{Z}, \mathtt{0}, \dots, \mathtt{9}\}$ be the set of *Characters*, then $\mathbb{S} = \Sigma^*$. Note that the number 1 is not equal to the character `1`. A word in Σ^* is also referred to as *string*. In this work, we use strings as *identifiers*.

$\mathbb{B}$ The set of truth values, $\mathbb{B} = \{\text{true}, \text{false}\}$. Note that true $\neq$ `true` and false $\neq$ `false` for the Strings `true`, `false` $\in \mathbb{S}$.

Throughout this work we use the following symbols to denote the respective logical connectives of two Boolean variables $a, b \in \mathbb{B}$:

$\wedge$ Denotes the logical conjunction, i.e. $a \wedge b$ is true if and only if a is true and b is true.

$\vee$ Denotes the logical disjunction, i.e. $a \vee b$ is true if and only if a is true or b is true.

$\neg$ Denotes the inversion of proportion or truth value, i.e. $\neg a$ is true if and only if a is false. Note, that unary operators have a higher precedence than binary operators.

$\Rightarrow$ Denotes the logical implication, i.e. $a \Rightarrow b$ is true if and only if $\neg a \vee b$ is true.

2.1.1 Tuples

A tuple of length $n \in \mathbb{N}$, or n-tuple, is a finite and ordered sequence of elements. Parentheses are used to enclose tuples, e.g. $(0, 1, 1, 2)$ is the 4-tuple consisting of the elements 0,1, and 2. Formally, the definition is based on the definition of *ordered pairs*. Let A and B be sets. An ordered pair $p = (a, b)$ over the *domain* $A \times B$ is an pair of elements $a \in A, b \in B$. It equals $p' = (a', b') \in A \times B$ if and only if $a = a'$ and $b = b'$, i.e. the order of the elements in p is determining identity as well as the identity of the domains. Formally, tuples can be defined as nested ordered pairs. The 0-tuple is represented by $\varnothing$. An n-tuple is represented by an ordered pair (a, b) where a is the first component of the tuple and b is an $(n-1)$-tuple. We write $(x_1, \ldots, x_n)$ to denote the n-tuple $(x_1, (\ldots, (x_n, \varnothing)))$. The Cartesian Product of the domains of each component determines the *type* of the tuple.

Note that, unlike the properties of sets where $\{ 0, 1, 2 \} = \{ 2, 1, 0 \}$, for tuples $(0, 1, 2) \neq (0, 2, 1)$ and $(0, 1, 1, 2) \neq (0, 1, 2)$ holds.

Lists

A special case of tuples are lists or sequences which are used as synonyms in this work. They are tuples where each element has the same type and serve as ordered collections of values. Unlike the properties of sets, elements can be contained in the collection multiple times. Similar to the Kleene-closure, we define an operation which gives us all lists of length k which can be constructed with the elements of a set.

2.1.4 Definition (Lists over a Set)**.** Let A be set. We define $A^{l(0)} = \varnothing$. The lists of length $i \in \mathbb{N}$, $i > 0$ over A is defined as

$$A^{l(i)} = \{ (a, b) \mid a \in A, b \in A^{l(i-1)} \} \ .$$

The set of tuples of arbitrary length is denoted $A^{l(*)}$ and defined as

$$A^{l(*)} = \bigcup_{i \in \mathbb{N}} A^{l(i)}$$

2.1.5 Definition (List)**.** Let A be a set. A list of length n over domain A is a tuple of length $n \in \mathbb{N}$ where each element comes from A. As an abbreviation the notation $(a_i)_{i=1}^{n}$ is used for a list $l = (a_1, \ldots, a_n) \in A^{l(*)}$. Furthermore, we define the following operations on lists:

Empty list The nil-constant creates an empty list:

$$\begin{aligned} &\mathrm{nil}() \colon A^{l(0)} \\ &\mathrm{nil}() = \varnothing \ . \end{aligned}$$

Length The len-operator gives the length of a list:

$$\begin{aligned} &\mathrm{len} \colon A^{l(n)} \to \mathbb{N} \\ &\mathrm{len}(a_1, \ldots, a_n) = n \ . \end{aligned}$$

Contruction The cons-operator allows to append elements to a list. Let $a \in A$, then

$$\begin{aligned} &\mathrm{cons} \colon A^{l(n)} \times A \to A^{l(n+1)} \\ &\mathrm{cons}(l, a) = (a_1, \ldots, a_n, a) \end{aligned}$$

Concatenation The infix operator $\oplus$ denotes the concatenation of two lists:

$$\begin{aligned} &\oplus \colon A^{l(n)} \times A^{l(m)} \to A^{l(n+m)} \\ &(a_1^1, \ldots, a_n^1) \oplus (a_1^2, \ldots, a_m^2) = (a_1^1, \ldots, a_n^1, a_1^2, \ldots, a_m^2) \ . \end{aligned}$$

Selection The elem_i-operator is a projection to the i^{th} element of a list l:

$$\begin{aligned} &\mathrm{elem}_i \colon A^{l(n)} \to A \\ &\mathrm{elem}_i(a_1, \ldots, a_n) = (a_i) \end{aligned}$$

for all $i \in \{ 1, \ldots, n \}$.

To-Set The set-operator gives all elements of a list in a set:

$$\mathrm{set}\, A^{l(n)} \to \mathrm{P}(A)$$

$$\text{set}(l) = \bigcup_{i \in \mathbb{N}} \{ a_i \} \ .$$

2.1.2 Functions

Let A and B be sets. A *relation* between two sets is a subset of the Cartesian product $A \times B$. A relation $R \subset A \times B$ encodes that element $a \in A$ is related to element $b \in B$ if $(a, b) \in R$. A function is a special type of relation between two sets. It associates each element from the first set set to exactly one element of the second set. To denote that a function f maps values from A to values in B, we write $f : A \to B$. A is called the *domain* of f, formally denoted by $\text{domain}(f)$ and B is called the *codomain* of f, formally denoted by $\text{codomain}(f)$.

In this work, the inline notation for functions is used. We write

$$\{ a_1 \mapsto b_1, \ldots a_n \mapsto b_n \}$$

to denote the function f with

$$\begin{aligned} f(a_1) &= b_1 \\ &\vdots \\ f(a_n) &= b_n \end{aligned}$$

.

A *partial function* from A to B is a function $g : A' \to B$ with $A' \subset A$. In contrast, the term *total function* is a synonym for *function*, used to emphasize on the fact that the whole domain must be mapped.

A function $f : A \to B$ can fulfill the following properties:

Injectivity f is said to be *injective* if

$$\forall a_1, a_2 \in A \colon f(a_1) = f(a_2) \Rightarrow a_1 = a_2 \ .$$

Surjectivity f is said to be *surjective* if

$$\forall b \in B \colon \exists a \in A \colon f(a) = b \ .$$

Bijectivity f is said to be *bijective* if f is injective and surjective.

Invertibility f is called *invertible* if and only if f is bijective. If f is invertible, a function $g : B \to A$ exists such that $g(f(a)) = a$ for all $a \in A$ and $f(g(b)) = b$ for all $b \in B$. g is called the *inverse function* of f. In this work, we denote the inverse function of a invertible function f by f^{-1}.

The following relations between domain and codomain can be inferred from injectivity, surjectivity, and bijectivity [1].

2.1.6 Remark (Domain and Codomain of injective Functions). Let A and A be sets. If an injective function $f : A \to B$ exists, then $|A| \leqslant |B|$. From injectivity follows $a_1 \neq a_2 \to f(a_1) \neq f(a_2)$. Since f maps all values from A to a unique value in B, B must contain at least as much elements as A.

2.1.7 Remark (Domain and Codomain of surjective Functions). Let A and B be sets. If a surjective function $f : A \to B$ exists, then $|B| \leqslant |A|$. Since all values in B need to have a value in A which is mapped to it, A must contain at least as much elements as B.

2.1.8 Remark (Domain and Codomain of bijective Functions). Let A and B be sets. If a bijective function $f : A \to B$ exists, then $|A| = |B|$. If $|A| \leqslant |B|$ and $|B| \leqslant |A|$ then $|A| = |B|$.

2.2 Records

A record is a collection of different named values. The formal definition is formulated based on nested ordered pairs.

2.2.1 Definition (Record Type)**.** A record type is a ordered pair (n, t) where $n \in \mathbb{S}$ is an identifier and t is an n-tuple $(t_1, \ldots, t_n)$ with $t_i = (n_i, D_i)$ an ordered pair with $n_i \in \mathbb{S}$ the name of the field and D_i a set $i \in \{1, \ldots, n\}$. D_i are the allowed values for the field i, i.e.the value of i^{th} field of a record of type n must be some $d \in D_i$.

Let $\mathcal{T} = (\texttt{RecordTypeName}, ((\texttt{FieldName}_1, D_1), \ldots, (\texttt{FieldName}_n, D_n)))$ be a record type. As an abbreviation, we write

$$\texttt{RecordTypeName}\ (\texttt{FieldName}_1 : D_1, \ldots, \texttt{FieldName}_n : D_n\ )$$

to denote $\mathcal{T}$.

[1] Author's note: these statements can be further generalized and extended but here we confine ourselves to the needed properties.

An instantiation of a record type is called record. In the following, the name of a record type is used to denote the set of all records of that type.

2.2.2 Definition (Record)**.** Let

$$\mathcal{T} = \texttt{RecordTypeName}\ (\texttt{FieldName}_1 : D_1, \ldots, \texttt{FieldName}_n : D_n\ )$$

be a record type. A record of $\mathcal{T}$ is a tuple with type $D_1 \times \ldots \times D_n$. To avoid any confusion, we write

$$R = \texttt{RecordTypeName}\ (v_1, \ldots, v_n\ )$$

to denote a record $(v_1, \ldots, v_n) \in D_1 \times \ldots \times D_n$ of type $\mathcal{T}$.

To conveniently access the fields of a record, implicitly defined selection functions are used.

2.2.3 Definition (Selector Function)**.** Let

$$\mathcal{T} = (\texttt{RecordTypeName}, ((\texttt{FieldName}_1, D_1), \ldots, (\texttt{FieldName}_n, D_n)))$$

be a record type. The following *selector functions* are implicitly defined for all $i \in \{\, 1, \ldots, n \,\}$:

$$\texttt{RecordTypeName.FieldName}_i : D_1 \times \ldots \times D_n \to D_i$$
$$\texttt{RecordTypeName.FieldName}_i((v_1, \ldots, v_n)) = v_i\ .$$

2.3 Graphs

In this work the following notations for graphs are used: a graph G consists of a finite set of vertices V and a finite set of edges E.

2.3.1 Definition (Finite Graph)**.** A finite graph is an ordered pair $G = (V, E)$ with V a finite set of vertices and either $E \subseteq V \times V$ (defining a directed graph), or $E \subseteq \{\, \{\, v_1, v_2 \,\} \mid v_1, v_2 \in V \,\}$ (defining an undirected graph).

A graph can have the following special forms: In the case of directed graphs write $v_1 \to v_2$ if and only if $(v_1, v_2) \in E$. In the case of undirected graphs we write $v_1 - v_2$ if and only if $\{\, v_1, v_2 \,\} \in E$.

Irrespective of whether a graph is directed or undirected, the edges and vertices of a graph can be labeled. Formally, with L a set of labels, this is achieved with labeling functions $l_E : E \to L$ or $l_V : V \to L$ respectively. We write $G = (V, E, L_V, L_E)$ for a graph with two labeling functions, $G = (V, E, L_V)$ for a graph with a vertex labeling function, and $G = (V, E, L_E)$ for a graph with an edge labeling function respectively.

A *walk* of length n in a graph $G = (V, E)$ is a sequence of edges $(u, v)_{i=1}^{n}$ with $(u, v)_i \in E$ where $v_i = u_{i+1}$ for all $i \in \{1, \ldots, (n-1)\}$. The vertex walk for a walk $(u, v)_{i=1}^{n}$ is the sequence $(u_1, \ldots, u_i, v_i)$. A *cycle* is a walk for which the first and last vertex of the vertex walk are identical.

2.4 Continuous and discrete time

For the formal analysis of timing properties an abstraction for time measurement is needed. In physics, there is a plethora of different concepts that would go way too far to be covered here. In computer science, two mappings of time to abstract clocks are common: *continuous* and *discrete*. The full list of time abstractions additionally includes *dense* and *logical* time.

The continuous time model models a clock with a differentiable function $C_{\text{con}} : \mathbb{R} \to \mathbb{R}$. This clock possibly has discontinuities where a reset happens. It has a precision which is given in $\frac{\mathrm{d}}{\mathrm{d}t}C(t_0)$ at all $t_0 \in \mathbb{R}_{\geqslant 0}$ where C is continuous and differentiable. This notion of time pays attention to the fact that physical values are possibly *incommensurable*, which is relevant for clocks which are not perfectly synchronized [54].

In the case of discrete and dense time the domain for the clock reads are not continuous but countably infinite sets. More precisely, for dense time the domain for points of time is $\mathbb{Q}$, for discrete time it is $\mathbb{N}$. Again a clock maps time to points of dense or discrete clock readings, i.e. a discrete function $C : \mathbb{R} \to \mathbb{D}$ where $\mathbb{D} = \mathbb{Q}$ or $\mathbb{D} = \mathbb{N}$. Discrete and dense clock values increase every time the interval of a *tick* is elapsed.

Besides the above-mentioned metric formalism with an underlying time domain, there is a fourth, which is more general in that it detaches from the idea of time flowing in equidistant measures. This concept is also known as *logical* time. The idea is to order events in a system by their

occurrence. It was first introduced by Lamport in 1979 [80]. The clocks to *measure* logical time are especially useful in situations where a specification of right system behavior in terms of timing can not rely on the presence of accurate physical clocks. Instead, a logical clock assigns numbers to events in the system, only satisfying the constraint, that an event A gets a lower number than an event B if A occurred before B.

Depending on the method and focus, a different abstraction of time is useful for the modeling and the analysis of CPS. In this work the discrete time abstraction is used for the formal analysis of the temporal behavior of cause-effect chains in task sets and within a cluster of connected ECUs. In the context of time, two notations are used: $\bar{t}$ is used to denote intervals of time, and $[t_0, t_1]$ denotes the interval containing all points of time in $\{\, t \mid t \in \mathbb{D} \wedge t_0 \leqslant t \leqslant t_1 \,\}$

2.5 Constraint Programming

Constraint programming is a technique to solve combinatorial problems based on a declarative problem description. The *programming* aspect should partly be viewed as the programming in mathematical programming. The user declaratively models the problem as the feasible solutions for a set of decision variables. However, on top of this, the user may specify a search strategy to optimize the systematic search for solutions to his specific problem [109]. Although constraint programming does, in general, not prescribe *how* to solve a problem, search strategies allow to adapt the technique to better perform on certain problems. A constraint solver takes the encoding of the real-world problem and the search strategy, and tries to find an assignment to the variables that satisfies all constraints. Defining the set of constraints to model a real-world problem seems easy at first glance, however there are some pitfalls for finding a good model, e.g. over-constraining [61].

While the question whether a satisfying assignment exists or not can be used to check whether the modeled real-world object has a certain property or not, adding a *objective function* to the problem description additionally allows to identify extrema, e.g. a worst-case situation. The solver will then try to find a feasible solution yielding the largest value

for the objective function. If such a solution is found, the assignment of the variables models the real-world situation leading to the worst case. Furthermore, a proof that no worse case can be found is implicitly part of the solution.

In this section, the formal foundation of constraint programming is introduced: the constraint satisfaction problem (CSP).

The first ingredient for a CSP are *Variables*.

2.5.1 Definition (CSP Variable)**.** A variable in a CSP has an identifier and a domain of valid values for this variable.

2.5.2 Definition (Constraint)**.** Let $S = \{ (x_1, D_1), \ldots, (x_n, D_n) \}$ be a set of CSP variables and $\mathcal{D} = \bigcup_{i \in \{ 1, \ldots n \}} D_i$. A constraint is a tuple $c = (R, S)$ where R is a relation over arithmetic expressions containing variables from S and constants from $\mathcal{D}$.

To search for and check solutions, the variables in constraints need to be replaced by values from their domain. This is formally described with the help of a *substitution*.

2.5.3 Definition (Substitution)**.** Let $c = (R, S)$ be a constraint. A substitution for c is a partial function $\varphi : S \to \mathcal{D}$. A substitution φ is called *consistent* if R holds true for the assigned values. A substitution φ is called *complete* if φ is a total function. If a substitution φ is consistent and complete, we write $\varphi \models c$.

A CSP consists of variables, domains for the variables, and constraints on the variables.

2.5.4 Definition (Constraint Satisfaction Problem)**.** A CSP is a triple $\mathcal{P} = (X, \mathcal{D}, C)$ where $X = ((x_1, D_1), \ldots, (x_n, D_n))$ is a list of variables, $\mathcal{D} = \bigcup_{i \in \{ 1, \ldots n \}} D_i$ is the universe all variables might attain values from, and $\mathcal{C}$ is a set of constraints such that

$$\forall (R, S) \in \mathcal{C} : S \subseteq \text{set}(X) \ .$$

An assignment to a CSP is tuple which contains a value for each variable.

2.5.5 Definition (Assignment for a CSP)**.** Let $\mathcal{P} = (X, \mathcal{D}, C)$ be a CSP with $X = ((x_1, D_1), \ldots, (x_n, D_n))$. An assignment to $\mathcal{P}$ is a n-tuple $(a_1, \ldots, a_n) \in D_1 \times \ldots \times D_n$.

Finally, a solution of a CSP is defined as an assignment which yields consistent substitutions for every constraint.

2.5.6 Definition (Solution of CSP)**.** Let $\mathcal{P} = (X, \mathcal{D}, C)$ be a CSP and let $A = (a_1, \ldots, a_n)$ be an assignment to $\mathcal{P}$. A is called a solution to $\mathcal{P}$ if

$$\forall (R, S) \in \mathcal{C} \colon \{\, x_i \mapsto a_i \mid i \in \{\, 1, \ldots, n \,\} \wedge (x_i, D_i) \in S \,\} \models (R, S)$$

The set of solutions for $\mathcal{P}$ is denoted $\mathrm{sols}(\mathcal{P})$.

To find solutions with certain properties, an *objective function* can be added to a CSP to rate the solutions found by the solver.

2.5.7 Definition (Constraint Optimization Problem)**.** Let $\mathcal{P} = (X, \mathcal{D}, C)$ be a CSP with $\mathcal{D} = (D_1, \ldots, D_n)$. A constraint optimization problem is a 4-tuple $\mathcal{P}^{\max} = (X, \mathcal{D}, C, \varsigma)$ with $\varsigma : (D_1, \ldots, D_n) \to \mathbb{Q}$ an objective function.

A solution to $\mathcal{P}^{\max}$ is an assignment $A = (a_1, \ldots, a_n)$ to $\mathcal{P}$ for which

$$\forall A' \in \mathrm{sols}(\mathcal{P}) \colon \varsigma(A') \leqslant \varsigma(A)$$

The set of solutions for $\mathcal{P}^{\max}$ is denoted $\mathrm{sols}(\mathcal{P}^{\max})$.

In this work, we will use CSP as a general term for constraint satisfaction and constraint optimization problems.

2.5.8 Example (CSP). Consider the the CSP $\mathcal{P} = (X, \mathcal{D}, C)$ with

$$\begin{aligned} X &= (x_1, x_2, x_3) \\ \mathcal{D} &= (\{\, 1, 2 \,\}, \{\, 2, 3 \,\}, \{\, 1, 2 \,\}) \\ c_1 &= x_1 < x_2 \vee x_1 > x_3 \\ c_2 &= x_2 > x_3 \\ c_3 &= x_1 = 4 \vee x_3 > x_1 \\ C &= \{\, c_1, c_2, c_3 \,\} \end{aligned}$$

A solution for $\mathcal{P}$ is the assignment $A = (1, 3, 2)$. To verify this, we have to look at the substitutions resulting from a:

$$\begin{aligned} \varphi_1 &= \{\, x_1 \mapsto 1, x_2 \mapsto 3, x_3 \mapsto 2 \,\} \\ \varphi_2 &= \{\, x_2 \mapsto 3, x_3 \mapsto 2 \,\} \end{aligned}$$

$$\varphi_3 = \{ x_1 \mapsto 1, x_3 \mapsto 2 \}$$

and check whether all constraints in $\mathcal{P}$ are satisfied:

$$\begin{aligned}
\varphi_1 &\models c_1 \iff \text{domain}(\varphi_1) = \{ x_1, x_2, x_3 \} \land 1 < 3 \lor 1 > 2 \\
\varphi_2 &\models c_2 \iff \text{domain}(\varphi_2) = \{ x_2, x_3 \} \land 3 > 2 \\
\varphi_3 &\models c_3 \iff \text{domain}(\varphi_3) = \{ x_1, x_3 \} \land 1 = 4 \lor 2 > 1
\end{aligned}$$

The small CSP given in Example 2.5.8 shows, that the problem of finding a solution gets complex quickly. In fact, the problem of finding a solution to a CSP is NP-hard in general. A compact, further discussion can be found in [42].

Chapter 3

Modeling Automotive Cyber-physical Systems

This chapter is dedicated to the modeling of CPS regarding their temporal behavior in order to conduct end-to-end timing analyses. The models introduced in this chapter are the basis for the formal models used in Chapter 4 and Chapter 5, and the measurement data analysis described in Chapter 6. We do not take sensor and actuator into account but focus on ECU software and network communication between sensing and actuating.

The end-to-end estimation process is therefore subdivided two parts: firstly, all relevant information for an exhaustive timing analysis need to be listed. Secondly, for each real-world object represented by its relevant properties, the best-fitting formal model needs to be defined or chosen and a transformation needs to be defined. This is the step in the verification procedure where usually a gap between formal models and the models used in the industry is observed. The former are usually described using mathematical notations. The latter are focused on data exchange between different parties involved in the development process. Scientific models usually strive high abstraction and generality, the data models of industry a focused on electronic data exchange and contain implementation details of different granularity.

As an example ECUs and networks might be abstracted to service-providing entities in an abstract mathematical model. Software tasks as well as network messages are abstracted to service consuming entities accordingly. Contrary to that, ECUs with their different processing units and networks with different types of physical channels can be modeled with completely dedicated object types in a less abstract approach. From another point of view, in an abstract model, the impact of multiple service

consuming entities might be aggregated to reduce the complexity of the analysis. This comes at cost of some overestimation in most cases. That is why an approach with several layers of abstraction is followed in this work. We make a plead for a separation of concerns: data collection and analysis. Both steps can work on different models to find a good balance between the two trade-offs: abstraction versus comprehensibility, and comprehensiveness versus solvability. The data model needs to have a strong orientation towards comprehensibility and comprehensiveness. The analysis model on the other hand needs to serve as an input for an algorithm executed by a machine which is easier to do with more abstract models.

Generally speaking, the properties needed to estimate the temporal behavior of a system come down to how often is the input data processed and how long it takes until a response is computed. The latter question is significantly harder than the first one. It does not only depend on the possible data flows existing between an input or request and an appropriate response but also on how often the respective routines are processed and the resulting possible interactions. As described above, the *cyber*-part of modern CPS usually comprises multiple heterogeneous ECUs interconnected by different buses.

This chapter is organized as follows: First of all related work with different backgrounds and different levels of abstraction is examined regarding the underlying model for a selection of the most promising concepts. Subsequently, the system model used in the rest of this work in formally defined in Section 3.2. It is divided into multiple parts: the relevant properties of the ECUs and networks of ECUs are discussed in Section 3.2.1 and Section 3.2.2 respectively. Building upon this, a formalization of *cause-effect* chains is introduced in Section 3.2.3. In Section 3.3 exemplary data sources for the system model are listed and the different parts are put in a broader context.

3.1 Related work

In this section related work from the field of modeling the temporal behavior of CPS is discussed. We distinguish two origins: data models

for standardized data exchange in the industry, and scientific models. Examples from both areas are discussed below.

In the scientific community, a plethora of system models where developed over time. They can be categorized according to various criteria which is discussed in more detail in Section 4.1. In this section we focus on the level of abstraction of the underlying models. As one would expect, models from the scientific community are more abstract. Theoretically, some even allow to consider software and networks together, like e.g. the Real-time Calculus (RTC). Similar ideas have been further developed, commonly known as compositional performance analysis (CPA) approaches. The underlying models are presented in Section 3.1.1. Furthermore, approaches exist, that are focused on either the networking properties or the software of single ECUs. Naturally, these models used are less abstract as described in Section 3.1.2 for a network-focused model and in Section 3.1.3 for a software-focused model respectively.

In industrial applications, models are usually more extensive, sometimes they even tend to be bulky as many stakeholder were involved in the development process. In any case they are less abstract. A well known model from the industry is the timing extensions of the methodology for describing automotive ECUs and bus systems of the AUTOSAR. It is further described in Section 3.1.4.

3.1.1 Real-time calculus and CPA

Real-time Calculus [136] is a method for formal performance analysis of embedded systems and networks. It is based on network calculus [27] which was introduced as an method to reason about data flows in networks based on so-called *input arrival curves* and *input service curves*. The result of the analysis are *output arrival curves* describing the outgoing event stream and *output service curves* describing the remaining capacity of the service providing entity. The modeling process for a system with one resource and one task, as well as all curves included is depicted in Figure 3.1. The relationship between the in- and output streams is the core of RTC: the description of the manipulation of the streams is described with the help of min-plus and max-plus algebra. These discrete algebraic systems can be used to describe discrete event systems. The interested reader is referred

to Part II of [16] for an profound introduction.

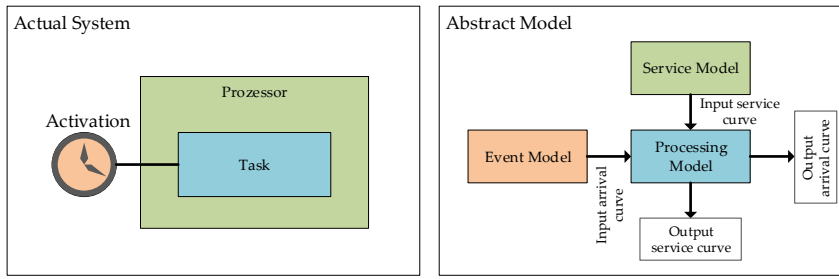


Figure 3.1. Data Model of Real-time Calculus

Another CPA approach developed around the same time as RTC is presented in [70]. The major difference is, that the approach presented in [70] allows to combine multiple input events for task activation to support the modeling of more complex systems. This approach has been further developed [113]. It was also the theoretical basis for a commercial tool [132].

However, all CPA approaches suffer from the problem of a context loss when the next compound is analyzed which is further discussed in Chapter 4.

As an interesting concept for our model, we note the idea to describe the arrival of tasks or network messages in terms of functions.

3.1.2 Timing analysis of vehicular networks

Another system model from the scientific community is the one introduced and used in [82, 81]. It has a strong focus on network transmission and is less abstract than the model underlying CPA. The core of the model is formed by so-called *message streams*. Each message stream is characterized by at least the following attributes:

- the length of the network message in bytes or the time it needs to transmit the message on the bus
- the minimum interval between the generation of two messages

▷ the relative deadline for a message

Furthermore, the model contains tasks between which message streams flow. A task is characterized by the following attributes:

▷ the execution time on a processing unit

▷ the minimum interval between two activations of the task

▷ the relative deadline of the task

As we can see, the authors distinguish between network communication and software tasks in their model. They use two rather specific models and consider the integration of network and software parts model-wise. This seems like a natural cut and we therefore note this partitioning for our data model. We will however see, that the network communication in todays automotive communication clusters has some properties which are essential for end-to-end latency analysis but not covered by the model proposed in [82, 81].

3.1.3 Synthesis of job-level dependencies

For the end-to-end timing analysis of *multi-rate effect chains* Becker et al. presented an approach in [20, 19]. Multi-rate here means, that a set of time-triggered tasks with different activation periods is analyzed. The underlying model has three parts: an application model, a chain model and a model for the dynamic behavior, i.e. the job-level dependencies. The application model is a set of tasks, where each task comprises the following elements:

1. an optional offset for the activation of a task
2. the period for the activation of a task
3. the worst-case response time of the task

The deadline of a task is implicitly assumed to be the period.

The chain model straight forward: a cause-effect chain is described as sequence of data-dependent tasks. The model for the dynamic behavior adds a new aspect: to calculate all possible data propagation paths, the

notions of a read and write interval is introduced. The read interval starts at the earliest possible point of time a job can read a value and ends at the last possible point of time it can read a value. In the context of a cause-effect chain this can be further refined since it is well-known which value is of interest. The write-interval is accordingly defined as the interval between the earliest point of time the output of the task is available and the last point of time the output of the task is available. These points of time can be derived from the information given in the instance of the application model.

From this approach, we note a third concept for our model: the differentiation between data dependencies in the application model, i.e. static task sequences corresponding to cause-effect chains, and the possible job-level instantiation at run-time. The static model provides the constraints to describe the possibilities of the dynamic model. The continuation of this thought leads to a further separation of data and analysis model.

3.1.4 AUTOSAR TIMEX

To do justice to the growing importance of timing analysis in automotive development processes, the AUTOSAR Timing Extensions (TIMEX) were added to the standard's methodology in 2009 [14].

The main objective of the extensions is to provide the means to describe timings and to specify *timing constraints* on them. The basis of the description is formed by so-called *events* which are combined to *event chains* in order to model end-to-end timings. The provided information are additionally intended to serve as the foundation for a validation of the temporal behavior [14]. The definition of events and consequently the definition of an event chain is broadly formulated in the TIMEX:

> *Events refer to locations in systems at which the occurrences of events are observed. The AUTOSAR Specification of Timing Extensions defines a set of predefined event types for such observable locations.* [...] *Event chains specify a causal relationship between events and their temporal occurrences. The notion of event chain enables one to specify the relationship between two events, for example when an event A occurs then the event B occurs, or in other words, the event B occurs if and only if the event A occurred before.* [...]

(Source: [14])

For events, the TIMEX allows to constrain the activation patterns. For event chains, latency constraints can be specified. Latency constraints are a bound on the reaction time or the data age between the first and last event of the chain. Additionally, execution time constraints and execution order constraints can be specified for runnable entities.

To make the concept of events more tangible, different so-called *timing views* for different phases of the development process are defined. They refer to the locations mentioned in the definition given above. The following timing views exist:

VfBTiming
: Timing on the level of the *Virtual Function Bus*. This view is concerned with the interaction of so-called *SwComponentTypes* which is a abstract base class for all AUTOSAR software components.

SwcTiming
: Timing on the level of *Software Components*. This view is concerned with the internal behavior of so-called *AtomicSwComponentTypes*. *AtomicSwComponentTypes* are software components which can not be further decomposed or distributed across multiple ECUs.

SystemTiming
: Timing on the uppermost level, the level which is described by a topology, a software implementation and a mapping on ECUs, and a communication design.

BswModuleTiming
: Timing on the level of *Basis Software Module*. This view is concerned with the internal behavior of so-called *BswModuleDescription* which describe the software modules underneath the AUTOSAR runtime environment.

EcuTminig
: Timing on the uppermost level within one ECU. In this view the timing can be described with the help of references to all ECU-relevant information, e.g. input and output values of basic software modules.

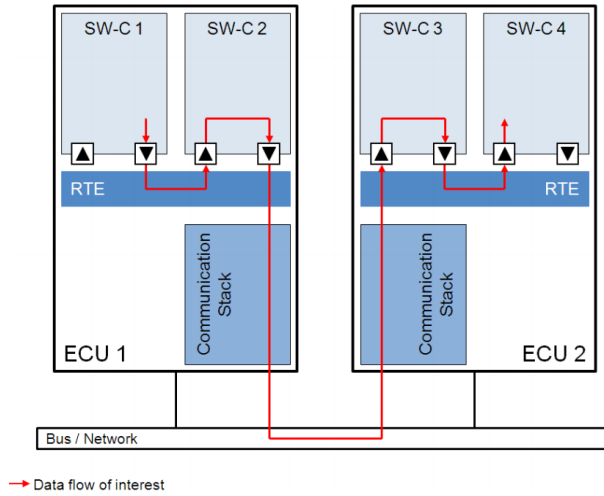


Figure 3.2. *SystemTiming*-view of AUTOSAR TIMEX (Source: [14])

Most of the different timing views are well suited to help OEMs to break down requirements and to exchange them with their suppliers focused on different parts of the systems. For timing analysis multiple views of the focused views have to be combined. The *SystemTiming*-view is intended for this purpose. Consequently, this view is the one with the most similarity to the perspective we take in our system model. The main difference is that we do not consider software components to be black boxes but look at their internal behavior. Additionally, we do not distinguish between application software and the software responsible for the communication stack.

3.2 System Model

In this section, the system model used in the rest of this work is formally defined. The heart of the model are communication clusters consisting of two elements: ECUs and networks which connect them. Although a higher degree of abstraction would be possible by reducing ECUs and networks

to service-providing entities, we decided to go separate concerns for two reasons: Firstly, the analysis process as we propose it is already divided in two parts due to different challenges in the verification process. The stronger argument is however that, secondly, an abstract model bringing together the service consuming entities of both domains with the needed detail would be cumbersome. For the network communication artifacts on the one side, sending triggers for PDUs of different OSI layers are needed. For runnable entities on the other side, the activation pattern and memory accesses are relevant. An overview of the differences is given in Table 3.1.

Table 3.1. End-to-end analysis - Software vs. Network

	Software	Network
Challenge	Find all possible relative offsets between reads and writes of runnable entities	Find the longest delay possible for one artifact to be triggered and send based on various conditions
Artifacts analyzed	Runnable entities processed in task contexts	Network packets on different layers
Structure of artifacts	Set of sequences of runnable entities	Hierarchical, tree-like structure for all frames
Most relevant temporal properties	Activation by timer or event and core-execution time	Sending trigger, trigger dependencies, and transmission time

3.2.1 Electronic Control Units

The first artifact in our model are ECUs. The ECU model has two parts: hardware and software. Properties of the hardware are, e.g., the amount of processing units determines how many tasks can be processed in parallel. The software is organized in tasks which call a sequence of runnable entities. The scheduling of tasks as well as the mapping to the processing units of ECUs affects the possible data flows which again have an direct impact on the possible latencies.

ECU Hardware

The timing-relevant properties of the hardware of an ECU include the physical connections to sensors and actuators, as well as communication

controllers, the speed of its processing units and the size and speed of its memories. As aforementioned, in this work, we are focused on the *cyber*-part of CPS. Thus, interfaces to sensors and actuators are reduced to an abstract event source or sink which are implicitly part of every ECU.

3.2.1 Definition (Communication Port)**.** Let $n, s \in \mathbb{S}$ be identifiers, let $c \in \mathbb{N}$, and let $d \in \{\, \texttt{in}, \texttt{out} \,\}$. A communication port is a tuple $P = (n, c, r, d)$ describing that the resource identified by r is provided (`out`) or consumed (`in`) via the the c^{th} channel of the network identified by n.

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{CommunicationPort}\ (\\ &\qquad \texttt{Network} : \mathbb{S},\\ &\qquad \texttt{Channel} : \mathbb{N},\\ &\qquad \texttt{Resource} : \mathbb{S},\\ &\qquad \texttt{Direction} : \{\, \texttt{in}, \texttt{out} \,\}\ )\end{aligned}$$

3.2.2 Definition (Processing Unit)**.** Let $c \in \mathbb{N}$ and $s \in \{\, \texttt{SPP}, \texttt{SPNP} \,\}$. A processing unit is a tuple $U = (c, s)$ describing a processing unit with an index c and a scheduling policy s where `SPP` stands for fixed-priority pre-emptive scheduling and `SPNP` stands for fixed-priority non pre-emptive scheduling.

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{ProcessingUnit}\ (\\ &\qquad \texttt{CoreNr} : \mathbb{N},\\ &\qquad \texttt{Scheduler} : \{\, \texttt{SPP}, \texttt{SPNP} \,\}\ )\end{aligned}$$

3.2.3 Definition (Electronic Control Unit)**.** Let P be a finite set of communication ports and let U be a finite set of processing units The tuple $\mathcal{E} = (w, P, U)$ is called ECU w with communication ports P and the processing units U.

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{ElectronicControlUnit}\ (\\ &\qquad \texttt{EcuName} : \mathbb{S},\\ &\qquad \texttt{Ports} : \texttt{CommunicationPort},\end{aligned}$$

$$\texttt{Processors} : \texttt{ProcessingUnit}\ )$$

ECU Software

The characteristics of the software processed in automotive ECUs is described in Section 1.2.4 in detail. At this point we briefly revisit the topic to extract the important properties for our model.

Each task has a sequence of so-called software runnables assigned for processing. Software runnables are list of statements or instructions that perform a sub-task, basically like a function, e.g. in the *C*-programming language. The processing of a runnable might depend on the mode of operation. In our approach, the different modes of operation need to be validated individually with their configuration of runnables.

A task is activated by OS-interrupts which occur periodically or event-driven e.g. when a sensor reads a specific input. Each time a task is activated, an *instance* of this task is processed. Within the system model we assume that the processing unit a task is processed on is determined at design time. If this is not the case, the verification process as described below has to be performed for all possible configurations by generating different instances of the model.

For tasks which are activated by a periodic timer, the *Periodic Activation Model* is used:

3.2.4 Definition (Periodic Activation Model)**.** Accordingly, we define the following record type:

$$\begin{aligned}\texttt{PeriodicActivationModel}\ (&\\ &\texttt{Offset} : \mathbb{N},\\ &\texttt{Period} : \mathbb{N}\ )\end{aligned}$$

For task which are activated by events where a minimal and maximal time between two readings can be specified, the *Bound Activation Model* is used:

3.2.5 Definition (Bound Activation Model)**.** Accordingly, we define the following record type:

$$\texttt{BoundActivationModel}\ ($$

$$\texttt{DtMin} : \mathbb{N},$$
$$\texttt{DtMax} : \mathbb{N}\ )$$

For tasks which only occur sporadically without an upper bound on the inter-arrival time, the *Sporadic Activaion Model* is used:

3.2.6 Definition (Sporadic Activation Model)**.** Accordingly, we define the following record type:

$$\texttt{SporadicActivationModel}\ (\ \texttt{DtMin} : \mathbb{N}\ )$$

A third option for task activation is that one task is being activated by a predecessor, which is also referred to as *task chaining*. Therefore, we define an additional timing model for chained activation:

3.2.7 Definition (Chained Activation Model)**.** Accordingly, we define the following record type:

$$\texttt{ChainedActivationModel}\ (\ \texttt{Predecessor} : \mathbb{S}\ )$$

Set

$$\begin{aligned}\mathcal{AM} = &\texttt{PeriodicActivationModel}\\ &\cup \texttt{BoundActivationModel}\\ &\cup \texttt{SporadicActivationModel}\\ &\cup \texttt{ChainedActivationModel}\end{aligned}$$

i.e. $\mathcal{AM}$ is the set of all possible activation models.

The time between activation of a task and termination the runnables of the task are processed sequentially. The time needed to process a runnable is called core execution time (CET). The CET of a task instance is the sum of the CETs of its runnables. While being processed, a runnable might be paused due to a higher priority task instance being scheduled first. Furthermore, a whole task instance might be delayed due to a higher priority task instance currently being processed. Therefore, the time span between activation and termination of a task instance is in general not only its CET but CET plus delay plus paused time. This time span is called *response time*

of the task. The CETs as well as the response times for a task instance vary. Therefore, we consider intervals between a best-case execution time and worst-case execution time, and best-case response time and worst-case response time. The WCRT of a task is constrained by a deadline, which we assume to be met by any task instance. The focus of this work lies on *hard* and *firm* real-time systems (cf. [116] for categorization), overload situations are however not considered in the analyses. In both categories of systems results are not useful if they are generated after the relative deadline of a task instance as this directly influences the end-to-end functionalities. Correct results cannot be expected anymore. Furthermore, deadlines are monitored in most cases and a reset is performed if a deadline is exceeded too often. The verification for no overload situations should be performed prior to the integration process in the quality assurance on module-level.

The communication between task instances which run on the same ECU happens within a shared memory, for tasks which are located on different ECU, the exchange of data is realized with the help of buses. The latter communication is based on network signals and is discussed in Section 3.2.2. The former one can be broken down to the level of software runnables. In this work, the abstraction of software signals is used for global variables which can potentially be written and read by all runnables. Access restrictions of the software architecture are not modeled explicitly.

For the access on software signals different communication patterns are possible. The communication pattern of a task or runnable restricts the read and write accesses on software signals. The following communication pattern are considered here:

Direct access (or explicit communication)
: The executable block can read and write on its global variables at any time.

Indirect access (or implicit communication)
: The executable block works on a local copy of all global variables which is obtained when the execution of the blocks starts and written back when the execution ends. Hence, only the last written value is transmitted to global memory. The points of time for communication can either be the start and end of the task container or the start and end of the runnable entity.

Deterministic access
The executable block works on a local copy of all global variables which is obtained when the execution of the block starts and written back at a well defined point of time, e.g. the relative deadline. Again, only the last valid value is written back.

Set

$$\mathcal{CP} = \{\, \texttt{explicit}, \texttt{implicit-runnable}, \texttt{implicit-task}, \texttt{deterministic} \,\}.$$

Based on the described properties, we define the record type for a *Runnable Entitiy* as follows:

3.2.8 Definition (Runnable Entity)**.** Let $r \in \mathbb{S}$, let P be a processing unit, and let $b, w \in \mathbb{N}$. Let $I \subset \mathbb{S}$ be a set of identifiers, $O \subset \mathbb{S}$ be a set of identifiers, and let $p \in \mathcal{CP}$. The Tuple $R = (r, p, I, O)$ is called a runnable entity r processed on P with BCET b and WCET w using input signals I and output signals O through communication pattern p.

Accordingly, we define the following record type:

$$\begin{aligned}\texttt{RunnableEntity}\ (&\\ &\texttt{RunnableName} : \mathbb{S},\\ &\texttt{CommPattern} : \mathcal{CP}\\ &\texttt{Input} : \mathrm{P}(\mathbb{S}),\\ &\texttt{Ouput} : \mathrm{P}(\mathbb{S})\ )\ .\end{aligned}$$

The record type for a *Software Task* is defined as follows:

3.2.9 Definition (Software Task)**.** Let $t \in \mathbb{S}$, let $p \in \mathbb{N}$, let $a \in \mathcal{AM}$, let $q \in \mathbb{B}$, let $(r)_{i=1}^{n}$ be a list of runnables and let $d \in \mathbb{N}$. The Tuple $T = (t, p, a, q, r, d)$ is called a software task t with priority p, activation pattern a, which is preemptable if $q =$ true and has the the runnables r and deadline d assigned.

Accordingly, we define the following record type:

$$\begin{aligned}\texttt{SoftwareTask}\ (&\\ &\texttt{TaskName} : \mathbb{S},\\ &\texttt{Priority} : \mathbb{N},\end{aligned}$$

$$\begin{aligned}
&\texttt{Activation} : \mathcal{AM},\\
&\texttt{Preemtable} : \mathbb{B}\\
&\texttt{Runnables} : \texttt{RunnableEntity}^{l(*)},\\
&\texttt{Deadline} : \mathbb{N}\ ) .
\end{aligned}$$

If all runnables of a task implement the deterministic communication pattern, we call this task a LET task.

Finally, software task need to be assigned to the processing units of ECUs. This mapping also determines the CET bounds for the runnable entities.

3.2.10 Definition (Software Mapping)**.** A software mapping is a function $\xi : \texttt{SoftwareTask} \rightarrow \texttt{ProcessingUnit}$ assigning a task to a processing unit.

The CET of a runnable entity depends on the the processing unit it is processed on. For multicore platforms the allocation and scheduling of runnables on other processing units potentially also has an impact [72]. Since tasks might be shifted to different processing units over the development cycles, we introduce *CET Measurements* allowing to model best and worst case execution times based on software mappings. Note, that the models developed in Chapter 4 and Chapter 5 only give upper bounds if the CETs provided for the runnables are safe upper bounds.

3.2.11 Definition (CET Measurement)**.** A CET measurement is a function $(\texttt{SoftwareTask} \rightarrow \texttt{ProcessingUnit}) \rightarrow (\texttt{RunnableEntity} \times \mathbb{N} \times \mathbb{N})$ taking a software mapping and returning the BCET and WCET of each runnable.

The software task record type also allows to model black-box tasks, if no runnables are assigned to it. Black box tasks are tasks where no implementation details are known, e.g. because they are developed by a supplier. In this case, analyses must work with the assumption, that the whole time from activation to deadline is used to generate results.

Although the delay of a physical input at sensing or the inertia of actuators are not subject to analysis here, they can nevertheless be modeled with the help of software tasks. For example, a sensor producing a value can be modeled with a runnable entity producing the signal which needs

no time to be processed. The temporal properties of the sensed value can be modeled in the activation model of the task. Accordingly, actuators can be model as chained tasks with one runnable consuming the signal for actuation. The time to react can be modeled in the execution time of this runnable. Sensor and actuator tasks should therefore be assigned to dedicated processing units to ensure that they do not interfere with software tasks.

3.2.2 ECU Networks

To connect ECUs to the communication clusters of a CPS, a model for the connecting networks is needed. Although only an approach for CAN-FD is presented in this work (see Chapter 5), for generality and with a view to future developments, we consider CAN, CAN-FD, FlexRay, Ethernet, MOST, and LIN as possible types of fieldbuses. Therefore, set $\mathcal{BT} = \{\texttt{CAN}, \texttt{CAN-FD}, \texttt{FR}, \texttt{Eth}, \texttt{MOST}, \texttt{LIN}\}$ be the set of bus types. This selection is motivated and explained in Chapter 1. The model for a fieldbus is a collection of network channels of a certain bus type.

3.2.12 Definition (Fieldbus)**.** Let $n \in \mathbb{S}$ be an identifier, let $t \in \mathcal{BT}$, and let $C \subset \mathbb{N}$. The tuple (n, t, C) is called a fieldbus n of type t with channels C.

Accordingly, we define the following record type:

$$\begin{aligned}\texttt{FieldBus}\,(&\\ &\texttt{BusName} : \mathbb{S},\\ &\texttt{Type} : \mathcal{BT},\\ &\texttt{Channels} : \mathrm{P}(\mathbb{N})\\ )\,.&\end{aligned}$$

With the help of fieldbuses we can now connect ECUs to ECU networks in our model. Directed graphs are used for the formal representation of a network.

3.2.13 Definition (ECU Network)**.** Let $\mathcal{E}$ be a set of electronic control units and let $\mathcal{N}$ be a set of fieldbuses. The *ECU Network* formed $\mathcal{N}$ by and $\mathcal{E}$ is a directed graph $G = (\mathcal{E}, A)$ with surjective edge labeling function $l_A : A \to \mathcal{N}$ having the following properties:

An edge exists between two communicating ECUs.

$$\begin{aligned}&\forall(w_1, P_1, U_1), (w_2, P_2, U_2) \in \mathcal{E} : w_1 \neq w_2 \wedge \rightarrow \\ &\forall(n_1, c_1, s_1, \texttt{out}), (n_2, c_2, s_2, \texttt{in}) \in P_1 \cap P_2 : \\ &((w_1, P_1, U_1), (w_2, P_2, U_2)) \in A\end{aligned} \tag{3.2.14}$$

All connecting fieldbuses are available and edges are labeled accordingly.

$$\begin{aligned}&\forall(w_1, P_1, U_1), (w_2, P_2, U_2) \in \mathcal{E} : w_1 \neq w_2 \rightarrow \\ &\forall(n_1, c_1, s_1, d_1), (n_2, c_2, s_2, d_2) \in P_1 \cap P_2 : \\ &\exists(n, t, C) \in \mathcal{N} : n = n_1 \wedge c_1 \in C \wedge \\ &l_A\left(((w_1, P_1, U_1), (w_2, P_2, U_2))\right) = (n, t, C)\end{aligned} \tag{3.2.15}$$

ECUs communicate using network packets. A network packet can be subject to end-to-end encryption (E2EE) or secure onboard communication (SecOC). SecOC means, that data is sent with an additional authentication added to messages to assure that messages are generated by a specific sender and have not been altered. Encryption and protection of messages need additional processing for decryption and checking authentication and therefore possibly impact the timing of messages. Furthermore, signals have different properties for triggering the transmission of an encapsulating PDU. The possible *transfer properties* are: $\mathcal{TP} = \{\, \texttt{always}, \texttt{never}, \texttt{on-change} \,\}$. If a signal has transfer property `always`, any writing access on the variable will trigger a transmission, if the transfer property is set to be `on-change`, transmission is only initiated if the value actually changed. No transmission is initiated if the property is configured to be `never`.

3.2.16 Definition (Network Signal)**.** Let $s \in \mathbb{S}$ be an identifier, let $l \in \mathbb{N}$, and let $t \in \mathcal{TP}$. The tuple (s, l, t) is called network signal s with transmission property t.

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{NetworkSignal}\ (\\ &\qquad\qquad \texttt{SignalName} : \mathbb{S},\\ &\qquad\qquad \texttt{Length} : \mathbb{N},\\ &\qquad\qquad \texttt{Transmission} : \mathcal{TP}\ )\ .\end{aligned}$$

In the next step, signals a are put into network packets. Such a packet can have one of the following trigger types: $\mathcal{TT}^{\mathrm{P}} = \{\, \texttt{always}, \texttt{never} \,\}$. The length of a network packet can be derived from the summed length of the contained signals plus the overhead for the protocol overhead information. The latter are depending on the chosen protocol and implementation. Therefore, they this information is not stored individually for each packet.

3.2.17 Definition (Network Packet)**.** Let $p \in \mathbb{S}$ be an identifier, let $a \in \mathcal{AM}$, let $e \in \mathbb{B}$, let $s \in \mathbb{B}$, let $t \in \mathcal{TT}^{\mathrm{P}}$, and let $S \subset \texttt{NetworkSignal}$ be a set of network signals. The tuple (p, a, e, s, t, S) is called network packet p containing network signals S which triggers the transmission of data according to t, has E2EE if $e =$ true, and is subject to SecOC if $p =$ true.

Accordingly, we define the following record type:

$$
\begin{aligned}
&\texttt{NetworkPacket}\ (\\
&\qquad \texttt{PacketName} : \mathbb{S},\\
&\qquad \texttt{SendingTimer} : \mathcal{AM},\\
&\qquad \texttt{Length} : \mathbb{N},\\
&\qquad \texttt{Encrypted} : \mathbb{B},\\
&\qquad \texttt{Protected} : \mathbb{B},\\
&\qquad \texttt{Triggers} : \mathcal{TT}^{\mathrm{P}},\\
&\qquad \texttt{Signals} : \mathrm{P}(\mathbb{S})\ )\ .
\end{aligned}
$$

To allow arbitrary hierarchies of network packets, network packets can be encapsulated in containers. Such a container network packet has different so-called sending triggers. It can be send due to a threshold for the filling-level being exceeded, due to a timeout, or due to direct sending request. The contained packets are taken from buffers and the collection semantics of the container specify which value is used for transmission. This can either be the last valid value or the first value after last transmission. Therefore, set $\mathcal{TT}^{\mathrm{C}} = \{\, \texttt{first}, \texttt{none} \,\}$ the set of possible trigger reasons due to contained network packets and set $\mathcal{CS} = \{\, \texttt{last-is-best}, \texttt{queued} \,\}$ the possible collection semantics.

3.2.18 Definition (Container Network Packet)**.** Let $c \in \mathbb{S}$ be an identifier, let $t \in \mathbb{N}$, let $t \in \mathbb{N}$, let $t \in \mathcal{TT}^{\mathrm{C}}$, let $s \in \mathcal{CS}$, and let $C \subset \texttt{NetworkPacket} \cup$

`ContainerNetworkPacket`, The tuple (c, t, s, C) is called container network packet c with contained trigger type t, collection semantics s and containees C.

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{ContainerNetworkPacket}\ (\\ &\qquad\qquad\texttt{ContainerName} : \mathbb{S},\\ &\qquad\qquad\texttt{TransmissionTrigger} : \mathcal{TT}^{\mathrm{C}}\\ &\qquad\qquad\texttt{ColletionSemantics} : \mathcal{CS}\\ &\qquad\qquad\texttt{Containees} : \mathrm{PDUs}\ )\end{aligned}$$

with $\mathrm{PDUs} = \mathrm{P}(\texttt{NetworkPacket} \cup \texttt{ContainerNetworkPacket})$.

3.2.19 Definition (Network Resource)**.** Let $r \in \mathbb{S}$, let $P \subset \texttt{NetworkPacket} \cup \texttt{ContainerNetworkPacket}$, The Tuple $R = (r, P)$ is called a network resource r

Accordingly, we define the following record type:

$$\begin{aligned}&\texttt{NetworkResource}\ (\\ &\qquad\qquad\texttt{ResName} : \mathbb{S},\\ &\qquad\qquad\texttt{Provided} : \mathrm{PDUs}\ )\end{aligned}$$

with $\mathrm{PDUs} = \mathrm{P}(\texttt{NetworkPacket} \cup \texttt{ContainerNetworkPacket})$.

In the following, $\mathrm{signals} : A \to \mathrm{P}(\texttt{NetworkSignal})$ with

$$A = \texttt{NetworkResource} \cup \texttt{NetworkPacket} \cup \texttt{ContainerNetworkPacket}$$

denotes the function collecting all network signals assigned to a network resource. It is recursively defined as

$$\mathrm{signals}(a) = \begin{cases} \bigcup_{p \in P} \mathrm{signals}(p) \\ \quad \text{if } a = (r, P) \in \texttt{NetworkResource} \\ \bigcup_{p \in C} \mathrm{signals}(p) \\ \quad \text{if } a = (c, t, s, C) \in \texttt{ContainerNetworkPacket} \\ S \\ \quad \text{if } a = (p, a, e, s, t, S) \in \texttt{NetworkPacket} \end{cases} \tag{3.2.20}$$

Network resources need additional information depending on the protocol, e.g. for CAN-FD an identifier to derive the priority for is needed. Formally, we assume that a function exists which returns all *timing relevant attributes* for a network resource in the form of a set of key-value-pairs. An implementation based on a relational database is presented in the section Section 3.3.

The mapping of network resources to network channels is defined through the outgoing communication ports of the ECUs. However, two things still need to be brought together: software tasks need to control the network controller for the transmission and reception of network resources, and the translation of software signals to network signals and vice versa. The mapping of software signals and network signals is handled by consistent naming, i.e. we assume that a software signal l and network signal s contain the same information if they have the same name. In the real system, the transition between a software signal and network signal is happening when data from the global memory of the ECU is copied into the buffers used in the communication stacks for bus communication. The buffer operations are performed by the so-called *Communication Tasks*.

3.2.21 Definition (Communication Task Mapping)**.** A communication task mapping $\psi : \texttt{CommunicationPort} \rightarrow \texttt{SoftwareTask}$ is a function mapping communication ports to software tasks.

Finally, the formal model is completed with the definition of the *Communication Cluster*.

3.2.22 Definition (Communication Cluster)**.** Let $G = (\mathcal{E}, A, l_A)$ be a ECU network. Let $\mathcal{R}$ be a set of network resources and ψ be a communication task mapping. Let $\mathcal{T}$ be a set of tasks and ξ be a software mapping.

The tuple $\Gamma = (G, \mathcal{R}, \mathcal{T}, \xi, \psi)$ is called *Communication Cluster*. Γ is *well-formed* if all identifiers except of network signals are unique and

Γ is complete with regards to network resources:

$$\forall (r, p) \in \mathcal{R} \colon \exists (w, P, U) \in \mathcal{E} \colon \exists (n, c, r', d) \in P \colon r = r' \wedge d = \texttt{out} \quad (3.2.23)$$

$$\forall (w, P, U) \in \mathcal{E} \colon \forall (n, c, r, d) \in P \colon \exists (r', p) \in \mathcal{E} \colon r = r' \quad (3.2.24)$$

Γ is complete with regards to communication task mapping:

$$\text{domain}(\psi) = \mathcal{R} \wedge \text{codomain}(\psi) = \mathcal{T} \tag{3.2.25}$$

Γ is complete with regards to signal mappings:

$$\begin{aligned}
&\forall (w, P, U) \in \mathcal{E} \colon \forall (n, c, r, d) \in P \colon d = \texttt{out} \Rightarrow \\
&\quad \forall res \in \mathcal{R}_{|R \mapsto \texttt{Name}(R) = r} \colon \forall sig \in \text{signals}(res) \colon \\
&\quad \exists (e, p, I, O) \in \text{set}(\texttt{SoftwareTask.Runnables}(\psi(n, c, r, d))) \colon \\
&\qquad sig \in O
\end{aligned} \tag{3.2.26}$$

$$\begin{aligned}
&\forall (w, P, U) \in \mathcal{E} \colon \forall (n, c, r, d) \in P \colon d = \texttt{in} \Rightarrow \\
&\quad \forall res \in \mathcal{R}_{|R \mapsto \texttt{Name}(R) = r} \colon \forall sig \in \text{signals}(res) \colon \\
&\quad \exists (e, p, I, O) \in \text{set}(\texttt{SoftwareTask.Runnables}(\psi(n, c, r, d))) \colon \\
&\qquad sig \in I
\end{aligned} \tag{3.2.27}$$

3.2.3 Cause-effect Chains

Recalling our main objective, we want to use a set of cause-effect chains to represent the data flows which are possible to occur for a system function. Different ECUs and networks are relevant when it comes to implement a system function as depicted in Figure 3.3. Multiple cause-effect chains describe the run-time behavior of a system function. In this section the formal encoding of cause-effect chains in our model is presented, however, first we discuss which mathematical structure fits best for this purpose.

With the system model defined above, this is achieved in two steps. Firstly we use directed graphs to describe the data dependencies between the different parts of the chain. In a second step, these graphs are flattened to cause effect chains which are are a sequences of of runnable entities, each consuming and producing a software signal. If the a cause-effect chain spans over multiple ECUs we call it *Distributed Cause-effect Chain*.

3.2.28 Definition (Data Dependency Graph)**.** Let $\mathcal{R}$ be a set of runnables and let $\mathcal{S} \subset \mathbb{S}$ be a set of identifiers. A *Data Dependency Graph* over $\mathcal{R}$ and $\mathcal{S}$ is a directed graph $G = (V, A)$ with label function $l_V : V \to \mathcal{R}$ and $l_A : A \to \mathcal{S}$.

The cause-effect chains belonging to a system function are finite walks

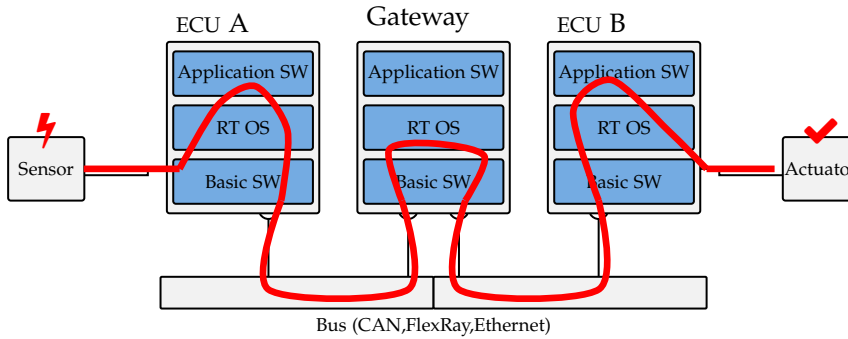


Figure 3.3. End-to end latency of an cause-effect chain (Source: [131])

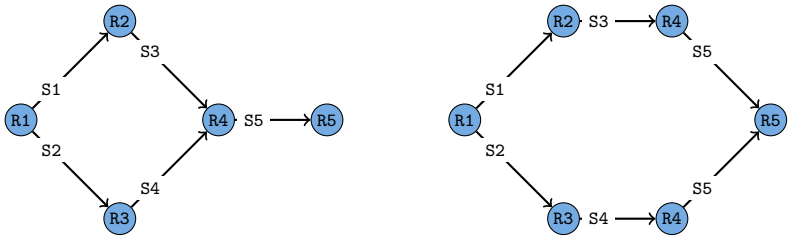
in the data dependency graph where each visited node is labeled with a runnable mapped to one ECU. However, the analysis of end-to-end latencies for cause-effect chains relies on flat sequences. The problem of data flows which split and join add an additional dimension of complexity. Since the set of walks is possibly infinite if a cycle exists, an engineer with knowledge about the implementation has to consulted on how often a node can be visited. The reason why we consider the data dependency graph in a first step is twofold: (1) it is a more convenient way to specify chains from a functional point of view and (2) a set of chains does lead to a unique graph.

Consider graphs depicted in Figure 3.4b and Figure 3.4a, to obtain a latency estimation for the functions described the following flat chains need to be analyzed: $\{ (R_1, R_2, R_4, R_5), (R_1, R_3, R_4, R_5) \}$. Therefore, from a non-functional timing perspective, the graphs make no difference, functionally however the situation might lead to different results.

Cause-effect chains are defined as follows:

3.2.29 Definition (Cause-effect Chain)**.** Let $\Gamma = ((\mathcal{E}, A, l_A), \mathcal{R}, \mathcal{T}, \xi, \psi)$ be a well-formed communication cluster. A *Cause-effect Chain* in Γ is a sequence of tuples $(c, R, p)_{i=1}^{n}$ where $c_i, p_i \in \mathbb{S}$ and $R_i \in$ `RunnableEntites` for all $i \in \{1, \ldots, n\}$ and the following holds:

$$\forall i \in \{1, \ldots, n-1\} : p_i = s_{i+1} \tag{3.2.30}$$



(a) Graph with early Join **(b)** Graph with late Join

Figure 3.4. Exemplary Graphs

$$\forall i \in \{1,\dots,n\} : c_i \in \texttt{RunnableEntity.Input}(r_i) \tag{3.2.31}$$

$$\forall i \in \{1,\dots,n\} : p_i \in \texttt{RunnableEntity.Output}(r_i) \tag{3.2.32}$$

$$\begin{aligned}&\exists (w,P,U) \in \mathcal{E} \colon \forall i \in \{0,\dots,n\} : \exists u \in U \colon \exists T \in \mathcal{T} \colon \\ &\quad \xi(T) = U \wedge R_i \in \texttt{Task.Runnables}(T)\end{aligned} \tag{3.2.33}$$

In general, not all runnables of a walk in a data dependency graph need to be mapped to the same ECU. If this is not the case, the cause-effect chain is distributed over multiple ECUs. The sub-chains on the different ECUs are connected via a network signal.

3.2.34 Definition (Distributed Cause-effect Chain)**.**
Let $\Gamma = ((\mathcal{E}, A, l_A), \mathcal{R}, \mathcal{T}, \xi, \psi)$ be a well-formed communication cluster. A *Distributed Cause-effect Chain* in Γ is a sequence $(C)_{i=1}^{n}$ where C_i a cause-effect chain for all $i \in \{1,\dots,n\}$. We use double subscript to denote the segments of the chain, more precisely for more convenient notation we assume the form:

$$\begin{aligned}&(((c_{1,1}, R_{1,1}, p_{1,1}), \dots, (c_{1,m_1}, R_{1,m_1}, p_{1,m_1})),\\ &\quad\vdots\\ &((c_{n,1}, R_{n,1}, p_{n,1}), \dots, (c_{n,m_n}, R_{n,m_n}, p_{n,m_n}))\end{aligned}$$

Then, following must hold for all $i \in \{1,\dots,n\}$:

$$\forall i \in \{1,\dots,n-1\} : p_{i,m_i} = c_{i+1,1} \tag{3.2.35}$$

$$\begin{aligned}
&\forall i \in \{1, \ldots, n-1\} : \exists (w_1, P_1, U_1), (w_2, P_1, U_2) \in \mathcal{E} : \\
&(\exists (b, c, r, d) \in P_1 : d = \texttt{out} \wedge (b, c, r, \texttt{in}) \in P_2 \wedge \\
&\quad \exists (s, l, t) \in \text{signals} : s = (p_{m_i})_i) \wedge \\
&(\exists u_1 \in U_1 : \exists t_1 \in \mathcal{T} : \xi(t_1) = u_1 \wedge R_{i,m_i} \in \texttt{Task.Runnables}(T_1)) \wedge \\
&(\exists u_2 \in U_2 : \exists t_2 \in \mathcal{T} : \xi(t_2) = u_2 \wedge R_{i,1} \in \texttt{Task.Runnables}(T_2))
\end{aligned} \tag{3.2.36}$$

In order to work with distributed cause-effect chains conveniently in the following let rxPort(s) denote the sending communication port of a signal in the course of a distributed cause-effect chain and let txPort(s) denote the receiving communication port of a signal in the course of a distributed cause-effect chain.

Latency and Jitter

For the definition of end-to-end latencies, instances of the modeled objects need to be considered. More precisely, we consider task instances (or jobs) repeatedly spawned and processed. For each task instance all runnable entities of that task are processed. We call this an instance of the runnable, formally $\tau^R[i]$ denotes the processing of runnable $R \in \mathcal{R}$ in the i^{th} instance of its task container. For each pair of runnable instance and signal we introduce two points of time for each input signal framing the read interval and two points of time for each output signal framing the write interval.

3.2.37 Definition (Read Interval)**.** Let $R \in \texttt{RunnableEntity}$, $i \in \mathbb{N}$ and $S \in \texttt{RunnableEntity.Input}(R)$, $\underline{\text{Read}}\left(\tau^R[i], S\right)$ denotes the first point of time S is possibly read by τ_i^R and $\overline{\text{Read}}\left(\tau^R[i], S\right)$ denotes the last point of time S is possibly read by τ_i^R.

3.2.38 Definition (Write Interval)**.** Let $R \in \texttt{RunnableEntity}$, $i \in \mathbb{N}$ and $S \in \texttt{RunnableEntity.Output}(R)$, $\underline{\text{Write}}\left(\tau^R[i], S\right)$ denotes the first point of time S is possibly written by τ_i^R and $\overline{\text{Write}}\left(\tau^R[i], S\right)$ denotes the last point of time S is possibly written by τ_i^R.

For the transmission of information in a chain distributed over multiple ECUs, communication intervals additionally need to be considered. They are defined for a combination of communication port and network signal.

3.2.39 Definition (Communication Interval)**.** Let $P_1, P_2 \in$ `CommunicationPort` and let $S \in$ `NetworkSignal`. $\underline{\text{Trans}}\,(P_1, S, P_2)$ denotes the minimum amount of time required to transfer a change in signal S from P_1 to P_2 and $\overline{\text{Trans}}\,(P_1, S, P_2)$ denotes the maximum amount of time required to transfer a change in signal S from P_1 to P_2. Both times are assumed to include the potential drift between the clocks of the ECUs.

Based on read and write intervals we define two different end-to-end latency semantics: data age denotes the interval between first change and last reaction, response time denotes the interval between first change and first reaction. The data age and the response time of the a cause-effect chain on task-level are depicted in Figure 3.5.

Due to offsets, additional delays can occur during the startup phase, that is why we only consider jobs start at index 1. This has an additional advantage for the definition of the response time of a cause-effect chain which would be more complicated if we started at index 0.

3.2.40 Definition (Data age of a cause-effect chain)**.** The data age of a cause-effect chain $(c, R, p)_{i=1}^{n}$ is the length of the time span between any change c_1 and the last associated reaction observed in p_n, more precisely, for a sequence $(j_1, \ldots, j_n)$ for which $j_i > 0$

$$\overline{\text{Read}}\left(\tau^{R_{i+1}}\left[j_{i+1}\right], c_{i+1}\right) \leqslant \underline{\text{Write}}\left(\tau^{R_i}\left[j_i + 1\right], p_i\right) \quad \forall i \in \{1, \ldots, n-1\} \tag{3.2.41}$$

$$\underline{\text{Read}}\left(\tau^{R_{i+1}}\left[j_{i+1}\right], c_{i+1}\right) \geqslant \underline{\text{Write}}\left(\tau^{R_i}\left[j_i\right], p_i\right) \qquad \forall i \in \{1, \ldots, n-1\} \tag{3.2.42}$$

the data age is

$$\overline{\text{Write}}\left(\tau^{R_n}\left[j_n\right], p_n\right) - \underline{\text{Read}}\left(\tau^{R_1}\left[j_1\right], c_1\right) .$$

The maximum data age is the highest possible difference in time which can be obtained with any sequence fulfilling Equation 3.2.41 and Equation 3.2.42.

3.2.43 Definition (Response time a cause-effect chain)**.** The response time of a cause-effect chain $(c, R, p)_{i=1}^{n}$ is the length of the time span between any change c_1 and the first associated reaction observed in p_n, more

precisely, for a sequence $(j_1, \ldots, j_n)$ for which $j_i > 0$ and

$$\underline{\text{Write}}\left(\tau^{R_i}\left[j_i\right], p_i\right) \geqslant \overline{\text{Read}}\left(\tau^{R_{i+1}}\left[j_{i+1}-1\right], c_{i+1}\right) \quad \forall i \in \{1, \ldots, n-1\} \tag{3.2.44}$$

$$\overline{\text{Write}}\left(\tau^{R_i}\left[j_i\right], p_i\right) \leqslant \overline{\text{Read}}\left(\tau^{R_{i+1}}\left[j_{i+1}\right], c_{i+1}\right) \quad \forall i \in \{1, \ldots, n-1\} \tag{3.2.45}$$

the response time is

$$\overline{\text{Write}}\left(\tau^{R_n}\left[j_n\right], p_n\right) - \underline{\text{Read}}\left(\tau^{R_1}\left[j_1\right], c_1\right) .$$

The maximum response time is the highest possible difference in time which can be obtained with any sequence fulfilling Equation 3.2.44 and Equation 3.2.45.

For the end-to-end latencies of distributed cause-effect chains, additionally the time to transfer network signals needs to be considered. We assume that the clocks on one ECU are sufficiently synchronized in the sense that a fixed and low bound on the clock drift between the processing resources can be given. For distributed chains, this is not necessarily the case. The clocks of two ECUs might drift. However, this is included in the definition of the communication interval of a signal. Therefore, the end-to-end latencies of distributed cause-effect chains are defined as in Definition 3.2.49 and Definition 3.2.46.

3.2.46 Definition (Data age of a distributed cause-effect chain)**.** Let $\Gamma = ((\mathcal{E}, A, l_A), \mathcal{R}, \mathcal{T}, \xi, \psi)$ be a well-formed communication cluster and $(C)_{i=1}^{n}$ be a distributed cause-effect chain in Γ. Let $(S)_{i=1}^{n-1}$ be the sequence of network signals exchanges form i to $i+1$ in the course of the distributed cause-effect chain.

The data age of the distributed cause-effect chain is the length of the time span between any change $c_{1,1}$ and the last associated reaction observed in p_{n,m_n}, more precisely, for a sequence $(j_{1,1}, \ldots, j_{1,m_1}, \ldots, j_{n,1}, \ldots, j_{n,m_n})$ for which Equation 3.2.41 and Equation 3.2.42 hold for all cause-

effect chains and furthermore for all $i \in \{1, \ldots, n-1\}$

$$\begin{aligned} &\overline{\text{Read}}\left(\tau^{R_{i+1,1}}\left[j_{i+1,1}\right], c_{i+1,1}\right) \leqslant \\ &\underline{\text{Write}}\left(\tau^{R_{i,m_i}}\left[j_{i,m_i}+1\right], p_{i,m_i}\right) + \overline{\text{Trans}}\left(\text{rxPort}(S_i), S_i, \text{txPort}(S_i)\right) \end{aligned} \tag{3.2.47}$$

$$\begin{aligned} &\underline{\text{Read}}\left(\tau^{R_{i+1,1}}\left[j_{i+1,1}\right], c_{i+1,1}\right) \geqslant \\ &\underline{\text{Write}}\left(\tau^{R_{i,m_i}}\left[j_{i,m_i}\right], p_{i,m_i}\right) + \underline{\text{Trans}}\left(\text{rxPort}(S_i), S_i, \text{txPort}(S_i)\right) \end{aligned} \tag{3.2.48}$$

the data age is

$$\overline{\text{Write}}\left(\tau^{R_{n,m_n}}\left[j_{n,m_n}\right], p_{n,m_n}\right) - \underline{\text{Read}}\left(\tau^{R_{1,1}}\left[j_{1,1}\right], c_{1,1}\right) .$$

The maximum data age is highest possible difference in time which can be obtained with any sequence fulfilling the requirements.

3.2.49 Definition (Response time of a distributed cause-effect chain)**.** Let $\Gamma = ((\mathcal{E}, A, l_A), \mathcal{R}, \mathcal{T}, \xi, \psi)$ be a well-formed communication cluster and $(C)_{i=1}^{n}$ be a distributed cause-effect chain in Γ. Let $(S)_{i=1}^{n-1}$ be the sequence of network signals exchanges form i to $i+1$ in the course of the distributed cause-effect chain.

The response time of the distributed cause-effect chain is the length of the time span between any change $c_{1,1}$ and the first associated reaction observed in p_{n,m_n}, more precisely, for a sequence $(j_{1,1}, \ldots, j_{1,m_1}, \ldots, j_{n,1}, \ldots, j_{n,m_n})$ for which Equation 3.2.44 and Equation 3.2.45 hold for all cause-effect chains and furthermore for all $i \in \{1, \ldots, n-1\}$

$$\begin{aligned} &\underline{\text{Write}}\left(\tau^{R_{i,m_i}}\left[j_{i,m_i}\right], p_{i,m_i}\right) + \underline{\text{Trans}}\left(\text{rxPort}(S_i), S_i, \text{txPort}(S_i)\right) \geqslant \\ &\overline{\text{Read}}\left(\tau^{R_{i+1,1}}\left[j_{i+1,1}-1\right], c_{i+1,1}\right) \end{aligned} \tag{3.2.50}$$

$$\begin{aligned} &\overline{\text{Write}}\left(\tau^{R_{i,m_i}}\left[j_{i,m_i}\right], p_{i,m_i}\right) + \overline{\text{Trans}}\left(\text{rxPort}(S_i), S_i, \text{txPort}(S_i)\right) \leqslant \\ &\overline{\text{Read}}\left(\tau^{R_{i+1,1}}\left[j_{i+1,1}\right], c_{i+1,1}\right) \end{aligned} \tag{3.2.51}$$

the response time is

$$\overline{\text{Write}}\left(\tau^{R_{n,m_n}}\left[j_{n,m_n}\right], p_{n,m_n}\right) - \underline{\text{Read}}\left(\tau^{R_{1,1}}\left[j_{1,1}\right], c_{1,1}\right) .$$

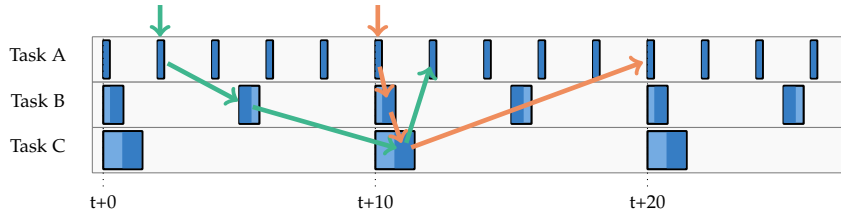


Figure 3.5. Response Time (green) and Data Age (orange) of the Chain Task A → Task B → Task C → Task A. Processing of task instances is in darker blue.From activation to start and from end to deadline is in lighter blue.

The maximum response time is highest possible difference in time which can be obtained with any sequence fulfilling the requirements.

Finally, the end-to-end latency for any system function with a given data dependency graph is obtained by analyzing all cause-effect chains regarding the desired end-to-end latency semantic and determine the maximum of all latencies.

3.2.4 Example

As mentioned in Section 1.3 one of the most important criteria for our model is the fitness for practical use. As also noted, the CPCM of automotive powertrains provides a telling example for this. Therefore, in Example 3.2.52 the topology of an exemplary powertrain communication cluster is encoded. In the following chapters, examples of industrial scale task-sets and network communication designs are mentioned. Showing an encoding would go beyond the size of a good example, however when comparing the attributes needed to create the CSP model and the attributes specified above, the reader will see that all necessary information is available.

3.2.52 Example (Automotive CPS - Pt Domain). Consider the communication cluster depicted in Figure 3.6. To model the cluster, the following ECUs are needed:

$$CPC = \texttt{ECU}(\texttt{PtGateway}, CP_{\text{CPC}}, PU_{\text{CPC}}),$$

$$
\begin{aligned}
INV_1 &= \texttt{ECU}(\texttt{Inverter1}, CP_{\text{Inv1}}, PU_{\text{Inv1}}),\\
INV_2 &= \texttt{ECU}(\texttt{Inverter2}, CP_{\text{Inv2}}, PU_{\text{Inv2}}),\\
TCM &= \texttt{ECU}(\texttt{Transmission}, CP_{\text{TCM}}, PU_{\text{TCM}}),\\
ECM &= \texttt{ECU}(\texttt{Engine}, CP_{\text{ECM}}, PU_{\text{ECM}})
\end{aligned}
$$

With $CP_\star$ being the communication ports of each ECU and the following sets of processing units:

$$
\begin{aligned}
PU_{\text{CPC}} &= \{\texttt{ProcessingUnit}(1, \texttt{SPP}), \texttt{ProcessingUnit}(2, \texttt{SPP}),\\
&\qquad \texttt{ProcessingUnit}(3, \texttt{SPP}), \texttt{ProcessingUnit}(4, \texttt{SPP})\}\\
PU_{\text{Inv1}} &= \{\texttt{ProcessingUnit}(1, \texttt{SPP}), \texttt{ProcessingUnit}(2, \texttt{SPP})\}\\
PU_{\text{Inv2}} &= \{\texttt{ProcessingUnit}(1, \texttt{SPP}), \texttt{ProcessingUnit}(2, \texttt{SPP})\}\\
PU_{\text{TCM}} &= \{\texttt{ProcessingUnit}(1, \texttt{SPP}), \texttt{ProcessingUnit}(2, \texttt{SPP}),\\
&\qquad \texttt{ProcessingUnit}(3, \texttt{SPP}), \texttt{ProcessingUnit}(4, \texttt{SPP})\}\\
PU_{\text{ECM}} &= \{\texttt{ProcessingUnit}(1, \texttt{SPP}), \texttt{ProcessingUnit}(2, \texttt{SPP}),\\
&\qquad \texttt{ProcessingUnit}(3, \texttt{SPP}), \texttt{ProcessingUnit}(4, \texttt{SPP})\}
\end{aligned}
$$

The ECU network $G = (\mathcal{E}, \mathcal{A})$ with

$$
\begin{aligned}
E &= (CPC, INV_1, INV_2, TCM, ECM)\\
V &= ((CPC, ECM), (CPC, TCM), (CPC, INV_1), (CPC, INV_2))
\end{aligned}
$$

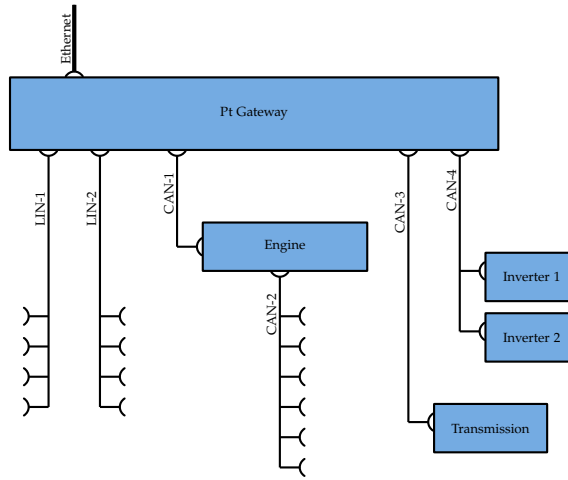


Figure 3.6. Network Topology of the powertrain (exemplary)

3.3 Data Collection and Storage

The data needed to model the system's temporal behavior as described above is usually not directly available, especially when it comes to create a model for an entire CPS. Communication design and software architecture are complex tasks and the development is distributed over several teams. Therefore, it is likely that data for ECUs and network comes from different sources. A single perspective is however not sufficient to obtain precise estimations of the temporal behavior. Consequently, the data needs to be aggregated to a consistent model even if it is divided again for analysis. The challenges of integration are similar to the challenges of the integration of the real system. At the borders of responsibilities, the integrity of the data has to be checked. Tool support with a sophisticated storage solution has become indispensable with today's ever shorter development cycles. The challenges of integrity checks and consistency are not discussed here, however, Figure 3.7 shows a entity-relationship model for a database allowing to store all information needed for the formal model presented in the previous section.

The records types can almost directly be translated to the according relationships. Besides, the database model has the following features:

Software Release The software tasks of an ECU are weak entities, connected to both, the ECU and a relationship for software releases. Thus, software artifacts can only be added if an ECU and software release exist and a new runnable is added for each release. Although this looks like a huge amount of data at first glance, the overhead compared to a mapping of un-versioned software artifacts to releases is relatively low since the amount of attributes of the software artifacts is low. On the upside, a potentially complicated versioning is avoided.

NCD Version Analogously to ECUs and their software, networks and communication artifacts are brought together through network communication design (NCD) versions.

Referencing Chains Since runnables with their software signals and network signals are weak entities connected to software and NCD releases respectively, referencing them through the hierarchy of identifiers means a lot of maintenance effort. Since these references are needed in cause-effect chains, they are implemented referencing only the artifact's identifier instead. This is indicated by the bright highlighting of the respective relationships in Figure 3.7.

Timing Relevant Attributes To specialize communication artifacts regarding bus-dependent attributes, the notion of timing relevant attributes is added. These relationships are not shown in Figure 3.7 but can be found in Appendix 6.6. They are tuples containing the key of the communication artifact, an additional key for the attribute name and a field for the value of the attribute.

A relational model described by *create*-scripts for the relationships depicted in Figure 3.7 can be found in Appendix 6.6.

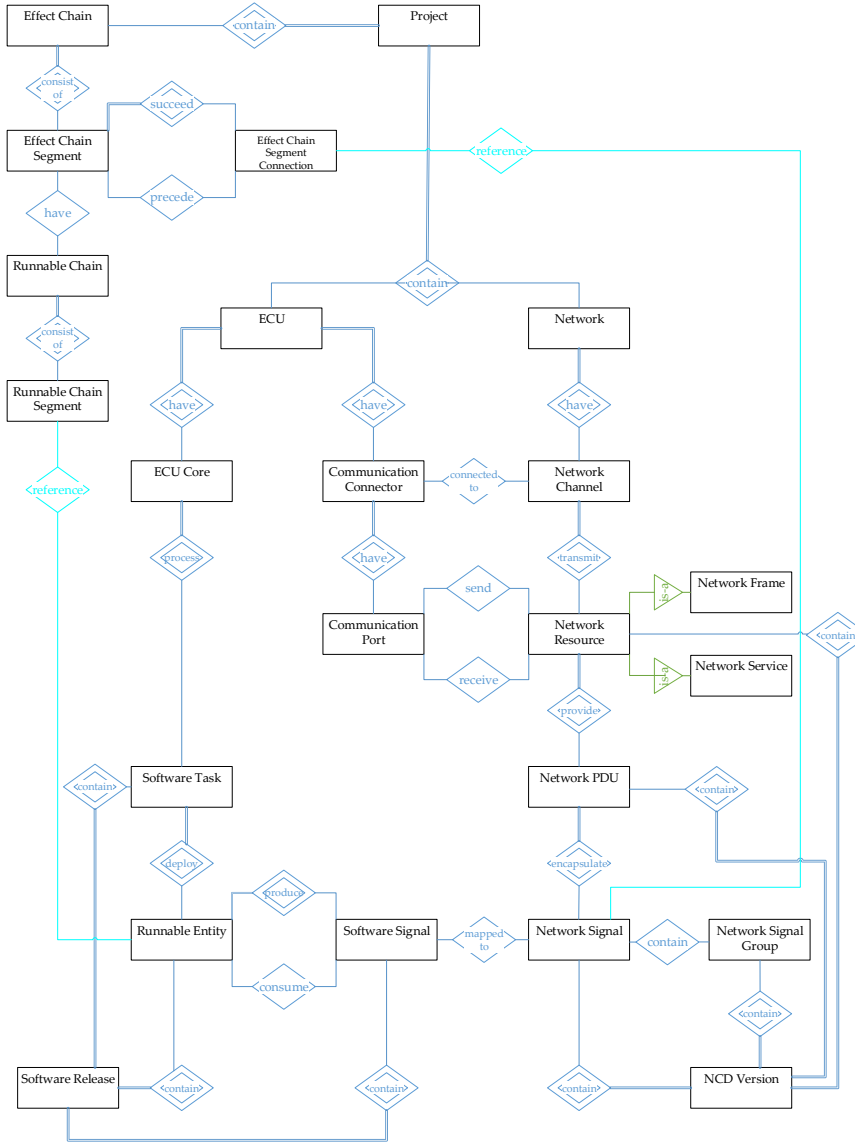


Figure 3.7. Database model

3.4 Summary and Conclusion

In this chapter we discussed a system model which allows to describe the temporal behavior of the *cyber*-part of automotive CPS. It is close to the actual implementation of automotive systems but also allows to omit details where information might not be available. Attention is payed to the iterative procedures in automotive development, possibly running at different speeds for different parts of the system. It is compatible with but not tied to the AUTOSAR methodology.

Chapter 4

Latencies of Cause-effect Chains

In the previous chapter, a system model to describe instances of end-to-end latency analyses was introduced. The component of the model which is the subject of the analysis are *cause-effect chains*. They consist of sequences of runnables which are assigned to tasks for processing and the activation pattern of the task determines how frequently a runnable is processed. The idea of the task-level constraint model for end-to-end timing analysis is to enumerate all possible relative offsets between the task instances of the chain with all possible points of time for data accesses. Figure 4.1 shows the possible relative offsets between for the chain Task A→Task B→Task C.

Since support for different levels of details for the task is important during the development process we start with a task-level model in Section 4.2. Subsequently, in Section 4.3, we add detailed information about runnables of the tasks where it is possible and useful in terms of precision gained for the analysis. Afterwards, the correctness of the model, two optimizations, and the solving process are described. Finally, the results of different experiments to show scalability and to compare precision are presented. We start with an overview of related work provided in the following section.

4.1 Related work

End-to-end timing analysis has a long history and accordingly, a lot of work has been presented in this field. Besides very abstract models like Petri nets, which were initially presented in [100] to model chemical processes, and timed automatons presented in 1991 [3], work comes especially from the area of scheduling theories. Over time, a plethora of work around end-to-end latency analyses for different kinds of task sets have been presented.

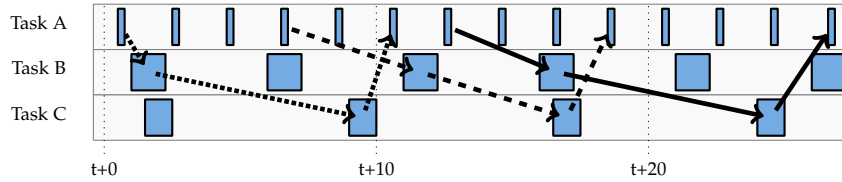


Figure 4.1. Example of a Task-level Chain with possible Instance-level Flows for Chain Task A→Task B→Task C

Mostly, presented approaches are specific to one kind of task activation or chain. An overview of recent work is shown in Table 4.1. Note, that only related work is listed which aims to analyze already implemented systems. Approaches to check end-to-end constraints while generating systems based on specifications, like e.g. [48, 47, 102] are not included.

Table 4.1. Estimation Approaches in Comparison

Approach \ Supports	Task Activation					
	Periodic	Offsets	Sporadic	Chained	Mixed Chains	Multi-core
Becker et al. (2016) [19, 20]	✓	✓	✗	✗	✗	✓
Schlatow et al. (2016) [111]	✗	✗	✗	✓	✗	✓
Girault et al. (2018) [57]	✗	✗	✗	✓	✗	✗
Martinez et al. (2018) [92]	✓	✓	✗	✗	✗	✓
Dürr et al. (2019) [41]	✓	✓	✓	✗	✗	✓
Köhler et al. (2020) [1]	✓	✓	✓	✓	✗	✓
This approach (2018) [49, 50]	✓	✓	✓	✓	✓	✓

Regarding the underlying approach, related work listed in Table 4.1 can be divided into to three categories: transition, compositional, and holistic analyses. The approach followed here belongs to the third category. The difference between transition and compositional, and holistic approaches is that the former divide the problem into smaller sub-problems while the latter look at the whole problem instance at once. The difference

[1] Work currently under review.

between compositional and transition approaches is a slightly different way of partitioning the problem. For compositional approaches, the to sub-problems are translated to an abstract processing model similar to the model of real-time calculus presented in Chapter 3. In transitional approaches the time for the transition between the different sections of the chain is summed with the time needed for the section itself. A variation of this is presented by Becker [20, 19] and Dürr [41]. Martinez et al. [92] follow a related approach but are focused on LET chains only. They bound the time between to task instances based on their periods and offsets. Schlatow et al. follow a purely compositional approach in [111]. Girault et al. [57] follow a different approach by using a busy-window analysis to bound the time a subsequent part of the chain can be delayed.

Another holistic approach to estimating end-to-end latencies is based on mixed integer linear programming (MILP) and was presented in [25, 83]. However, the scheduling mechanism considered differs from static priority preemptive scheduling as it assigns different, fixed time intervals for each task instance within one hyper period of the processing resource.

Besides the work from the scientific community, a range of industrial tools for the analysis of heterogeneous multiprocessor systems exists [107, 7, 130]. Some of them can also be used to perform timing analysis on multi-rate cause effect chains.

4.2 Task-level Model

The task-level encoding is based on different points of time for a finite set of task instances. To obtain the model, first of all a cause-effect chain record is translated into a set of parameters for the constraint model. Subsequently, the variables describing the different points of time are constrained based on these values. Finally, the situation modeling the worst-case response time is searched within all possible valuations.

The parameters of each task in the constraint model are listed in Table 4.2. How these parameters can be obtained from for a cause-effect chain in a communication cluster is described in Section 4.5.2.

Table 4.2. Task Variables with Domain

Variable	Description	Domain
period	Period of a periodically activated task	$\mathbb{N}$
offset	Offset of a periodically activated task	$\mathbb{N}$
pred	Pointer to the predecessor of a chained task	T
dt_min	Minimum temporal distance of two consecutive activations of a sporadic task (bounded or interrupt)	$\mathbb{N}$
dt_max	Maximum temporal distance of two consecutive activations of a sporadic task with bounded occurrence	$\mathbb{N}$
prio	The priority of the task for fixed priority scheduling	$\mathbb{N}$
core	The index of the processing unit on which the task is scheduled (we assume an enumeration.)	$\mathbb{N}$
preemptable	A flag to define whether a task can be interrupted	$\{\,\texttt{t},\texttt{f}\,\}$
deadline	The deadline of the task	$\mathbb{N}$

4.2.1 Task Instance Encoding

For encoding tasks and task instances are numbered consecutively. Let T be the set of indices for T and let m_i hold the number of instances which need to be considered for a safe estimation for all $i \in \mathrm{T}$. How the values for m_i are determined is described in Section 4.5.2. Let $\mathrm{J}_i = \{\,1, \ldots, m_i\,\}$ for all $i \in \mathrm{T}$. In the constraint model, each task instance $j \in \{\,1, \ldots, m_i\,\}$ of task $i \in \mathrm{T}$ is represented by three points of time: activation $\alpha_{i,j}$, start $\sigma_{i,j}$, and end $\varepsilon_{i,j}$. Furthermore, a task instance has a total time it was paused from processing which is modeled by the variable $\iota_{i,j}$. The variables of a task are depicted in Figure 4.2. Task instances are ordered by activation time, this means,

$$j < k \Rightarrow \alpha_{i,j} \leqslant \alpha_{i,k} \qquad \forall i \in \mathrm{T}\ j, k \in \mathrm{J}_i\ . \tag{4.2.1}$$

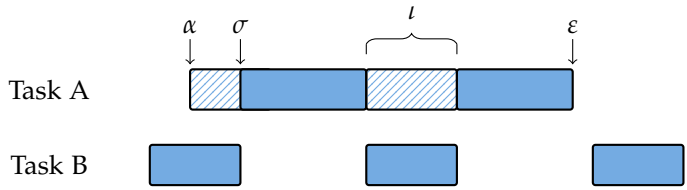


Figure 4.2. Variables of a task instance

The constraints for the variables of a task instance are given in Constraint 4.2.2 to Constraint 4.2.14. They model the following behavior:

Task Instance Activation (Constraint (4.2.2), Constraint (4.2.3), Constraint (4.2.4), Constraint (4.2.5), Constraint (4.2.6), Constraint (4.2.7))
Instances of tasks are activated according to the activation model of the task.

Task Instance Scheduling (Constraints (4.2.10),(4.2.11) and (4.2.12))
A task instance activated is delayed if a task instance of another task is currently being processed which either has a higher priority or can not be preempted. If a task instance is not further delayed, it is started. If a task instance is started while a instance of another task with lower priority is being processed, the latter gets paused.

Task Instance WCRT (Constraints (4.2.13) and (4.2.14))
A task instance is completely processed after its paused time plus BCET and before its paused time plus WCET.

The task activation of a task i depends on its activation model. Task activated by a periodic activation model are activated at multiples of their period starting at the offset:

$$\alpha_{i,j} = \text{offset}(i) + j \cdot \text{period}(i) \qquad \forall j \in \mathrm{J}_i \,. \tag{4.2.2}$$

In the case of chained activation, an instance of task i is activated every time an instance of its predecessor $\text{pred}(i)$ terminates. This is,

$$\alpha_{i,j} = \varepsilon_{\text{pred}(i),j} \qquad \forall j \in \mathrm{J}_i \,. \tag{4.2.3}$$

For a task $i \in \mathrm{T}$ which is activated non-periodically but according to a bounded activation model, the time between two activations lies in the

interval $[\text{dt\_min}(i), \text{dt\_max}(i)]$. Thus,

$$\alpha_{i,1} \geqslant 0 \tag{4.2.4}$$
$$\alpha_{i,1} \leqslant \text{dt\_max}(i) \tag{4.2.5}$$
$$\alpha_{i,j} \geqslant \alpha_{i,j-1} + \text{dt\_min}(i) \qquad \forall j \in \mathrm{J}_i \tag{4.2.6}$$
$$\alpha_{i,j} \leqslant \alpha_{i,j-1} + \text{dt\_max}(i) \qquad \forall j \in \mathrm{J}_i\ . \tag{4.2.7}$$

The set points of time at which all other task instance are completely processed which have been activated before j^{th} instance of task i was started and have a higher priority is:

$$D_{i,j}^{\text{HP}} = \left\{ \varepsilon_{\ell,k} \;\middle|\; \sigma_{\ell,k} \leqslant \sigma_{i,j} \wedge \text{core}(i) = \text{core}(\ell) \wedge \text{prio}(i) < \text{prio}(\ell) \right\}\ . \tag{4.2.8}$$

The set points of time at which all other task instance are completely processed which have been started before j^{th} instance of task i was activated and cannot be preempted is:

$$D_{i,j}^{\text{NP}} = \left\{ \varepsilon_{\ell,k} \;\middle|\; \sigma_{\ell,k} < \alpha_{i,j} \wedge \text{core}(i) = \text{core}(\ell) \wedge \text{preemptable}(\ell) = \texttt{false} \right\}\ . \tag{4.2.9}$$

A task instance is started when it is activated or after other delaying task instances got completely processed if such instances exist:

$$\sigma_{i,j} = \max\left(D_{i,j}^{\text{HP}} \cup D_{i,j}^{\text{NP}} \cup \{\, \alpha_{i,j} \,\} \right) \qquad \forall i \in \mathrm{T}, j \in \mathrm{J}_i\ . \tag{4.2.10}$$

To determine the time a task instance was paused due to other task instances with higher priority being processed first, we define a function mapping all $i, \ell \in \mathrm{T}$, $j \in \mathrm{J}_i$ and $k \in \mathrm{J}_\ell$ to 1 if instance j of i is paused by instance k of ℓ and 0 otherwise:

$$\text{pauses}^{\text{HP}}(i,j,\ell,k) = \begin{cases} 1 & \text{if } \sigma_{\ell,k} > \sigma_{i,j} \wedge \varepsilon_{\ell,k} < \varepsilon_{i,j} \wedge \\ & \quad \text{core}(i) = \text{core}(\ell) \wedge \\ & \quad \text{prio}(i) < \text{prio}(\ell) \\ 0 & \text{otherwise} \end{cases} \tag{4.2.11}$$

Now, if an task instance caused a pause of another task instance, the former instance is certainly processed completely before the processing of

the latter is resumed. Therefore, the interrupted instance is paused for the whole core execution time of the preempting instance. For all $i \in \mathrm{T}$ and $j \in \mathrm{J}_i$ we thus define the following constraint:

$$\iota_{i,j} = \sum_{\ell \in \mathcal{T} \setminus \{i\}} \left(\sum_{k=1}^{m_\ell} \mathrm{pauses}^{\mathrm{HP}}(i,j,\ell,k) \cdot (\varepsilon_{\ell,k} - \sigma_{\ell,k} - \iota_{\ell,k}) \right). \tag{4.2.12}$$

Finally, the processing of a task instance is completed between the BCET and WCET of the task plus the time it was paused. This is, we can give an estimation for when a task instance may terminate and therefore bound point of time for termination by the following two constraints:

$$\varepsilon_{i,j} \geqslant \sigma_{i,j} + \mathrm{bcet}(i) + \iota_{i,j} \qquad \forall i \in \mathrm{T}, j \in \mathrm{J}_i \tag{4.2.13}$$

$$\varepsilon_{i,j} \leqslant \alpha_{i,j} + \mathrm{deadline}(i) \qquad \forall i \in \mathrm{T}, j \in \mathrm{J}_i. \tag{4.2.14}$$

4.2.2 Chain Encoding

Finally, we use the established variables of the constraint model to implement the constraints and estimate the latency semantics described in Definition 3.2.43 and Definition 3.2.40. The model parameter for the chain subject to the end-to-end latency analysis is a sequence of task indices $(p)_{i=1}^{\mathrm{len}}$ where $\mathrm{len} \in \mathbb{N}$ is the length of the chain. To encode the chain on instance level two additional values are added to our model: n_i and x_i for all $i \in \{1, \ldots, \mathrm{len}\}$. The x_i array contains the point of time at which data is ready for the next task in the chain in the case of response times and the point of time to which data is not overwritten in the case of data age. The n_i array contains the index of the instance of task p_i which participated in the worst-case chain instance for all $i \in \{1, \ldots, \mathrm{len}\}$. The constraints on x and n depend on the latency semantic and on the communication pattern of the tasks in the chain. The different constraints are listed in Table 4.3. Note, that implicit communication on runnable-level is treated like explicit communication as this is an over-estimation for the read and write intervals.

The constraints for explicit communication and response time are:

$$n_1 \in \mathrm{J}_{p_1} \tag{4.2.15}$$

$$n_k = \min\left(\{\, j \mid j \in \mathrm{J}_{p_k} \wedge \sigma_{p_k,j} \geqslant x_{k-1} \,\}\right) \qquad \forall k \in \{2, \ldots, \mathrm{len}\} \tag{4.2.16}$$

Table 4.3. Constraints for Latency Semantics and Communication Pattern on Task-level

	Explicit	Implicit	Deterministic
Response Time	4.2.15 - 4.2.18	4.2.15, 4.2.16, 4.2.19	4.2.15, 4.2.20, 4.2.21
Data Age	4.2.22 - 4.2.25	4.2.22, 4.2.23, 4.2.26	4.2.22, 4.2.27, 4.2.28

$$x_k \geqslant \sigma_{p_k,n_k} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.17}$$
$$x_k \leqslant \varepsilon_{p_k,n_k} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.18}$$

The constraints for implicit communication and response time are Constraint 4.2.15, Constraint 4.2.16 and:

$$x_k = \varepsilon_{p_k,n_k} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.19}$$

The constraints for deterministic communication and response time are Constraint 4.2.15 and:

$$n_k = \min\left(\{\, j \mid j \in \mathrm{J}_{p_k} \wedge \alpha_{p_k,j} \geqslant x_{k-1} \,\}\right) \qquad \forall k \in \{2,\ldots,\text{len}\} \tag{4.2.20}$$
$$x_k = \alpha_{p_k,n_k} + \text{deadline}(i) \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.21}$$

The constraints for explicit communication and data age are:

$$n_1 \in \mathrm{J}_{p_1} \tag{4.2.22}$$
$$n_k = \max\left(\{\, j \mid j \in \mathrm{J}_{p_k} \wedge \sigma_{p_k,j} < x_{k-1} \,\}\right) \qquad \forall k \in \{2,\ldots,\text{len}\} \tag{4.2.23}$$
$$x_k \geqslant \sigma_{p_k,(n_k+1)} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.24}$$
$$x_k \leqslant \varepsilon_{p_k,(n_k+1)} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.25}$$

The constraints for implicit communication and data age are Constraint 4.2.15, Constraint 4.2.16 and:

$$x_k = \varepsilon_{p_k,(n_k+1)} \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.26}$$

The constraints for deterministic communication and data age are Constraint 4.2.15 and:

$$n_k = \max\left(\{\, j \mid j \in \mathrm{J}_{p_k} \wedge \alpha_{p_k,j} < x_{k-1} \,\}\right) \qquad \forall k \in \{2,\ldots,\text{len}\} \tag{4.2.27}$$
$$x_k = \alpha_{p_k,(n_k+1)} + \text{deadline}(p_i) \qquad \forall k \in \{1,\ldots,\text{len}\} \tag{4.2.28}$$

The constraints on the x array show that in the case of implicit and explicit communication, precision can be gained if more details about the actual run time behavior of the tasks is added. Hence, in the next section a runnable-level modeling for the analysis of end-to-end latencies in software task chains is presented.

4.3 Runnable-level Model

In the previous section, the basic idea of the encoding of task and chain instances was presented. To add more details about the actual behavior of tasks at run time, we study their insides. Information about the runnable entities allow for more precise modeling. However, since the number of runnable entities assigned to a task possibly lies in the range of multiple hundreds, generating a quadratic amount of constraints for each runnable entity of each task instance is beyond scalability of today's constraint programming (CP) solvers. Strategies to retain precision where needed and being coarse where it has no impact are discussed in Section 4.6.

To formulate the additional constraints necessary for runnable-level modeling, let $\text{rnbls}(i)$ denote the set of runnable entities for task $i \in \text{T}$. Each runnable entity has four input parameter for the model: a BCET, a WCET, its sequence number, and a flag to determine whether processing of the runnable can be interrupted by the scheduler. Two runnables of the same task cannot have the same sequence number, formally we therefore require

$$r_a \neq r_b \Rightarrow \text{nr}(r_a) \neq \text{nr}(r_b) \qquad \forall i \in \text{T},\ r_a, r_b \in \text{rnbls}(i)\ . \tag{4.3.1}$$

4.3.1 Task Instance Encoding

Additional variables are added to the model for each runnable entity of each task instance. The variable $\sigma^R_{i,j,r}$ holds the point of time the processing of a runnable entity r in the j^{th} instance of task i started, $\varepsilon^R_{i,j,r}$ the point of time the processing finished. Furthermore, $\iota^R_{i,j,r}$ holds the time span the processing of r was paused. We use the same symbols as for tasks, therefore task variables get a superscript T for better differentiation, e.g.

Table 4.4. Runnable entity variables with domain

Variable	Description	Domain
bcet	The BCET of a runnable entity	$\mathbb{N}$
wcet	The WCET of a runnable entity	$\mathbb{N}$
nr	The sequence number of the runnable entity	$\mathbb{N}$
preemptable	A flag to define whether a runnable entity can be interrupted	{ t, f }

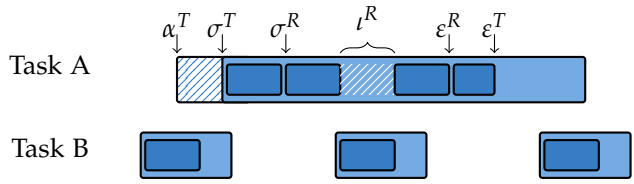


Figure 4.3. Variables of task instances and runnable instances

$\sigma^T_{i,j}$ from now on denotes the point of time the processing of j^{th} instance of of task i started. Figure 4.3 depicts the variables of a runnable entity in the context of a task instance.

With the newly introduced variables, additionally to the behavior modeled with then constraints in Section 4.2, the following behavior is modeled:

Runnable Entities (Constraint (4.3.2)) If a task is scheduled for processing, an instance of the first runnable of the task is started. The runnable entities of a task are activated in sequence, each runnable after its predecessor was completely processed.

Detailed Scheduling (Constraint (4.3.5)) When a task instance gets activated, it may pause the processing of a runnable currently being processed runnable if preferred for execution by the scheduling policy and both are assigned to the same processing unit. The processing time of all runnable instance of the preempting task is added to the paused time of the preempted runnable. Analogously to the task-level model, the

interrupting task instance is certainly processed completely before processing of any paused instance is resumed.

Runnable WCRT (Constraints (4.3.6), (4.3.7)) The time between start and end of processing a runnable entity is at least its paused time plus it BCET and at most its paused time plus its WCET. The results of a runnable are produced between its start and end and written back according to the communication pattern of the runnable.

With the start of the processing of a task instance, the first runnable of the task is processed. Each following runnable is processed after the processing of its predecessors finished. Therefore,

$$\sigma^R_{i,j,r} = \max(\{\, \sigma^T_{i,j} \,\} \cup \{\, \varepsilon^R_{i,j,o} \mid o \in \mathrm{rnbls}(i) \wedge \mathrm{nr}(o) < \mathrm{nr}(r) \,\}) \tag{4.3.2}$$

must hold for all $i \in \mathrm{T}$ and $j \in \mathrm{J}_i$ and $r \in \mathrm{rnbls}(i)$.

Analogously to the task-level model, to calculate the total paused time, ι^R, we introduce a function to determine whether the processing of a runnable entity causes a pause of the processing of another runnable entity. On runnable-level, pauses^{HP} is defined as

$$\mathrm{pauses}^{\mathrm{HP}}_{i,j,r,\ell,k,o} = \begin{cases} 1 & \text{if } \sigma^T_{\ell,k,r} \geqslant \sigma^T_{i,j} \wedge \varepsilon^T_{\ell,k} \leqslant \varepsilon^T_{i,j} \wedge \sigma^R_{\ell,k,o} > \sigma^R_{i,j,r} \wedge \\ & \quad \mathrm{core}(i) = \mathrm{core}(\ell) \wedge \mathrm{prio}(i) < \mathrm{prio}(\ell) \wedge \\ & \quad \mathrm{preemptable}(r) = \mathtt{t} \\ 0 & \text{otherwise} \end{cases} \tag{4.3.3}$$

for all $i, \ell \in \mathrm{T}$, $j \in \mathrm{J}_i$, $k \in \mathrm{J}_\ell$, $r \in \mathrm{rnbls}(i)$, and $o \in \mathrm{rnbls}(\ell)$. More precisely, $\mathrm{pauses}^{\mathrm{HP}}_{i,j,r,\ell,k,o}$ maps to 1 if the runnable o of the k^{th} instance of task ℓ causes a pause of the runnable r of the j^{th} instance of task i. Additionally, an auxiliary variable, $\gamma^R_{i,j,r}$ denoting the actual CET of r is introduced:

$$\gamma^R_{i,j,r} = \varepsilon^R_{i,j,r} - \sigma^R_{i,j,r} - \iota^R_{i,jr} \qquad \forall i, \ell \in \mathrm{T},\ j \in \mathrm{J}_i,\ r \in \mathrm{rnbls}(i). \tag{4.3.4}$$

The pause time ι^R then is the sum of the CETs of all pausing runnables:

$$\iota^R_{i,j,r} = \sum_{\ell \in \mathcal{T} \setminus \{\, i \,\}} \sum_{k=1}^{m_\ell} \sum_{o \in \mathrm{rnbls}(\ell)} i^{\mathrm{HP}}_{i,j,r,\ell,k,o} \cdot \gamma^R_{\ell,k,o} \quad \forall i \in \mathrm{T},\ j \in \mathrm{J}_i,\ r \in \mathrm{rnbls}(i) \tag{4.3.5}$$

The point of time the processing of a runnable entity is finished is constrained by its CET and its paused time:

$$\varepsilon^R_{i,j,r} \geqslant \sigma^R_{i,j,r} + \iota^R_{i,j,r} + \text{bcet}(r) \qquad \forall i \in \text{T}, j \in \text{J}_i, r \in \text{rnbls}(i) \tag{4.3.6}$$

$$\varepsilon^R_{i,j,r} \leqslant \sigma^R_{i,j,r} + \iota^R_{i,j,r} + \text{wcet}(r) \qquad \forall i \in \text{T}, j \in \text{J}_i, r \in \text{rnbls}(i)\ . \tag{4.3.7}$$

The end of processing of the last runnable marks the termination of the task instance ε^T:

$$\varepsilon^T_{i,j} = \max\left(\{\, \varepsilon^R_{i,j,r} \mid r \in \text{rnbls}(i) \,\}\right)\ . \tag{4.3.8}$$

4.3.2 Chain Encoding

Contrary to the expectations of an attentive reader, the encoding of the cause-effect chain in the runnable-level CP model is not based on a sequence of runnable entities. For more convenient modeling, we rather assume a pre-processing step that arranges the chain in the following way: $(p^T)_{k=1}^{\text{length}}$ with $p^T_k \in \text{T}$ for all $k \in \{\, 1, \ldots, \text{length} \,\}$ is the cause-effect chain on task-level and $(p^R)_{k=1}^{\ell}$ with $p^R_k \subseteq \text{rnbls}(p_k)$ is the cause-effect chain on runnable-level for each segment of the chain. The pre-processing step is described in more detail in Section 4.5.2. This saves us the problem of checking whether the chain on runnable-level is arranged in the correct order within each task. Moreover, constraints on the chain variables can be implemented using max- and min-operations on sets which are more easy to handle for the constraint solver. Last but not least, the variables for task-level chain encoding can be reused with the same semantics. The arrays n_i and x_i f.a. $i \in \{\, 1, \ldots, \text{len} \,\}$ are used exactly the same as in task-level encoding, but the constraints on n and x are updated to reflect the newly gained precision in the case of explicit or runnable-level implicit communication. An overview over the updated constraints for response time and data age are shown in Table 4.3

The constraints for explicit communication and response time are:

$$\begin{aligned} & n_1 \in \text{J}_i \\ & \qquad \text{with } i = p^T_1 \end{aligned} \tag{4.3.9}$$

Table 4.5. Constraints for Latency Semantics and Communication Pattern on Runnable-level

	Explicit	Implicit		Deterministic
		Task-level	Runnable-level	
Response Time	4.3.9 - 4.3.12	4.3.9, 4.3.10, 4.3.13, 4.3.14	4.3.9, 4.3.10, 4.3.16	4.3.9, 4.3.17, 4.3.18
Data Age	4.3.19 - 4.3.22	4.3.19, 4.3.20, 4.3.23, 4.3.24	4.3.19, 4.3.20, 4.3.25	4.3.19, 4.3.26, 4.3.27

$$n_k = \min\left(\{\, j \mid j \in \mathrm{J}_i \wedge \min\left(\{\, \sigma^R_{i,j,r} \mid r \in p^R_k \,\}\right) \geqslant x_{k-1} \,\}\right) \quad \text{with } i = p^T_k \;\; \forall k \in \{\, 2, \ldots, \mathrm{len} \,\} \tag{4.3.10}$$

$$x_k \geqslant \min\left(\{\, \sigma^R_{i,j,r} \mid r \in p^R_k \,\}\right) \quad \text{with } i = p^T_k, j = n_k \;\; \forall k \in \{\, 1, \ldots, \mathrm{len} \,\} \tag{4.3.11}$$

$$x_k \leqslant \max\left(\{\, \varepsilon^R_{i,n_k,r} \mid r \in p^R_k \,\}\right) \quad \text{with } i = p^T_k, j = n_k \;\; \forall k \in \{\, 1, \ldots, \mathrm{len} \,\} \tag{4.3.12}$$

The constraints for implicit communication on runnable-level and response time are Constraint 4.3.9, Constraint 4.3.10 and

$$x_k = \min\left(\{\, \sigma^R_{i,j,r} \mid r \in p^R_k \,\}\right) \quad \text{with } i = p^T_k, j = n_k \;\; \forall k \in \{\, 1, \ldots, \mathrm{len} \,\} \tag{4.3.13}$$

$$x_k = \max\left(\{\, \varepsilon^R_{i,j,r} \mid r \in p^R_k \,\}\right) \quad \text{with } i = p^T_k, j = n_k \;\; \forall k \in \{\, 1, \ldots, \mathrm{len} \,\} \tag{4.3.14}$$

The constraints for implicit communication on task-level and response time are Constraint 4.3.9, Constraint 4.3.10 and:

$$x_k = \varepsilon^T_{i,j} \tag{4.3.15}$$

$$\text{with } i = p_k^T, j = n_k \quad \forall k \in \{1, \ldots, \text{len}\} \tag{4.3.16}$$

The constraints for deterministic communication and response time are Constraint 4.2.15 and:

$$\begin{aligned} n_k &= \min\left(\{\, j \mid j \in \mathrm{J}_{p_k} \wedge \alpha_{i,j}^T \geqslant x_{k-1} \,\}\right) \\ &\qquad \text{with } i = p_k^T \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.17}$$

$$\begin{aligned} x_k &= \alpha_{i,j}^T + \text{deadline}(i) \\ &\qquad \text{with } i = p_k^T, j = n_k \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.18}$$

The constraints for explicit communication and data age are:

$$\begin{aligned} &n_1 \in \mathrm{J}_i \\ &\qquad \text{with } i = p_1^T \end{aligned} \tag{4.3.19}$$

$$\begin{aligned} n_k &= \min\left(\{\, j \mid j \in \mathrm{J}_i \wedge \min\left(\{\, \sigma_{i,j,r}^R \mid r \in p_k^R \,\}\right) < x_{k-1} \,\}\right) \\ &\qquad \text{with } i = p_k^T \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.20}$$

$$\begin{aligned} x_k &\geqslant \max\left(\{\, \sigma_{i,j,r}^R \mid r \in p_k^R \,\}\right) \\ &\qquad \text{with } i = p_k^T, j = (n_k + 1) \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.21}$$

$$\begin{aligned} x_k &\leqslant \max\left(\{\, \varepsilon_{i,j,r}^R \mid r \in p_k^R \,\}\right) \\ &\qquad \text{with } i = p_k^T, j = (n_k + 1) \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.22}$$

The constraints for implicit communication on runnable-level and data age are Constraint 4.2.15, Constraint 4.2.16 and:

$$\begin{aligned} x_k &\geqslant \max\left(\{\, \sigma_{i,j,r}^R \mid r \in p_k^R \,\}\right) \\ &\qquad \text{with } i = p_k^T, j = (n_k + 1) \quad \forall k \in \{1, \ldots, \text{len}\} \end{aligned} \tag{4.3.23}$$

$$x_k \leqslant \max\left(\{\, \varepsilon^R_{i,j,r} \mid r \in p^R_k \,\}\right) \quad \text{with } i = p^T_k, j = (n_k + 1) \;\; \forall k \in \{\, 1, \ldots, \text{len} \,\} \tag{4.3.24}$$

The constraints for implicit communication on task-level and data age are Constraint 4.2.15, Constraint 4.2.16 and:

$$x_k = \varepsilon^T_{i,j} \quad \text{with } i = p^T_k, j = (n_k + 1) \;\; \forall k \in \{\, 1, \ldots, \text{len} \,\} \tag{4.3.25}$$

The constraints for deterministic communication and data age are Constraint 4.2.15 and:

$$n_k = \max\left(\{\, j \mid j \in \mathrm{J}_i \wedge \alpha^T_{i,j} < x_{k-1} \,\}\right) \quad \text{with } i = p^T_k \;\; \forall k \in \{\, 1, \ldots, \text{len} \,\} \tag{4.3.26}$$

$$x_k = \alpha^T_{i,j} + \text{deadline}(i) \quad \text{with } i = p^T_k, j = (n_k + 1) \;\; \forall k \in \{\, 1, \ldots, \text{len} \,\} \tag{4.3.27}$$

4.4 Model Validation

Although the constraint model presented in the previous sections is precisely described and the constraints are carefully chosen, it can not be ruled out that bugs in the model exists like bugs possibly exist in the code of computer programs. In fact, the constraints as stated above allow for an imprecision: a result in which a runnable is terminated while being preempted. This is not possible for real-world instances of a task because the return statement needs to be processed. Since such results have no counterpart in any real run of the system they possibly lead to more overestimation. In this case, they can be removed from the solution space by adding an additional constraint. Let (i, j, r) denote the r-th runnable in the j-th instance of task i, and (l, k, o) accordingly. Then, the following constraint enforces that if (i, j, r) is preempted by (l, k, o), it must be terminated after (l, k, o):

$$(\sigma^R_{l,k,o} > \sigma^R_{i,j,r} \wedge \sigma^R_{l,k,o} < \varepsilon^R_{i,j,r}) \rightarrow (\varepsilon^R_{i,j,r} > \varepsilon^R_{l,k,o}) \tag{4.4.1}$$

However, this gives rise to the question if there are any other overestimations or possibly even bugs excluding valid results. To check the constraint model for problems, we use an additional set of constraints modeling the same behavior for cross-checking. We use the well-known concept of the *busy-window* to check whether the outcome is the same. Let $B_{i,j}$ denote the starting times of task instances which may delay the start of the j-th instance of task i, i.e. $B_{i,j} = \{ \sigma^T_{\ell,k} \mid \ell \in \mathcal{T}_{\mid \mathrm{core}(i)} \wedge \varepsilon^T_{\ell,k} > \alpha^T_{i,j} \wedge \varepsilon^T_{\ell,k} \leqslant \varepsilon^T_{i,j} \}$. Then, the busy window starts at time $bw^{\mathrm{start}}_{i,j}$ with

$$bw^{\mathrm{start}}_{i,j} = \min \left(\{ \alpha^T_{i,j} \} \cup B_{i,j} \right)$$

The sum of executions times of task instances running in a busy windows is given by

$$bw^{\mathrm{length}}_{i,j} = \sum_{(\ell,k) \in A_{i,j}} \left(\varepsilon^T_{\ell,k} - \sigma^T_{\ell,k} - \iota^T_{\ell,k} \right) \tag{4.4.2}$$

with

$$A_{i,j} = \left\{ (\ell,k) | \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h) = \mathrm{core}(i)}, k \in \mathrm{J}_\ell \wedge \sigma^T_{\ell,k} \geqslant bw^{\mathrm{start}}_{i,j} \wedge \varepsilon^T_{\ell,k} \leqslant \sigma^T_{i,j} \right\} \tag{4.4.3}$$

Using this additional constraints showed, that the current constraints on σ can be satisfied by a solution in which a task is started too late although its resource was idling in the mean time. Again, this is an overestimation leading to additional results we do not want to consider for tight estimations. This imprecision can be resolved by adding additional constraints. A straight forward idea is utilizing the busy window for this problem. However, this introduces two variables for each task instance holding discrete time values and depending on many parameters. This results in a massive increase of complexity, increasing the size of the problem for both, constraint compiler and constraint solver. Therefore, we propose a formalization where no additional discrete time variables are

introduced. For this purpose let

$$\begin{aligned}\mathrm{hplb}_{i,j,p} =& \exists \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h)=\mathrm{core}(i)} : \exists k \in \{1, \ldots, m_\ell\} : \\ & \mathrm{prio}(\ell) > p \wedge \sigma^T_{\ell,k} < \sigma^T_{i,j} \wedge \varepsilon^T_{\ell,k} \geqslant \sigma^T_{i,j} \\ \vee & \\ & \exists \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h)=\mathrm{core}(i)} : \exists k \in \{1, \ldots, m_\ell\} : \exists r \in \mathrm{rnbls}(\ell) : \\ & \mathrm{preemptable}(r) = \mathrm{false} \wedge \sigma^R_{\ell,k,r} < \alpha^T_{i,j} \wedge \varepsilon^R_{\ell,k,r} \geqslant \sigma^T_{i,j}\end{aligned} \tag{4.4.4}$$

be a decision variable for all $i \in \mathrm{T}$, $j \in \mathrm{T}_i$, and $p \in \{\min(\{\mathrm{prio}(i) \mid i \in \mathcal{T}\}), \ldots, \max(\{\mathrm{prio}(i) \mid i \in \mathcal{T}\})\}$. This is, $\mathrm{hplb}_{i,j,p}$ holds true if the processing unit of task i is busy processing a task instance having a higher priority than p or which is not preemptable, before instance j of task i is started.

Thus, adding the constraint

$$\begin{aligned}&(\sigma^T_{i,j} > \alpha^T_{i,j}) \rightarrow \forall \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h)=\mathrm{core}(i)} : \forall k \in \{1, \ldots, m_i\} : \\ &\quad (\sigma^T_{\ell,k} \leqslant \sigma^T_{i,j} \wedge \alpha^T_{\ell,k} > \alpha^T_{i,j}) \rightarrow (\mathrm{prio}(\ell) > \mathrm{prio}(i) \wedge \mathrm{hplb}_{\ell,k,\mathrm{prio}(i)})\end{aligned} \tag{4.4.5}$$

for all $i \in \mathrm{T}$ and $j \in \mathrm{J}_i$ prevents a task being delayed by a task which was started after the processing unit was idle while it was already activated.

Another problem we observed while *debugging* is, that the model allows for an unintended priority inversion. This means, that a task instance is being processed despite another task instance with higher priority was activated but not started. Since both scenarios are accepted by the constraints, this is another imprecision leading to less tight estimations. To prevent this, the constraints

$$\begin{aligned}&\forall \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h)=\mathrm{core}(i)} : \forall k \in \{1, \ldots, m_i\} : \\ &\quad (\sigma^T_{\ell,k} \geqslant \alpha^T_{i,j} \wedge \sigma^T_{\ell,k} \leqslant \sigma^T_{i,j}) \rightarrow (\mathrm{prio}(\ell) > \mathrm{prio}(k))\end{aligned} \tag{4.4.6}$$

and

$$\begin{aligned}&\forall \ell \in \mathrm{T}_{|h \mapsto \mathrm{core}(h)=\mathrm{core}(i)} : \forall k \in \{1, \ldots, m_i\} : \forall r \in \mathrm{rnbls}(\ell) : \\ &\quad (\varepsilon^R_{\ell,k,r} > \alpha^T_{i,j} \wedge \varepsilon^T_{\ell,k,r} \leqslant \sigma^T_{i,j}) \rightarrow \\ &\qquad (\mathrm{prio}(\ell) > \mathrm{prio}(k) \vee (\mathrm{preemptable}(r) = \mathtt{f} \wedge \sigma^R_{\ell,k,r} < \alpha^T_{i,j}))\end{aligned} \tag{4.4.7}$$

are added f.a. $i \in T$ and $j \in J_i$. Constraint 4.4.6 ensures that, if a task is started between the activation and the start of another task, it needs to have a higher priority. Constraint 4.4.7 ensures that, if a runnable finishes between the activation and the start of a task, either its task has a higher priority or it is not preemptable.

4.5 Solving

To encode our model, we use the high-level constraint language MiniZinc[2] Using MiniZinc has two main advantages. Firstly, the language has a lot of pre-defined structures which allow a direct modeling of mathematical formulations, e.g. the Example 2.5.8 can easily be encoded in MiniZinc as shown in Listing 4.1. Secondly, a MiniZinc-model is data and solver independent. Only when it comes to the compilation to the low-level constraint language FlatZinc, the model parameters and the constraint model are brought together. FlatZinc is then supported by a wide range of solver-backends. To show the advantage of a high-level modeling language, the FlatZinc for Example 2.5.8 is shown in Listing 4.2.

```
var { 1, 2 } : x1;
var { 2, 3 } : x2;
var { 1, 2 } : x3;

constraint ( x1 < x2 \/ x1 > x3 );
constraint ( x2 > x3 );
constraint ( x3 = 4 \/ x3 > x1 );
```

Listing 4.1. MiniZinc Example

```
array [1..2] of int: X_INTRODUCED_0_ = [-1,1];
array [1..2] of int: X_INTRODUCED_2_ = [1,-1];
var 1..2: x1:: output_var;
var 2..3: x2:: output_var;
var 1..2: x3:: output_var;
```

[2] https://www.minizinc.org/

```
var bool: X_INTRODUCED_1_ ::var_is_introduced :: is_defined_var;
var bool: X_INTRODUCED_3_ ::var_is_introduced :: is_defined_var;
constraint array_bool_or([X_INTRODUCED_1_,X_INTRODUCED_3_],true);
constraint int_lin_le(X_INTRODUCED_0_,[x2,x3],-1);
constraint int_lin_le(X_INTRODUCED_0_,[x3,x1],-1);
constraint int_lin_le_reif(X_INTRODUCED_0_,[x1,x3],-1,
    X_INTRODUCED_1_):: defines_var(X_INTRODUCED_1_);
constraint int_lin_le_reif(X_INTRODUCED_2_,[x1,x2],-1,
    X_INTRODUCED_3_):: defines_var(X_INTRODUCED_3_);
solve  satisfy;
```

Listing 4.2. FlatZinc for MiniZinc Example

4.5.1 MiniZinc Model

In this section we take a look at how to translate the constraint model introduced in Section 4.2 and Section 4.3 into MiniZinc. The most important aspect of the MiniZinc model are explained and the MiniZinc representation for the core of the constraint model is presented. All constraints are based on the parameters and variables of tasks and their instances. Listing 4.3 shows the variables for a task instance in MiniZinc.

```
% variables for each task instance
array[i_Task,Min_Inst..Max_Inst] of var Timepoint : alphaT;
array[i_Task,Min_Inst..Max_Inst] of var Timepoint : sigmaT;
array[i_Task,Min_Inst..Max_Inst] of var Timepoint : epsilonT;
array[i_Task,Min_Inst..Max_Inst] of var 0..Time_Dom_Max : iotaT;
array[i_Task,Min_Inst..Max_Inst] of var 0..Time_Dom_Max :
    runningTime;
```

Listing 4.3. Variables of a task instance in MiniZinc

All constraints listed above can be directly translated into MiniZinc, however, some modeling principles must be taken into account in order to create scalable FlatZinc models. An example of a constraint which can

directly be translated into MiniZinc is the WCRT constraint on ε as shown in Listing 4.4

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i] where
    isRelevant[i]) (
  epsilonT[i,j] >= sigmaT[i,j] + iotaT[i,j] + Bcet[i] /\
  epsilonT[i,j] <= sigmaT[i,j] + iotaT[i,j] + Wcet[i] /\
  epsilonT[i,j] <= alphaT[i,j] + Deadline[i]
);
```

Listing 4.4. Constraints on ε in MiniZinc

The constraints on α are formulated with a case distinction for the different types of activation models. Within the different cases the MiniZinc representation of the constraints again is directly translated from the mathematical formulation as shown in Listing 4.5.

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i] where
    isRelevant[i]) (
  if (ActType[i] = 0) then     % periodic
    alphaT[i,j] = Offset[ActModel[i]] + ((j) * Period[ActModel[i]])

  elseif (ActType[i] = 1) then % chaining
    assert(length ([p | p in i_Task where Name[p] = Predecessor[
    ActModel[i]]]) = 1, "exactly one predecessor! ") /\
    let { var int: predI = min([p | p in i_Task where Name[p] =
    Predecessor[ActModel[i]]]) } in
      alphaT[i,j] = epsilonT[predI,j]

  elseif (ActType[i] = 2) then % intervall
    if (j == InstMin[i]) then
      alphaT[i,j] >= Min_Time /\
      alphaT[i,j] <= Min_Time + BoundMaxDt[ActModel[i]]
    else
      alphaT[i,j] >= alphaT[i,j-1] + BoundMinDt[ActModel[i]] /\
      alphaT[i,j] <= alphaT[i,j-1] + BoundMaxDt[ActModel[i]]
    endif
  else % ActType[i] = 3
```

```
    if (j == InstMin[i]) then
      alphaT[i,j] >= Min_Time
    else
      alphaT[i,j] >= alphaT[i,j-1] + MinDeltaT[ActModel[i]]
    endif
  endif
);
```

Listing 4.5. Constraints on α in MiniZinc

The constraints on σ can also be almost directly translated as shown in Listing 4.6. The `maximum_int` predicate constrains the first argument to be the maximum value of the second argument which needs to be a collection of values. Here we see an additional condition for other task instances to be counted in: a function called `TimeFrameOverlap`. This functions only returns true of the processing of instance j of task i is possibly affected by instance k of task ℓ through a delay or an interrupt. The same function is also used in the constraints for ι which are shown in Listing 4.7. It is used to massively reduce the amount of constraints generated, as interference constraints for tasks instances which do not possible interfere are omitted. An estimation for this can be done before translating into FlatZinc. This is further discussed in Section 4.5.2. The summation an the conditions for ι can be translated directly again.

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i] where
     isRelevant[i]) (
  maximum_int(sigmaT[i, j],
    %  activation time
    [alphaT[i, j]]

    % Non-preemptive
    ++ [ if (sigmaT[l, k] < alphaT[i, j])  then  epsilonT[l, k]
     else 0 endif |
      l in i_Task , k in InstMin[l]..InstMax[l]  where
      (i!=l /\ Core[i] = Core[l] /\ Prmptbl[l] = 0) /\
     TimeFrameOverlap(i,j,l,k)]
```

```
    % Higher prio
    ++ [ if (sigmaT[l, k] <=  sigmaT[i, j])  then  epsilonT[l, k]
     else 0 endif |
      l in i_Task , k in InstMin[l]..InstMax[l]  where
      (i!=l /\ Core[i] = Core[l] /\ Prio[l] > Prio[i]) /\
     TimeFrameOverlap(i,j,l,k)]
  )
);
```

Listing 4.6. Constraints on σ in MiniZinc

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i] where
    isRelevant[i]) (
  if (Prmptbl[i] = true) then
    iotaT[i,j] =
      sum(l in i_Task where i != l /\ Core[l] = Core[i] /\ Prio[l]
    > Prio[i] /\ isRelevant[l]) (
        sum(k in InstMin[l]..InstMax[l] where TimeFrameOverlap(i,j,
    l,k)) (
          if (sigmaT[l, k] >= sigmaT[i, j] /\ epsilonT[l, k] <=
    epsilonT[i, j]) then
            epsilonT[l, k] - sigmaT[l, k] - iotaT[l, k]
          else
            0
          endif
        )
      )
  else
    iotaT[i,j] = 0
  endif
);
```

Listing 4.7. Constraints on ι in MiniZinc

For the runnable-level MiniZinc model, constraints on start of runnables have to added. Furthermore, the representation of ι constraints is now

based on runnables instead of tasks. The constraints are shown in Listing 4.8 and Listing 4.9.

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i], r in
    1..card(Runnables[i]) where isRelevant[i]) (
  iotaR[i,j,r] =
    sum(l in i_Task where i != l /\ Core[l] = Core[i] /\
              Prio[l] > Prio[i] /\ Prmptbl[i] = true) (
      sum(k in InstMin[l]..InstMax[l] where TimeFrameOverlap(i,j,l,
    k)) (
        if (sigmaT[l, k] >= sigmaT[i, j] /\ epsilonT[l, k] <=
    epsilonT[i, j]) then
          if (sigmaT[l, k] >= sigmaR[i, j, r] /\ epsilonT[l, k] <=
    epsilonR[i, j, r]) then
            cetT[l,k]
          else
            0
          endif
        else
          0
        endif
      )
    )
);
```

Listing 4.8. Runnable-level Constraints on σ in MiniZinc

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i], r in
    1..card(Runnables[i]) where isRelevant[i]) (
  sigmaR[i,j,r] =
    max([sigmaT[i,j]] ++ [epsilonR[i,j,o] | o in 1..card(Runnables[
    i]) where o < r])
);
```

Listing 4.9. Runnable-level Constraints on ι in MiniZinc

Finally, we come to the encoding of chains. Since the translation is direct and straight forward, we only consider one case here: the runnable-level encoding for the response time latency semantics is shown in Listing 4.10.

```
constraint forall (p in 2..len_Path) (
  let { int: i = T_Path[p] } in

  if (CommMode[i] = "explicit") then
    minimum_int (n[p],
    [ if (min([sigmaR[i,j,R_Nr[r]] | r in R_Path[p]]) >= x[p-1])
     then j else Max_Inst + 1 endif |
    j in InstMin[i]..InstMax[i]])

  elseif (CommMode[i] = "implicit_runnable") then
    minimum_int (n[p],
    [ if (min([alphaR[i,j,R_Nr[r]] | r in R_Path[p]]) >= x[p-1])
     then j else Max_Inst + 1 endif |
    j in InstMin[i]..InstMax[i]])

  elseif (CommMode[i] = "implicit_task") then
    minimum_int (n[p],
    [ if (alphaT[i,j] >= x[p-1]) then j else Max_Inst + 1 endif |
    j in InstMin[i]..InstMax[i]])

  elseif (CommMode[i] = "deterministic") then
    minimum_int (n[p],
    [ if (alphaT[i,j] >= x[p-1]) then j else Max_Inst + 1 endif |
    j in InstMin[i]..InstMax[i]])

  else
    assert(false, "unkown communication mode: " ++ CommMode[i])
  endif
);
```

Listing 4.10. Runnable-level Chain Encoding for Response Time in MiniZinc

4.5.2 DataZinc Generation

Almost all parameters used in the constraint model for task- and runnable-level can be extracted from the model defined in Chapter 3. Only the minimum and maximum instance and some derived parameters for the optimizations described in Section 4.6 need to be calculated. In this section we present the algorithms to do this.

For solving we consider a finite amount of task and runnable instances within a finite time span with discrete points of time. The first step to generate a data file for the MiniZinc model is to determine the length of this time span. It must be long enough to safely cover all possible relative offsets between tasks but if it is too long, the time needed for compiling and solving will sharply increase. It is a well-known fact that the relative offsets for task instances of a periodic task set repeat after the hyper period or the least common multiple of all periods. As described in the previous sections, the constraints on α assure that all relative offsets for sporadic tasks are observed during an instance of the chain. Therefore, the worst-case occurrence of the chain will start at some point of time within the hyper period of all tasks participating in the chain. Furthermore, tasks which might influence a task on the chain need to be considered as they possibly influence the relative offsets between two tasks on the chain. We therefore calculate the set of possibly influencing tasks as formally described in Algorithm 1. The hyper period of the union of all sets of relevant tasks for the tasks participating in the chain makes up the first part of the time span for analysis. The second part is a rough upper bound on the length of the chain. The time span for analysis must be large enough such that an instance of the chain completely fits into it, independent of its starting time. For response time, such a bound can for example be derived by summing up trivial upper bounds on the time between two arrivals and the worst case response time of every task on the chain.

For tasks with a sporadic activation pattern the estimation gets more imprecise with time, since the occurrence of the next instance depends on the occurrence of the current instance. If the time span for activation is varying within a range, the time span of the next activation is varying even more. Therefore, in order to obtain safe estimations, the minimum and maximum amount of instances considered has to be chosen carefully

Algorithm 1: Set of Influencing Tasks

```
1 RelevantPeriod Task T, Tasks 𝒯
2     ℛ ← { T }
3     f ← false
4     while f = false do
5         ℛ_new ← { O | O ∈ 𝒯 ∧ Task.Preemtable(O) = false }
6         ℛ_new ← ℛ ∪
              { O | O ∈ 𝒯 ∧ Task.Priority(O) ⩾ Task.Priority(T) }
7         f ← ℛ_new ⊆ ℛ
8         ℛ ← ℛ ∪ ℛ_new
9     return ℛ
```

because huge discrete time domains massively impact the performance of the solver.

Using the `GpTimeFrame`-array as shown in Listing 4.11 and generated as formally described in Algorithm 3, the constraints given in Listing 4.12 are added to the model to implement the model tweak described Section 4.6.

```
GPTimeFrame = array3d(i_Task,Min_Inst..Max_Inst,1..2, [
%...
]);
```

Listing 4.11. GPTimeFrame Array in MiniZinc

```
constraint forall (i in i_Task, j in InstMin[i]..InstMax[i]) (
GPTimeFrame[i,j,1] <= alphaT[i,j] /\ alphaT[i,j] <= GPTimeFrame[i
   ,j,2] /\
GPTimeFrame[i,j,1] <= sigmaT[i,j] /\ sigmaT[i,j] <= GPTimeFrame[i
   ,j,2] /\
GPTimeFrame[i,j,1] <= epsilonT[i,j] /\ epsilonT[i,j] <=
   GPTimeFrame[i,j,2] /\

forall (r in 1..Max_Rnbl)(
if (r <= card(Runnables[i])) then
```

Algorithm 2: Calculate First and Last Occurrence

InstMinMax *Time* t_{min}, *Time* t_{max}, *Task* T, *Tasks* $\mathcal{T}$
- $a \leftarrow$ `Task.ActivationModel`(T)
- $(i_{\min}, i_{\max}) = (0, 0)$
- **if** $a \in$ `PeriodicActivationModel` **then**
  - $p \leftarrow$ `PeriodicActivationModel.Period`(a)
  - $o \leftarrow$ `PeriodicActivationModel.Offset`(a)
  - $(i_{\min}, i_{\max}) \leftarrow \left(\left\lceil \frac{t_{\min} - o}{p} \right\rceil, \left\lceil \frac{t_{\max} - o}{p} \right\rceil \right)$
- **else if** $a \in$ `ChainedActivationModel` **then**
  - $id \leftarrow$ `ChainedActivationModel.Predecessor`(a)
  - $P \leftarrow$ find task with id id in $\mathcal{T}$
  - $(i_{\min}, i_{\max}) \leftarrow$ **InstMinMax**$(t_{\min}, t_{\max}, P, \mathcal{T})$
- **else**
  - $l \leftarrow \infty$
  - **if** $a \in$ `BoundActivationModel` **then**
    - $l \leftarrow$ `BoundActivationModel.MinDt`(a)
  - **else if** $a \in$ `SporadicActivationModel` **then**
    - $l \leftarrow$ `SporadicActivationModel.MinDt`(a)
  - $(i_{\min}, i_{\max}) \leftarrow \left(\left\lceil \frac{t_{\min}}{l} \right\rceil, \left\lceil \frac{t_{\max}}{l} \right\rceil \right)$
- **return** $(i_{\min}, i_{\max})$

```
GPTimeFrame[i,j,1] <= sigmaR[i,j,r] /\ sigmaR[i,j,r] <=
   GPTimeFrame[i,j,2] /\
GPTimeFrame[i,j,1] <= epsilonR[i,j,r] /\ epsilonR[i,j,r] <=
   GPTimeFrame[i,j,2] /\
cetR[i,j,r] = epsilonR[i,j,r] - sigmaR[i,j,r] - iotaR[i,j,r]
else
sigmaR[i,j,r] = -1 /\
cetR[i,j,r] = 0 /\
iotaR[i,j,r] = 0 /\
epsilonR[i,j,r] = -1
endif)
);
```

Listing 4.12. GPTimeFrame Array in MiniZinc

4.5.3 FlatZinc Solving and Results

For solving, the MiniZinc model is linked with the DataZinc generated for the problem instance and compiled to the low-level constraint language FlatZinc. The specification of FlatZinc is given in [21] and the compilation process is described in [96]. We use the compiler available at the MiniZinc github page[3]. As the solver back-end we chose the parallel version of the lazy-clause generation solver *Chuffed* as presented in [43].

From Definition 2.5.6 we know, that a solution for the CSP formed by the constraints stated above is an assignment which maps each variable to a value while satisfying all constraints on that variable. In our case this means that the situation modeled by the result can be reconstructed from the solution of the CSP. As depicted in Figure 4.2 and Figure 4.3 different points of time describing the task instances contributing to the worst-case latency are available. They can be used to visualize the result for further analysis in a representation of the temporal sequence like Figure 4.1.

Since we search for the solutions yielding a maximal end-to-end latency, the CSP we formulated actually is a constraint optimization problem. As we know from its definition, $\text{sols}(\mathcal{P}^{\max})$ only contains solutions where no result exists which yields a higher objective function. This means that a safe upper bound for the end-to-end latency is estimated under the assumptions of the constraint model.

4.6 Model Optimization

In this section we revisit the constraint model defined above to add some additional tweaks for faster solving. The first optimization targets the number of generated constraints. When generated naively, the number of constraints for, e.g. Equation 4.3.3 shows quadratic growth when adding new task instances. To preserve scalability, we add a preprocessing step. For each task instance we estimate the maximal time span in which it is possibly active. Constraints modeling interference with other task instances only need to be generated for task instances where this time span overlaps. We add the function shown in Listing 4.13 and use it to instruct the

[3] https://github.com/MiniZinc/libminizinc

FlatZinc-compiler omit the generation of constraints for two task instances j and k of tasks i and ℓ where this function does not return true. The `GPTimeFrame`-array contains the maximum active time span for each task instance. Its generation is described in Section 4.5.2.

```
function bool: TimeFrameOverlap(int: i, int: j, int: l, int: k) =
  GPTimeFrame[l, k, 2] >= GPTimeFrame[i, j, 1] /\
  GPTimeFrame[i, j, 2] >= GPTimeFrame[l, k, 1] ;
```

Listing 4.13. TimeFrameOverlap Function in MiniZinc

Another factor for the number of constraints in Equation 4.3.3 is the amount of runnables per task. To reduce this number, we aggregate multiple runnables to a single runnable where it has no impact, i.e. where no signals for the chain are consumed or produced but only core execution time happens. For tasks which are not participating in the chain, all runnables can be aggregated to one runnable modeling the same load with small impacts on the precision. The impacts are only relevant in the case of runnables with can not be preempted. Since a task instance might be preempted between two runnable entities, the aggregated runnable needs to be preemptable. This adds new possibilities for preemption where the individual runnables where not preemptable in the first place. However, if the precision is needed for targeted analyses, a more fine-grained aggregation is possible, e.g. aggregating several runnables to three ones where one is still non-preemptable. The BCET of the synthesized runnables is the sum of the BCETs of the individual runnables. Analogously, the WCET is the sum of the WCETs. For task participating in the chain, the first possible read and the last possible write access on the signals consumed or produced in the course of the chain are relevant. Therefore, the runnables of a participating task can be divided in three categories: runnables processed before the first possible read, runnables processed in between, and runnables processed after the last possible write. Each category of runnables can be aggregated as described above, resulting in at least three synthetic runnables for a task on the chain.

A second optimization is also concerned with the number of constraints being generated but tackles the problem at a higher level. Assume a cause-

effect chain spanning over three resources: R_1, R_2, R_3. If the run of the chain starts at R_1, then has a transition to R_2 and finally a transition to R_3, then task instances at R_1 do not need to be considered anymore given that no task on R_3 is activated by a task on R_1. Continuing in the same vein, task instances on R_1 also do not need to be considered if the further course of the chain only takes place on R_2. Therefore, the last task instance considered on R_1 can be lowered such that an overlap of instances is only given at all points of time where a transition from R_1 to R_2 is possible since considering more instances of any task on R_1 does not possibly influence the result under our assumptions. With the same argument, a higher number for the first instance of any task on R_2 and R_3 can be chosen.

A third optimization we developed is concerned with the search strategy of the solver. Searching in the wrong area of the set of satisfying assignments can be very time consuming. Therefore, in order to decrease the time to find an optimal solution, we instruct the solver to start its search assuming a large value for the objective. This strategy fails if the upper bound on the objective is too far away from the optimal solution. Therefore, we add different constraints to help the solver infer an upper bound on the objective fast. The main reason why it is hard to infer such a bound a priori is that it is hard to estimate the possible inferences for tasks instances. This is why adding constraints for the maximal delay and the maximal pause time were added to support the fast estimation of a upper bound for the objective. For this purpose, the task instances possibly delaying and the task instances possibly causing a pause of a task instance are needed. These are for instance j of task i the tasks which have a higher priority

$$I_{i,j}^{\mathrm{HP}} = \left\{ (\ell, k) \mid \varepsilon_{\ell,k}^{T} \geqslant \sigma_{i,j}^{T} \wedge \sigma_{\ell,k}^{T} \leqslant \varepsilon_{i,j}^{T} \wedge \mathrm{prio}(\ell) > \mathrm{prio}(i) \right\} \tag{4.6.1}$$

and the tasks which possibly have a runnable that is not preemptable

$$I_{i,j}^{\mathrm{NP}} = \left\{ (\ell, k) \mid \varepsilon_{\ell,k}^{T} \geqslant \sigma_{i,j}^{T} \wedge \sigma_{\ell,k}^{T} \leqslant \varepsilon_{i,j}^{T} \wedge \mathrm{preemptable}(\ell) = \mathtt{f} \right\} \tag{4.6.2}$$

which are possibly running in the same time interval.

Then, the delay of instance j of task i can be bound with the help of

the WCETs of these tasks:

$$\left(\sigma_{i,j}^{T} - \alpha_{i,j}^{T}\right) \leqslant \sum_{(l,k)\in I_{i,j}^{\mathrm{HP}}} \mathrm{wcet}(l) + \sum_{(l,k)\in I_{i,j}^{\mathrm{NP}}} \mathrm{wcet}(l)\ . \tag{4.6.3}$$

Additionally, the worst-case response time of the same task instance, i.e. the time the task needs to react to an input, can be bound by:

$$\varepsilon_{i,j}^{T} \leqslant \alpha_{i,j}^{T} + \mathrm{wcet}(i) + \sum_{(l,k)\in I_{i,j}^{\mathrm{HP}}} \mathrm{wcet}(l) + \sum_{(l,k)\in I_{i,j}^{\mathrm{NP}}} \mathrm{wcet}(l). \tag{4.6.4}$$

4.7 Model Application

In this section we prove the practical applicability by carrying out different evaluations. We use three different sources for task sets: (1) the task set published in [50], (2) the task set published in [60], and (3) task sets generated with the parameters described in [78].

4.7.1 Case Studies

In this section we test the MiniZinc model with two different industrial-scale task sets in three experiments. In the first experiment we compare the results obtained with our CP approach with the CPA implementation *pyCPA*[4]. In the second experiment we analyze the potential impact of sporadic task occurrences. The third experiment is considered with the question how much precision is gained on runnable level and draw a conclusion for the applicability in automated system optimization.

The experiments were carried out on a desktop computer with an Intel(R) Core(TM) i9-7940X CPU @ 3.10GHz and 128GB of DDR4 memory.

The first and the second experiment are based on the task set shown in Table 4.8 and Table 4.9. The results for both alternatives for the activation models are listed in Table 4.6. The results of the first experiment show the superior precision of the holistic constraint approach. The results of the second experiment show the importance of the support of bonded and

[4]https://bitbucket.org/pycpa/

sporadic activation models. Even if they are not included in the chain, non-periodic task possibly impact the estimation. As expected our experiment shows that modeling such tasks with a periodic activation model does not yield safe bounds for the end-to-end estimation of the chain.

The third experiment is based on a task set with runnable-level details originally published in [60]. In [112] Schlatow et al. propose different priority assignments and core mappings in order to optimize data ages for two of the three chains specified for the data set. This is the only other analysis with this level of detail currently known to us. For comparison, we carried out an estimation on runnable-level and on task-level. The main question is, how the results with the different levels of detail differ. The results are shown in Table 4.7. They allow to draw important conclusions regarding the need of precision in the context of optimization. Fortunately, Schlatow et al. published an interim result of the optimization process. The mapping and priority assignment $E\pi\phi$ yields, according to the results of Schlatow et al., a slightly worse data age for both chains. The improvement from variant $E\pi\phi$ to $E\pi\phi A$ for Chain 2 is confirmed by our estimations. However, in the case of Chain 3 the estimations using a task-level data flow disagree with the results obtained using a runnable-level data flow regarding an improvement. This shows the importance highly precise estimations in the context of optimization. An actual improvement might not be achieved if the margin of the optimization lies within the imprecision of the estimation method. Consequently, an optimization might not actually improve the performance of the system if the estimation method is not precise.

4.7.2 Precision and Performance Evaluation

Since we only tested the approach on industrial use-cases so far, a more general claim about the scalability of the approach can not be made. Furthermore, we want to investigate the impact of context losses for analysis precision. Therefore, we carry out another experiment with generated task sets in this section. The task sets were generated as follows:

- A system consists of $\{2, \ldots, 6\}$ resources with synchronized clocks.
- On each resource either periodic BET or LET tasks with implicit deadlines

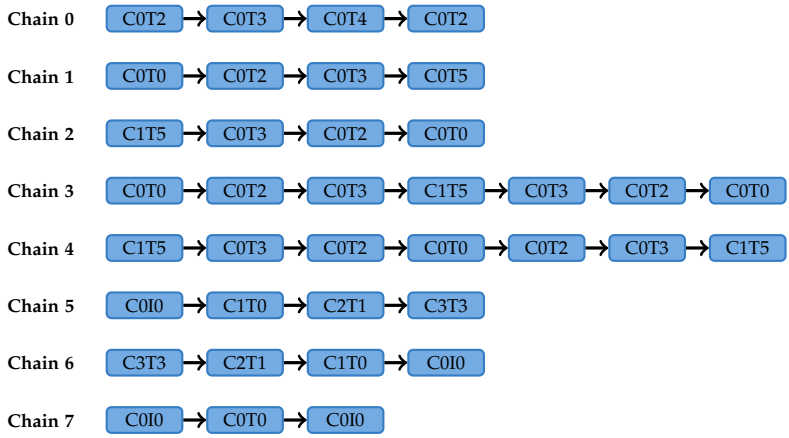
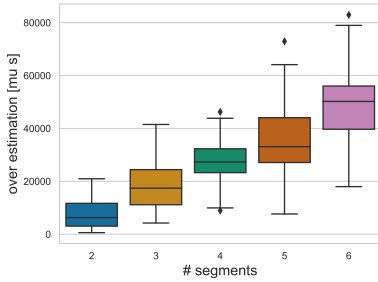


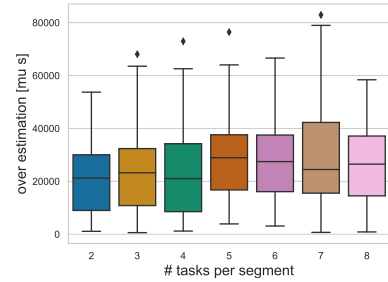
Figure 4.4. Test Chains

are scheduled. The probability that a resource schedules BET tasks is 0.5 in the case of mixed chains. The number of tasks per resource is 20. Priorities are assigned rate-monotonic.

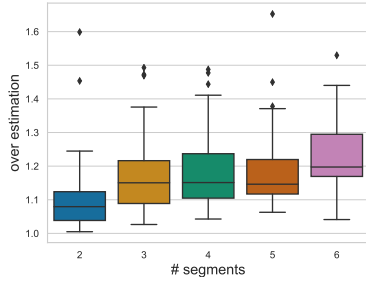
- Task periods are selected from the set of periods proposed in a generic automotive benchmark published [78]. The angle-synchronous tasks were randomly assigned periods between one and five milliseconds. This is done in order to obtain WCRTs from the generated WCETs more easily.
- WCETs of BET tasks are generated with the *UUnifast*-algorithm [24] where the resource utilization is set to 0.69.
- The WCET of a LET task with period p_T is randomly selected from the set $\{ 250\mu s, 300\mu s, \ldots, p_T \}$.
- Finally, one cause-effect chain spanning over all resources in the system and including $\{ 2, 4, 8 \}$ tasks on each resource is generated. We only use the periods from 1ms to 20ms for the chains since these make up almost all time-critical chains. Although time-critical cause-effect chains through task with a lower frequency exist, they usually have, according to the slower processing frequency, relatively weak requirements.



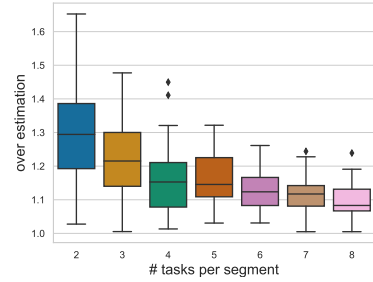
(a) Absolute Overestimation by Number of Segments



(b) Absolute Overestimation by Number of Tasks per Segment



(c) Relative Overestimation by Number of Tasks per Segment



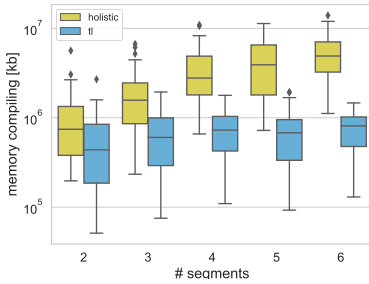
(d) Relative Overestimation by Number of Tasks per Segment

Figure 4.5. Precision

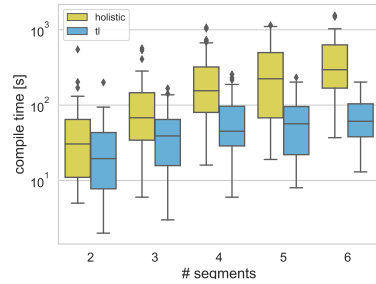
▷ All tasks are generated without an offset in order to obtain WCRTs from the generated WCETs more easily. This also makes the problem instances harder for the solver as more interference within the task instances occurs.

We carried out the experiments to investigate on the following questions:

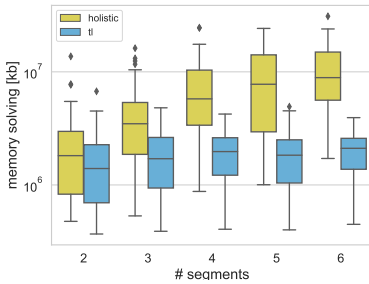
1. How does the resource-need for compilation and solving increase when the length of the chain is increased by either adding more tasks on one resource or adding additional resources.



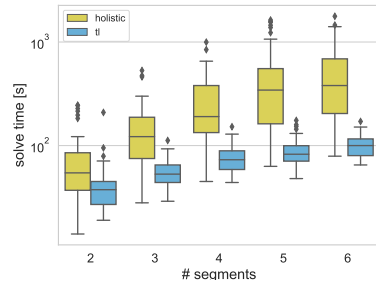
(a) Memory Needed for Compilation



(b) Time Needed for Compilation

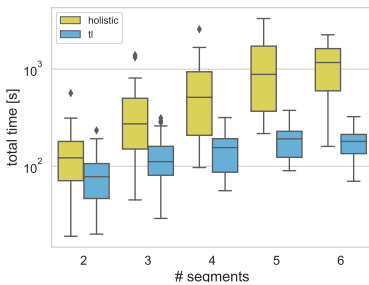


(c) Memory Needed for Solving

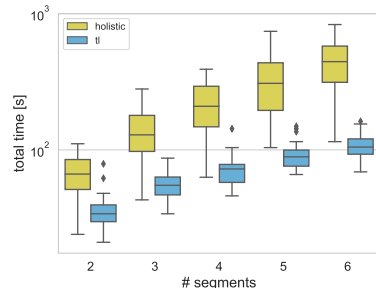


(d) Time Needed for Solving

Figure 4.6. Resource Consumption for Compilation and Solving



(a) Total Analysis Time BET Chains



(b) Total Analysis Time LET Chains

Figure 4.7. Total Analysis Time - BET vs. LET Chains

2. How does a context-loss affect the tightness of the estimations.

3. What is the difference between the analysis of chains consisting of only LET, only BET or mixed LET and BET task regarding performance and precision of the presented approach.

The experiments were carried out on a desktop computer equipped with two Intel(R) Xeon(R) Gold 6242 CPUs and 256GB of memory.

To analyze the resource need for the analysis of the complete chain and compare it to the resource need for the split chain, we analyzed each resource individually in the latter case and summed up the results plus an bound on the transition latencies between the resources. Since we have no offsets for any task instance, this is the period the task in the case of LET and the period plus the difference between the WCRT and BCRT of a task instance otherwise. Different conclusions can be drawn from the results. First of all, we see that both, compilation of the constraint model and solving of the FlatZinc model, scale a little worse than linearly but significantly better than quadratic with respect to the number of segments as depicted in Figure 4.6. The time and memory consumption for analysis rises significantly slower in the case of divided analysis, however we also see a sharp rise in the absolute and relative overestimation when compared to the precise results of the holistic analysis. The trend is depicted in Figure 4.5. With an increasing number of segments analyzed, the number of context losses increases. The overestimation decreases if more tasks are taking part on the chain on each segment. From a practical application perspective, the outliers to the maximal relative over-estimation especially for a small amount of context losses is problematic. Potentially having an overestimation of over 50% for any context loss does not justify the savings in computational effort, especially if the approach can be adjusted for specific cases as proven in the previous section.

Regarding the third question for this experiment we generated two additional task sets following the same procedure as before but one containing only BET tasks and one only containing LET tasks. The difference in the total analysis time is depicted in Figure 4.7. It shows a little less distribution in the time needed for analysis, but more importantly that the total analysis time is greatly reduced in the case of LET-only chains. Table 4.10 also reflects this. Here we observe a drop from about 7% timeouts

at compilation and about 9% timeouts at solving of BET-only chains to 0% in total for LET chains.

4.8 Summary and Conclusion

In this chapter, we have added a new layer to the formal model defined in the previous chapter. The encoding in a constraint program allows us to deploy a CP solver to directly obtain worst-case estimations with an example of the situation they occurred in. Several measures to check validity and improve performance have been presented and we have shown that this approach is unmatched in precision. Although the approach does not scale for arbitrary system sizes we have shown that it scales for task set sizes relevant to analyze cause-effect chains in the software current and next generation of automotive CPS.

Algorithm 3: Calculate Time Frame of possible Interaction

```
1  GPTimeFrame Time t_min, Time t_max, Task T, Tasks 𝒯
2      (i_min, i_max) ← InstMinMax(t_min, t_max, P, 𝒯)
3      tfs ← nil
4      a ← Task.ActivationModel(T)
5      if a ∈ ChainedActivationModel then
6          id ← ChainedActivationModel.Predecessor(a)
7          P ← find task with id id in 𝒯
8          ((pf_{i_min,1}, pf_{i_min,2}), ..., (pf_{i_max,1}, pf_{i_max,2})) ←
             GPTimeFrame(t_min, t_max, P, 𝒯)
9      for i ∈ (i_min, ..., i_max) do
10         (tf_min, tf_max) = (t_min, t_max)
11         d ← Task.Deadline(T)
12         if a ∈ PeriodicActivationModel then
13             p ← PeriodicActivationModel.Period(a)
14             o ← PeriodicActivationModel.Offset(a)
15             l ← o + (i · p)
16             (tf_min, tf_max) ← (l, l + d)
17         else if a ∈ ChainedActivationModel then
18             (tf_min, tf_max) ← (pf_{i,1}, pf_{i,2} + d)
19         else
20             l ← tdom_max
21             if a ∈ BoundActivationModel then
22                 l ← BoundActivationModel.MinDt(a)
23             else if a ∈ SporadicActivationModel then
24                 l ← SporadicActivationModel.MinDt(a)
25             a ← i − i_min
26             b ← (i_max − i_min) − i
27             mi ← t_min + (a · l)
28             ma ← t_max − (b · l) + d
29             (tf_min, tf_max) ← (mi, ma)
30         tfs ← cons(tfs, (tf_min, tf_max))
31     return tfs
```

Table 4.6. Results for Experiment 1 and Experiment 2

Chain	Response time (μs)			Data age (μs)		
	pyCPA	task-level[1]	task-level[2]	pyCPA	task-level[1]	task-level[2]
Chain 0	22160	10650	10650	20910	10650	10650
Chain 1	21420	10740	10800	15170	6740	6800
Chain 2	15410	4750	4750	19930	8660	8720
Chain 3	34620	15250	15250	32890	15160	15220
Chain 4	36110	15240	15300	34380	15240	15300
Chain 5	27685	15780	15780	7685	5780	10549
Chain 6	16520	10015	11748	31520	25015	25734
Chain 7	5265	5015	6355	5265	5015	6355

task-level[1] : Task set listed in Table 4.8 with activation model A05
task-level[2] : Task set listed in Table 4.8 with activation models A14/A15

Table 4.7. Results for Experiment 3

Experiment	Chain	Data age (μs)		
		results from [112]	task-level	runnable-level
default	Chain 2	-	148283	120101
$E\pi\phi$	Chain 2	160415	148283	120101
$E\pi\phi A$	Chain 2	134207	128283	110101
default	Chain 3	-	4927	3148
$E\pi\phi$	Chain 3	5249	4200	3148
$E\pi\phi A$	Chain 3	4683	4981	2422

Table 4.8. Task set for benchmark

Name	Core	Prio	Bcet	Wcet	Deadline	Act. Model	Preemptive
C0T0	0	20	100	160	250	A13	false
C0T1	0	19	100	240	1000	A09	false
C0T2	0	18	100	700	1250	A05	false
C0T3	0	17	200	250	3500	A12	false
C0T4	0	16	100	250	2000	A08	true
C0I0	0	30	10	15	500	A05/A14	false
C0I1	0	29	10	15	500	A05/A15	false
C0I2	0	28	10	15	500	A05/A15	false
C0I3	0	27	10	15	500	A05/A15	false
C1T0	1	20	100	160	250	A13	false
C1T1	1	19	100	180	1000	A09	false
C1T2	1	18	100	190	1000	A06	false
C1T3	1	17	100	220	1000	A05	false
C1T4	1	16	175	250	2750	A11	false
C1T5	1	15	100	240	2500	A07	true
C1T6	1	14	125	250	5000	A10	true
C1T7	1	13	130	250	10000	A04	true
C1I0	1	30	10	15	500	A05/A15	false
C1I1	1	29	10	15	500	A05/A15	false
C1I2	1	28	10	15	500	A05/A15	false
C1I3	1	27	10	15	500	A05/A15	false
C2T0	2	20	100	160	500	A09	false
C2T1	2	19	100	180	5000	A05	true
C2T2	2	18	125	150	10000	A03	true
C2T3	2	17	150	200	100000	A00	true
C2I0	2	30	10	15	500	A05/A15	false
C2I1	2	29	10	15	500	A05/A15	false
C2I2	2	28	10	15	500	A05/A15	false
C2I3	2	27	10	15	500	A05/A15	false
C3T0	3	20	100	160	500	A09	false
C3T1	3	19	100	180	5000	A05	true
C3T2	3	18	125	290	10000	A04	true
C3T3	3	17	200	250	20000	A02	true
C3T4	3	16	210	300	100000	A01	true
C3T5	3	15	250	300	100000	A01	true
C3I0	3	30	10	15	500	A05/A15	false
C3I1	3	29	10	15	500	A05/A15	false
C3I2	3	28	10	15	500	A05/A15	false
C3I3	3	27	10	15	500	A05/A15	false

Table 4.9. Activation models for task set in Table 4.8

(a) Periodic Activation Models

Name	**Period** (*ms*)	**Offset** (*ms*)
A00	1000.0	2.5
A01	100.0	0.0
A02	20.0	5.0
A03	10.0	5.0
A04	10.0	7.5
A05	5.0	0.0
A06	5.0	0.5
A07	5.0	1.0
A08	5.0	1.5
A09	5.0	2.0
A10	5.0	2.5
A11	5.0	3.5
A12	5.0	4.0
A13	1.0	0.5

(b) Sporadic/Bound Activation Models

Name	**min. dt** (*ms*)	**max. dt** (*ms*)
A14	5.0	6.0
A15	5.0	–

Table 4.10. Timeouts BET vs. LET vs. mixed Chains

Approach	**Chains**	**Timeout Rate** (%)		**#Task Set**
		Compiler	Solver	
Hol	LET	0.00	0.00	280
	BET	7.14	9.29	140
	Mixed	1.07	3.57	140
TL	LET	0.00	0.00	280
	BET	0.00	5.00	140
	Mixed	0.00	1.07	140

Chapter 5

Latencies of Distributed Cause-effect Chains

Recalling Definition 3.2.46 and Definition 3.2.49, one part of the estimation is still missing, the time to transmit a signal s when send from communication port a to communication port b: $\overline{\text{Trans}}\,(a, s, b)$. In this chapter we therefore discuss the problem of estimating the time needed to communicate a change in a signal value from one ECU to the next. The scenario is depicted in Figure 5.1. A distributed cause-effect chain runs through different task instances until it reaches the border of ECU A. The information about the *cause* having happened is then transmitted to ECU B. In Chapter 3 a hierarchy of different communication artifacts has been defined to model the network communication in a distributed automotive CPS. This hierarchy is often neglected in state-of-the-art analyses on network transmission latencies. A lot of approaches have been presented to estimate the transmission time on the physical layer. While this is an important part of the total time needed for transmission, a second, equally important source of delay is not taken into account. PDUs are not necessarily sent directly when the signal changes. They have different so-called *trigger* conditions. The actual mapping of signals to PDUs and PDUs to PDUs of a lower layer is determined at run time and depends on different packing mechanisms. The combination of event-based packing, immediate sending, and dynamic mapping leads to complex situations where it is not directly evident whether an updated value is sent with the very next encapsulating PDU. In Figure 5.1 this might lead to the situation that the change in the signal value is not transmitted with the third but with the fourth instance of Frame 1. As a consequence, it might not be received by the third but the fourth instance of Task 5. Although it does not have an impact in this

Sequence: Task 1 → Task 4 → Frame 1 → Task 5 → Task 7 → Task 5 → Frame 2 → Task 4

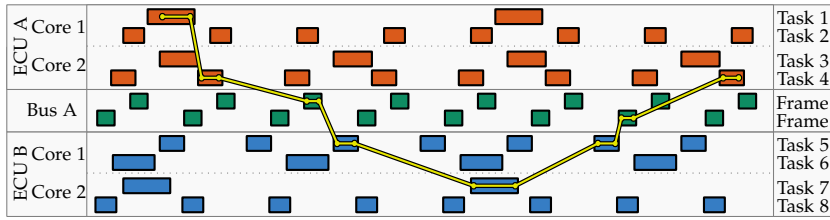


Figure 5.1. Example of a cause-effect chain with one possible instance-level flow

particular example, this can lead to longer end-to-end latencies in general and therefore must be addressed by end-to-end analyses.

5.1 Related work

Since software and network communication latencies need to be considered to obtain end-to-end lantencies, related work comes from two categories. Approaches to estimate software latencies are discussed in Chapter 4. The other category of related work is formed by approaches to estimate the latency due to network communication, i.e. the delay between a network controller of one ECU and the network controller of the next ECU in the chain. Respective analyses have been presented for different automotive field busses [36, 97]. so-called *holistic* approaches are additionally concerned with parts of the ECU's software [81]. However, currently they rely on single-core analyses [82]. To reduce complexity and therefore make analysis applicable, *compositional* approaches were developed. In compositional performance analysis (CPA) different local scheduling analyses are combined to obtain end-to-end estimations [106]. CPA got a lot of research attention, and was used e.g. for the end-to-end response time analysis in automotive systems [113]. In its basic form however, CPA suffers from the problem that multiple worst-cases are possibly considered simultaneously although they can not occur at the same time. Recent work in the area underlines applicability for industrial-scale use cases [56, 128]

and improves, i.e. reduces, pessimism on the estimations [76].

5.2 Network Analysis

In this section we analyze the temporal behavior of network communication artifacts on different levels.

5.2.1 Combining Activation Models

Similar to the *arrival curves* introduced in [33] we use *timing models* to describe the nature of how different events occur. However, unlike the *cumulative function* of *Network Calculus* [27] or the *interval bound functions* of *Real-time Calculus* [129] we do not use them to derive request and response counts but use them to bound the interval of time in which an event might occur.

5.2.1 Definition (Timing Model)**.** A timing model is a function $m : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ which maps the i^{th} occurence of to the first and the last possible point of time the event might occur.

For elements of the codomain of timing models we use the projection functions π_1 and π_2 to access the respective element, e.g. let $p = (a, b) \in \mathbb{N} \times \mathbb{N}$, then $\pi_1(p) = a$ and $\pi_2(p) = b$.

With an eye on CP encoding, it is important that a finite representation of the timing models is available. That being said, we distinguish two types of timing models: (1) *Periodic Timing Models* and (2) *Sporadic Timing Models*. Periodic models are used to describe events triggered by periodic clocks. The time in which these events might occur does not vary. Sporadic models allow the specification of temporally less predetermined events. Here, the possible point of time for an occurrence of the event can only be be narrowed down to a time span. Note, that sporadic events with no upper bound on the time between two occurrences are not considered here.

5.2.2 Definition (Periodic Timing Models)**.** A periodic timing model $t^{\mathbf{P}}_{o,p,n} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a timing model parameterized with three arguments offset $o \in \mathbb{N}$ and period and number of occurrences $p, n \in \mathbb{N}_{>0}$ with

$t^{\mathbf{P}}_{o,p,n}(i) = (a_i, a_i)$ where $a_i = o + \left\lceil \frac{i}{n} \right\rceil \cdot p$ for all $i \in \mathbb{N}$. The family of periodic timing models is defined by $\mathcal{TM}^{\mathbf{P}} = \{ t^{\mathbf{P}}_{o,p,n} \mid o \in \mathbb{N}, p, n \in \mathbb{N}_{>0} \}$.

5.2.3 Definition (Sporadic Timing Models)**.** A sporadic timing model $t^{\mathbf{S}}_{l,u,n} : \mathbb{N} \to \mathbb{N} \times \mathbb{N}$ is a timing model parameterized with three arguments minimum inter-arrival time and number of occurrences $l, n \in \mathbb{N}_{>0}$ and maximum inter-arrival time $u \in \mathbb{N}_{>l}$ with $t^{\mathbf{S}}_{l,u,o}(i) = (l_i, u_i)$ where

$$l_i = \left\lceil \frac{i}{n} \right\rceil \cdot l \text{ and } u_i = \left(\left\lceil \frac{i}{n} \right\rceil + 1 \right) \cdot u$$

for all $i \in \mathbb{N}$. The family sporadic timing models is defined by $\mathcal{TM}^{\mathbf{S}} = \{ t^{\mathbf{S}}_{l,u,n} \mid l, n \in \mathbb{N}_{>0}, u \in \mathbb{N}_{>l} \}$

Let $\mathcal{TM} = \mathcal{TM}^{\mathbf{P}} \cup \mathcal{TM}^{\mathbf{S}}$ be the set of timing models.

To enable a cut of the analysis, it is necessary to be able to combine different timing models for an event, e.g. when we want to describe the timing of signal changes when the is signal value is written from different runnables. To combine multiple possible time intervals for an event, we define the $\sqcup$-operator for timing models.

5.2.4 Definition (Union of Timing Models)**.** Let $t_0, t_1 \in \mathcal{TM}$ be timing models. We distinguish three cases:

Case 1 $t_0 = t^{\mathbf{P}}_{o_0,p_0,n_0}$ is a periodic timing model and $t_1 = t^{\mathbf{P}}_{o_1,p_1,n_1}$ is a periodic timing model.

$$t_0 \sqcup t_1 = \begin{cases} t^{\mathbf{P}}_{o,p',n'} & \text{if } o_0 = o_1 \wedge \mod (p_{\max}, p_{\min}) = 0 \\ t^{\mathbf{S}}_{l,u,n'} & \text{else} \end{cases}$$

with $p_{\min} = \min \{ p_0, p_1 \}$, $p_{\max} = \max \{ p_0, p_1 \}$, $l = \min \{ p_0, p_1 \}$, $u = \max \{ o_0, o_1 \} + p_{\max}$, $n' = n_0 + n_1$, and $p' = p_{\min}$.

Case 2 $t_0 = t^{\mathbf{S}}_{l,u,n_0}$ is a sporadic timing model and $t_1 = t^{\mathbf{P}}_{o,p,n_1}$ is a periodic timing model.

$$t_0 \sqcup t_1 = t_1 \sqcup t_0 = t^{\mathbf{S}}_{l',u',n'}$$

with $l' = \min \{ l, p \}$, $u' = \max \{ o + p, u \}$, and $n' = n_0 + n_1$.

Case 3 $t_0 = t^{\mathbf{S}}_{l_0,u_0,n_0}$ is a sporadic timing model and $t_1 = t^{\mathbf{S}}_{l_1,u_1,n_1}$ is a sporadic timing model.

$$t_0 \sqcup t_1 = t^{\mathbf{S}}_{l',u',n'}$$

with $l' = \min(\{\, l_0, l_1, |l_0 - l_1| \,\})$, $u' = \max(\{\, u_0, u_1 \,\})$, and $n' = n_0 + n_1$.

Since we are interested in safe bounds for the end-to-end latencies, it needs to be assured that the union of two timing models contains all possible periods of time from both timing models. For this we define the order relation for containment in Definition 5.2.5 and show that the union of two timing models A and B contains all occurrences from A and B in terms of the possible points of time in Theorem 5.2.7.

5.2.5 Definition (Order Relation for Timing Models)**.** We define an order $\sqsupseteq \subseteq \mathcal{TM} \times \mathcal{TM}$ Let $t_0, t_1 \in \mathcal{TM}$ be timing models,

$$t_0 \sqsupseteq t_1 \Leftrightarrow \forall i \in \mathbb{N} \colon \exists j \in \mathbb{N} \colon \pi_1\left(t_0(i)\right) \geqslant \pi_1\left(t_1(j)\right) \wedge \pi_2\left(t_0(i)\right) \leqslant \pi_2\left(t_1(j)\right).$$

5.2.6 Lemma (Commutativity of $\sqcup$)**.** *Let $t_0, t_1 \in \mathcal{TM}$ be timing models. Then, the following holds: $t_0 \sqcup t_1 = t_1 \sqcup t_0$.*

Proof. The cases 1 and 3 of Definition 5.2.4 are build solely on commutative functions. Case 2 is commutative by definition. □

5.2.7 Theorem (Monoticity of $\sqcup$)**.** *Let $t_0, t_1 \in \mathcal{TM}$ be timing models. It holds $t_0 \sqsupseteq t_0 \sqcup t_1$ and $t_1 \sqsupseteq t_0 \sqcup t_1$.*

Proof. We prove $t_0 \sqsupseteq t_0 \sqcup t_1$. The same cases as in the definition of $\sqcup$ are distinguished:

Case 1 $t_0 = t^{\mathbf{P}}_{o_0,p_0,n_0}$ is a periodic timing model and $t_1 = t^{\mathbf{P}}_{o_1,p_1,n_1}$ is a periodic timing model. Let $p_{\min} = \min\{\, p_0, p_1 \,\}$, $p_{\max} = \max\{\, p_0, p_1 \,\}$, $p' = p_{\min}$, and $n' = n_0 + n_1$. This case has two subcases:

Case 1.1 Assume $o_0 = o_1$ and $\mod(p_{\max}, p_{\min}) = 0$. The claim in this case is:

$$\forall i \in \mathbb{N} \colon \exists j \in \mathbb{N} \colon \pi_1\left(t^{\mathbf{P}}_{o_0,p_0,n_0}(i)\right) \geqslant \pi_1\left(t^{\mathbf{P}}_{o,p',n'}(j)\right) \wedge$$

$$\pi_2\left(t^{\mathbf{P}}_{o_0,p_0,,n_0}(i)\right) \leqslant \pi_2\left(t^{\mathbf{P}}_{o,p',n'}(j)\right).$$

For an arbitrary $i \in \mathbb{N}$, this means that $j \in \mathbb{N}$ needs to be found such that

$$\left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \leqslant \left\lceil \tfrac{j}{n'} \right\rceil \cdot p' \text{ and } \left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \geqslant \left\lceil \tfrac{j}{n'} \right\rceil \cdot p'$$

If $p' = p_0$, the two inequations can be satisfied by setting $j = \left\lceil \frac{i}{n_0} \right\rceil \cdot n'$. Otherwise, $p' = p_1$ and due to the assumptions of Case 1.1, there must be a $d \in \mathbb{N}$ such that $\frac{p'}{p_0} = d$. Therefore, setting $j = \left\lceil \frac{i}{n_0} \right\rceil \cdot n' \cdot d$ satisfies both inequations.

Case 1.2 Assume $o_0 \neq o_1$ or $\mod(p_{\max}, p_{\min}) \neq 0$. Let $l = \min\{p_0, p_1\}$ and $u = \max\{o_0, o_1\} + p_{\max}$, then the claim in this case is:

$$\forall i \in \mathbb{N}\colon \exists j \in \mathbb{N}\colon \pi_1\left(t^{\mathbf{P}}_{o_0,p_0,n_0}(i)\right) \geqslant \pi_1\left(t^{\mathbf{S}}_{l,u,n'}(j)\right) \wedge \pi_2\left(t^{\mathbf{P}}_{o_0,p_0,n_0}(i)\right) \leqslant \pi_2\left(t^{\mathbf{S}}_{l,u,n'}(j)\right).$$

For an arbitrary $i \in \mathbb{N}$, this means that $j \in \mathbb{N}$ needs to be found such that

$$\left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \geqslant \left\lceil \tfrac{j}{n'} \right\rceil \cdot l \text{ and } \left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \leqslant \left(\left\lceil \tfrac{j}{n'} \right\rceil + 1\right) \cdot u.$$

Setting $j = \left\lceil \frac{i}{n_0} \right\rceil \cdot n'$ will satisfy this equation because

$$\begin{aligned} \leqslant o_0 + p_0 \geqslant \ell &\Rightarrow \forall n \in \mathbb{N}\colon n \cdot p_0 \geqslant n \cdot \ell \\ &\Rightarrow \forall n \in \mathbb{N}\colon o_0 + (n \cdot p_0) \geqslant n \cdot \ell \end{aligned}$$

and

$$\begin{aligned} o_0 + p_0 \leqslant u &\Rightarrow \forall n \in \mathbb{N}\colon (n \cdot (p_0 + o_0)) \leqslant n \cdot u \\ &\Rightarrow \forall n \in \mathbb{N}\colon o_0 + (n \cdot p_0) \leqslant n \cdot u. \end{aligned}$$

Case 2 Let $t^{\mathbf{S}}_{l,u,n_0}$ be a sporadic and $t_1 = t^{\mathbf{P}}_{o,p,n_1}$ be a periodic timing model. Furthermore, let $l' = \min\{l, o\}$, $u' = \max\{o + p, u\}$, and $n' = n_0 + n_1$. In this case we have to prove two sub-cases.

Case 2.1 $t_0 \sqsupseteq t_0 \sqcup t_1$. In this case, the claim is

$$\forall i \in \mathbb{N}\colon \exists j \in \mathbb{N}\colon \pi_1\left(t^{\mathbf{S}}_{l,u,n_0}(i)\right) \geqslant \pi_1\left(t^{\mathbf{S}}_{l',u',n'}(j)\right) \wedge$$

$$\pi_2\left(t^{\mathbf{S}}_{l,u,n_0}(i)\right) \leqslant \pi_2\left(t^{\mathbf{S}}_{l',u',n'}(j)\right).$$

For an arbitrary $i \in \mathbb{N}$, this means that $j \in \mathbb{N}$ needs to be found such that

$$\left\lceil \tfrac{i}{n_0} \right\rceil \cdot l_0 \geqslant \left\lceil \tfrac{j}{n'} \right\rceil \cdot l' \wedge$$
$$\left(\left\lceil \tfrac{i}{n_0} \right\rceil + 1\right) \cdot u_0 \leqslant \left(\left\lceil \tfrac{j}{n'} \right\rceil + 1\right) \cdot u'.$$

Setting $j = \left\lceil \frac{i}{n_0} \right\rceil \cdot n'$ will satisfy both inequations since $l' \leqslant l$ and $u' \geqslant u$ as per definition of l' and u'.

Case 2.1 $t_1 \sqsupseteq t_0 \sqcup t_1$. In this case, the claim is

$$\forall i \in \mathbb{N} \colon \exists j \in \mathbb{N} \colon \pi_1\left(t^{\mathbf{P}}_{o,p,n_0}(i)\right) \geqslant \pi_1\left(t^{\mathbf{S}}_{l',u',n'}(j)\right) \wedge$$
$$\pi_2\left(t^{\mathbf{P}}_{o,p,n_0}(i)\right) \leqslant \pi_2\left(t^{\mathbf{S}}_{l',u',n'}(j)\right).$$

For an arbitrary $i \in \mathbb{N}$, this means that $j \in \mathbb{N}$ needs to be found such that

$$\left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \leqslant \left\lceil \tfrac{j}{n'} \right\rceil \cdot l' \wedge$$
$$\left\lceil \tfrac{i}{n_0} \right\rceil \cdot p_0 \leqslant \left(\left\lceil \tfrac{j}{n'} \right\rceil + 1\right) \cdot u'.$$

Since again $l' \leqslant o + p$ and $u' \geqslant o + p$ this case follows analogously to Case 1.2.

Case 3 $t_0 = t^{\mathbf{S}}_{l_0,u_0,n_0}$ is a sporadic timing model and $t_1 = t^{\mathbf{S}}_{l_1,u_1,n_1}$ is a sporadic timing model. Let $l' = \min\{l_0, l_1\}$, $u' = \max\{u_0, u_1\}$, and $n' = n_0 + n_1$. Then, the claim is:

$$\forall i \in \mathbb{N} \colon \exists j \in \mathbb{N} \colon \pi_1\left(t^{\mathbf{S}}_{l_0,u_0,n_0}(i)\right) \geqslant \pi_1\left(t^{\mathbf{S}}_{l',u',n'}(j)\right) \wedge$$
$$\pi_2\left(t^{\mathbf{S}}_{l_0,u_0,n_0}(i)\right) \leqslant \pi_2\left(t^{\mathbf{S}}_{l',u',n'}(j)\right).$$

This case follows analogously to Case 2.1.

Commutativity of $\sqcup$ also provides $t_1 \sqsupseteq t_0 \sqcup t_1$ for all three cases. □

Theorem 5.2.7 allows us to combine arbitrary timing models for events. In the next section we use this to describe the complex temporal behavior of different PDU triggers.

Based on the system model described in Chapter 3 our formal model to describe the temporal behavior of the communication artifacts comprises four types of elements: (1) signals, (2) PDUs, (3) frames and (4) communication tasks. For each element type we distinguish two types of variables: (1) input parameter, and (2) modeling variables. The input parameter of a task include a timing model for its activation and a deadline. Any frame which is in triggered state when the task is activated is copied from global memory to the buffer of the communication controller. We do not analyze the time needed for copying buffers and calculating message authentication codes here. Although the model can be extended to be used for such analyses, the processing time of messages in the communication task is just assumed to meet all deadlines here. Finally, we assume that the communication controller immediately tries to put the frame on the physical medium. The input parameter for a signal consists of a single timing model and a reference to a PDU. Signal changes are generated by the runnables of the transmitting ECU. For each producing runnable, a timing model describing the points of time the signal value possibly changes is derived from the activation model of the task. These timing models are *summed up* with the help of the $\sqcup$-operation. The resulting timing model is the aforementioned input. The input parameters of a PDU are more diverse. First of all, the different triggering options have to be considered. Besides a direct triggering by contained signals, this can be a threshold for the filling level, and a timeout. Furthermore, for PDUs which are encapsulated in dynamically filled container PDUs, the *collection semantics* are needed to describe possible behaviors. The collection semantics can either be *last-is-best* or *queued*. Queued collection semantics guarantee that every instance of the contained PDU is visible on the wire (cf. [12]). Thirdly, the maximum length and the size of the threshold for triggering need to be known to determine a triggering due to the filling level. Finally, a reference to the encapsulating frame is included in the set of input parameters for a PDU. The input parameters of a frame comprise its priority for arbitration, its length and a reference to the communication task responsible for its transmission. These parameters are specific for CAN-FD and might need to be adjusted for other physical layer protocols.

Besides its input parameters, each model element has different types of modeling variables subjected to the modeling constraints. Firstly, there

Table 5.1. Time Variables of Network Model Elements

Element	Variable	Event description
Signal	$\varphi^S_{i,j}$	The time span for the j^{th} change of signal i
PDU	$\alpha^P_{i,j}$	The time span in which the j^{th} instance of PDU i is triggered
PDU	$\sigma^P_{i,j}$	The time span in which the j^{th} instance of PDU i is moved to the lower level buffer
Frame	$\alpha^F_{i,j}$	The time span in which the j^{th} instance of frame i is triggered
Frame	$\sigma^F_{i,j}$	The time span in which the j^{th} instance of frame i is tried to be sent by its sending communication task
Frame	$\epsilon^F_{i,j}$	The time span in which the j^{th} instance of frame i is fully received by its receiving communication task
Task	$\alpha^P_{i,j}$	The time span for the activation of the j^{th} instance of task i
Task	$\epsilon^P_{i,j}$	The time span for the completion of the j^{th} instance of task i

are the *time* related variables listed in Table 5.1. Secondly, each instance of any element is encapsulated in an instance of a lower-layer element. This is modeled in the parameter n for the types signal (n^S), PDU (n^P) and frame (n^F). It contains a reference to the instance of the container for each occurrence of the respective element. For frames the semantic is slightly different as n contains the instance of the transmitting communication task in this case. Thirdly, in order to obtain safe estimations, a minimum amount of occurrences of each element needs to be considered. Therefore, assume that $\mathbb{T}$ covers a sufficient period of time in which all combinations of relative offsets between occurrences of the timing models appear. Bounds on the length of this period are discussed below. Furthermore, set $T_{\text{sup}} = \sup(\mathbb{T})$ a value to indicate invalid points of time. The maximum number of occurrences can be computed for most model elements, if this time interval is fixed. For all the remaining elements, i.e. the container PDUs, the number of occurrences has to be derived from the occurrences of the contained elements. Let $\Omega^S_i = \{\, \text{occ\_min}^S_i, \ldots, \text{occ\_max}^S_i \,\}$ be the index set for the occurrences of signal i. Let Ω^P, Ω^F, and Ω^T hold the index set of occurrences for PDUs, frames, and tasks respectively.

The update of a signal value is modeled with the help of a timing model as described above. Let t^S_i be the timing model of signal i. Note that t^S_i can be sporadic or periodic. The S indicates that it belongs to a signal here. The following constraint is added to the model for all $j \in \Omega^S_i$:

$$\varphi^S_{i,j} \geqslant \pi_1\left(t^S_i(j)\right) \wedge \varphi^S_{i,j} \leqslant \pi_2\left(t^S_i(j)\right) \tag{5.2.8}$$

When a signal value was updated, the so-called *update bit* is set. The update of the value then eventually triggers the sending of a PDU. However, since we are interested in the transmission time for the changed value, we also need to model in which occurrence of its designated container (PDU^S) the update is transmitted. This is done by adding the following constraints for all signals i and their occurrences $j \in \Omega_i^S$:

$$\forall k \in \Omega_\ell^P \colon (\varphi_{i,j}^S > \sigma_{\ell,k-1}^P \wedge \varphi_{i,j}^S \leqslant \sigma_{\ell,k}^P) \rightarrow (n_{i,j}^S = k) \; . \tag{5.2.9}$$

As described above two different events have to be considered for the triggering of PDUs : triggering due to timeout modeled by α^{P-T}, and triggering due to transmission request by containees modeled by α^{P-E}. If no clock triggering is configured, α^{P-T} is set to T_{sup}. Analogously, if a PDU is not triggered by any containee, α^{P-E} is set to T_{sup}. The containees of a non-container PDU i are signals. Accordingly, the following constraint is added for all $j \in \Omega_i^P$:

$$\alpha_{i,j}^{P-E} = \min \{ \, \varphi_{\ell,k}^S \mid \ell \in \text{sigs}_i^P, k \in \Omega_\ell^S \wedge n_{\ell,k}^S = j \, \} \; . \tag{5.2.10}$$

For container PDUs, possibilities for triggering are more complex. The following points of time are considered conditionally: the point of time the first containee was triggered $\alpha_{i,j}^{P-C1}$, the point of time the first containee was triggered plus the timeout of the container $\alpha_{i,j}^{P-CT}$, and the point of time the length of the contained PDUs exceeds the threshold of the container $\alpha_{i,j}^{P-Cn}$. To detect a triggering of a container PDU due to exceeding of the threshold, the filling level needs to be determined. To this end, we introduce an additional variable foreach instance j of a PDU i, $len_{i,j}^P$, which contains the total length of the PDU. Additionally, for a pair of PDUs i and j and each instance j of i and k of ℓ we add an auxiliary variable $c_{i,j,\ell,k}^P$ which holds 1 if k is contained in j and 0 otherwise, i.e.

$$c_{i,j,\ell,k}^P = \begin{cases} 1 & \text{if } \ell \in \text{PDU}_i^P \wedge n_{\ell,k}^P = j \\ 0 & \text{else} \end{cases} \tag{5.2.11}$$

for all PDUs i, ℓ and $j \in \Omega_i^P$, $k \in \Omega_i^L$.

Depending on the collection semantics, an instance of a PDU might be

overwritten if its container is not send between two updates. This means, that c^P is 1 for two instances of the contained PDU. To model the fact that an instance k of a PDU ℓ might be overwritten by a subsequent instance in the instance j of its container PDU i, we add an additional variable $o^P_{i,j,\ell,k}$ with

$$o^P_{i,j,\ell,k} = \begin{cases} 1 & \text{if } c^P_{i,j,\ell,k} = 1 \wedge \exists k' \in \Omega^P_\ell : k' > k \wedge n^P_{\ell,k'} = j \\ 0 & \text{else} \end{cases} \tag{5.2.12}$$

for all PDUs i, ℓ and $j \in \Omega^P_i$, $k \in \Omega^P_\ell$.

The length of a non-container PDU is fixed, based on the contained signals and a fixed-size header. The length of a container PDU depends on its PDU layout. If it has a *static* layout, the length is fixed. Otherwise, if it has a *dynamic* layout, the collection semantics of the contained PDUs is the crucial factor. If the collection semantic is *last-is-best* the content in the container can be overwritten. Otherwise, if the collection semantic is *queued*, multiple instances of the same PDU can be transmitted in one container. Note, that we assume that containers cannot be nested. The actual length of a container can therefore finally be calculated by summing the length of all not-overwritten containee instances, i.e.

$$len^P_{i,j} = \text{length}^{P-\text{H}}_i + \sum_{\substack{\ell \in \text{PDU}^P_i, \\ k \in \Omega^P_\ell}} (1 - o^P_{i,j,\ell,k}) \cdot c^P_{i,j,\ell,k} \cdot len^P_{i,j} \tag{5.2.13}$$

for all PDUs i, ℓ and $j \in \Omega^P_i$, $k \in \Omega^P_\ell$ where $\text{length}^{P-\text{H}}_i$ is the length of the header. In order to ensure correct modeling of the collection semantics, the following constraints are added conditionally:

$$n_{i,j} \leqslant n_{i,j+1} \qquad \text{if } i \text{ is collected } \textit{last-is-best} \tag{5.2.14}$$

$$n_{i,j} < n_{i,j+1} \qquad \text{if } i \text{ is collected } \textit{queued} \tag{5.2.15}$$

To determine the possible point of time for the triggering of a container PDU, the minimum of the values is used:

$$\alpha^P_{i,j} = \min \left\{ \alpha^{P-C1}_{i,j}, \alpha^{P-CT}_{i,j}, \alpha^{P-Cn}_{i,j} \right\}. \tag{5.2.16}$$

If one of the triggers is not applicable, the respective value is set to T_{sup}. This means, if $\alpha^P_{i,j} = T_{\text{sup}}$ the instance of the container PDU has not been

triggered and must not be considered further if not triggered due to a timeout. Finally, the triggering of an instance j of an PDU i happens at the first point of time it is possibly triggered by any of the described triggers, i.e.

$$\alpha_{i,j}^{P} = \min \{ \alpha_{i,j}^{P-E}, \alpha_{i,j}^{P-T} \} \tag{5.2.17}$$

for all PDUs i and PDU occurrences $j \in \Omega_i^P$.

Given $\alpha_{i,j}^{P}$ it can be described in which occurrences of the encapsulating PDU (PDU^P) j is possibly transmitted. However, since we are considering container PDUs with dynamic layouts decided at runtime, a PDU is not necessarily sent within the next instance of a container. In other words, if the container PDU is already filled to capacity, the containee has to wait until the next instance is sent. To model this, we constrain all instances of the container which are sent between the instance encapsulating the an instance j of an PDU i and the point of time j was triggered to be too full to contain j. The variable $n_{i,j}^{P}$ holds index of the encapsulating instance for all PDUs i and $j \in \Omega_i^P$. The variable $fn_{i,j}^{P}$ holds the index of the first instance which is a candidate for encapsulation, i.e. for all PDUs i and PDU occurrences $j \in \Omega_i^P$ which are mapped to a container PDU ℓ,

$$fn_{i,j}^{P} = \min \left(\{ \text{occ\_max}_{\ell}^{P} \} \cup \{ k \mid k \in \Omega_{\ell}^{P} \wedge \alpha_{\ell,k}^{P} \geqslant \alpha_{i,j}^{P} \} \right). \tag{5.2.18}$$

If the collection semantic of the PDU i into the container PDU ℓ is *queued*, the following constraint is added for each occurrence j in $j \in \Omega_i^P$:

$$\forall k \in \Omega_{\ell}^{P} \colon (k \geqslant fn_{i,j}^{P} \wedge k > n_{i,j-1}^{P} \wedge k < n_{i,j}^{P}]) \rightarrow (len_{\ell,k}^{P} + len_{i,j}^{P} > \text{length}_{\ell}^{P}). \tag{5.2.19}$$

Otherwise, if the collection semantic of the PDU i into the container PDU ℓ is *last-is-best*, the constraints for j in $j \in \Omega_i^P$ are depending on whether an instance of the same PDU is already part of the container. If this is the case, the content would simply be overwritten. Otherwise, if there exists no $j' \in \Omega_i^P \setminus \{ j \}$ such that $n_{i,j'}^{P} = k$, the following constraint needs to be added:

$$\forall k \in \Omega_{\ell}^{P} \colon (k \geqslant fn_{i,j}^{P} \wedge k < n_{i,j}^{P}) \rightarrow (len_{\ell,k}^{P} + len_{i,j}^{P} > \text{length}_{\ell}^{P}). \tag{5.2.20}$$

It is important to note that in this case, instances of PDU i must be excluded when computing the length of ℓ as previously contained instances would be overwritten.

Together with the constraint for the length of a container PDU, the Constraints 5.2.19 and 5.2.20 assure that no non-full container can be sent if one of its contained PDU is queued for sending and could fit into the container instance lengthwise.

Once triggered, the frame is queued for transmission on the bus. Thereupon, the frame data is copied from global memory to the buffer of the communication controller by the next instance of the responsible communication task. Therefore, in order to get the relative temporal distance between sending and receiving communication task, the possible point of time it was queued for transmission needs to be tracked for an instance of a frame. The following constraint is added for all frames i and their instances $j \in \Omega_i^F$:

$$\alpha_{i,j}^F = \min \{ \alpha_{\ell,k}^P \mid \ell \in \mathrm{PDU}_i^F, k \in \Omega_\ell^P \wedge n_{\ell,k}^P = j \} . \tag{5.2.21}$$

In the following, we describe the constraints needed to model CAN FD networks. They can easily be plugged in by adding the corresponding constraints. The constraints listed up to this point can consequently be reused for other bus types. Let iTx be the communication task for frame i and jTx be the first instance of this iTx after occurrence j of i was queued for transmission. Furthermore, let $\alpha_{iTx,jTx}^T$ be the activation time of this task and frame_i^{HP} the set of frames which have a higher priority than i. The following constraint is added to model the start of transmission:

$$\sigma_{i,j}^F = \max \{ \alpha_{iTx,jTx}^T \} \cup \{ \epsilon_{\ell,k}^F \mid \ell \in \mathrm{frame}_i^{HP}, k \in \Omega_\ell^F \wedge \sigma_{\ell,k}^F \leqslant \sigma_{i,j}^F \} . \tag{5.2.22}$$

For CAN FD, the transmission time of a frame is resulting from an arbitration phase and the time the bits are transmitted on the physical medium. In the arbitration phase, all nodes of the network agree on the node allowed to transmit next. The node trying to transmit the frame with the highest priority is allowed to continue sending after arbitration. From Constraint 5.2.22 it can be inferred that the occurrence j of frame i has the highest priority at point of time $\sigma_{i,j}^F$. Therefore, the end of transmission $\epsilon_{i,j}^F$

can be calculated in the following way:

$$\left\lfloor t_{\text{arb}} + (len^F_{i,j} \cdot t_{\text{bit}}) \right\rfloor \leqslant \epsilon^F_{i,j} \leqslant \left\lceil t_{\text{arb}} + (len^F_{i,j} \cdot t_{\text{bit}}) \right\rceil \tag{5.2.23}$$

where t_{arb} worst-case time needed for arbitration and t_{bit} is the time needed to transfer one bit of data. The length of the instance j of frame i, $len^F_{i,j}$, can be computed analogously to the PDUs (cf. Constraint 5.2.13). With the help of $\epsilon^F_{i,j}$ the instance of the communication task at the receiving ECU can be determined. The time between the initial change of the signal and the deadline of this task is the time needed for a value change of this signal to be transferred from the global memory of one ECU to the next. Note, that we do not analyze the behavior of the communication tasks here. Although details about their internal behavior can be added to the model, we currently just assume that all processing is done within the respective deadlines.

Given all constraints for a set of network artifacts, a constraint solver is deployed to obtain the worst-case transmission time. The solver searches all satisfying assignments for the one modeling the longest transmission time. This emulates an exhaustive search if all situations are covered. To guarantee this, all possible relative offset between the PDU containing the objective signal and all PDUs possibly causing a delay of this PDU needs to be covered. If all relevant PDUs are triggered by periodic timers or periodically changing signals, the least common multiple of these periods covers all relative offsets. If sporadic events are included, a lot more relative offsets are possible. However, in this case the solver freely chooses values between the first and last possible point of time for the event to happen (cf. Constraint 5.2.8). For practical application, it is furthermore important that $\mathbb{T}$ covers a sufficiently large time interval such that it is possible that all PDUs which are triggered can also be sent, i.e. an instance of the communication task responsible for sending must be added for each frame possibly in queue.

5.3 Solving

An introduction to the solving process is given in Section 4.5. Again, most of the constraints can directly be translated. Nevertheless, some particularities apply for the network model. They are are described in Section 5.3.1. How the DataZinc is generated for a cause-effect chain from our formal data model is described in Section 5.3.1.

5.3.1 MiniZinc Model

The constraints modeling the temporal behavior of signal changes and the resulting PDU triggerings are again implemented in a MiniZinc model. The constraints on the timing models of the signals can be implemented analogously to the timing models of tasks as presented in Listing 4.5. All other constraints have a rather longish representation in MiniZinc. Due to their length we can not discuss all constraints in detail here. A version of the full model including the case study data used in Section 5.3.3 is available online [29]. To include a small example at least, two of the core constraints are discussed in this section and can be found in Appendix 6.6

The constraints to model which occurrence of an PDU is used to transfer a change in a signal value are listed in Listing 7. We can see that the different rules have a direct and speaking representation in MiniZinc.

The constraints to model the triggering of a PDU are given in Listing 7 and Listing 8. Here we see even more complex rules. In the author's opinion, the MiniZinc representation is now at the limits of what can be called easy comprehensible at first glance. Nevertheless, the different cases and rules for triggering can be identified on a closer look.

5.3.2 DataZinc Generation

The generation of DataZinc is based on the information stored in the *TimingAttributes*-relationships (see Appendix 6.6). The only information additionally needed is the timing model of each signals. They can be derived using Algorithm 4. Note, that this algorithm does not support sporadic activation models as they have no upper bound on the time between two changes. This potentially leads to huge overestimation, however,

Algorithm 4: Generate Timing Model for a Signal

```
1  TimingModel Signal S, Task T, Tasks 𝒯
2      m ← ∅
3      a ← Task.ActivationModel(T)
4      d ← Task.Deadline(T)
5      for R ∈ SoftwareTask.Runnables(T) do
6          if NetworkSignal.SignalName ∈ RunnableEntity.Output
            then
7              n ← ∅
8              if a ∈ PeriodicActivationModel then
9                  n ← PeriodicActivationModel.Period(a)
10                 o ← PeriodicActivationModel.Offset(a)
11                 n ← i ↦ ((i · n) + o, (i · n) + o + d)
12             else if a ∈ ChainedActivationModel then
13                 id ← ChainedActivationModel.Predecessor(a)
14                 P ← find task with id id in 𝒯
15                 n ← TimingModel(S, P, 𝒯)
16             else if a ∈ BoundActivationModel then
17                 l ← BoundActivationModel.MinDt(a)
18                 u ← BoundActivationModel.MaxDt(a)
19                 n ← i ↦ ((i · n) + o, (i · n) + o + d)
20             if m = ∅ then
21                 m ← n
22             else
23                 m ← m ⊔ n
24     return m
```

signals generated by interrupts with no upper bound a usually sampled in periodic tasks before they are send via a network.

5.3.3 FlatZinc Solving and Results

FlatZinc solving is performed in the same way as described in Section 4.5.

Again the solution provided by the CP-solver allows to draw conclusions on the situation resulting in the worst-case latency. In this case, the

temporal order of the sending trigger on different layers can be tracked. It can be visualized which PDU is encapsulated in which lower-layer PDU and at which point of time the PDUs are processed, e.g. in a representation like depicted in Figure 3.3.

Following the same argumentation as in the previous chapter, an upper bound on for the end-to-end latency under the assumptions we made is estimated, if the constraint model is correct.

5.4 Model Application

In this section we want to prove the practical applicability by carrying out different evaluations.

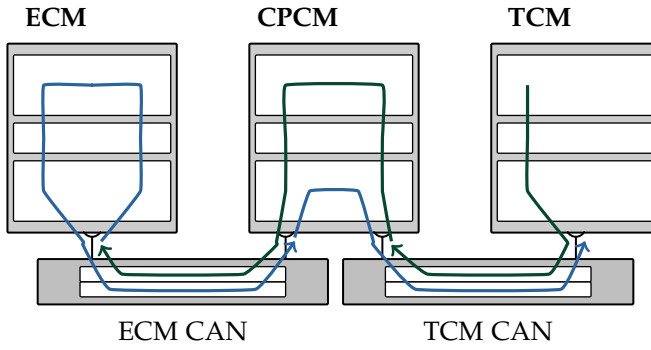


Figure 5.2. Case-Study: Topology and Chain

We analyzed the worst-case time for the transmission of eight different signals, four contained in dynamic container PDUs, four being part of the cause-effect chain depicted in Figure 5.2. The first-mentioned signals represent status updates sent from the ECM and TCM to the CPCM. The cause-effect chain on the other hand is a control loop and has two parts. The first part of the chain describes the data flow when the TCM sends a request for torque to the ECM. The CPCM checks the requests before forwarding it to the ECM. After providing torque to the extend possible, the ECM reports to the CPCM. The answer of the ECM is directly routed from

Table 5.2. Resource Usage for Compiling and Solving the Model

Sender	Receiver	Signal	Compiler		Solver			Result
			s	kbyte	s	kbyte		μs
ECM	CPC	Status A	2:29	4,435,568	2:30	57,462,088	*	15500
ECM	CPC	Status B	2:28	4,441,504	2:16	60,797,024	*	20500
TCM	CPC	Status C	0:55	1,723,920	0:46	8,772,148		15500
TCM	CPC	Status D	0:52	1,723,896	1:03	9,863,388		15500
TCM	CPC	Request 1	0:52	1,723,808	0:41	8,499,072		11250
CPC	ECM	Request 2	2:24	4,432,880	3:02	19,298,896		12500
ECM	CPC	Response 1	2:27	4,466,500	2:45	91,870,500		5500
CPC	TCM	Response 2	0:56	1,734,188	0:49	9,050,628		6000

*Running instances of the solver were interrupted after the optimum was found.

the ECM CAN to the TCM CAN for fast reception at the TCM.

The experiments were carried out on a desktop computer equipped with an Intel(R) Core(TM) i9-7940X CPU and 128GB of memory. The time and memory needed to compile the MiniZinc model as well as the time and memory needed for solving the FlatZinc model by 16 instances of *Chuffed* working in parallel are shown in Table 5.2. The amount of memory needed for the verification of the transmission time of Response 1 suggest that the model needs further investigation on scalability regarding memory usage. Unfortunately, a generic benchmark to generate configurations for CAN to carry out more general tests is currently not available.

5.5 System-level End-to-end Analysis

Recalling Definition 3.2.46 and Definition 3.2.49, only part of the estimations is missing which is mentioned briefly in Definition 3.2.39. The scheduling on each ECUs is based on an oscillator having a nominal clock speed. These clocks possibly do not meet their exact frequencies which leads a drift between the clocks. Depending on whether synchronization mechanism are used to put a bound on the maximum possible drift or not, the differences in the speeds of the ECUs have to be considered in the estimation. The model presented in this chapter therefore uses one ECU as the current time base and allows to specify a drift between the communication tasks. This drift is bound by the maximum of the frequencies of the

involved communication tasks.

An end-to-end estimation is finally achieved by deploying the software model and the network model alternatingly to estimate the latency of software chains and the time to transmit the changes in signal values between ECUs. Since the start and end of the analyzed sub-problems overlap at the communication tasks, the context loss in this is minimal: the software chain on the next ECU surely starts with the receiving communication task and from here on, all relative offsets between the other tasks of the ECU have to be considered either way.

5.6 Summary and Conclusion

In this chapter we closed the gap in the present methodology for end-to-end latency estimation by developing an approach allowing to estimate the time needed to transmit a change in a signal value from one ECU to another. We discussed how end-to-end latencies can be obtained in in synchronized and un-synchronized CAN-FD communication clusters. Future work may be conducted in the direction of the included software shares in the communication tasks, e.g. due to message authentication and further investigation on the trade-off between precision and computational effort when comparative figures from other approaches are available.

Chapter 6

Measurement Data Analysis

In this chapter we approach the problem of latency analysis from a different perspective. While Chapter 4 and Chapter 5 are concerned with techniques for the formal analysis of end-to-end latencies, in this chapter a method to analyze the systems performance based on measurement data is presented. It is based on the specification language EC.Lang which is used to specify searches for cause-effect chain instances in measurement data, i.e. in a set of system runs. The search is based on the changes in signal values which occur in the course of the chain. For a clear distinction between the formal analysis of the previous chapter and the data based analysis in this chapter, we refer to these changes as *events* in the following. The result of the analysis is a set of latencies, collected and aggregated for all event chain instances found.

6.1 Related work

EC.Lang closes the gap existing between well-studied temporal logic and an easy-to-use language for the specification of measurement data evaluations tailored for event chains.

Approaches to meet the demand for using time within a formal specification of sequential programs exist for a long time, e.g. [85, 59, 23]. More recent approaches for combining control and data-flow with *time* are concerned with the formal specification of distributed systems [37, 65].

At about the same time timing-aware programming languages were first introduced, temporal logic was used the first time as a basis for the formal verification of the correct behavior of time-constraint systems [44, 77, 2]. Note that *quantative* here refers to the assertions being quantitative. Recent advantages in this area include the combination of signal

temporal logic (STL) with timed regular expressions (TRE) [98] and time-stamp temporal logic [93]. Furthermore, real-time regular expressions with *event*-support have been presented in [63]. The approach described in [63] supports a discrete-time and real-time *goto* operation and *event concatenation*. Event concatenation can be used to connect events to chains and the *goto*-operation allows to move back and forth in evaluation-time. However, it does not allow to use the value of a signal at the time of a first event to be retrieved when a second event is matched. For the extended signal temporal logic a quantitative semantic was presented in [46]. It allows to collect numerical performance measures such as the average response time. The collected results are collected in a *match set* with start and end point of time in which the signals match the timed regular expression. However, this does not allow for the different parts of the chain to be tracked individually. In Section 6.3.3 we discuss why more differentiated semantics in the context of evaluations for event chain is needed.

Based on different dialects of diverse temporal logics, tools have been implemented to support formal verification of MATLAB/Simulink models, e.g. Breach [40] and S-TaLiRo [5]. Furthermore, tools allowing for the general analysis of measurement data based on timed regular expressions have been presented. Montre [135] is a tool supporting TRE pattern matching with no graphical user interface (GUI). AMT 2.0 is a Java-based GUI tool supporting offline monitoring of TRE and STL with several features like, e.g. parameterized property templates [98].

Our work addresses the formal specification of offline analyses. Closely related to this are languages used for the formal specification of run time monitoring, e.g. [18, 34]. Furthermore, the temporal stream-based specification language (TeSSLa) [90, 30] needs to be mentioned. TeSSLa is a specification language for monitoring signal streams with a strong focus on online monitoring. It includes a *last*-operator which allows to query the last value of an so-called event stream after a certain point of time.

Although solutions which allow to analyze the context between two events have been proposed earlier, EC.LANG is the first approach dedicated to complex chains with quantitative semantics. In Section 6.3.3 we discuss the particular need of a *chain context* for response-time evaluation of event chains. With *chain context* we mean that, in order to check whether a response matches a request, the quantity of the request at request-time

needs to be known when possible responses are evaluated.

6.2 Measurement Data Acquisition

As discussed in Chapter 1, an important prerequisite for dependable results is that the system runs analyzed cover a significant part of systems stimuli. This is a general problem of measurement-based approaches which is not further addressed here. However, one consequence is that

EC.LANG is designed to be independent from the source of measurement data. Nevertheless, the following assumptions must hold: (1) the time values of all measurement points are synchronized and (2) the reading is complete when the analysis starts.

Measurement data can consequently be acquired by software in the loop (SIL), hardware in the loop (HIL), and in-car experiments or even simulations. A plethora of commercial tools is available to obtain measurement data at different stages of the development process, like e.g.

- *Silver* a SIL software by QTronic Gmbh
- *TA Tool Suite* a simulation tool suite by Vector Informatik GmbH
- *T1* a tool for software instrumentation by GLIWA GmbH
- *CANOE* a measurement and simulation tool Vector Informatik GmbH
- *ControlDesk* a HIL software by dSPACE GmbH
- *INCA* a measurement and calibration tool by ETAS GmbH

6.3 EC.Lang

In this section EC.LANG is presented. It was first published in [53].

6.3.1 Syntax

In this section we define the syntax of EC.LANG. The syntax comprises three types of expressions: *Signal Expressions*, *Propositional Signal Expressions*, and

Chain Segment Expression. *Signal Expressions* are used to combine signals and constants arithmetically to define additional, derived signals. *Propositional Signal Expressions* map signals values to a truth value at points of time in $\mathbb{T}$. They are used to specify the occurrence of events in course of an event chain. Finally, these events are combined to chains in *Chain Segment Expressions*.

Signal Expressions are recursively defined based on signals, constants, and a fixed unary operator *Delta* which operates on signals only. The *Delta*-operator is used to obtain the change in value of a signal with respect to an interval of time. This is needed to identify jumps in signals which are the cause of almost every event. *Signal Expressions* can be connected via arithmetic operators.

6.3.1 Definition (Signal Expression)**.** The set of signal expressions EXP_S of type $k \in \mathcal{T}$ is recursively defined as:

1. (*Signals*)
   The identifier of a signal can be used as a placeholder for the signal value at some point of time. More precisely, for $(id, D) \in \mathcal{M}$ a signal of type k, id is a signal expression of type k.

2. (*Constants*)
   Every constant $c \in \mathbb{D}_k$ is a signal expression of type k.

3. (*Binary Arithmetic Expression*)
   Let $\circ \in \{ +, -, \cdot, \div \}$ and $op_0, op_1 \in \mathrm{EXP}_S$. Then,
   $$op_0 \circ op_1 \in \mathrm{EXP}_S$$
   if the type of op_0 equals the type of op_1.

4. (*Delta Expression*)
   Let $op \in \mathrm{EXP}_S$ and $dt \in \mathbb{Q}_{>0}$. Then,
   $$\mathrm{delta}\,(op, dt) \in \mathrm{EXP}_S\ .$$

Signal Propositional Expressions are recursively defined based on simple propositions, e.g. relational expressions. *Signal Propositional Expressions* can be connecteted via logical connectives.

6.3.2 Definition (Propositional Signal Expression)**.** Let EXP_S be a set of signal expressions. The set of propositional expressions EXP_P over EXP_S is recursively defined as:

1. (*True and False*)
   `false` $\in \text{EXP}_P$ and `true` $\in \text{EXP}_P$.

2. (*Negation*)
   Let $op \in \text{EXP}_P$. Then, $\neg(op) \in \text{EXP}_P$.

3. (*Binary Propositional Signal Expression*)
   Let $\circ \in \{ \wedge, \vee \}$ and $op_0, op_1 \in \text{EXP}_P$. Then,
   $$op_0 \circ op_1 \in \text{EXP}_P \ .$$

4. (*Relational Expression*)
   Let $\circ \in \{ =, \leqslant, <, > \}$ and $op_0, op_1 \in \text{EXP}_S$ of type k. Then,
   $$op_0 \circ op_1 \in \text{EXP}_P \ .$$

Signal Expressions and *Propositional Signal Expressions* are the basis for the core of EC.Lang: Chain Segment Expression (CSE). A CSE either denotes a start of a chain or a hop to the next segment. The start of a chain is given when the signals show a specific, well-defined behavior, e.g. the signal value changes by a certain amount within a certain amount of time. We say, an instance of a chain is triggered, the behavior defined in a start-expression is observed. EC.Lang now allows to keep track of the value of an arithmetic expression at the point of time the chain was triggered. This value is subsequently available to determine whether the request was answered appropriately at following hops. Since part of the correct response might also include the time needed to answer, additionally, a predicate to check whether the response meets timing constraints can be specified. The end of a chain is implicitly given if an expression does not have a successor.

6.3.3 Definition (Chain Segment Expression)**.** Let EXP_P be a set of propositional expressions over a set of signal expressions EXP_S. The set of chain segment expressions EXP_C is recursively defined as:

1. (*Start of Chain*)
   Let $X \in \mathbb{S}$ be an identifier, $e \in \mathrm{EXP}_S$ be a signal expression, and $c \in \mathrm{EXP}_P$ be a propositional expression. Then,
   $$(X \leftarrow e \texttt{ if } c) \in \mathrm{EXP}_C \,.$$
   We call X the identifier of the expression.

2. (*Hop*)
   Let $X, Y \in \mathbb{S}$ be identifiers, $e \in \mathrm{EXP}_S$ be a signal expression, $c \in \mathrm{EXP}_P$ be a propositional expression. λ_v be a predicate with $\lambda_v : (\Gamma \times \Gamma) \to \{\,\mathrm{false}, \mathrm{true}\,\}$, and λ_t be a predicate with $\lambda_t : (\mathbb{Q} \times \mathbb{Q}) \to \{\,\mathrm{false}, \mathrm{true}\,\}$.
   Then,
   $$(X \leftarrow Y \odot e \texttt{ if } c \texttt{ where } \lambda_t \texttt{ and } \lambda_v) \in \mathrm{EXP}_C \,.$$
   X is called the identifier of the expression and the expression identified by Y is called its predecessor.

6.3.2 Semantic

Each type of EC.Lang expression introduced in the previous section has different semantics. The semantics of a signal expression are functions mapping discrete points of time to valuations of a type. To obtain a valuation between two sampling points, we rely on the last valid value measured. Accordingly, for a signal, the valuation at point of time t is the last valid value before t.

6.3.4 Definition (Semantic of Signal Expressions)**.**

The semantic function $[\![\,\cdot\,]\!]_{\mathrm{EXP}_S} : \mathrm{EXP}_S \to (\mathbb{T} \to \Omega)$ maps every $E \in \mathrm{EXP}_S$ to a function returning valuations for points of time. It is recursively defined as:

1. (*Signals*)
   $[\![(id)]\!]_{\mathrm{EXP}_S} = v$ with $(id, D) \in \mathcal{M}$ the signal id of type k and
   $$v(t) = \begin{cases} \bot_k & \text{if } t < \min\left(T(D)\right) \\ v_i & \text{otherwise} \end{cases}$$
   where $i = \min\left(\{\,|D|\,\} \cup \left\{j \mid t_j \in T(D) \wedge t_j \geqslant t\right\}\right)$.

2. (*Constants*)
   $[\![c]\!]_{\text{EXP}_S} = v$ with $v(t) = c$.

3. (*Binary Arithmetic Expression*)
   $[\![op_0 \circ op_1]\!]_{\text{EXP}_S} = t \mapsto [\![op_0]\!]_{\text{EXP}_S}(t) \circ [\![op_1]\!]_{\text{EXP}_S}(t)$

   Note, that syntactic restrictions assure that only expressions of the same type might be operands of an arithmetic expression.

4. (*Delta expression*)

$$[\![\text{delta}\,(op, dt)]\!]_{\text{EXP}_S} = t \mapsto ([\![op]\!]_{\text{EXP}_S}(t) - [\![op]\!]_{\text{EXP}_S}(t - dt))$$

Based on the semantic of *Signal Expressions* the semantic function of *Propositional Signal Expressions* determines whether a predicate over one or multiple signals holds true at some point of time $t \in \mathbb{T}$.

6.3.5 Definition. Semantic of Propositional Signal Expression
The semantic function $[\![\cdot]\!]_{\text{EXP}_P} : \text{EXP}_P \to (\mathbb{T} \to \{ \text{false}, \text{true} \})$ maps every $E \in \text{EXP}_P$ to a function which evaluates whether the proposition holds at a point of time. It is recursively defined as:

(*True and False*)
$[\![\texttt{false}]\!]_{\text{EXP}_P} = t \mapsto \text{false}$ and $[\![\texttt{true}]\!]_{\text{EXP}_P} = t \mapsto \text{true}$.

(*Negation*)

$$[\![\neg(op)]\!]_{\text{EXP}_P} = t \mapsto \begin{cases} \text{true} & \text{if } [\![op]\!]_{\text{EXP}_P} = \text{false} \\ \text{false} & \text{otherwise} \end{cases} \tag{6.3.6}$$

(*Binary Propositional Signal Expression*)
Let $\circ \in \{ \wedge, \vee \}$ be a logical connective. The semantic of the binary propositional expression using operator $\circ$ is:

$$[\![op_0 \circ op_1]\!]_{\text{EXP}_P} = t \mapsto [\![op_0]\!]_{\text{EXP}_P}(t) \circ_{\text{Bool}} [\![op_1]\!]_{\text{EXP}_P}(t) \tag{6.3.7}$$

where $\circ_{\text{Bool}}$ stands for the corresponding Boolean operation.

(*Relational Expression*)
Let $\circ \in \{ =, \leqslant, <, > \}$ be a relational operator. The semantic of the relational expression using operator $\circ$ is:

$$[\![op_0 \circ op_1]\!]_{\mathrm{EXP}_P} = t \mapsto [\![op_0]\!]_{\mathrm{EXP}_S}(t) \circ_k [\![op_1]\!]_{\mathrm{EXP}_S}(t) \tag{6.3.8}$$

where $\circ_k$ is the corresponding relational operator for type op_0 and op_1. Note that syntactic restrictions assure, that only expressions of the same type get compared.

Finally, based on *Signal Expression* and *Propositional Signal Expression* we define the semantics of CSEs. The semantic function for CSEs makes use of nested lists. Therefore, let

$$\mathcal{D}_\ell = \{ ((t_1, v_1), \ldots, (t_\ell, v_\ell)) \mid t_j \in \mathbb{Q}, v_j \in \Omega \cup \{ \perp \} \text{ f.a. } j \in \{ 1, \ldots, \ell \} \}$$

denote set of lists of trace data points of length ℓ for all $\ell \in \mathbb{N}$. Furthermore, let

$$\mathcal{T} = \{ (d_1, \ldots, d_m) \mid m \in \mathbb{N}, d_i \in \mathcal{D}_\ell \text{ f.a. } i \in \{ 1, \ldots, m \} \text{ and } \ell \in \mathbb{N}_{\geqslant 1} \}$$

denote the set of lists of lists of trace data points where the inner lists have equal lengths. Since the codomain of the semantic function is a set of lists of lists, the author wants to recall that the definitions related to lists are given in Chapter 2.

To evaluate a CSE, i.e. calculate its semantic, a set of further CSEs is needed to calculate the semantics of potential predecessors. The semantic of the successor is then *appended* as defined in Definition 6.3.9. In that, $\mathcal{P}(\mathrm{EXP}_C)$ denotes the power set of the set of CSEs. We assume that each set of CSEs $\mathcal{E}$ is well-formed in the sense that f.a. $E \in \mathcal{E}$, if E is a *Hop Expression*, the predecessor of E is also contained in $\mathcal{E}$ and that all identifiers of CSEs are unique.

6.3.9 Definition (Semantic of CSEs)**.**
The semantic function $[\![\cdot]\!]_{\mathrm{EXP}_C} : \mathcal{P}(\mathrm{EXP}_C) \times \mathrm{EXP}_C \to \mathcal{T}$ maps a set of CSEs and a single CSE E to a list of lists where each inner list is the semantic of the predecessor of E extended by the chain segment described in E. It is

recursively defined as:

1. (*Start of Chain*)
   Let $E = X \leftarrow e \texttt{ if } c$. Then,

$$[\![(\mathcal{E}, E)]\!]_{\mathrm{EXP_C}} = (T_i)_{i=1}^{\ell} \tag{6.3.10}$$

   with a sequence $T_i = ((t, v))$ where

$$v = [\![e]\!]_{\mathrm{EXP_S}}(t) \tag{6.3.11}$$

   for every $t \in \mathbb{T}$ with

$$[\![c]\!]_{\mathrm{EXP}_P} = \text{true} \wedge [\![e]\!]_{\mathrm{EXP_S}}(t) \neq \bot_k \,. \tag{6.3.12}$$

2. (*Hop*)
   Let $E = X \leftarrow Y \odot e \texttt{ if } c \texttt{ where } \lambda_t \texttt{ and } \lambda_v$ and

$$Pred = [\![y]\!]_{\mathrm{EXP_C}}\,, y \in \mathcal{E} \text{ and } y \text{ identified with } X \tag{6.3.13}$$

   Then,

$$[\![(\mathcal{E}, E)]\!]_{\mathrm{EXP_C}} = (T_i)_{i=1}^{\ell} \tag{6.3.14}$$

   with a sequence of the form

$$T_i = ((t_0, v_0), \ldots, (t_m, v_m), (t, v)) \tag{6.3.15}$$

   where

$$((t_0, v_0), \ldots, (t_m, v_m)) = \mathrm{elem}_i\,(Pred) \tag{6.3.16}$$

   and

$$t_{\mathrm{prev}} = \begin{cases} -\infty & \text{if } i = 1 \\ \pi_1\,(\mathrm{elem}_m\,(\mathrm{elem}_{i-1}\,(Pred))) & \text{else} \end{cases}. \tag{6.3.17}$$

   and

$$\begin{aligned} R = \big\{ t \in \mathbb{T} \;\mid\; & t \geqslant t_m \wedge t \geqslant t_{\mathrm{prev}} \wedge \\ & [\![e]\!]_{\mathrm{EXP_S}}(t) \neq \bot_k \wedge \\ & \lambda_v(v_m, v) = \text{true} \;\wedge \\ & \lambda_t(t_m, t) = \text{true} \,\big\} \end{aligned} \tag{6.3.18}$$

and

$$v = [\![e]\!]_{\mathrm{EXP}_S}(t) \tag{6.3.19}$$

$$t = \begin{cases} \min(R) & \text{if } R \neq \varnothing \\ \bot & \text{otherwise} \end{cases} \tag{6.3.20}$$

f.a. $i \in \{1, \ldots, \ell\}$, $\ell = \mathrm{len}(Pred) + 1$ and $m = \mathrm{len}(\mathrm{elem}_1(Pred))$.

6.3.3 Advantage of EC.LANG

For the analysis of a non-trivial event chain it is desirable to be able to track the different segments of the chain regarding the individual response times. When looking at longer chains, specifications based on logical operations *Globally* ($\Box$) and *Eventually* ($\Diamond$) can result in some considerable overhead if evaluated naively. Furthermore, a specification using a timed temporal logical does not allow to keep track of the number of currently open requests. Let's take a chain incorporating three signals as an example. Assume it originates in a request in a signal A and propagates to a request in signal B which finally leads to a change in a response-signal C. In the following we will refer to requests and responses as *events*. Assume the availability of predicates which describe the occurrence of the above-mentioned events. More precisely, let φ^X be terms to check whether an occurrence of the respective event $X \in \{A, B, C\}$ happened at some point of time. To check whether two occurrences are related, let $\psi^{(Y,Z)}$ be a term to check whether an occurrence of an event in Y matches an event in Z for all $(Y, Z) \in \{(A, B), (B, C)\}$. To express the constraint that an event in A is followed by a matching event in B within an Interval $I_{A,B}$ can then be expressed as:

$$\Box \Big(\underbrace{\varphi^A \rightarrow \Diamond_{I_{A,B}} \left(\varphi^B \wedge \psi^{(A,B)} \right)}_{=\varsigma} \Big) . \tag{6.3.21}$$

Further, the following formula expresses the expected continuation of the chain in signal C within an interval of $I_{B,C}$:

$$\Box \left(\varsigma \rightarrow \Diamond_{I_{B,C}} \left(\varphi^C \wedge \psi^{(B,C)} \right) \right) . \tag{6.3.22}$$

Now, if it is observed that Formula 6.3.22 does not hold at some point of time, it is not directly evident whether Formula 6.3.21 already failed or not. In other words, it is not clear whether the transition from signal A to signal B caused the response time miss or the transition from signal B to signal C. To keep track of this, both formulas need to be evaluated. If this is done naively, it can lead to some considerable overhead. The advantage of EC.Lang's semantics is, that they allow for a detailed breakdown of the individual shares of the chain segments to the end-to-end response time. Moreover, classically, the semantics of a formula of timed temporal logic is used to express qualitative constraints. The result is either true, meaning all requirements are met or false, meaning an assertion for a requirement failed. However, besides hard real-time demands, real-life systems additionally feature more vague requirements, like "the customer should not encounter a noticeable delay". Here, *should not encounter* means, that in a majority of cases the delay should be below some threshold. To track these kinds of requirements, quantitative semantics are more suitable. For timed temporal logics this is realized with the help of so-called *match-sets* which contain the start and the end of the interval time where the formula evaluates to true. Therefore, it is necessary to evaluate the individual formulas completely for an analysis of the individual chain segments. However, a more severe drawback of the specification in Formula 6.3.21 and Formula 6.3.22 is, that it does not allow to use the value of A at the point of time where φ^A evaluates to true in $\psi^{(B,C)}$. It is only possible to use the value of signal A at the point of time φ^B is evaluated. Thus, a specifying

> *If the value of A changes, after some time, the value of B equals the value of A*

is possible but

> *If the value of A changes, after some time, the value of B equals the value of A at the point of time of the initial change*

is not. To keep track of event chains, it is necessary to specify the second requirement since the value of A might change the time between its initial change and the time B *reacts*. In the following section we describe how this can be achieved with EC.Lang by means of an example.

6.4 Example

The example we consider here is part of the torque request and deliver functionality of an communication cluster like the one presented in the end of Chapter 3. Figure 6.1 depicts the data flow of the chain. First, the TCM sends a request to change torque to the CPCM, e.g. for a gear change. The CPCM splits the request into three individual torque requests for the ECM, and the controllers of the electrical motors (INV0, INV1). All motors try to adjust the torque to the possible extent and send a response with the delivered torque accordingly. In our example, a change to a lower value in TrqReq reflects the request for reduced torque. If such a request arrives at the CPCM, an according jump is expected to be observed. This change then propagates to the values of $TrqReq_C$, $TrqReq_{E_0}$ and $TrqReq_{E_1}$. Eventually, after some time, the sum of the three signal values equals *TrqReq*. The allocation of the torque requests inside the CPCM can be tracked further with the help of ECU-internal variables. To keep things readily comprehensible, we stay at network-level here. The variation in the reaction time of a combustion engine is greater than the one of an electrical motor. Therefore, the full amount of torque demanded does not always have an exact equal counterpart in the deliver signal and response times might vary. For measurement data analysis this means, that equality of requested and delivered values includes a small margin. If the deviation gets too large, i.e. it extends the margin, the missing values need to be identified correctly. If no reaction occurs, the request might have been dropped and the reason has to be tracked down. However, in the majority of cases the response signal eventually equals the request signal after some time. In some cases, due to the different physical properties of the motors, the sum of Trq_C, Trq_{E_0}, and Trq_{E_1} does not directly jump to the amount of torque requested in *TrqReq*. The difference in time between the final response and the initial request is the time required for the system to react to a torque demand of the transmission.

6.4.1 Example (Evaluation of Example encoded in EC.LANG).

$$Start \leftarrow TrqReq \texttt{ if } \mathrm{abs}(\mathrm{delta}(TrqReq, 0.001)) > 5$$

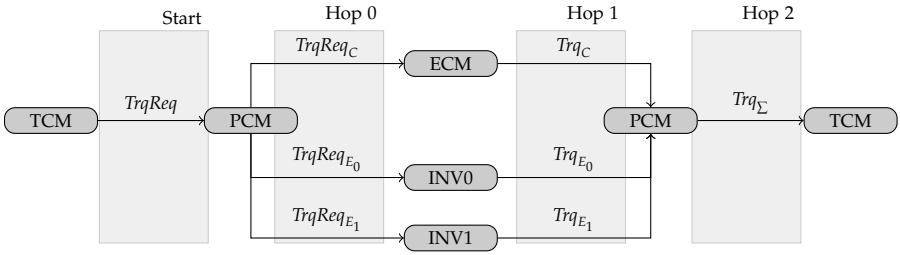


Figure 6.1. Torque Request and Deliver

$$
\begin{aligned}
Hop_1 \leftarrow\ & Start \odot TrqReq_C + TrqReq_{E_0} + TrqReq_{E_1} \\
& \texttt{if } \mathrm{abs}(\mathrm{delta}(TrqReq_C, 0.001)) > 1 \\
& \quad \wedge\ \mathrm{abs}(\mathrm{delta}(TrqReq_{E_0}, 0.001)) > 1 \\
& \quad \wedge\ \mathrm{abs}(\mathrm{delta}(TrqReq_{E_1}, 0.001)) > 1 \\
& \texttt{where } (v_0, v_1) \mapsto (v_0 = v_1) \\
& \texttt{and } (t_0, t_1) \mapsto (t_1 - t_0 \geqslant 0.005 \wedge t_1 - t_0 \leqslant 0.010) \\
Hop_2 \leftarrow\ & Hop_1 \odot Trq_C + Trq_{E_0} + Trq_{E_1} \\
& \texttt{if } \mathrm{abs}(\mathrm{delta}(Trq_C, 0.001)) > 1 \\
& \quad \wedge\ \mathrm{abs}(\mathrm{delta}(Trq_{E_0}, 0.001)) > 1 \\
& \quad \wedge\ \mathrm{abs}(\mathrm{delta}(Trq_{E_1}, 0.001)) > 1 \\
& \texttt{where } (v_0, v_1) \mapsto (\mathrm{abs}(v_0 - v_1) \leqslant 2.0) \\
& \texttt{and } (t_0, t_1) \mapsto (t_1 - t_0 \geqslant 0.010 \wedge t_1 - t_0 \leqslant 0.015) \\
Hop_3 \leftarrow\ & Hop_2 \odot Trq_\Sigma \\
& \texttt{if } \mathrm{abs}(\mathrm{delta}(Trq_\Sigma, 0.001)) > 1 \\
& \texttt{where } (v_0, v_1) \mapsto (v_0 = v_1) \\
& \texttt{and } (t_0, t_1) \mapsto (t_1 - t_0 \geqslant 0.005 \wedge t_1 - t_0 \leqslant 0.010)
\end{aligned}
$$

Algorithm 5: Build Dependency Graph for Expressions

```
1 BuildDependencyGraph EC.Lang-String S
2     E ← Translate(S); E ← ∅, V ← ∅;
3     forall Expression E in E do
4         if E is Start-Expression then
5             V ← V ∪ { e };
6         else if E is Hop-Expression then
7             p ← Predecessor(e); V ← V ∪ { e }; E ← E ∪ { (p, e) };
8     return G = (E, V)
```

6.5 Implementation

Our implementation of an EC.LANG-compiler is based on the ANTLR parser generator[1]. On top of the compiler we further implemented an evaluation engine and a GUI tool called Measurement Analyzer 2.

6.5.1 EC.LANG Compiler

The compilation is divided into two steps. First, all relevant signals loaded, and secondly, a dependency graph for the chain expressions is built.

We implemented two extensions of Antlr4's `TreeVisitor`-class. The first links the signal names of the CSEs with the respective signals from the measurement data file. The second one compiles a string of EC.LANG to the respective dependency graph G. Each node in G is labeled with a CSE. An edge in G connects two vertices, if the expression of the target node depends on the expression of the source node. The generation of the dependency graph is shown in Algorithm 5. A node in the generated graph has in-degree 0 if and only if it is representing a *Start Expression*. Otherwise, it has an incoming edge, namely the expression describing the preceding chain segment.

[1] https://github.com/antlr/antlr4

6.5.2 Evaluation Engine

The algorithm which is used to evaluate a dependency graph describing an EC.LANG-specification is shown in Algorithm 6. The evaluation again consists of two steps. First, all nodes with in-degree 0 are collected. As described above, all of these nodes must be labeled with a *Start Expression*. The semantics of expressions can be evaluated in parallel since they are guaranteed to have no mutual dependencies. The evaluated expressions are stored in the set of reachable expressions (R).

Algorithm 6: Compute Evaluations for Dependency Graph

```
1  VisitDependencyGraph Dependency Graph G = (V, E)
2      E ← ∅; R_start ← { n | n ∈ V where n has indegree 0 };
3      forall Start-Expression SE in R do
4          E ← E ∪ { (SE, [[SE]]) };
5      c ← false;
6      while c = false do
7          R_new ← { n | n ∈ V ∧ ∃p ∈ R : (p, n) ∈ E };
8          forall Hop-Expression HE in R_new do
9              E ← E ∪ { (HE, [[HE]]) };
10         c ← R_new ⊆ R; R ← R ∪ R_new;
11     return E
```

Subsequently, all *Hop Expressions* that are reachable via some edge originating at a node in R are visited iteratively. For each visiting step, evaluation of the newly reachable nodes can also be done in parallel. The evaluation stops if no new nodes are reachable. In our implementation the semantics of each expression is put in a map with its identifier as the key. For subsequent analyses each section of the chain can be examined in detail, e.g. to obtain an average response time between the start of a chain and a succeeding segment.

6.5.3 Performance of the Evaluation Engine

We created five files with randomly generated trace data for 20, 40, 60, 80, and 100 minutes. Each file comprises four signals: *Request*, *ResponseA-0*, *ResponseA-1*, and *Response-B*. The signals can have a value in $\{0, \ldots, 255\}$. The request signal randomly increases and decreases over time. If the request reaches the value 255, it drops to 0. Additionally, it drops to 0 with a low probability. The first response comes in two responses which need to be added to detect the randomly delivered response. The reaction time is at least thirty milliseconds. The divided response is summed up in *ResponseB* after some additional twenty milliseconds. With a decreasing probability reaction times are exceeded. Listing 6.1 shows the EC.Lang specification tested on the data.

```
Start <- 'Request'
if abs(delta('Request', 0.011)) >= 0.2 /\
abs(delta('Request', 0.011)) <= 2.0;

Split <- Start & 'ResponseA-0' + 'ResponseA-1'
if abs(delta('ResponseA-0', 0.011)) > 0.0 \/
abs(delta('ResponseA-1', 0.011)) > 0.0
where (t0, t1) |-> t1-t0 > 0.01 /\
t1-t0 <= 0.06
and (v0, v1) |-> v0 < v1 * 1.10 /\
v0 > v1 * 0.90;

Join <- Split & 'ResponseB'
if abs(delta('ResponseB', 0.011)) > 0.0
where (t0, t1) |-> t1-t0 > 0.01 /\
t1-t0 < 0.04
and (v0, v1) |-> abs(v0 - v1) < 0.01;
```

Listing 6.1. EC.Lang Specification used for Evaluation

Table 6.1 shows the evaluation speed achieved on a laptop computer with an Intel i7-8850H CPU and 32GB of memory. The columns show the time needed to evaluate the corresponding part of the chain. The

Table 6.1. Performance of the evaluation engine

Data set	Number of requests	Evaluation time (seconds)			
		Start	*Split*	*Join*	Sum
20m	50774	0.21	4.34	3.34	7.88
40m	105958	0.42	16.74	14.10	31.25
60m	153758	0.76	37.29	30.46	68.52
80m	212187	0.88	68.39	55.91	125.17
100m	258707	1.22	102.20	83.91	187.34

higher evaluation time of the expression labeled *Split* is due to the higher range of accepted reaction times and the fact that values are summed up deterministically in *ResponseB*. Although the ratio between measurement duration and time needed for analysis does show a linear increase, the experiment shows that the evaluation engine offers sufficient performance for the practical use. For experiments on measurement data obtained from long-term test however, the data files should be split into multiple smaller files for evaluation. This also allows a parallel processing on multiple machines. The current restrictions on performance can be overcome, if an upper bound for the time in which a potential response to a request should be search for can is specified. Since the evaluation engine in the current implementation always search to the end of the file if no corresponding response was found for a request, the length of trace in combination with the amount of errors potentially impacts performance.

6.6 Summary and Conclusion

With EC.Lang we developed a language which allows its user to specify the evaluation of distributed cause-effect chains based on measurement data. We discussed the need for an approach with dedicated chain support. The syntax of EC.Lang is designed to be simple so that they can be used after short induction phase. EC.Lang has quantitative semantics which allow to track down the segment of the chain with the largest contribution to the latency.

Appendix A - Database Script

```
create table ProjectInfo (
  prj_id             varchar( 64),
  prj_name           varchar(255),
  prj_creator        varchar( 64),
  prj_creationDate   datetime    ,
  prj_description    varchar(255)
  primary key (prj_id)
);

create table NCDRelease (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  ncd_publicationWeek  smallint,
  primary key(prj_id, ncd_id),
  foreign key(prj_id) references ProjectInfo(prj_id)
)

create table SWRelease (
  prj_id             varchar( 64),
  ecu_id             varchar( 64),
  release_id         varchar( 64),
  sprint_id          varchar( 64),
  realase_date       datetime    ,
  primary key(prj_id, ecu_id, release_id, sprint_id),
  foreign key(prj_id, ecu_id) references ECU(prj_id, ecu_id)
)
```

Listing 1. T-SQL Script to Create Project Relationships

```
create table Network (
  prj_id             varchar( 64),
```

```
  nw_id              varchar( 64),
  nw_name            varchar(255),
  nw_type            varchar( 64)
  primary key(prj_id, nw_id),
  foreign key(prj_id) references ProjectInfo(prj_id)
);

create table NWChannel (
  prj_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  ch_name            varchar(255),
  ch_nr              smallint    ,
  ch_type            varchar( 64),
  primary key(prj_id, nw_id, ch_id),
  foreign key(prj_id, nw_id) references Network(prj_id, nw_id)
);


create table NWResource (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  isSocket           bit,
  isFrame            bit,
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id),
  foreign key(prj_id, nw_id, ch_id) references NWChannel(prj_id,
   nw_id, ch_id),
  foreign key(prj_id, ncd_id) references NCDRelease(prj_id,
   ncd_id),
  constraint Frame_Service_XOR check (isSocket ^ isFrame is NOT
   NULL )
);

create table NWResource_TimingAttributes (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
```

```
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  attr_id            varchar( 64),
  attr_parent        varchar( 64),
  attr_value         varchar( 64),
  attr_value_type    varchar( 64),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id, attr_id,
   attr_value),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id) references
   NWResource(prj_id, ncd_id, nw_id, ch_id, res_id)
);

create table NWFrame (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  frame_name         varchar(255),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id) references
   NWResource(prj_id, ncd_id, nw_id, ch_id, res_id)
);

create table NWSocket (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  socket_name        varchar(255),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id) references
   NWResource(prj_id, ncd_id, nw_id, ch_id, res_id)
);

create table NWPDU (
  prj_id             varchar( 64),
```

```
  ncd_id              varchar( 64),
  nw_id               varchar( 64),
  ch_id               varchar( 64),
  res_id              varchar( 64),
  pdu_id              varchar( 64),
  pdu_name            varchar(255),
  pdu_length          int,
  pdu_type            varchar( 64),
  pdu_isContainer     bit,
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id) references
   NWResource(prj_id, ncd_id, nw_id, ch_id, res_id)
);

create table NWPDU_Container (
  prj_id              varchar( 64),
  ncd_id              varchar( 64),
  nw_id               varchar( 64),
  ch_id               varchar( 64),
  res_id              varchar( 64),
  pdu_id_container    varchar( 64),
  pdu_id_contained    varchar( 64),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id,
   pdu_id_container, pdu_id_contained),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id,
   pdu_id_container) references NWPDU(prj_id, ncd_id, nw_id, ch_id
   , res_id, pdu_id),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id,
   pdu_id_contained) references NWPDU(prj_id, ncd_id, nw_id, ch_id
   , res_id, pdu_id)
);

create table NWPDU_TimingAttributes (
  prj_id              varchar( 64),
  ncd_id              varchar( 64),
  nw_id               varchar( 64),
  ch_id               varchar( 64),
  res_id              varchar( 64),
  pdu_id              varchar( 64),
```

```
  attr_id            varchar( 64),
  attr_parent        varchar( 64),
  attr_value         varchar( 64),
  attr_value_type    varchar( 64),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
   attr_id, attr_value),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
   references NWPDU(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
);

create table NWSignal (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  pdu_id             varchar( 64),
  signal_id          varchar( 64),
  signal_name        varchar(255),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
   signal_id),
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
   references NWPDU(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
);

create table NWSignal_TimingAttributes (
  prj_id             varchar( 64),
  ncd_id             varchar( 64),
  nw_id              varchar( 64),
  ch_id              varchar( 64),
  res_id             varchar( 64),
  pdu_id             varchar( 64),
  signal_id          varchar( 64),
  attr_id            varchar( 64),
  attr_parent        varchar( 64),
  attr_value         varchar( 64),
  attr_value_type    varchar( 64),
  primary key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
   signal_id, attr_id, attr_value),
```

```
    foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
     signal_id) references NWSignal(prj_id, ncd_id, nw_id, ch_id,
     res_id, pdu_id, signal_id)
  );

  create table NWSignalGroup (
    prj_id              varchar( 64),
    ncd_id              varchar( 64),
    nw_id               varchar( 64),
    ch_id               varchar( 64),
    res_id              varchar( 64),
    pdu_id              varchar( 64),
    group_id            varchar( 64),
    group_name          varchar(255),
    primary key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
     group_id),
    foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
     references NWPDU(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
  );

  create table NWSignal_NWSignalGroup_Map (
    prj_id              varchar( 64),
    ncd_id              varchar( 64),
    group_id            varchar( 64),
    nw_id               varchar( 64),
    ch_id               varchar( 64),
    res_id              varchar( 64),
    pdu_id              varchar( 64),
    signal_id           varchar( 64),
    primary key(prj_id, ncd_id, group_id, signal_id),
    foreign key(prj_id, ncd_id) references NCDRelease(prj_id,
     ncd_id),
    foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
     group_id)
    references NWSignalGroup(prj_id, ncd_id, nw_id, ch_id, res_id,
     pdu_id, group_id),
    foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
     signal_id)
    references NWSignal(prj_id, ncd_id, nw_id, ch_id, res_id,
```

```
    pdu_id, signal_id)
);

  create table NWSignal_SWSignal_Map (
  prj_id              varchar( 64),
  /* SW signal identifier */
  ecu_id              varchar( 64),
  release_id          varchar( 64),
  sprint_id           varchar( 64),
  sw_signal_id        varchar( 64),
  /* NW signal identifier */
  nw_id               varchar( 64),
  ch_id               varchar( 64),
  ncd_id              varchar( 64),
  res_id              varchar( 64),
  pdu_id              varchar( 64),
  nw_signal_id        varchar( 64),
  direction           varchar(  2),
  primary key(prj_id, ecu_id, sw_signal_id, nw_id, ch_id, res_id,
    pdu_id),
  foreign key(prj_id, ecu_id, release_id, sprint_id) references
   SWRelease(prj_id, ecu_id, release_id, sprint_id),
  foreign key(prj_id, ecu_id, release_id, sprint_id, sw_signal_id
   ) references SWSignal(prj_id, ecu_id, release_id, sprint_id,
   signal_id) on delete cascade,
  foreign key(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id,
   nw_signal_id) references NWSignal(prj_id, ncd_id, nw_id, ch_id,
    res_id, pdu_id, signal_id) on delete cascade,
  constraint valid_direction_swsignal_nwsignal_map check (
   direction in ('TX','RX'))
);
```

Listing 2. T-SQL Script to Create Network Relationships

```
create table ECU (
  prj_id              varchar( 64),
  ecu_id              varchar( 64),
```

```
  ecu_name          varchar(255),
  primary key (prj_id, ecu_id)
);

create table CommunicationConnector (
  prj_id            varchar( 64),
  ecu_id            varchar( 64),
  nw_id             varchar( 64),
  ch_id             varchar( 64),
  comm_task_name    varchar(255),
  primary key(prj_id, ecu_id, nw_id, ch_id),
  foreign key(prj_id, ecu_id) references ECU(prj_id, ecu_id),
  foreign key(prj_id, nw_id, ch_id) references NWChannel(prj_id,
   nw_id, ch_id)
);

create table CommunicationConnector_TimingAttributes (
  prj_id            varchar( 64),
  ecu_id            varchar( 64),
  nw_id             varchar( 64),
  ch_id             varchar( 64),
  attr_id           varchar( 64),
  attr_parent       varchar( 64),
  attr_value        varchar( 64),
  attr_value_type   varchar( 64),
  primary key(prj_id, ecu_id, nw_id, ch_id, attr_id, attr_value),
  foreign key(prj_id, ecu_id, nw_id, ch_id) references
   CommunicationConnector(prj_id, ecu_id, nw_id, ch_id)
);

create table CommunicationPort (
  prj_id            varchar( 64),
  ecu_id            varchar( 64),
  ncd_id            varchar( 64),
  nw_id             varchar( 64),
  ch_id             varchar( 64),
  res_id            varchar( 64),
  port_name         varchar(255),
  direction         varchar(  6),
```

```
  primary key (prj_id, ecu_id, ncd_id, nw_id, ch_id, res_id,
   direction),
  foreign key(prj_id, ncd_id) references NCDRelease(prj_id,
   ncd_id),
  foreign key(prj_id, ecu_id, nw_id, ch_id) references
   CommunicationConnector(prj_id, ecu_id, nw_id, ch_id),
  constraint valid_direction_port check (direction in ('IN','OUT'
   ))
);

create table ECU_Core (
  prj_id             varchar( 64),
  ecu_id             varchar( 64),
  core_nr            smallint,
  primary key(prj_id, ecu_id, core_nr),
  foreign key(prj_id, ecu_id) references ECU(prj_id, ecu_id)
);
```

Listing 3. T-SQL Script to Create ECU Relationships

```
create table SWTask (
  prj_id             varchar( 64),
  ecu_id             varchar( 64),
  core_nr            smallint,
  release_id         varchar( 64),
  sprint_id          varchar( 64),
  task_id            varchar( 64),
  task_name          varchar(255),
  priority           smallint,
  deadline_mus       int,
  preemptable        bit,
  primary key(prj_id,  ecu_id, core_nr, release_id, sprint_id,
   task_id),
  foreign key(prj_id, ecu_id, release_id, sprint_id) references
   SWRelease(prj_id, ecu_id, release_id, sprint_id),
  foreign key(prj_id, ecu_id, core_nr) references ECU_Core(prj_id
   , ecu_id, core_nr)
```

```
);

create table SWSignal (
  prj_id             varchar( 64),
  ecu_id             varchar( 64),
  release_id         varchar( 64),
  sprint_id          varchar( 64),
  signal_id          varchar( 64),
  signal_name        varchar(255),
  foreign key(prj_id, ecu_id) references ECU(prj_id, ecu_id),
  foreign key(prj_id, ecu_id, release_id, sprint_id) references
   SWRelease(prj_id, ecu_id, release_id, sprint_id),
  primary key(prj_id, ecu_id, release_id, sprint_id, signal_id)
);

create table SWRunnable (
  prj_id             varchar( 64),
  ecu_id             varchar( 64),
  core_nr            smallint,
  release_id         varchar( 64),
  sprint_id          varchar( 64),
  task_id            varchar( 64),
  runnable_id        varchar( 64),
  runnable_name      varchar(255),
  runnable_nr        int not null,
  runnable_bcet_mus  int,
  runnable_wcet_mus  int,
  primary key (prj_id, ecu_id, core_nr, release_id, sprint_id,
   task_id, runnable_id),
  constraint swrunnable_unambiguous_order unique (prj_id, ecu_id,
    core_nr, release_id, sprint_id, task_id, runnable_nr),
  foreign key(prj_id, ecu_id) references ECU(prj_id, ecu_id),
  foreign key(prj_id, ecu_id, release_id, sprint_id) references
   SWRelease(prj_id, ecu_id, release_id, sprint_id),
  foreign key (prj_id, ecu_id, core_nr, release_id, sprint_id,
   task_id) references SWTask(prj_id, ecu_id, core_nr, release_id,
    sprint_id, task_id),
);
```

```
create table SWRunnable_SWSignal_Map (
  prj_id              varchar( 64),
  ecu_id              varchar( 64),
  core_nr             smallint,
  release_id          varchar( 64),
  sprint_id           varchar( 64),
  task_id             varchar( 64),
  runnable_id         varchar( 64),
  signal_id           varchar( 64),
  direction           varchar(  6),
  primary key(prj_id, ecu_id, runnable_id, signal_id, direction),
  foreign key(prj_id, ecu_id, release_id, sprint_id) references
   SWRelease(prj_id, ecu_id, release_id, sprint_id),
  foreign key(prj_id, ecu_id, core_nr, release_id, sprint_id,
   task_id, runnable_id)
    references SWRunnable(prj_id, ecu_id, core_nr, release_id,
   sprint_id, task_id, runnable_id) on delete cascade,
  foreign key(prj_id, ecu_id, release_id, sprint_id, signal_id)
   references SWSignal(prj_id, ecu_id, release_id, sprint_id,
   signal_id) on delete cascade,
  constraint valid_direction_swsignal check (direction in ('IN','
   OUT','LOCAL'))
);
```

Listing 4. T-SQL Script to Create Software Relationships

```
create table TimingModel_Periodic(
  prj_id              varchar( 64),
  tm_id               int,
  tm_name             varchar(255),
  offset_mus          int,
  period_mus          int,
  primary key(prj_id, tm_id) ,
  constraint TimingModel_Periodic_non_zero_id check (tm_id  > 0)
);

create table TimingModel_Sporadic(
```

```
  prj_id              varchar( 64),
  tm_id               int,
  tm_name             varchar(255),
  delta_t_min_mus     int,
  delta_t_max_mus     int,
  primary key(prj_id, tm_id) ,
  constraint TimingModel_Sporadic_non_zero_id check (tm_id  > 0)
);

create table TimingModel_Chained(
  prj_id              varchar( 64),
  tm_id               int,
  tm_name             varchar(255),
  ecu_id_ref          varchar( 64),
  core_nr_ref         smallint,
  release_id_ref      varchar( 64),
  sprint_id_ref       varchar( 64),
  task_id_ref         varchar( 64),
  runnable_id_ref     varchar( 64),
  primary key(prj_id, tm_id),
  /* reference to runnable */
  foreign key(prj_id, ecu_id_ref, core_nr_ref, release_id_ref,
   sprint_id_ref, task_id_ref, runnable_id_ref)
  references SWRunnable(prj_id, ecu_id, core_nr, release_id,
   sprint_id, task_id, runnable_id) on delete cascade,
  constraint TimingModel_Chained_non_zero_id check (tm_id  > 0)
);

create table TaskActivationMap (
  prj_id              varchar( 64),
  ecu_id              varchar( 64),
  core_nr             smallint,
  release_id          varchar( 64),
  sprint_id           varchar( 64),
  task_id             varchar( 64),
  tm_chain_ref        int DEFAULT 0,
  tm_periodic_ref     int DEFAULT 0,
  tm_sporadic_ref     int DEFAULT 0,
  foreign key (prj_id,ecu_id, core_nr, release_id, sprint_id,
```

```
    task_id) references SWTask(prj_id, ecu_id, core_nr, release_id,
     sprint_id, task_id),
    primary key (prj_id, ecu_id, release_id, sprint_id, core_nr,
     task_id),
    constraint XOR1 check ((tm_chain_ref + tm_periodic_ref +
     tm_sporadic_ref) > 0) ,
    constraint XOR2 check ((tm_chain_ref    * tm_periodic_ref) = 0)
      ,
    constraint XOR3 check ((tm_periodic_ref * tm_sporadic_ref) = 0)
      ,
    constraint XOR4 check ((tm_sporadic_ref * tm_chain_ref) = 0)
  );

  create table NWPDUSendingTrigger (
    prj_id              varchar( 64),
    ncd_id              varchar( 64),
    nw_id               varchar( 64),
    ch_id               varchar( 64),
    res_id              varchar( 64),
    pdu_id              varchar( 64),
    trigger_variant     varchar( 64),
    tm_periodic_ref     int DEFAULT 0,
    tm_sporadic_ref     int DEFAULT 0,
    foreign key (prj_id, tm_periodic_ref) references
     TimingModel_Periodic(prj_id, tm_id),
    foreign key (prj_id, tm_sporadic_ref) references
     TimingModel_Sporadic(prj_id, tm_id),
    foreign key (prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id)
     references NWPDU(prj_id, ncd_id, nw_id, ch_id, res_id, pdu_id),
    primary key (prj_id, pdu_id, ncd_id, nw_id, ch_id, res_id,
     trigger_variant)
  );
```

Listing 5. T-SQL Script to Create Timing Relationships

```
  create table EffectChain (
    prj_id              varchar( 64),
```

```
  ec_id              varchar( 64),
  ec_name            varchar(255),
  ec_description     varchar( 64),
  primary key (prj_id, ec_id)
);

create table EC_Segment (
  prj_id             varchar( 64),
  ec_id              varchar( 64),
  part_nr            smallint,
  ecu_id             varchar( 64),
  primary key (prj_id, ec_id, part_nr, ecu_id),
  foreign key (prj_id, ecu_id) references ECU(prj_id, ecu_id)
);

create table EC_Connection (
  prj_id             varchar( 64),
  ec_id              varchar( 64),
  /* connections are ordered */
  conn_nr            smallint,
  /* reference to sending ECU segment */
  part_tx_nr         smallint,
  ecu_tx_id          varchar( 64),
  /* reference to receiving ECU segment */
  part_rx_nr         smallint,
  ecu_rx_id          varchar( 64),
  /* reference to network by id (unique) */
  nw_id              varchar( 64),
  /* Reference to signal by name (possibly multiple signals) */
  signal_name        varchar(255),
  primary key (prj_id, ec_id, conn_nr),
  foreign key (prj_id, nw_id) references Network(prj_id, nw_id),
  foreign key (prj_id, ec_id, part_tx_nr, ecu_tx_id) references
   EC_Segment(prj_id, ec_id, part_nr, ecu_id),
  foreign key (prj_id, ec_id, part_rx_nr, ecu_rx_id) references
   EC_Segment(prj_id, ec_id, part_nr, ecu_id)
);

create table SWGraphNode (
```

```
  prj_id              varchar( 64),
  ec_id               varchar( 64),
  part_nr             smallint,
  ecu_id              varchar( 64),
  node_nr             smallint,
  /* reference to runnable */
  runnable_ref        varchar(255),
  primary key (prj_id, ec_id, part_nr, ecu_id, node_nr),
  foreign key (prj_id, ec_id, part_nr, ecu_id) references
   EC_Segment(prj_id, ec_id, part_nr, ecu_id)
);

create table SWGraphEdge (
  prj_id              varchar( 64),
  ec_id               varchar( 64),
  part_nr             smallint,
  ecu_id              varchar( 64),
  node_nr_a           smallint,
  node_nr_b           smallint,
  /* reference to connecting signal */
  signal_ref          varchar(255),
  primary key (prj_id, ec_id, part_nr, ecu_id, node_nr_a,
   node_nr_b),
  foreign key (prj_id, ec_id, part_nr, ecu_id) references
   EC_Segment(prj_id, ec_id, part_nr, ecu_id)
);
```

Listing 6. T-SQL Script to Create Effect Chain Relationships

Appendix B - MiniZinc Models

```
constraint forall (i in i_Signals, j in Signal_OccMin[i]..
    Signal_OccMax[i]) (
  let { int: l = Signal_PDU[i]; } in
  if (Signal_TriggersPDU[i] = "ALWAYS") then
    if (Signal_Updated[i, j] <= PDU_InTxBuffer[l, PDU_OccMin[l]])
     then
      Signal_PDU_Contained[i, j] = PDU_OccMin[l]
    else
      Signal_PDU_Contained[i, j] > PDU_OccMin[l]
    endif
    /\
    forall (k in PDU_OccMin[l]+1..PDU_OccMax[l]) (
      if (Signal_Updated[i, j] > PDU_InTxBuffer[l, k-1] /\
        Signal_Updated[i, j] <= PDU_InTxBuffer[l, k]) then

        Signal_PDU_Contained[i, j] = k
      else
        true
      endif)

  else % On-Change or Pending
    if (Signal_PDU_Contained[i, j] != -1) then
      Signal_PDU_Contained[i, j] >= PDU_OccMin[l] /\
      Signal_PDU_Contained[i, j] <= PDU_OccMax[l] /\

      if (Signal_Updated[i, j] <= PDU_InTxBuffer[l, PDU_OccMin[l]])
      then
        Signal_PDU_Contained[i, j] = PDU_OccMin[l]
      else
        Signal_PDU_Contained[i, j] > PDU_OccMin[l]
      endif
      /\
      forall (k in PDU_OccMin[l]+1..PDU_OccMax[l]) (
```

```
        if (Signal_Updated[i, j] > PDU_InTxBuffer[l, k-1] /\
          Signal_Updated[i, j] <= PDU_InTxBuffer[l, k]) then

          Signal_PDU_Contained[i, j] = k
        else
          true
        endif)
      else
      true
    endif
  endif
);
```

Listing 7. Constraints on n^S in MiniZinc

```
constraint forall (i in i_PDUs, j in Min_Occ_PDU..Max_Occ_PDU) (

  if (PDU_Type[i] == "IPDU" /\ j >= PDU_OccMin[i] /\ j <=
     PDU_OccMax[i]) then
    minimum_int(PDU_Triggered[i,j], [  PDU_Triggered_Time[i,j],
     PDU_Triggered_Signal[i,j] ] ) % ,
  elseif (PDU_Type[i] = "SECPDU" /\ j >= PDU_OccMin[i] /\ j <=
     PDU_OccMax[i]) then

    minimum_int(PDU_Triggered[i,j], [Time_Dom_Sup] ++
    [ if (PDU_Parent_Contained[l, k] = j) then PDU_Triggered[l, k]
     else Time_Dom_Sup endif
    | l in i_PDUs, k in PDU_OccMin[l]..PDU_OccMax[l] where
     PDU_Parent[l] = i])

    elseif (PDU_Type[i] = "CONTAINER" /\ j >= PDU_OccMin[i] /\ j <=
      PDU_OccMax[i]) then
    let {

      var int: direct_trigger = min([Time_Dom_Sup] ++ [
      if (PDU_Parent_Contained[l, k] = j) then PDU_Triggered[l,k]
     else Time_Dom_Sup endif
```

```
  | l in i_PDUs, k in PDU_OccMin[l]..PDU_OccMax[l]
  where PDU_Parent[l] = i /\ PDU_Parent_TriggerType[l] = "
 ALWAYS"]);

  var int: trigger_first_pdu = min([Time_Dom_Sup] ++ [
  if (PDU_Parent_Contained[l, k] = j) then PDU_Triggered[l,k]
 else Time_Dom_Sup endif
  | l in i_PDUs, k in PDU_OccMin[l]..PDU_OccMax[l] where
 PDU_Parent[l] = i]);

  var int: trigger_last_pdu = max([Time_Dom_Inf] ++ [
  if (PDU_Parent_Contained[l, k] = j) then PDU_Triggered[l,k]
 else Time_Dom_Inf endif
  | l in i_PDUs, k in PDU_OccMin[l]..PDU_OccMax[l] where
 PDU_Parent[l] = i]);

} in

minimum_int(PDU_Triggered[i,j],
[ Time_Dom_Sup ] ++ [ PDU_Triggered_Time[i,j] ] ++ [
 direct_trigger ]

++

% immediate transmission
[ if (PDU_Container_TriggerType[i] = "FIRST" /\
 trigger_first_pdu < Time_Dom_Sup) then

trigger_first_pdu
else
Time_Dom_Sup
endif ]

++

% container timeout
[ if (PDU_Container_TriggerType[i] = "DEFAULT" /\
 trigger_first_pdu < Time_Dom_Sup) then
```

```
    trigger_first_pdu + PDU_Container_Timeout[i]
    else
    Time_Dom_Sup
    endif ]

    ++

    [ if ((PDU_Container_ThresholdSize[i] > 0 /\ PDU_ActualLength[i
     ,j] >= PDU_Container_ThresholdSize[i]) \/ PDU_ActualLength[i,j]
      = PDU_Length[i]) then

    trigger_last_pdu
    else
    Time_Dom_Sup
    endif ]
    )

  else
  if (j >= PDU_OccMin[i] /\ j <= PDU_OccMax[i]) then
  assert(false, "Invalid PDU type: '" ++ PDU_Type[i] ++ "'")
  else
  PDU_Triggered[i,j] = Time_Dom_Sup
  endif

  endif
);
```

Listing 8. Constraints on n^S in MiniZinc

Bibliography

[1] *2016 IEEE real-time and embedded technology and applications symposium (RTAS), vienna, austria, april 11-14, 2016*. IEEE Computer Society, 2016. ISBN: 978-1-4673-8639-5. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7460013.

[2] Rajeev Alur, Costas Courcoubetis, and David L. Dill. "Model-checking in dense real-time". In: *Inf. Comput.* 104.1 (1993), pp. 2–34. DOI: 10.1006/inco.1993.1024. URL: https://doi.org/10.1006/inco.1993.1024.

[3] Rajeev Alur and David L. Dill. "The theory of timed automata". In: *Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991, Proceedings*. Ed. by J. W. de Bakker, Cornelis Huizing, Willem P. de Roever, and Grzegorz Rozenberg. Vol. 600. Lecture Notes in Computer Science. Springer, 1991, pp. 45–73. ISBN: 3-540-55564-1. DOI: 10.1007/BFb0031987. URL: https://doi.org/10.1007/BFb0031987.

[4] Albert Amos. "Comparison of event-triggered and time-triggered concepts with regard to distributed control systems". In: *Proceedings of the Embedded World Conference* 2004 (2004), pp. 235–252. URL: https://pdfs.semanticscholar.org/0675/28af2cda260f8eb32ef6d90d3e1e5f04cdfa.pdf.

[5] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. "S-taliro: A tool for temporal logic falsification for hybrid systems". In: *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*. Ed. by Parosh Aziz Abdulla and K. Rustan M. Leino. Vol. 6605. Lecture Notes in Computer Science. Springer, 2011, pp. 254–257. ISBN: 978-3-642-19834-2. DOI: 10.1007/978-3-642-19835-9_21. URL: https://doi.org/10.1007/978-3-642-19835-9%5C_21.

[6] Silviu Apostu, Ondrej Burkacky, Johannes Deichmann, and Georg Doll. *Automotive ecus for adas and autonomous driving systems, north america and europe*. Apr. 2017. URL: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/automotive-software-and-electrical-electronic-architecture-implications-for-oems.

[7] Arcticus Systems AB. *Arcticus systems – products*. Dec. 2017. URL: https://www.arcticus-systems.com/products/.

[8] Automotive Grade Linux. *Automotive grade linux platform debuts on the 2018 toyota camry*. May 2017. URL: https://www.automotivelinux.org/announcements/2017/05/30/automotive-grade-linux-platform-debuts-on-the-2018-toyota-camry.

[9] AUTomotive Open System ARchitecture. *Document title virtual functional bus*. Dec. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_VFB.pdf.

[10] AUTomotive Open System ARchitecture. *History – AUTOSAR*. 2020. URL: https://www.autosar.org/about/history/.

[11] AUTomotive Open System ARchitecture. *Layered software architecture*. Dec. 2017. URL: https://autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[12] AUTomotive Open System ARchitecture. *Specification of i-pdu multiplexer*. Dec. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_IPDUMultiplexer.pdf.

[13] AUTomotive Open System ARchitecture. *Specification of rte software*. Dec. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_SWS_RTE.pdf.

[14] AUTomotive Open System ARchitecture. *Specification of timing extensions*. Dec. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_TimingExtensions.pdf.

[15] Global Automotive and Transportation Research Team at Frost & Sullivan. *Automotive software and electrical/electronic architecture: implications for oems*. Apr. 2018. URL: https://store.frost.com/automotive-ecus-for-adas-and-autonomous-driving-systems-north-america-and-europe-2017.html.

[16] François Baccelli, Guy Cohen, Geert Jan Olsder, and Jean-Pierre Quadrat. *Synchronization and linearity: an algebra for discrete event systems*. John Wiley & Sons Ltd, 1992. URL: http://cermics.enpc.fr/~cohen-g/documents/BCOQ-book.pdf.

[17] Michael Barr. "Pulse width modulation". In: *Embedded Systems Programming* 14.10 (2001), pp. 103–104. URL: http://homepage.cem.itesm.mx/carbajal/Microcontrollers/ASSIGNMENTS/readings/ARTICLES/barr01_pwm.pdf.

[18] Howard Barringer, Allen Goldberg, Klaus Havelund, and Koushik Sen. "Rule-based runtime verification". In: *Verification, Model Checking, and Abstract Interpretation, 5th International Conference, VMCAI 2004, Venice, Italy, January 11-13, 2004, Proceedings*. Ed. by Bernhard Steffen and Giorgio Levi. Vol. 2937. Lecture Notes in Computer Science. Springer, 2004, pp. 44–57. ISBN: 3-540-20803-8. DOI: 10.1007/978-3-540-24622-0_5. URL: https://doi.org/10.1007/978-3-540-24622-0%5C_5.

[19] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. "End-to-end timing analysis of cause-effect chains in automotive embedded systems". In: *J. Syst. Archit.* 80 (2017), pp. 104–113. DOI: 10.1016/j.sysarc.2017.09.004. URL: https://doi.org/10.1016/j.sysarc.2017.09.004.

[20] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. "Synthesizing job-level dependencies for automotive multi-rate effect chains". In: *22nd IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, RTCSA 2016, Daegu, South Korea, August 17-19, 2016*. IEEE Computer Society, 2016, pp. 159–169. DOI: 10.1109/RTCSA.2016.41. URL: https://doi.org/10.1109/RTCSA.2016.41.

[21] Ralph Becket. *Specification of flatzinc*. Version 1.6. 2008. URL: https://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf.

[22] Lucia Lo Bello. "The case for ethernet in automotive communications". In: *SIGBED Review* 8.4 (2011), pp. 7–15. DOI: 10.1145/2095256.2095257. URL: https://doi.org/10.1145/2095256.2095257.

[23] Gérard Berry and Laurent Cosserat. "The ESTEREL synchronous programming language and its mathematical semantics". In: *Seminar on Concurrency, Carnegie-Mellon University, Pittsburg, PA, USA, July 9-11, 1984*. Ed. by Stephen D. Brookes, A. W. Roscoe, and Glynn Winskel. Vol. 197. Lecture Notes in Computer Science. Springer, 1984, pp. 389–448. ISBN: 3-540-15670-4. DOI: 10.1007/3-540-15670-4_19. URL: https://doi.org/10.1007/3-540-15670-4%5C_19.

[24] Enrico Bini and Giorgio C. Buttazzo. "Measuring the performance of schedulability tests". In: *Real-Time Systems* 30.1-2 (2005), pp. 129–154. DOI: 10.1007/s11241-005-0507-9. URL: https://doi.org/10.1007/s11241-005-0507-9.

[25] Frédéric Boniol, Michaël Lauer, Claire Pagetti, and Jérôme Ermont. "Freshness and reactivity analysis in globally asynchronous locally time-triggered systems". In: *NASA Formal Methods, 5th International Symposium, NFM 2013, Moffett Field, CA, USA, May 14-16, 2013. Proceedings*. Ed. by Guillaume Brat, Neha Rungta, and Arnaud Venet. Vol. 7871. Lecture Notes in Computer Science. Springer, 2013, pp. 93–107. DOI: 10.1007/978-3-642-38088-4_7. URL: https://doi.org/10.1007/978-3-642-38088-4%5C_7.

[26] Kai Borgeest. "Hardware". In: *Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement*. Wiesbaden: Springer Fachmedien Wiesbaden, 2014, pp. 135–211. ISBN: 978-3-8348-2145-4. DOI: 10.1007/978-3-8348-2145-4_6. URL: https://doi.org/10.1007/978-3-8348-2145-4_6.

[27] Jean-Yves Le Boudec and Patrick Thiran. *Network calculus: A theory of deterministic queuing systems for the internet*. Vol. 2050. Lecture Notes in Computer Science. Springer, 2001. ISBN: 3-540-42184-X. DOI: 10.1007/3-540-45318-0. URL: https://doi.org/10.1007/3-540-45318-0.

[28] Vadim Cebotari and Stefan Kugele. "On the nature of automotive service architectures". In: *IEEE International Conference on Software Architecture Companion, ICSA Companion 2019, Hamburg, Germany, March 25-26, 2019*. IEEE, 2019, pp. 53–60. DOI: 10.1109/ICSA-C.2019.00017. URL: https://doi.org/10.1109/ICSA-C.2019.00017.

[29] MARS Steering Committee. *Models for formal analysis of real systems*. Mar. 2020. URL: http://mars-workshop.org//repository/025-Latencies.html.

[30] Lukas Convent, Sebastian Hungerecker, Martin Leucker, Torben Scheffel, Malte Schmitz, and Daniel Thoma. "Tessla: temporal stream-based specification language". In: *Formal Methods: Foundations and Applications - 21st Brazilian Symposium, SBMF 2018, Salvador, Brazil, November 26-30, 2018, Proceedings*. Ed. by Tiago Massoni and Mohammad Reza Mousavi. Vol. 11254. Lecture Notes in Computer Science. Springer, 2018, pp. 144–162. DOI: 10.1007/978-3-030-03044-5_10. URL: https://doi.org/10.1007/978-3-030-03044-5%5C_10.

[31] Riccardo Coppola and Maurizio Morisio. "Connected car: technologies, issues, future trends". In: *ACM Comput. Surv.* 49.3 (2016), 46:1–46:36. DOI: 10.1145/2971482. URL: https://doi.org/10.1145/2971482.

[32] BroadCom Corporation. *Broadr-reach physical layer transceiver specification for automotive applications*. V3.0. 2014. URL: http://www.ieee802.org/3/1TPCESG/public/BroadR_Reach_Automotive_Spec_V3.0.pdf.

[33] Rene L. Cruz. "A calculus for network delay, part I: network elements in isolation". In: *IEEE Trans. Information Theory* 37.1 (1991), pp. 114–131. DOI: 10.1109/18.61109. URL: https://doi.org/10.1109/18.61109.

[34] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. "LOLA: runtime monitoring of synchronous systems". In: *12th International Symposium on Temporal Representation and Reasoning (TIME 2005), 23-25 June 2005, Burlington, Vermont, USA*. IEEE Computer Society, 2005, pp. 166–174. ISBN: 0-7695-2370-6. DOI: 10.1109/TIME.2005.26. URL: https://doi.org/10.1109/TIME.2005.26.

[35] Robert I. Davis and Alan Burns. "A survey of hard real-time scheduling for multiprocessor systems". In: *ACM Comput. Surv.* 43.4 (2011), 35:1–35:44. DOI: 10.1145/1978802.1978814. URL: https://doi.org/10.1145/1978802.1978814.

[36] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. "Controller area network (CAN) schedulability analysis: refuted, revisited and revised". In: *Real-Time Systems* 35.3 (2007), pp. 239–272. DOI: 10.1007/s11241-007-9012-7. URL: https://doi.org/10.1007/s11241-007-9012-7.

[37] Patricia Derler, Edward A. Lee, and Slobodan Matic. "Simulation and implementation of the PTIDES programming model". In: *12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications, 27-29 October 2008, Vancouver, BC, Canada, Proceedings*. Ed. by David J. Roberts, Abdulmotaleb El-Saddik, and Alois Ferscha. IEEE Computer Society, 2008, pp. 330–333. ISBN: 978-0-7695-3425-1. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4700086.

[38] Elmar Dilger, Thomas Führer, and Bernd Müller. "Distributed fault-tolerant and safety-critical application in vehicles - A time-triggered approach". In: *Computer Safety, Reliability and Security, 17th International Conference, SAFECOMP'98, Heidelberg, Germany, October 5-7, 1998, Proceedings*. Ed. by Wolfgang D. Ehrenberger. Vol. 1516. Lecture Notes in Computer Science. Springer, 1998, pp. 267–283. DOI: 10.1007/3-540-49646-7_21. URL: https://doi.org/10.1007/3-540-49646-7%5C_21.

[39] Elmar Dilger, Thomas Führer, Bernd Müller, and Stefan Poledna. "The x-by-wire concept: time-triggered information exchange and fail silence support by new system services". In: *SAE transactions* (1998), pp. 1037–1046.

[40] Alexandre Donzé. "Breach, A toolbox for verification and parameter synthesis of hybrid systems". In: *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*. Ed. by Tayssir Touili, Byron Cook, and Paul B. Jackson. Vol. 6174. Lecture Notes in Computer Science. Springer, 2010, pp. 167–170. ISBN: 978-3-642-14294-9. DOI: 10.1007/978-3-642-14295-6. URL: https://doi.org/10.1007/978-3-642-14295-6.

[41] Marco Dürr, Georg Von Der Brüggen, Kuan-Hsun Chen, and Jian-Jia Chen. "End-to-end timing analysis of sporadic cause-effect chains in distributed systems". In: *ACM Trans. Embed. Comput. Syst.* 18.5s (Oct. 2019), 58:1–58:24. ISSN: 1539-9087. DOI: 10.1145/3358181. URL: http://doi.acm.org/10.1145/3358181.

[42] Thorsten Ehlers. "SAT and CP - parallelisation and applications". PhD thesis. University of Kiel, Germany, 2017. URL: http://macau.uni-kiel.de/receive/dissertation%5C_diss%5C_00021179.

[43] Thorsten Ehlers and Peter J. Stuckey. "Parallelizing constraint programming with learning". In: *Integration of AI and OR Techniques in Constraint Programming - 13th International Conference, CPAIOR 2016, Banff, AB, Canada, May 29 - June 1, 2016, Proceedings*. Ed. by Claude-Guy Quimper. Vol. 9676. Lecture Notes in Computer Science. Springer, 2016, pp. 142–158. DOI: 10.1007/978-3-319-33954-2_11. URL: https://doi.org/10.1007/978-3-319-33954-2%5C_11.

[44] Ernest Allen Emerson, Aloysius Mok, Aravinda Prasad Sistla, and Jai Srinivasan. "Quantitative temporal reasoning". In: *Computer Aided Verification, 2nd International Workshop, CAV '90, New Brunswick, NJ, USA, June 18-21, 1990, Proceedings*. Ed. by Edmund M. Clarke and Robert P. Kurshan. Vol. 531. Lecture Notes in Computer Science. Springer, 1990, pp. 136–145. ISBN: 3-540-54477-1. DOI: 10.1007/BFb0023727. URL: https://doi.org/10.1007/BFb0023727.

[45] Nico Feiertag, Kai Richter, Johan Nordlander, and Jan Jonsson. "A compositional framework for end-to-end path delay calculation of automotive systems under different path semantics". In: *Proceedings of the IEEE Real-Time System Symposium - Workshop on Compositional Theory and Technology for Real-Time Embedded Systems, Barcelona, Spain, November 30, 2008*. 2008. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:ltu:diva-30306.

[46] Thomas Ferrère, Oded Maler, Dejan Nickovic, and Dogan Ulus. "Measuring with timed patterns". In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part II*. Ed. by Daniel Kroening and Corina S. Pasareanu. Vol. 9207. Lecture Notes in Computer Science. Springer, 2015, pp. 322–337. ISBN: 978-3-319-21667-6. DOI: 10.1007/978-3-319-21668-3_19. URL: https://doi.org/10.1007/978-3-319-21668-3%5C_19.

[47] Julien Forget, Frédéric Boniol, Emmanuel Grolleau, David Lesens, and Claire Pagetti. "Scheduling dependent periodic tasks without synchronization mechanisms". In: *16th IEEE Real-Time and Embedded Technology and Applications Symposium, RTAS 2010, Stockholm, Sweden, April 12-15, 2010*. Ed. by Marco Caccamo. IEEE Computer Society, 2010, pp. 301–310. DOI: 10.1109/RTAS.2010.26. URL: https://doi.org/10.1109/RTAS.2010.26.

[48] Julien Forget, Frédéric Boniol, David Lesens, and Claire Pagetti. "A real-time architecture design language for multi-rate embedded control systems". In: *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*. Ed. by Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung. ACM, 2010, pp. 527–534. DOI: 10.1145/1774088.1774196. URL: https://doi.org/10.1145/1774088.1774196.

[49] Max J. Friese, Thorsten Ehlers, and Dirk Nowotka. "Estimating latencies of task sequences in multi-core automotive ecus". In: *13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018*. IEEE, 2018, pp. 1–10. DOI: 10.1109/SIES.2018.8442095. URL: https://doi.org/10.1109/SIES.2018.8442095.

[50] Max J. Friese, Thorsten Ehlers, and Dirk Nowotka. "Improving estimations for latencies of cause-effect chains". In: *6th IFIP TC 10 International Embedded Systems Symposium, IESS 2019, Friedrichshafen, Germany, September 9-11, 2019*. (to appear). Springer, 2019, pp. 1–10.

[51] Max J. Friese, Stephan Grünfelder, Michael Paulitsch, and Alexander Weiss. *Tracing von eingebetteten multicore-systemen*. 2020. URL: https://www.hanser-automotive.de/zeitschrift/archiv/artikel/entwicklungs-tools-10718513.html?search.highlight=cedartools.

[52] Max J. Friese and Dirk Nowotka. "Estimating end-to-end latencies in automotive cyber-physical systems". In: *Proceedings of the 4th Workshop on Models for Formal Analysis of Real Systems, MARS@ETAPS 2020, Dublin, Ireland, April 26, 2020*. Ed. by Ansgar Fehnker and Hubert Garavel. Vol. 316. EPTCS. 2020, pp. 134–148. DOI: 10.4204/EPTCS.316.6. URL: https://doi.org/10.4204/EPTCS.316.6.

[53] Max J. Friese, Johannes Traub, and Dirk Nowotka. "Ec.lang - a language for specifying response time analyses of event chains". In: *2020 IEEE International Conference on Software Testing, Verification and Validation, ICST Industry Track 2020, Porto, Portugal, October 25-27, 2020*. (to appear). IEEE, 2020.

[54] Carlo A. Furia, Dino Mandrioli, Angelo Morzenti, and Matteo Rossi. *Modeling time in computing*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2012. ISBN: 978-3-642-32331-7.

DOI: 10.1007/978-3-642-32332-4. URL: https://doi.org/10.1007/978-3-642-32332-4.

[55] Paul Gao, Hans-Werner Kaas, Det Mohr, and Dominik Wee. "Automotive revolution–perspective towards 2030". In: *Advanced Industries, McKinsey & Company* (2016).

[56] Kai-Björn Gemlau, Johannes Schlatow, Mischa Möstl, and Rolf Ernst. "Compositional analysis for the waters industrial challenge 2017". In: *International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. Dubrovnik, Croatia, July 2017.

[57] Alain Girault, Christophe Prévot, Sophie Quinton, Rafik Henia, and Nicolas Sordon. "Improving and estimating the precision of bounds on the worst-case latency of task chains". In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 37.11 (2018), pp. 2578–2589. DOI: 10.1109/TCAD.2018.2861016. URL: https://doi.org/10.1109/TCAD.2018.2861016.

[58] Andreas Gzemba. *Most - the automotive multimedia network. From most25 to most150*. Vol. 2. Franzis Verlag GmbH, 2011. ISBN: 978-3-645-65061-8.

[59] Nicolas Halbwachs, Fabienne Lagnier, and Christophe Ratel. "The synchronous data flow programming language LUSTRE". In: *Proceedings of the IEEE* 79.9 (Sept. 1991), pp. 1305–1320. ISSN: 0018-9219. DOI: 10.1109/5.97300.

[60] Arne Hamann, Dirk Ziegenbein, Simon Kramer, and Martin Lukasiewycz. *Demonstration of the fmtv 2016 timing verification challenge*. Apr. 2016. URL: http://www.ecrts.org/forum/download/file.php?id=35%20%5C&sid=619cd45cc904f6f8a66fc526919a2ebd.

[61] Frank van Harmelen, Vladimir Lifschitz, and Bruce W. Porter, eds. *Handbook of knowledge representation*. Vol. 3. Foundations of Artificial Intelligence. Elsevier, 2008. ISBN: 978-0-444-52211-5. URL: http://www.sciencedirect.com/science/bookseries/15746526/3.

[62] Florian Hartwich. "CAN with flexible data-rate". In: *Proc. iCC*. 2012, pp. 1–9.

[63] John Havlicek and Scott Little. “Realtime regular expressions for analog and mixed-signal assertions”. In: *International Conference on Formal Methods in Computer-Aided Design, FMCAD '11, Austin, TX, USA, October 30 - November 02, 2011*. Ed. by Per Bjesse and Anna Slobodová. FMCAD Inc., 2011, pp. 155–162. ISBN: 978-0-9835678-1-3. URL: http://dl.acm.org/citation.cfm?id=2157679.

[64] Julien Hennig, Hermann von Hasseln, Hassan Mohammad, Stefan Resmerita, Stefan Lukesch, and Andreas Naderlinger. “Poster abstract: towards parallelizing legacy embedded control software using the LET programming paradigm”. In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, p. 51. ISBN: 978-1-4673-8639-5. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7460013.

[65] Thomas A. Henzinger, Benjamin Horowitz, and Christoph M. Kirsch. “Giotto: A time-triggered language for embedded programming”. In: *Embedded Software, First International Workshop, EMSOFT 2001, Tahoe City, CA, USA, October, 8-10, 2001, Proceedings*. Ed. by Thomas A. Henzinger and Christoph M. Kirsch. Vol. 2211. Lecture Notes in Computer Science. Springer, 2001, pp. 166–184. DOI: 10.1007/3-540-45449-7_12. URL: https://doi.org/10.1007/3-540-45449-7%5C_12.

[66] Martin Hillenbrand. “Funktionale Sicherheit nach ISO 26262 in der Konzeptphase der Entwicklung von Elektrik/Elektronik Architekturen von Fahrzeugen”. German. PhD thesis. 2012. 398 pp. ISBN: 978-3-86644-803-2. DOI: 10.5445/KSP/1000025616.

[67] Martin Hiller. “Future E/E architectures trends and the role of multi-cores”. Presentation at the EMCC 2019. May 2019.

[68] Harry Hsieh, Felice Balarin, Luciano Lavagno, and Alberto L. Sangiovanni-Vincentelli. “Efficient methods for embedded system design space exploration”. In: *Proceedings of the 37th Conference on Design Automation, Los Angeles, CA, USA, June 5-9, 2000*. Ed. by Giovanni De Micheli. ACM, 2000, pp. 607–612. DOI: 10.1145/337292.337593. URL: https://doi.org/10.1145/337292.337593.

[69] Federal Government Commissioner for Information Technology. *V-modell xt*. 2019. URL: https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html.

[70] Marek Jersak. "Compositional performance analysis for complex embedded applications". PhD thesis. University of Braunschweig - Institute of Technology, 2005. URL: http://opus.tu-bs.de/opus/volltexte/2005/707/index.html.

[71] Alexandru Kampmann, Bassam Alrifaee, Markus Kohout, Andreas Wüstenberg, Timo Woopen, Marcus Nolte, Lutz Eckstein, and Stefan Kowalewski. "A dynamic service-oriented software architecture for highly automated vehicles". In: *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019, Auckland, New Zealand, October 27-30, 2019*. IEEE, 2019, pp. 2101–2108. DOI: 10.1109/ITSC.2019.8916841. URL: https://doi.org/10.1109/ITSC.2019.8916841.

[72] Timon Kelter, Heiko Falk, Peter Marwedel, Sudipta Chattopadhyay, and Abhik Roychoudhury. "Bus-aware multicore WCET analysis through TDMA offset bounds". In: *23rd Euromicro Conference on Real-Time Systems, ECRTS 2011, Porto, Portugal, 5-8 July, 2011*. Ed. by Karl-Erik Årzén. IEEE Computer Society, 2011, pp. 3–12. DOI: 10.1109/ECRTS.2011.9. URL: https://doi.org/10.1109/ECRTS.2011.9.

[73] Siddhartha Kumar Khaitan and James D. McCalley. "Design techniques and applications of cyberphysical systems: A survey". In: *IEEE Systems Journal* 9.2 (2015), pp. 350–365. DOI: 10.1109/JSYST.2014.2322503. URL: https://doi.org/10.1109/JSYST.2014.2322503.

[74] Christoph M. Kirsch and Ana Sokolova. "The logical execution time paradigm". In: *Advances in Real-Time Systems (to Georg Färber on the occasion of his appointment as Professor Emeritus at TU München after leading the Lehrstuhl für Realzeit-Computersysteme for 34 illustrious years)*. Ed. by Samarjit Chakraborty and Jörg Eberspächer. Springer, 2012, pp. 103–120. ISBN: 978-3-642-24348-6. DOI: 10.1007/978-3-642-24349-3_5. URL: https://doi.org/10.1007/978-3-642-24349-3%5C_5.

[75] Leonie Köhler, Max Jonas Friese, Simon Bagschik, and Rolf Ernst. "Computing real-time properties of heterogeneous and distributed

cause-effect chains". Work submitted to EMSOFT 2020 and currently under review. Apr. 2020.

[76] Leonie Köhler, Borislav Nikolic, Rolf Ernst, and Marc Boyer. "Increasing accuracy of timing models: from CPA to CPA+". In: *Design, Automation & Test in Europe Conference & Exhibition, DATE 2019, Florence, Italy, March 25-29, 2019*. Ed. by Jürgen Teich and Franco Fummi. IEEE, 2019, pp. 1210–1215. ISBN: 978-3-9819263-2-3. DOI: 10.23919/DATE.2019.8714770. URL: https://doi.org/10.23919/DATE.2019.8714770.

[77] Ron Koymans. "Specifying real-time properties with metric temporal logic". In: *Real-Time Systems* 2.4 (1990), pp. 255–299. DOI: 10.1007/BF01995674. URL: https://doi.org/10.1007/BF01995674.

[78] Simon Kramer, Dirk Ziegenbein, and Arne Hamann. "Real world automotive benchmarks for free". In: *6th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS)*. 2015.

[79] Stefan Kugele, Philipp Obergfell, Manfred Broy, Oliver Creighton, Matthias Traub, and Wolfgang Hopfensitz. "On service-orientation for automotive software". In: *2017 IEEE International Conference on Software Architecture, ICSA 2017, Gothenburg, Sweden, April 3-7, 2017*. IEEE Computer Society, 2017, pp. 193–202. ISBN: 978-1-5090-5729-0. DOI: 10.1109/ICSA.2017.20. URL: https://doi.org/10.1109/ICSA.2017.20.

[80] Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Commun. ACM* 21.7 (1978), pp. 558–565. DOI: 10.1145/359545.359563. URL: http://doi.acm.org/10.1145/359545.359563.

[81] Rodrigo Lange, Alexsandro Cristovão Bonatto, Francisco Vasques, and Rômulo Silva de Oliveira. "Timing analysis of hybrid flexray, CAN-FD and CAN vehicular networks". In: *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Florence, Italy, October 23-26, 2016*. IEEE, 2016, pp. 4725–4730. ISBN: 978-1-5090-3474-1. DOI: 10.1109/IECON.2016.7793791. URL: https://doi.org/10.1109/IECON.2016.7793791.

[82] Rodrigo Lange, Rômulo Silva de Oliveira, and Francisco Vasques. "A reference model for the timing analysis of heterogeneous automotive networks". In: *Computer Standards & Interfaces* 45 (2016), pp. 13–25. DOI: 10.1016/j.csi.2015.10.004. URL: https://doi.org/10.1016/j.csi.2015.10.004.

[83] Michaël Lauer, Frédéric Boniol, Claire Pagetti, and Jérôme Ermont. "End-to-end latency and temporal consistency analysis in networked real-time systems". In: *IJCCBS* 5.3/4 (2014), pp. 172–196. DOI: 10.1504/IJCCBS.2014.064667. URL: https://doi.org/10.1504/IJCCBS.2014.064667.

[84] Wolfhard Lawrenz and Nils Obermöller, eds. *Can*. 5th ed. VDE Verlag, 2011. ISBN: 9783800738052. URL: https://content-select.com/index.php?id=bib_view&ean=9783800738052.

[85] Paul Le Guernic, Thierry Gautier, Michel Le Borgne, and Claude Le Maire. "Programming real-time applications with SIGNAL". In: *Proceedings of the IEEE* 79.9 (Sept. 1991), pp. 1321–1336. ISSN: 0018-9219. DOI: 10.1109/5.97301.

[86] Edward Ashford Lee. "Cyber physical systems: design challenges". In: *11th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2008), 5-7 May 2008, Orlando, Florida, USA*. IEEE Computer Society, 2008, pp. 363–369. ISBN: 978-0-7695-3132-8. DOI: 10.1109/ISORC.2008.25. URL: https://doi.org/10.1109/ISORC.2008.25.

[87] Gabriel Leen and Donal Heffernan. "Expanding automotive electronic systems". In: *IEEE Computer* 35.1 (2002), pp. 88–93. DOI: 10.1109/2.976923. URL: https://doi.org/10.1109/2.976923.

[88] Gabriel Leen and Donal Heffernan. "Time-triggered controller area network". In: *Computing Control Engineering Journal* 12.6 (Dec. 2001), pp. 245–256. ISSN: 0956-3385. DOI: 10.1049/cce:20010601.

[89] Gabriel Leen and Donal Heffernan. "TTCAN: a new time-triggered controller area network". In: *Microprocess. Microsystems* 26.2 (2002), pp. 77–94. DOI: 10.1016/S0141-9331(01)00148-X. URL: https://doi.org/10.1016/S0141-9331(01)00148-X.

[90] Martin Leucker, César Sánchez, Torben Scheffel, Malte Schmitz, and Alexander Schramm. "Tessla: runtime verification of non-synchronized real-time streams". In: *Proceedings of the 33rd Annual ACM Symposium on Applied Computing, SAC 2018, Pau, France, April 09-13, 2018*. Ed. by Hisham M. Haddad, Roger L. Wainwright, and Richard Chbeir. ACM, 2018, pp. 1925–1933. DOI: 10.1145/3167132.3167338. URL: https://doi.org/10.1145/3167132.3167338.

[91] Hyung-Taek Lim, Lars Völker, and Daniel Herrscher. "Challenges in a future ip/ethernet-based in-car network for real-time applications". In: *Proceedings of the 48th Design Automation Conference, DAC 2011, San Diego, California, USA, June 5-10, 2011*. Ed. by Leon Stok, Nikil D. Dutt, and Soha Hassoun. ACM, 2011, pp. 7–12. ISBN: 978-1-4503-0636-2. DOI: 10.1145/2024724.2024727. URL: https://doi.org/10.1145/2024724.2024727.

[92] Jorge Martinez, Ignacio Sanudo Olmedo, and Marko Bertogna. "Analytical characterization of end-to-end communication delays with logical execution time". In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 37.11 (2018), pp. 2244–2254. DOI: 10.1109/TCAD.2018.2857398. URL: https://doi.org/10.1109/TCAD.2018.2857398.

[93] Mohammadreza Mehrabian, Mohammad Khayatian, Aviral Shrivastava, John C. Eidson, Patricia Derler, Hugo A. Andrade, Ya-Shian Li-Baboud, Edward Griffor, Marc Weiss, and Kevin Stanton. "Timestamp temporal logic (TTL) for testing the timing of cyber-physical systems". In: *ACM Trans. Embedded Comput. Syst.* 16.5 (2017), 169:1–169:20. DOI: 10.1145/3126510. URL: https://doi.org/10.1145/3126510.

[94] Erich Meier. *V-modelle in automotive-projekten*. 2008. URL: https://www.all-electronics.de/wp-content/uploads/migrated/article-pdf/116443/ael08-01-036.pdf.

[95] Varun M. Navale, Kyle Williams, Athanassios Lagospiris, Michael Schaffert, and Markus-Alexander Schweiker. *(r)evolution of e/e architectures*. Apr. 2015. DOI: https://doi.org/10.4271/2015-01-0196. URL: https://doi.org/10.4271/2015-01-0196.

[96] Nicholas Nethercote. *Converting minizinc to flatzinc*. Version 1.6. 2014. URL: https://www.minizinc.org/downloads/doc-1.6/mzn2fzn.pdf.

[97] Moritz Neukirchner, Mircea Negrean, Rolf Ernst, and Torsten T. Bone. "Response-time analysis of the flexray dynamic segment under consideration of slot-multiplexing". In: *7th IEEE International Symposium on Industrial Embedded Systems, SIES 2012, Karlsruhe, Germany, June 20-22, 2012*. IEEE, 2012, pp. 21–30. ISBN: 978-1-4673-2685-8. DOI: 10.1109/SIES.2012.6356566. URL: https://doi.org/10.1109/SIES.2012.6356566.

[98] Dejan Nickovic, Olivier Lebeltel, Oded Maler, Thomas Ferrère, and Dogan Ulus. "AMT 2.0: qualitative and quantitative trace analysis with extended signal temporal logic". In: *Tools and Algorithms for the Construction and Analysis of Systems - 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part II*. Ed. by Dirk Beyer and Marieke Huisman. Vol. 10806. Lecture Notes in Computer Science. Springer, 2018, pp. 303–319. ISBN: 978-3-319-89962-6. DOI: 10.1007/978-3-319-89963-3_18. URL: https://doi.org/10.1007/978-3-319-89963-3%5C_18.

[99] Thomas Nolte, Hans Hansson, and Lucia Lo Bello. "Automotive communications-past, current and future". In: *Proceedings of 10th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA 2005, September 19-22, 2005, Catania, Italy*. IEEE, 2005. ISBN: 0-7803-9401-1. DOI: 10.1109/ETFA.2005.1612631. URL: https://doi.org/10.1109/ETFA.2005.1612631.

[100] Carl Adam Petri. "Fundamentals of a theory of asynchronous information flow". In: *Information Processing, Proceedings of the 2nd IFIP Congress 1962, Munich, Germany, August 27 - September 1, 1962*. North-Holland, 1962, pp. 386–390.

[101] Andy D. Pimentel and Peter van Stralen. "Scenario-based design space exploration". In: *Handbook of Hardware/Software Codesign*. Ed. by Soonhoi Ha and Jürgen Teich. Springer, 2017, pp. 271–299. ISBN: 978-94-017-7266-2. DOI: 10.1007/978-94-017-7267-9_10. URL: https://doi.org/10.1007/978-94-017-7267-9%5C_10.

[102] Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti. "Off-line mapping of multi-rate dependent task sets to many-core platforms".

In: *Real-Time Systems* 51.5 (2015), pp. 526–565. DOI: 10.1007/s11241-015-9232-1. URL: https://doi.org/10.1007/s11241-015-9232-1.

[103] Sophie Quinton. "Evaluation and comparison of real-time systems analysis methods and tools". In: *Formal Methods for Industrial Critical Systems - 23rd International Conference, FMICS 2018, Maynooth, Ireland, September 3-4, 2018, Proceedings*. Ed. by Falk Howar and Jiri Barnat. Vol. 11119. Lecture Notes in Computer Science. Springer, 2018, pp. 284–290. ISBN: 978-3-030-00243-5. DOI: 10.1007/978-3-030-00244-2_19. URL: https://doi.org/10.1007/978-3-030-00244-2%5C_19.

[104] Konrad Reif. *Automobilelektronik. Eine Einführung für Ingenieure*. Springer Vieweg, 2014. ISBN: 978-3-658-05048-1. DOI: 10.1007/978-3-658-05048-1. URL: https://doi.org/10.1007/978-3-658-05048-1.

[105] Dominik Reinhardt, Udo Dannebaum, Michael Scheffer, and Matthias Traub. "High performance processor architecture for automotive large scaled integrated systems within the european processor initiative research project". In: Apr. 2019, pp. 1–8. DOI: 10.4271/2019-01-0118. URL: https://doi.org/10.4271/2019-01-0118.

[106] Kai Richter, Dirk Ziegenbein, Marek Jersak, and Rolf Ernst. "Model composition for scheduling analysis in platform design". In: *Proceedings of the 39th Design Automation Conference, DAC 2002, New Orleans, LA, USA, June 10-14, 2002*. ACM, 2002, pp. 287–292. DOI: 10.1145/513918.513993. URL: https://doi.org/10.1145/513918.513993.

[107] Laurent Rioux, Rafik Henia, and Nicolas Sordon. "Using model-checking for timing verification in industrial system design". In: *2017 IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2017, Tokyo, Japan, March 13-17, 2017*. IEEE Computer Soc., 2017, pp. 377–378. ISBN: 978-1-5090-6676-6.

[108] Robert N. Charette. *This car runs on code*. Feb. 2009. URL: https://spectrum.ieee.org/transportation/systems/this-car-runs-on-code.

[109] Francesca Rossi, Peter van Beek, and Toby Walsh, eds. *Handbook of constraint programming*. Vol. 2. Foundations of Artificial Intelligence. Elsevier, 2006. ISBN: 978-0-444-52726-4. URL: http://www.sciencedirect.com/science/bookseries/15746526/2.

[110] Eric Sax, ed. *Automatisiertes testen eingebetteter systeme in der automobilindustrie*. ger. Carl Hanser Verlag, 2008. ISBN: 9783446419018. URL: https://content-select.com/index.php?id=bib_view&ean=9783446419018.

[111] Johannes Schlatow and Rolf Ernst. "Response-time analysis for task chains in communicating threads". In: *2016 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS), Vienna, Austria, April 11-14, 2016*. IEEE Computer Society, 2016, pp. 245–254. ISBN: 978-1-4673-8639-5. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7460013.

[112] Johannes Schlatow, Mischa Möstl, Sebastian Tobuschat, Tasuku Ishigooka, and Rolf Ernst. "Data-age analysis and optimisation for cause-effect chains in automotive control systems". In: *13th IEEE International Symposium on Industrial Embedded Systems, SIES 2018, Graz, Austria, June 6-8, 2018*. 2018, pp. 1–9.

[113] Simon Schliecker, Jonas Rox, Mircea Negrean, Kai Richter, Marek Jersak, and Rolf Ernst. "System level performance analysis for real-time automotive multicore and network architectures". In: *IEEE Trans. on CAD of Integrated Circuits and Systems* 28.7 (2009), pp. 979–992. DOI: 10.1109/TCAD.2009.2013286. URL: https://doi.org/10.1109/TCAD.2009.2013286.

[114] ITU Telecommunication Standardization Sector. *X.200 : information technology - open systems interconnection - basic reference model: the basic model*. 1994. URL: https://www.itu.int/rec/T-REC-X.200-199407-I/en.

[115] Robert Shaw and Brendan Jackman. "An introduction to FlexRay as an industrial network". In: *2008 IEEE International Symposium on Industrial Electronics*. June 2008, pp. 1849–1854.

[116] Kang G Shin and Parameswaran Ramanathan. "Real-time computing: a new discipline of computer science and engineering". In: *Proceedings of the IEEE* 82.1 (1994), pp. 6–24.

[117] Andreas Spillner. "The W-MODEL. Strengthening the bond between development and test". In: *Int. Conf. on Software Testing, Analysis and Review*. 2002, pp. 15–17.

[118] International Organization for Standardization. *ISO 17356-1:2005 road vehicles – open interface for embedded automotive applications – part 1: general structure and terms, definitions and abbreviated terms*. Nov. 2005. URL: https://www.iso.org/standard/33006.html.

[119] International Organization for Standardization. *ISO 17356-2:2005 road vehicles – open interface for embedded automotive applications – part 2: osek/vdx specifications for binding os, com and nm*. Nov. 2005. URL: https://www.iso.org/standard/33007.html.

[120] International Organization for Standardization. *ISO 17356-3:2005 road vehicles – open interface for embedded automotive applications – part 3: osek/vdx operating system (os)*. Nov. 2005. URL: https://www.iso.org/standard/40079.html.

[121] International Organization for Standardization. *ISO 17356-4:2005 road vehicles – open interface for embedded automotive applications – part 4: osek/vdx communication (com)*. Nov. 2005. URL: https://www.iso.org/standard/40118.html.

[122] International Organization for Standardization. *ISO 26262-6:2018 – road vehicles – functional safety – part 6: product development at the software level*. Dec. 2018. URL: https://www.iso.org/standard/68388.html.

[123] International Organization for Standardization. *ISO/IEC – 7498-1:1994 – information technology — open systems interconnection — basic reference model: the basic model*. Jan. 2000. URL: https://www.iso.org/standard/20269.html.

[124] International Organization for Standardization. *Road vehicles – diagnostic communication over controller area network (DoCAN) — part 2: transport protocol and network layer services*. Nov. 2011. URL: https://www.iso.org/standard/54499.html.

[125] International Organization for Standardization. *Road vehicles — communication on FlexRay — part 2: communication layer services*. June 2010. URL: https://www.iso.org/standard/46047.html.

[126] Miroslaw Staron. *Automotive software architectures. An introduction*. Vol. 1. Springer International Publishing, 2017. ISBN: 978-3-319-58610-6. DOI: 10.1007/978-3-319-58610-6. URL: https://doi.org/10.1007/978-3-319-58610-6.

[127] Steve C. Talbot and Shangping Ren. "Comparision of fieldbus systems can, ttcan, flexray and LIN in passenger vehicles". In: *29th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS 2009 Workshops), 22-26 June 2009, Montreal, Québec, Canada*. IEEE Computer Society, 2009, pp. 26–31. DOI: 10.1109/ICDCSW.2009.15. URL: https://doi.org/10.1109/ICDCSW.2009.15.

[128] Daniel Thiele, Johannes Schlatow, Philip Axer, and Rolf Ernst. "Formal timing analysis of can-to-ethernet gateway strategies in automotive networks". In: *Real-Time Systems* 52.1 (2016), pp. 88–112. DOI: 10.1007/s11241-015-9243-y. URL: https://doi.org/10.1007/s11241-015-9243-y.

[129] Lothar Thiele, Samarjit Chakraborty, and Martin Naedele. "Real-time calculus for scheduling hard real-time systems". In: *IEEE International Symposium on Circuits and Systems, ISCAS 2000, Emerging Technologies for the 21st Century, Geneva, Switzerland, 28-31 May 2000, Proceedings*. IEEE, 2000, pp. 101–104. DOI: 10.1109/ISCAS.2000.858698. URL: https://doi.org/10.1109/ISCAS.2000.858698.

[130] Timing-Architects Embedded Systems GmbH. *Ta tool suite | vector*. Jan. 2018. URL: https://www.vector.com/int/en/products/products-a-z/software/ta-tool-suite/.

[131] Matthias Traub. "Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug". German. PhD thesis. 2010. 197 pp. ISBN: 978-3-86644-582-6. DOI: 10.5445/KSP/1000020379.

[132] Matthias Traub, Vera Lauer, Thomas Weber, Marek Jersak, Kai Richter, and Jürgen Becker. "Timing-analysen für die untersuchung von vernetzungsarchitekturen". In: *ATZelektronik* 4.3 (2009), pp. 36–41.

[133] Matthias Traub, Alexander Maier, and Kai L. Barbehon. "Future automotive architecture and the impact of IT trends". In: *IEEE Software* 34.3 (2017), pp. 27–32. DOI: 10.1109/MS.2017.69. URL: https://doi.org/10.1109/MS.2017.69.

[134] Matthias Traub, Hans-Jörg Vögel, Eric Sax, Thilo Streichert, and Jérôme Härri. "Digitalization in automotive and industrial systems". In: *2018 Design, Automation & Test in Europe Conference & Exhibition, DATE 2018, Dresden, Germany, March 19-23, 2018*. Ed. by Jan Madsen and Ayse K. Coskun. IEEE, 2018, pp. 1203–1204. ISBN: 978-3-9819263-0-9. DOI: 10.23919/DATE.2018.8342198. URL: https://doi.org/10.23919/DATE.2018.8342198.

[135] Dogan Ulus. "Montre: A tool for monitoring timed regular expressions". In: *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*. Ed. by Rupak Majumdar and Viktor Kuncak. Vol. 10426. Lecture Notes in Computer Science. Springer, 2017, pp. 329–335. ISBN: 978-3-319-63386-2. DOI: 10.1007/978-3-319-63387-9_16. URL: https://doi.org/10.1007/978-3-319-63387-9%5C_16.

[136] Marcel Verhoef, Ernesto Wandeler, Lothar Thiele, and Paul Lieverse. "System architecture evaluation using modular performance analysis - A case study". In: *International Symposium on Leveraging Applications of Formal Methods, ISoLA 2004, October 30 - November 2, 2004, Paphos, Cyprus. Preliminary proceedings*. Ed. by Tiziana Margaria, Bernhard Steffen, Anna Philippou, and Manfred Reitenspieß. Vol. TR-2004-6. Technical Report. Department of Computer Science, University of Cyprus, 2004, pp. 209–219.

[137] Hendrik Weigel, Philipp Lindner, and Gerd Wanielik. "Vehicle tracking with lane assignment by camera and lidar sensor fusion". In: *2009 IEEE Intelligent Vehicles Symposium*. June 2009, pp. 513–520. DOI: 10.1109/IVS.2009.5164331.

[138] Werner Zimmermann and Ralf Schmidgall. *Bussysteme in der Fahrzeugtechnik. Protokolle, Standards und Softwarearchitektur*. Vol. 4. Springer Fachmedien Wiesbaden, 2014. ISBN: 978-3-658-02419-2. DOI: 10.1007/978-3-658-02419-2. URL: https://doi.org/10.1007/978-3-658-02419-2.

[139] Helge Zinner, Julian Brand, and Daniel Hopf. *Automotive e/e architecture evolution and the impact on the network*. IEEE802 Plenary. Mar. 2019. URL: http://ieee802.org/1/files/public/docs2019/dg-zinner-automotive-architecture-evolution-0319-v02.pdf.