TECHNISCHE UNIVERSITÄT MÜNCHEN
Institut für Informatik
Lehrstuhl für Echtzeitsysteme und Robotik

A Model-Based Framework for System-Wide Plug-and-Play with Flexible Timing Verification for Automotive Systems

Hauke Stähle

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender:	Prof. Dr. Hans Michael Gerndt
Prüfer der Dissertation:	1. Prof. Dr.-Ing. habil. Alois Knoll
	2. Hon.-Prof. Dr.-Ing. Gernot Spiegelberg, Universität Budapest/Ungarn

Die Dissertation wurde am 12.11.2015 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 08.03.2016 angenommen.

Abstract

The integration of hardware and software components into today's vehicles from a variety of suppliers is a complex process and becomes more and more challenging. The amount of code and data as well as the number of interconnections increases rapidly and pushes the complexity of the on-board electronic systems and the involved infrastructure to new limits. This stands in contrast with the constantly growing demand for new functionalities to enhance safety, comfort, and efficiency. Integration expenses are an increasing problem during design time and the addition of hardware and software to a vehicle after sale is limited today, because the systems are developed in a static manner.

To ease the integration process and to allow the addition of functionality after sale, a model-based framework is proposed in this work that combines the plug-and-play concept with an automatic timing verification to fulfill the real-time requirements of automotive systems. The idea is to divide the functionality of a vehicle into individual features that can be freely composed. Each feature consists of a set of hardware and software components as well as communication and timing requirements, which are automatically matched and verified in the resulting system setup. Addition of further features and automatic re-verification is possible at any point in time with an adjustable approximation level. The approach is capable to process event-based communication patterns and is based on the data-centric design principle, i.e., data senders and receivers are loosely coupled.

The contributions of this work comprise the introduction of the system-wide plug-and-play approach, the definition of a minimal set of suitable models, transformation patterns for a mapping to exemplary technologies, and the introduction of a method to specify timing requirements for unknown setups. They further include the development of a performance verification tool based on the Real-Time Calculus framework, its enhancement for the automatic processing of cyclic resource dependencies, and the design of an approach to control the approximation of the analysis and trade tightness of the derived timing bounds for computation time.

The feasibility of the approach is shown by a running example based on the electric vehicle demonstrator (eCar) throughout this work. The performance of the extensions and approximation approaches for the verification process are examined in detail by a series of experiments.

Zusammenfassung

Die Integration von Hardware- und Softwarekomponenten verschiedenster Zulieferer in heutige Fahrzeuge ist ein komplexer Vorgang und wird immer schwieriger. Die Menge an Software und Daten sowie die Anzahl der Verbindungen steigt rapide an und bringt die Komplexität der Elektronik im Fahrzeug und der benötigten Infrastruktur an ihre Grenzen. Dies steht im Widerspruch zu dem stetig wachsenden Bedarf an neuen Funktionen um Sicherheit, Komfort und Effizienz zu erhöhen. Die Kosten für die Integration zur Entwurfszeit sind ein zunehmendes Problem und das Hinzufügen von neuer Hardware und Software in ein Fahrzeug ist heutzutage nach dem Kauf nur eingeschränkt möglich, weil die Systeme statisch entwickelt werden.

Um den Integrationsprozess zu vereinfachen und um das Hinzufügen von Funktionen nach dem Kauf zu ermöglichen, wird in dieser Arbeit ein modellgetriebener Ansatz erläutert, welcher den Plug-and-Play-Gedanken mit einer automatisierten zeitlichen Verifikation verknüpft, um den Echtzeitanforderungen eines Fahrzeugsystems gerecht zu werden. Die Idee besteht aus einer Aufteilung der Funktionen eines Fahrzeugs in einzelne Features, welche frei miteinander kombiniert werden können. Jedes Feature kann eine Menge von Hardware- und Softwarekomponenten beinhalten sowie Anforderungen bezüglich der Kommunikation und des zeitlichen Verhaltens, welche automatisiert verarbeitet und in der resultierenden Systemausprägung verifiziert werden. Das Hinzufügen von weiteren Features und die erneute automatisierte Verifikation sind zu jedem Zeitpunkt möglich mit einer freien Wahl des Approximationsgrades. Der Ansatz ist in der Lage, ereignisbasierte Kommunikationsmuster zu verarbeiten und basiert auf dem datenzentrischen Entwurfsprinzip, bei dem die Sender und Empfänger von Daten lose miteinander verbunden sind.

Die Beiträge dieser Arbeit umfassen die Einführung des systemweiten Plug-and-Play-Ansatzes, die Definition einer minimalen Menge von geeigneten Modellen, Transformationsmuster für eine Abbildung auf exemplarische Technologien und die Beschreibung einer Methode zur Spezifikation von zeitlichen Anforderungen in unbekannten Konfigurationen. Sie beinhalten weiterhin die Entwicklung eines Werkzeugs, basierend auf dem Real-Time Calculus, zur Verifikation des zeitlichen Verhaltens und dessen Erweiterung zur automatisierten Verarbeitung von zyklischen Abhängigkeiten sowie dem Entwurf einer Möglichkeit, die Approximation der Analyse zu steuern und somit die Genauigkeit der berechneten Zeitgrenzen gegen Berechnungsaufwand einzutauschen.

Die Realisierbarkeit des Ansatzes wird mittels eines durchgängigen Beispiels innerhalb dieser Arbeit aufgezeigt, welches auf dem elektrischen Fahrzeugdemonstrator (eCar) basiert. Die Eigenschaften der Erweiterungen und Approximationsmethoden für den Verifikationsvorgang werden durch eine Reihe von Experimenten untersucht.

Danksagung

Die vorliegende Arbeit entstand während meiner Anstellungen als wissenschaftlicher Mitarbeiter am Lehrstuhl für Echtzeitsysteme und Robotik an der Technischen Universität München sowie am Forschungs- und Transferinstitut fortiss. Mein Dank geht an Professor Alois Knoll für das Ermöglichen dieser Arbeit, seine wissenschaftliche Betreuung und unterstützende Organisation. Ebenso danke ich Professor Gernot Spiegelberg für seinen Einsatz als Projektleiter von Diesel Reloaded, die vielen aufgebrachten Stunden und die Chance an seinem Weitblick teilhaben zu dürfen. Mein Dank gilt Claudia Meis und Dr. Ljubo Mercep für die gegenseitige Hilfe in unserem gemeinsamen Projekt und darüber hinaus. Dr. Christian Buckl möchte ich für die konstruktiven Ratschläge in allen Belangen danken, für die Möglichkeit, in seiner Forschungsgruppe mitwirken zu dürfen und für den stets motivierenden Umgang. Dr. Kai Huang gebührt mein Dank für seinen Optimismus, seine wissenschaftliche Kompetenz und für die hilfreichen Kommentare zu dieser Arbeit. André Gaschler und Michael Geisinger danke ich für die wertvollen Vorschläge nach dem Durchsehen dieser Ausarbeitung. Meinen Kollegen Benjamin Wiesmüller, André Leimbrock und Michael Geisinger danke ich für ihre Geduld und Verständnis, als ich mich in den letzten Schritten dieser Dissertation nur mit reduzierter Kraft unserem Vorhaben widmen konnte. Ein umfangreiches Lob gilt den von mir betreuten Studenten für ihr entgegengebrachtes Vertrauen und Engagement. Dies sind in alphabetischer Reihenfolge: Roshan Chulyada, Christian Dietz und Raphael Haase. Darüber hinaus möchte ich dem gesamten Lehrstuhl für Echtzeitsysteme und Robotik an der TUM sowie der Gruppe Cyber-Physical Systems am fortiss danken für den freundlichen und kooperativen Umgang miteinander.

Meiner Freundin gilt ein besonderer Dank für ihre entgegengebrachte Wärme, die notwendige Ablenkung und die Übernahme vieler Aufgaben, damit ich mich intensiv dieser Arbeit widmen konnte. Bei meinen Freunden bedanke ich mich für das erholende Einbringen von anderen Gedanken in mein Leben.

Zuletzt, aber nicht minder wichtig, möchte ich meiner Familie danken, welche mich aus der Ferne stets in meinen Vorhaben unterstützt und an meinen Erfolg glaubt.

München, im Oktober 2015 *Hauke Stähle*

Contents

List of figures

List of tables

List of acronyms

AUTOSAR Automotive Open System Architecture.

BCET Best-Case Execution Time.

CAN Controller Area Network.
CPU Central Processing Unit.

DAG Directed Acyclic Graph.

eCar Experimental Platform for Innovative Electric Car Architectures.
ECU Electronic Control Unit.
ET Event-Triggered.

FIFO First-In-First-Out.
FPNP Fixed-Priority Non-Preemptive.

GPC Greedy Processing Component.

HMI Human-Machine Interface.

ICT Information and Communication Technology.

IEEE Institute of Electrical and Electronics Engineers.
ISO International Organization for Standardization.
ISR Interrupt Service Routine.

MD Maximum Delay.

NC Network Calculus.

OMG Object Management Group.

PJD Period, Jitter, minimum Distance.

RTC Real-Time Calculus.

TD see TDMA.
TDMA Time Division Multiple Access.
TT Time-Triggered.

UML Unified Modeling Language.

WCET Worst-Case Execution Time.

List of acronyms

List of symbols and operators

Notation	Description
$\mathbb{N}$	Set of natural numbers.
$\mathbb{B}$	Set of Boolean values.
$\mathbb{F}$	Set of wide-sense increasing functions with $f(t) = 0 \; \forall \; t \leq 0$.
Δ	A time interval.
α	Pair of arrival curves.
β	Pair of service curves.
$\{\alpha, \beta\}^u, \{\alpha, \beta\}^l$	Upper/Lower arrival/service curve.
γ	Pair of workload curves.
$\widehat{\alpha}, \widehat{\beta}$	Scaled curves.
C	Specification of a segment-wise defined curve.
$\mathbb{S}_a$	Set of all segments of aperiodic part of C.
$\mathbb{S}_p$	Set of all segments of periodic part of C.
s	A single segment.
(c_{x0}, c_{y0})	Starting point of periodic part of C.
$(c_{\Delta 0}, c_{\Delta 0})$	Repetition vector for periodic part of C.
C_ρ	Long-term slope of curve C.
$c(\Delta)$	Function of unfolded curve C.
Φ	Limit for unfolding of a curve C.
$\otimes$, $\overline{\otimes}$	Min-plus/Max-plus convolution.
$\oslash$, $\overline{\oslash}$	Min-plus/Max-plus deconvolution.
$x^\star$	Subadditive closure of x.
$x^{\overline{\star}}$	Superadditive closure operation of x.
$p()$	Curve approximation function.
$\lfloor \; \rfloor$	Floor function.
$\lceil \; \rceil$	Ceiling function.
$f()$	Stream filter function.
$\mathbb{P}$	Set of filter parameters.
$\mathbb{E}$	Set of analysis results.
d	Delay.

Notation	Description
d_{e2e}	End-to-end delay.
b	Backlog size.
$D()$	Delay-bound function (max. horizontal distance).
$B()$	Backlog-bound function (max. vertical distance).
MBS	Maximal busy-period size.
r^{FF}	Fraction factor (approximation parameter).
$\tilde{x}$	Approximation of x.
$\mathfrak{M}$	An ANALYSIS model.

Chapter 1

Introduction

The electronic architecture of a vehicle today is a highly complex distributed system. Many of the functions are implemented in software running on several electronic control units (ECUs), which are distributed across the whole vehicle and that are interconnected via heterogeneous networks for data exchange. This electronic architecture supports and controls the mechanical setup of the vehicle to enhance safety, comfort, efficiency, and other factors. The number of ECUs already reached 30 to 100 in one vehicle, with up to 100 million lines of code running on them [1]. The number of functions and variants of vehicles will continue to increase in the future [2]. With the amount of functions, the degree of interconnection will increase as well and the functions will become more dependent on each other. This raises complexity and costs for the integration as the system grows further. The current development processes are not suited to implement this growing complexity with reasonable costs. Fig. 1.1 compares the complexity inherent to the functionality, which cannot be changed, and the complexity that is caused by the currently applied technologies and architecture of the system. The gap between those two metrics defines a room for improvement to reduce unnecessary costs and effort [3, 4].

This thesis proposes a method to reduce the gap between the actual and needed complexity by a system-wide plug-and-play approach with an integrated, automatic verification process. This ultimately helps to lower integration costs and increases flexibility during development and configuration of a vehicle, even so far that it allows to change the setup after sale.

1.1 Factors increasing the complexity and amount of functions

Following, reasons for the ever-growing amount of functions and complexity in the automotive domain are given, divided into four categories: Social mega-trends, political decisions, individual comfort and safety, and competition. The statements are an extension of a study about the future vehicle information and communication technology architecture [3].

1.1.1 Social mega-trends

Social mega-trends are driven by individuals and societies whose actions stem new requirements for the electronic architecture. In the last decade, especially three social mega-trends

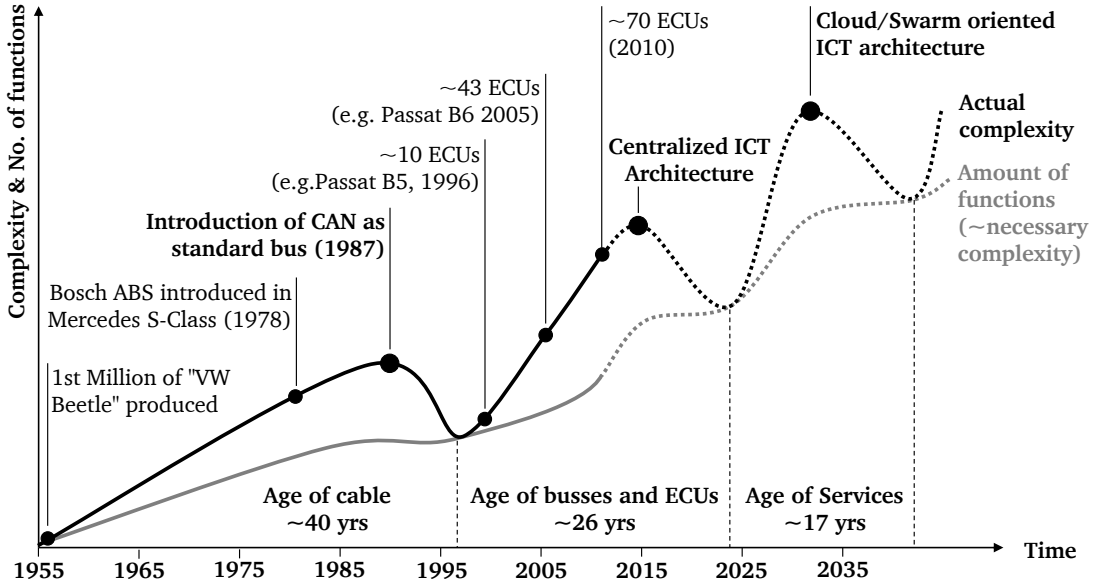


Figure 1.1: Development of automotive electronic architecture complexity (adopted from [3]). The gap between the actual and the necessary complexity defines a room for improvement for future architectures.

were visible: Environmental care, increasing urbanization, and the demographic change of society, which are elaborated in the following. **Environmental care** summarizes the actions and thoughts to reduce the environmental impact of a human being, e.g., to reduce carbon dioxide exhausted by combustion engines or the effort it takes to build and recycle a personal vehicle. The solutions include the usage of vehicles that run on alternative energy sources than fossil fuels, like electric vehicles or vehicles with a mixture of combustion and electric engines. This increases the number of vehicle types a manufacturer has to provide in order to meet the expectations of costumers. This demands a flexible approach that can handle different configurations seamlessly [6]. The environmental aspects are not bounded by the borders of the vehicle. The approaches intend to give a holistic solution to electric mobility, i.e., considering electric power generation, transmission, and storage as well. Exemplary for the research activities in this area, the InnoTruck is shown in Fig. 1.2, which functions as a smart grid if electric vehicles are connected to it and optimizes the overall energy balance [5, 7]. **Urbanization** refers to the movement of households from sparse populated areas into urban areas. People living in cities have different requirements for transportation than those living at the countryside. During the growth of the cities, this difference even increases. The acceptance of dependence on public transportation is much more developed in urban areas as the public transportation system is usually better equipped and space for individual vehicles is expensive and rare. Nevertheless, sometimes demand for an individual vehicle arises to solve certain tasks. To solve this situation, car-sharing concepts are available in cities that offer the possibility to easily rent vehicles, which are distributed across the city. This combines the flexibility of an own vehicle while keeping

Figure 1.2: The research and technology demonstrator InnoTruck as an example for a holistic approach to electric mobility. Several electric vehicles can be connected to the InnoTruck to show different aspects of micro smart-grids [5].

costs low. But car-sharing concepts bring up new challenges for the electronic architecture of vehicles: Customers want to have instant access to personal data, independent of the actual vehicle that is used in that moment. Consequently, the authentication of the driver has to be established with the ability to securely store and download data from remote points [4]. As the life expectancy of the people in developed countries continues to increase and in contrast the number of newborns shrinks every year, the distribution of society changes – known as the **demographic change**. The wish for mobility persists in all ages, effectively meaning that the average age of a driver increases as the average age of the society does [8]. In this context, vehicles should support older people to keep their wish of mobility. This can be established by intelligent and tolerant vehicles, which help the driver and intervene in case of dangerous situations. That puts new challenges to the electronic architecture of the vehicles as the responsibility of an accident-free journey is transferred from the driver to the vehicle.

1.1.2 Political decisions

Beyond social mega-trends, political decisions influence the electronic architecture. The international **standard ISO 26262** [9] is meant as a guideline for the development of safety-critical automotive systems. It covers the aspect of functional safety and contains safety requirements, a methodology for the development, and implementation patterns. The standard directly impacts the development process and electronic architecture of vehicles, see for example [10]. Besides, the government defines functionalities that have to be implemented in order to increase the safety or other aspects of a vehicle or traffic. An example is the **eCall system** that was made compulsory for all new vehicles from 2018 on within the European Union [11]. The eCall system can automatically connect to an emergency service in case of an accident and transfer relevant data like the position of the vehicle. It is very likely that other systems will be made compulsory in the future, especially more advanced active safety systems that automatically intervene the commands of the driver. Political regulations further demand vehicle manufacturers to reduce the **ex-**

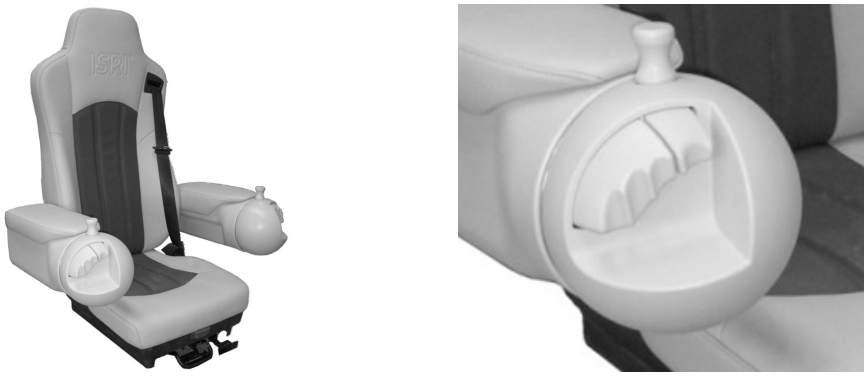


Figure 1.3: Sidesticks of the InnoTruck as an example for the human-machine interface of a drive-by-wire system [15, 16].

hausting footprint of the produced fleet. This is possible, on the one hand, by a switch to electric or hybrid vehicles [12] and, on the other hand, by an increase of the efficiency of the drive-train by mechanical and electronic improvements. Because pollution of cities is an increasing problem, it is expected that this trend will continue in the future.

1.1.3 Individual comfort and safety

While political decisions and social mega-trends limit or influence the decisions of individuals to buy vehicles, personal factors are not less important, which focus on the individual experience while driving, like comfort and safety. **Intervention** refers to the process of taking control by an electronic system as part of an active safety system. Whenever a dangerous situation is detected, the driver is first warned and in case the driver does not react, steering, braking, and acceleration are directly controlled by the electronics. An exemplary study of the impact of such an autonomous braking system is provided in [13]. The number of such intervening functions probably increases in the future to enhance safety. These systems usually rely on multiple states of the vehicle from different sources, e.g., the wish of the driver, the road condition, and an environmental representation, which leads to a strong coupling of the involved electronic systems. **Autonomous** or semi-autonomous vehicles execute a driving task with no or little inference from a driver. A driving task in this context refers to a control of the vehicle for a certain amount of time and/or distance, in extreme a complete journey. Autonomous vehicles render another challenge on the electronic architecture, because the driving task has to be constructed in a fail-operational manner [14], i.e., the driving task must not fail even if a fault in the system occurs. For example, it must not be interrupted by a malfunction of a sensor, actuator, or electronic control unit. **Drive-by-wire** refers to a setup, where the primary connection between driver and vehicle – regarding the execution of driving commands – is realized with an electronic system. E.g., instead of a direct mechanical connection between steering wheel and axle, a sensor at the steering wheel recognizes the commands, an electronic control unit processes the data, and control actuators then in turn manipulate the axle. Drive-by-wire functionality simplifies implementation of autonomous and intervening vehicles as direct execution of control commands is possible [17], but are complex to implement. Vehicles become in-

creasingly **connected** and access data from different sources. They connect to the internet to access real-time data like traffic information or communication services like e-Mail. This connection is not limited to internet services: Car-to-car and car-to-infrastructure communication is an upcoming topic to enhance the safety and comfort of vehicles [13]. This has several impacts on the electronic architecture of the vehicle as multiple radio communication standards have to be supported and data transfers have to be established dynamically. For example, a video call might be routed through the network of the vehicle, but only if the available resources allow so.

1.1.4 Manufacturer competition

While the above mentioned trends are directly consumer-visible, the electronic architecture is subject to modifications to reduce development effort and costs, and to increase reliability and scalability. These are implicit changes that influence the competition of the manufacturers. Reduction of production and development **costs** of a vehicle is an omni-present objective in order to stay competitive. New solutions for the electronic architecture are demanded that provide the same functionality while cutting the expenses to a minimum. This can be achieved, for example, by the integration of the functions of several electronic control units into one unit and therefore reducing the hardware costs by the sharing of resources [18]. Physical integration is not the only dimension as the development effort contributes increasingly to the costs of a vehicle, especially when considering the growing amount of distributed and dependent functions that make the development process difficult. **Diagnosis** services of vehicles should enable a service technician and the manufacturer to easily spot causes of faults. Diagnosis standards like the On-Board-Diagnostics (OBD) are already existing, but those cover only a subset of the functionality of the vehicle. While the diagnosis system is active, it might utilize a relevant part of the available data rate and therefore might influence the timing of other functions. With new electronic architecture concepts, which handle diagnosis ability as a first class objective, performance of such services can be improved. To extend the functionality and to overcome errors in the firmware of electronic control units, the possibility of **updates** has to be present [4]. During the update process the firmware is exchanged by deploying a new version into the non-volatile memory. This process is usually performed via the existing infrastructure, where the communication buses – depending on their data rate – can significantly slow down this process. If the electronic architecture is re-designed, this bottleneck can be mitigated [19]. The number of **variants** of vehicles is steadily increasing to fit the costumers' needs as close as possible. Handling those variants and their configuration becomes difficult as more combinations have to be considered. By extending the capabilities of the electronic architecture to support the verification and integration process, the variability problem can be mitigated successfully. An additional argument is the limited extendibility of vehicles. As of 2015, it is hardly possible to extend the functionality after sale. This includes the addition of sensors, actuators, and/or software components, possibly by different vendors. **Certification** describes the proof of functional or non-functional behavior of the system. It is carried out by measurements, long-time testings, or with formal approaches. The certification of safety-relevant components is frequently the most difficult part in the certification

process. The electronic architecture can be designed in a way that it either supports this process or even does it automatically, either before or during runtime [20].

1.1.5 Summary of influencing factors

The mentioned points show that the complexity of the electronic architecture is likely to increase further in the future. Countermeasures have to be developed to reduce the development and integration effort, in order to still be able to enhance the functionality while keeping the induced costs to a minimum. This thesis contributes to this demand by providing an alternative design and verification approach.

1.2 Trends influencing the automotive electronic architecture

This section introduces current trends in the technology and structure of automotive electronic systems, which have the potential to reduce the development and integration effort in the future.

1.2.1 Transition from static to dynamic systems

Vehicles used to be static systems – once built and configured, the setup did not change throughout its whole lifetime. As software began to play an important role in the functionality of a vehicle, diagnosis and updates of the system became necessary to mitigate possible faults and to fix safety-related bugs that were found after delivery. This evolved to partly-adaptable systems, where predefined hardware can be added and software customizations are possible. These changes of the system are either already known during the design time of the vehicle, or the modification is only possible in a limited way, e.g., by the installation of additional "Apps" on dedicated, mostly user-interactive, control units. We expect that this behavior will change in the future, towards more flexible systems that can be extended by hardware and software unknown during the design time of a vehicle. In order to support this, the vehicle has to be equipped with self-describing and reconfiguration capabilities, in short, it has to become adaptive. The change to dynamic systems is not limited to automotive systems, as can be seen by the upcoming topic of *smart* cyber-physical systems. They extend the definition of cyber-physical systems (CPS), which are "*[...] an integration of computation with physical processes whose behavior is defined by* both *cyber and physical parts [...]*" [22]. In contrast, smart CPS focus on a cross-, self-, and live-domain [21] as shown in Fig. 1.4. The main properties of these systems include that they have an understanding of themselves, their requirements and the environment in that they are embedded. Based on this information, flexible and dynamic systems can be built. We expect these concepts to be transferred into the automotive area to overcome future requirements.

The transition to dynamic systems goes hand-in-hand with a change of the messaging paradigm towards data-centric communication [23], which decouples the senders and receivers of messages. The entities communicate via so-called topics that define the structure of the data and provide a unique name for referencing. The actual communication routes of the system are either calculated automatically during design time or during runtime. This enables a flexible integration of components into the system.

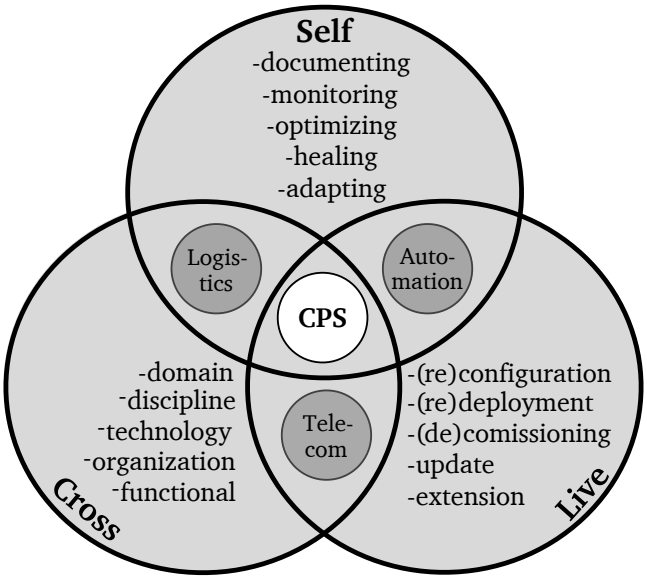


Figure 1.4: Properties of smart cyber-physical systems (adopted from [21]). Smart cyper-physical systems have a self-understanding of their capabilities and state, are deployed across multiple technologies and offer the ability for a seamless reconfiguration. It is expected that automotive systems will be equipped with similar features in order to become more flexible.

1.2.2 Transition from vertical, distributed to horizontal, integrated designs

One possibility to mitigate the increasing complexity is a shift of the architecture towards a centralized design. This shift is two-fold: On the physical level by an integration of several function onto a single electronic control unit [24], and on the logical level by a re-arrangement of the functional dependencies into horizontal layers, see for example the service architecture of [25].

At the **physical level**, centralization refers to a transition of distributed processing entities towards integrated ones to reduce hardware and software costs by resource sharing, see Fig. 1.5. Furthermore, it is possible to upgrade the existing system by addition of functions on the centralized nodes. Such a design typically consists of *centralized processing units* as the integration point for most functions, *smart aggregates* as sensor and actuator entities, and a *communication network* to interconnect the different nodes [26]. One example for a centralized physical architecture is the architecture of the RACE (Robust and Reliant Automotive Computing Environment for Future eCars) project [27], shown in Fig. 1.6. A circular topology for the communication system was chosen, forming one ring for the central processing platforms and two rings for aggregates of the front and back of the vehicle. The rings are interconnected with each other via gateways. Communication between central processing platforms is implemented redundantly by a double-ring. Effectively, four different paths exist: Two by each physical ring and one in each direction (clock-wise and counter-clock-wise). Depending on the safety requirements, the rings to connect aggregates are open or closed and realized either redundant or simple. One special feature of central

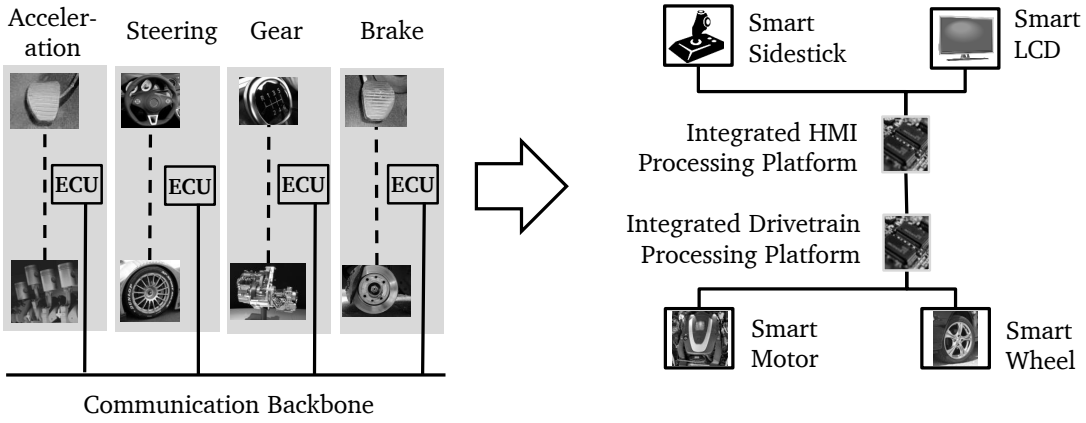


Figure 1.5: Centralization of the physical architecture by integration (adopted from [6]). Costs are reduced by sharing resources on the integrated processing platforms and the integration process is simplified. Smart aggregates abstract the functionality of sensors and actuators and offer a modularization point for a flexible system setup.

processing platforms is the double-lane architecture. Each central processing platform consists of two processing lanes. Safety-critical data is processed by both lanes simultaneously and an error is detected by a comparison of the results of both lanes. In case of an error, the central processing platform is considered faulty and switched off. Therefore, with one central processing platform, a system with fail-silent behavior can be implemented. If two central processing platforms are utilized, each with a double-lane architecture, it is possible to design a fail-operational system by a hot stand-by configuration.

At the **logical level**, centralization refers to a combination of arbitration and fusion points by a shift from the vertical architecture paradigm to a horizontal architecture paradigm. In the classic vertical paradigm, one electronic control unit is responsible to provide one specific functionality of the vehicle combined with the mechatronic components, as seen at the left side of Fig. 1.7. To enhance functionality, the electronic control units were interconnected to share state and event information. With this approach, functions running on electronic control units are highly coupled with each other across the vehicle, because they need knowledge from each other for a proper arbitration and fusion of data. System management is distributed among multiple processing entities, a global state view and strategy does not exist. With a centralization of the logical architecture, these kinds of problems are avoided. The idea is to decouple the strategy level from the execution level. The two levels are interconnected by generic or generated layers for data fusion and arbitration, as depicted at the right side of Fig. 1.7. With this concept, it is possible to exchange or extend functions of the strategy level as well as sensors and actuators of the execution level with little mutual influence. For example, consider a drive-by-wire system: Sensors from the driving wheel do not necessarily need to be exclusively connected with actuators that control the posture of the wheels. Other functions (e.g., safety related intervention) might control these actuators and a decoupling is useful in that case. An extreme implementation of this principle was presented in the SPARC (Secure Propulsion Using Advanced Redun-

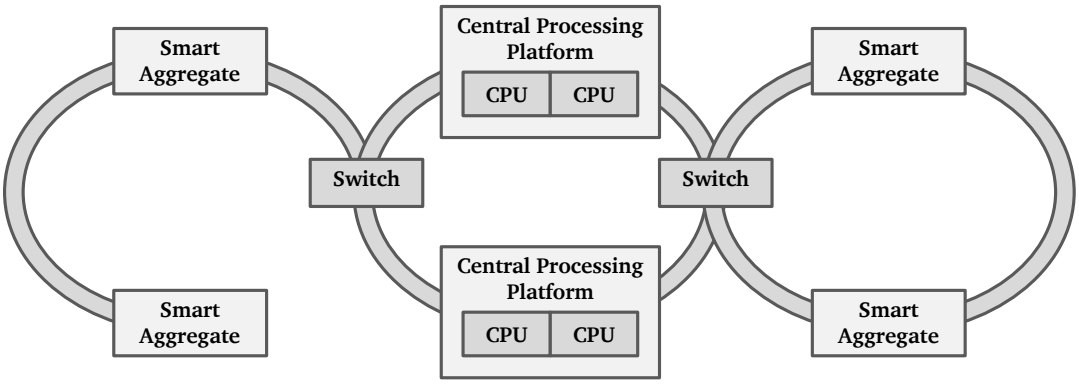


Figure 1.6: RACE architecture as an example of a physically centralized design (adapted from [27]). The setup enables the execution of fail-operational functions.

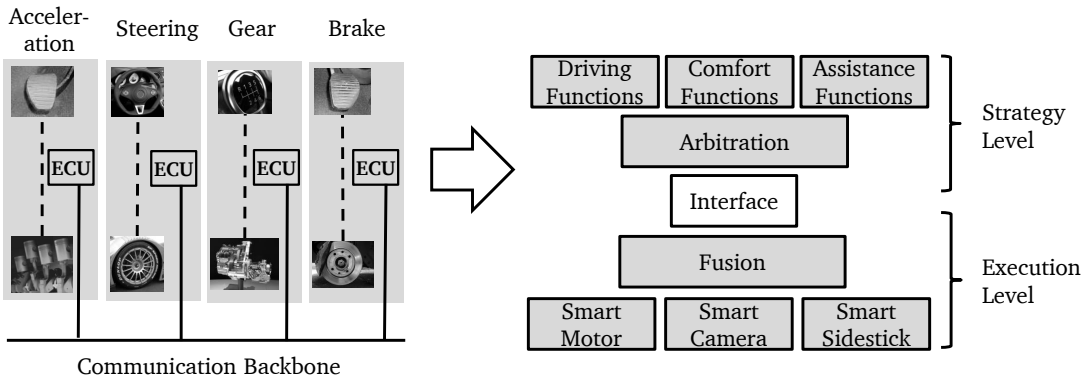


Figure 1.7: Centralization of the logical architecture by horizontal design. In the horizontal design, the direct relations between sensors/actuators and functions is broken up in favor of functional layers. This decouples the strategy from the execution level and therefore increases the flexibility.

dant Control) project [28, 29]. In that project, the interface between the strategy level and the execution level consists only of a single vector that includes the desired direction and velocity of the vehicle. With that minimalistic interface, a partly or complete change of the execution level, e.g., engine or even vehicle type, is possible.

5-module approach as example for an integrated, horizontal design. The 5-module architecture [26] as proposed by Prof. Spiegelberg is an example for a centralized design. The modules and their relations are sketched in Fig. 1.8. This architecture describes a partitioning of functions of a vehicle on a logical and physical level into five separate modules: Human-machine interface, virtual co-driver, drivetrain, comfort systems, and an infrastructure meta-module. From a logical point of view, the human-machine interface as well as the virtual co-driver are mapped to the strategy level of the system. These modules provide a driving vector that is forwarded to the execution level, which consists of the comfort systems and the drivetrain module. A fifth module, called infrastructure meta-module, ren-

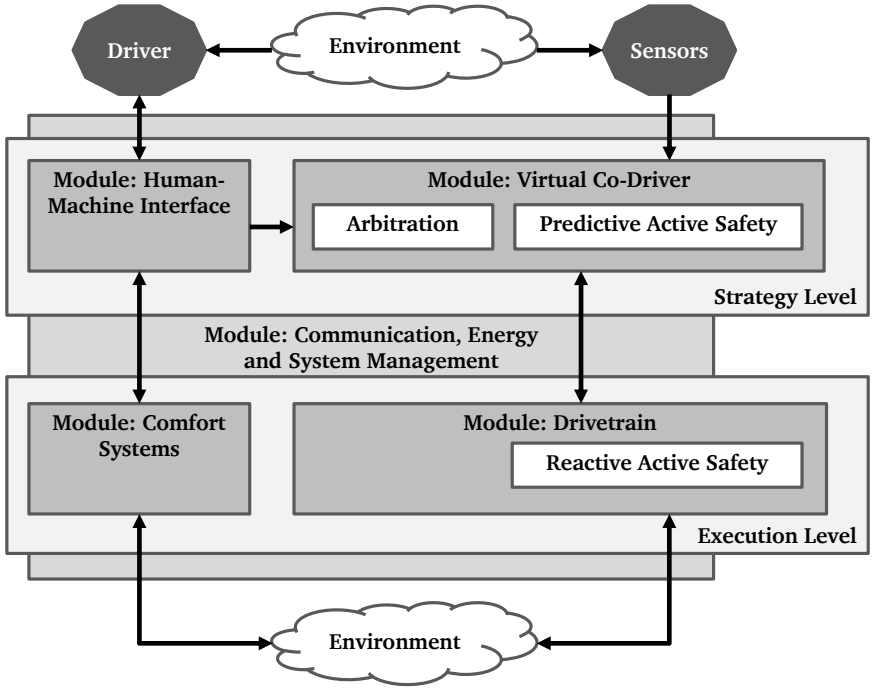


Figure 1.8: The 5-module approach combines physical and logical architecture aspects (adopted from [26]). Each module represents a certain set of functions that are executed on an integrated platform. The possibility of an exchange of individual modules makes this approach flexible in the configuration.

ders the access point for remote communication services and implements generic services for platform monitoring and management. Beyond the logical structure also the physical structure is defined by this approach as every module is mapped onto an individual platform module. The infrastructure meta-module serves as the motherboard for the other modules.

1.2.3 Automotive architecture trends summary

Changes of the automotive architecture have the potential to mitigate the increasing complexity. Solutions include a physical integration of functions and a horizontal design paradigm for modularity and adaptability. The discussed principles rely on a possibility for an automatic and adaptable verification of the system to support different setups effortlessly. The remainder of this work contributes to this demand by the specification of an adaptive integration and verification concept.

1.3 Research goals

Based on the increasing amount of functions and complexity in vehicles and the changes of the automotive electronic architecture in general, the question arises, how processes and systems can be improved to support more flexibility and to reduce integration effort. These aspects are expressed with two research goals:

Goal 1: Reduce integration effort by an automatic integration and verification process. The first goal is directly derived from the increasing complexity of the automotive architecture caused by an increasing number of functions. The goal is to develop a sound approach, which enables the automatic verification of timing constraints for systems that are composed in a plug-and-play manner. The approach should not be limited to the design time of a vehicle, but should also be practicable after shipping to allow the addition of hardware and software components after sale without expert knowledge.

Goal 2: Make the verification flexible in tightness and computation time. The second goal is to approximate the verification process to make it flexible in the sense that tightness can be traded for analysis computation time. This enables the application of the approach in various scenarios: During the design time of the vehicle, when tight results are needed and analysis time is not an issue, and during operation, when an instant verification should be performed but the tightness of the results does not matter as they have only a limited lifespan, e.g., when a temporary audio or video stream is established via the infrastructure of the vehicle.

1.4 Main contributions

The main contributions of this thesis are summarized in the following:

- **Methodology for system-wide plug-and-play with timing guarantees.** The proposed approach is based on a model-driven development of the vehicle's subsystems. A minimal set of seven adequate meta-models to represent the properties of these subsystems is introduced, which are the base of the approach. It includes the detailed explanation of a method to specify timing requirements for data-centric communication, where the concrete relations between data senders and receivers are unknown. The feasibility of the method is shown with a running example throughout this work.

- **Transformations and patterns for execution platform mapping.** To derive a representation suitable for deployment and timing analysis, the different models have to be combined and the entities have to be mapped onto an execution platform. The involved strategies to calculate data paths across several subsystems is developed and patterns for the mapping to concrete technologies are presented. The result is a holistic, deployable, and analyzable model of the system as input for the verification framework.

- **Seamless framework for the timing verification.** A verification framework is developed based on the mathematical foundations of the Real-Time Calculus [30], which is capable to gain performance metrics for heterogeneous, event- or time-triggered systems. The framework differentiates itself from other implementations by an integrated and unique approach to automatically process resource cycles and the ability to calculate the closure of curves in a simplified way, which is a prerequisite to handle components with finite buffer semantics. It includes an automatic mechanism to verify the timing requirements provided by the analyzable model.

- **Approximation techniques to trade analysis computation time for result tightness.** The time needed for the verification process depends in general on the setup of the system and involved timing parameters. Because this may be unsuitable for a quick decision process, it is shown how computational complexity can be reduced by several approximation strategies. Those include the possibility to seamlessly exchange tightness of the results for analysis time, while still producing valid bounds, i.e., a worst-case end-to-end delay is always over-approximated and a best-case end-to-end delay is always under-approximated. Further, a method is introduced that enables verification on memory-restricted platforms in trade for analysis time. The performance of the suggested modifications are evaluated with a series of well-defined experiments.

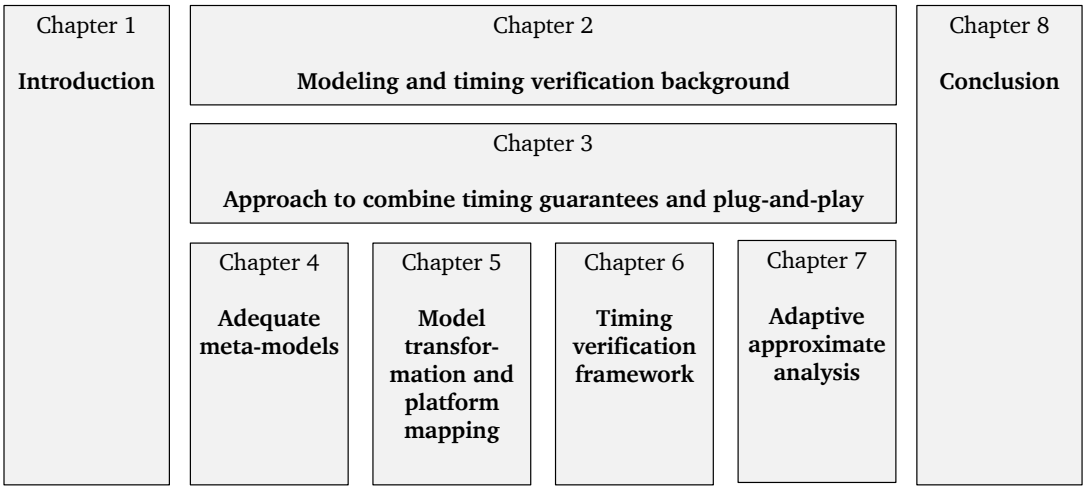


Figure 1.9: Structure of this thesis.

1.5 Structure of this document

This thesis is divided into eight chapters, as illustrated in Fig. 1.9. Background information for system modeling and timing verification is presented in Ch. 2. The actual approach to combine timing guarantees and plug-and-play is introduced in Ch. 3. The implementation of the approach is detailed in Ch. 4 to Ch. 7. The meta-models used to model the individual parts of the automotive electronic system are described and discussed in Ch. 4. The transformations for the transfer of the abstract to a technology-dependent representation are shown in Ch. 5. The actual verification framework to analyze a system and check the conformance to timing requirements is given in Ch. 6. The last step is the ability of the framework to seamlessly trade analysis computation time for tightness of the results, which is presented in Ch. 7. The last chapter concludes the thesis, gives an outlook on other domains the approach can be mapped to, and discusses possible future research activities.

Chapter 2

Modeling and timing verification background

This chapter gives background information on the basic principles that are required to model the system for the plug-and-play approach in the following chapters. They include component-based and model-driven development, and data-centric communication. Further, time modeling and representation is discussed and appropriate methods for a timing analysis are compared. The choice for a particular analysis framework (Real-Time Calculus [30]) is justified and its concepts and mathematical foundations are introduced.

2.1 Component-based development

Component-based development [31] is built on an encapsulation of functionality in so-called components with a hidden implementation, defined interfaces, and possibly an internal state. Composition of individual components and the definition of relations between them yields an actual system setup. The black-box behavior of components hides complexity from the system integrator, ultimately speeding up the development process. An example is shown in Fig. 2.1, where the functions data acquisition (sensor component), data processing (process component), and actuating (actuator component) are encapsulated into components with external relations to each other. Points for data input and output of a component are referred to as ports, the direction is indicated by an arrow. Highlighted in [32], component-based development is characterized by four main principles: *Reusability*

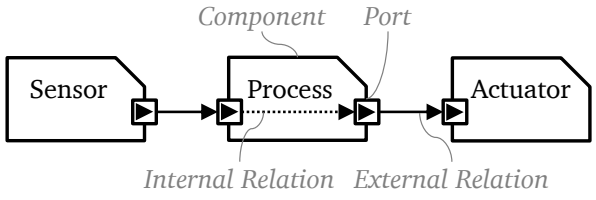


Figure 2.1: Example of a component-based system design. Implementation and state is encapsulated into components, input and output ports define interfaces used for communication with other entities.

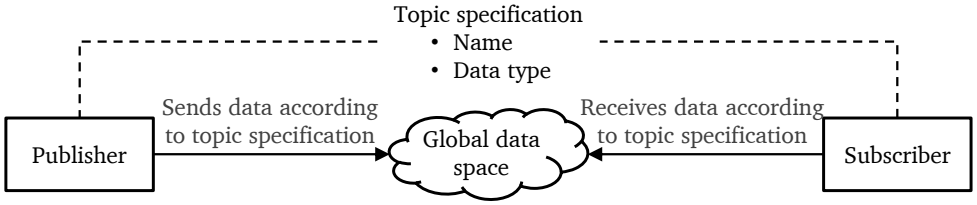


Figure 2.2: Entities of the data-centric communication paradigm as used in this work. Senders (*publishers*) and receivers (*subscribers*) of data are decoupled. The matching is based on *topics* that include a unique name and a data format. Subscribers, publishers, and topics can be enriched by attributes to specify quality-of-service requests and offers.

of components across several systems, *substitutability* of components while maintaining system correctness, *extensibility* by addition of further components to the system or evolution of components themselves, and *composability* of functional and extra-functional properties. It is pointed out that the composition and substitution of components, while keeping the extra-functional properties, is one of the major challenges of component-based development. This thesis aligns with a number of approaches [33, 34, 35] that mitigate this point.

2.2 Data-centric communication

Data-centric communication [23] decouples senders and receivers of data in a system and is intended to provide real-time communication in a flexible manner. The focus changes from a direct specification of communication relations to a specification what kind of data are needed or offered, which can be enriched by attributes. The tooling or infrastructure is responsible to find an appropriate matching of communication entities and to enforce quality-of-service requests. The matching process falls back on so-called *topics*, where each topic is associated with a data type specification and a unique name. In this context, senders to a certain topic are called *publishers*, while receivers are named *subscribers*. From the view of publishers and subscribers, data is transferred from and to a *global data space*, as visualized in Fig. 2.2. A set of topic specifications is called *dictionary* and forms a global agreement on the available data semantics and formats. We assume in this thesis that communication is handled according to the data-centric paradigm, in order to be able to construct systems in a plug-and-play manner. The data-centric communication principle was standardized by the Object Management Group (OMG) with the specification of the Data Distribution Service (DDS) [36]. It includes the definition of quality-of-service attributes to control the transmission, receiving, and storage behavior. However, these attributes lack the possibility to specify and enforce strict end-to-end latencies, which will be covered in this thesis.

2.3 Model-driven engineering

Model-driven engineering [37, 38] is a methodology for software development. It utilizes models as abstract representations of systems with well-defined semantics instead of focusing directly on algorithms. Models have a limited expressiveness, but are rich enough to be applied as a design, communication and implementation tool across several stakeholders. Common is a representation as a graph with attributed and annotated edges and

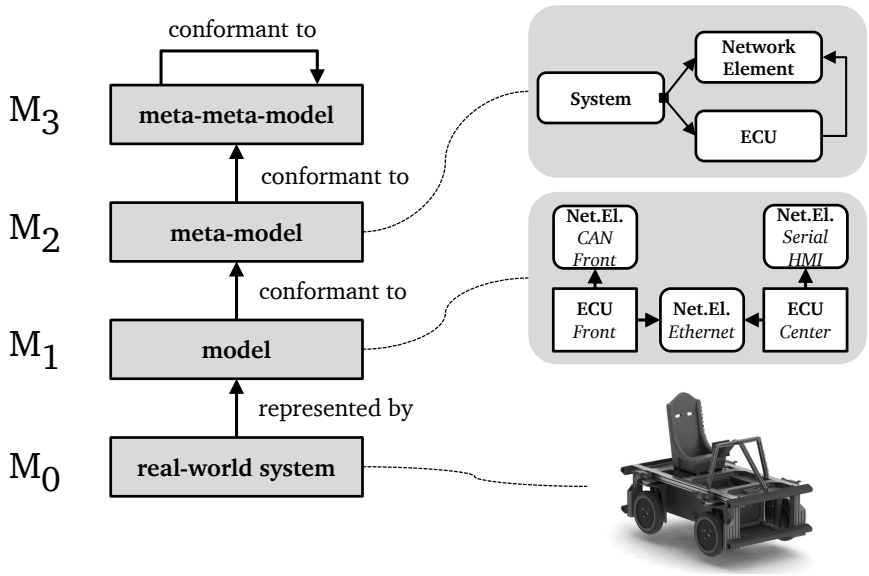


Figure 2.3: Modeling layers (adapted from [37]).

nodes. The meaning of the modeling elements has to be negotiated before the actual design phase. Several views may exist for a specific model, covering separate aspects like user-interaction, timing behavior or entity hierarchies. An approach to provide standards for model-driven engineering is the Model Driven Architecture (MDA) initiative by the OMG [39]. This standard includes specifications for the description and handling of models and associated operations for processing them. For example, an interchange format is specified to transfer models between different machines. It also focuses on a clear separation of platform-independent modeling (PIM) and platform-specific modeling (PSM). While PIM is mostly a technology-independent representation of the system functionality, PSM is mapped to a concrete technology, allowing a detailed analysis of the behavior. It is possible to generate code from a PSM to derive an executable implementation. As a third layer on top, the Computation Independent Model (CIM) offers means to describe a system in a more natural way, independent of any design decisions, e.g., without a separation of functions into logical units. Beyond the industrial implementations, the Eclipse Modeling Framework (EMF) [40] is a widely spread toolset for model-driven engineering, which is in most parts compatible with the proposed Model Driven Architecture by the OMG.

2.3.1 Modeling layers

The Model Driven Architecture defines four layers for the modeling of a system, where three layers represent an abstract view and one layer refers to the real-world system [37], see Fig. 2.3 . The four layers are: The meta-meta-model (M_3), the meta-model (M_2), the model (M_1) and the actual system (M_0). The abstraction level decreases along the layers from the meta-meta-model to the actual system. The model represents the real-word system and is conformant to the meta-model, i.e., the model can be described by means of entities of the

meta-model. The meta-model is conformant to the meta-meta-model, i.e., the meta-model can be described by elements of the meta-meta-model.

The **meta-meta-model (M_3)** is a facility to describe meta-models (also called domain-models). It is the most abstract representation of the modeling layers and is conformant to itself. A meta-meta-model can be constructed by elements the meta-meta-model itself provides. The usage of a meta-meta-model makes domain-models compatible with each other in a technical sense (but not semantically). This is a requirement for the implementation of generic tools that process models and/or domain-models. Because each element of a model or domain-model corresponds to an element of the meta-meta-model, a generic interface for the processing is provided. A widely used specification of a meta-meta-model is the Meta-Object Facility (MOF) [41] of the OMG that is part of the Model Driven Architecture initiative, which already became an international standard[1]. The MOF is divided into two parts, a reduced specification, *Essential MOF* (EMOF), and the complete specification, *Complete MOF* (CMOF). The Eclipse Modeling Framework comes with its own meta-meta-model, called Ecore. Ecore is in most parts a subset of the Essential MOF specification and compatible with it. **Meta-models (M_2)** (or domain-models) define the design space for an actual model that represents the real-world system. The meta-model is conformant to a meta-meta-model. The Eclipse Modeling Framework provides tools for the creation of meta-models based on Ecore within a graphical user interface, including automatic validation. A prominent meta-model specification is the Unified Modeling Language (UML) [42], which defines a standard for the design of software and other systems. It defines views of a system that cover various aspects: For example, use cases are deployed to define requirements, class diagrams show the hierarchy of software classes and state machines represent finite automata to model the implementation. Altogether, 14 different diagrams are defined, which cover structural and behavioral aspects of the system to model. The Unified Modeling Language provides extension points to either shape the functionality to a specific domain or to enhance it. These extensions are called profiles and several of them exist. In the automotive domain, the AUTOSAR profile for UML [43] is the most influential one, which is the base for sharing software designs between manufacturers and suppliers. Other profiles for UML include SysML [44], EAST-ADL [45] and MARTE [46]. A detailed discussion of selected meta-models is presented in Appx. A.1.1. The **model (M_1)** (also called instance) is the bottom layer of the modeling world and directly represents a **real-world system (M_0)**. Several models can exist that conform to the same meta-model, representing different real-world systems. Of course, the model is an abstraction from the real-world system and describes only particular aspects. It is in the responsibility of the domain expert to outfit the meta-model with enough features to be able to model a real-world system in a suitable way. It is not necessary that a holistic model describes a real-world system. Several descriptions, based on several meta-models, might co-exist and reference each other. For example, descriptions for energy, thermal, communication, and packaging aspects may exist.

[1] ISO/IEC 19508:204

2.3.2 Model operations

Models can be processed automatically by model operations [38]. This is an important step for the tool-supported design and refinement of models. Two important operations, *model-to-model transformation* and *model-to-text transformation*, are discussed in more detail in the following. **Model-to-model transformations** are seen as one of the most important operations involved in model-driven engineering [47]. Their purpose is to map entities of a model conforming to a meta-model to entities of a model that conforms to a different or the same meta-model. These mappings can be implemented by rules that have constraints for their activation and produce one or multiple elements in the resulting model based on one or multiple elements of the input model. It is possible to utilize multiple input and output models for one transformation. A standard for model-to-model transformations was defined by the OMG with the Query/View/Transformation (QVT) specification [48]. This specification defines a syntax, how elements of a model can be selected and mapped onto another model. The QVT specification includes two parts: A description of a declarative language, called QVT-Relation, and a description of an imperative language, called QVT-Operational. Diverse implementations for both parts of the QVT specification exist. Wide-spread examples are ATLAS [47] and QVTo [49], which are both part of the EMF. Further options of model-to-model transformation frameworks include XTend[1] and the Epsilon Transformation Language [50] that are not aligned with the QVT specification. Due to its simplicity and good usability, the model transformations in this thesis are implemented in QVTo. **Model-to-text transformations** are operations that convert elements of the modeling world into text-based representations, commonly by instantiating text templates. These text-based representations include program source code that can be utilized to create firmware images that are deployed onto hardware platforms. Model-to-text transformations render an important step for the seamless creation of software code out of the model-driven engineering process. In the ideal case, program source code can be completely generated and does not have to be reviewed or modified anymore before deployment. By the usage of code generation templates, the model does not have to be changed in order to adapt the implementation to a different hardware platform. However, for validation and verification, it might still be necessary to work directly on the code or even its compiled derivate, either manually or with automatic tools. A variety of tools exist around the EMF that support model-to-text transformations, for example Xpand[2] and its successor Xtext[3].

2.4 Time modeling, representation, and constraints

In this section, it is discussed how time can be modeled and represented for the analysis of systems. Due to various digital physical clocks in a distributed embedded system, time can be a vague term as it might lead to different interpretations. To specify constraints of a system in the time-domain, a conflict-free understanding of the meaning has to be established. Basically, we distinguish real time, digital physical clocks and logical clocks in computer-

[1] https://eclipse.org/xtend/, accessed 30-10-2015
[2] https://eclipse.org/modeling/m2t/?project=xpand, accessed 30-10-2015
[3] https://eclipse.org/Xtext/, accessed 30-10-2015

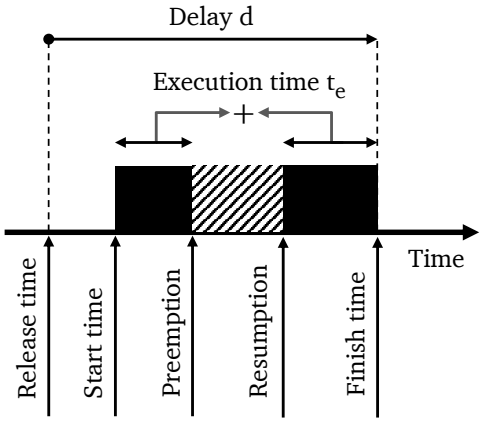


Figure 2.4: Times associated with a task execution (adapted from [22]). Of interest is usually the delay d, i.e., the time between the release of the task and the point, where the processing is finished and the results are available.

based systems [51, 52]. We refer with **real time** to a Newtonian time base, which *"assumes a globally shared absolute time, where any reference anywhere [...] will yield the same value"* [22]. The real time is monotonic, i.e., always increasing, and effects due to relative velocity and gravitational time dilatation are neglected. **Digital physical clocks** are a reproduction of the real time in computer systems. The representation is imperfect, as a digital physical clock can have an offset to the real time or be faster or slower, leading to a drift. If several clock sources are present in a system, which is usually the case in a distributed setup like within a vehicle, they are prone to clock skew problems and have to be actively synchronized in order to form a consistent time base. Depending on the implementation, digital physical clocks can jump or even go backwards, i.e., when a clock was "too fast" and has to be turned back during the synchronization process. **Logical clocks** are used to consistently order events in a distributed system. Logical clocks are not necessarily in synchronization to the real time but guarantee a consistent time base for a complete system. Protocols exist that handle the synchronization between multiple entities and avoid conflicts [53].

For this work, it is assumed that timings are given in the real time domain. Although the nodes must have a digital physical clock to keep track of time, it is assumed that no difference to the real time exists. This is a simplification, but the imperfections can be respected partly during the modeling process. For example, an incorrect clock rate may be modeled by a modification of the worst-case and best-case execution times of tasks. Problems related to jumps of clocks, i.e., the multiple emission of events or the absence of certain events, are not considered in this work.

An overview of time and constraint modeling in selected frameworks is omitted here but presented in Appx. A.1.2.

2.4.1 Timing and constraints of software components

The times involved in the processing of events by software components are visualized in Fig. 2.4 [22]. The *delay* is the difference of the *finish time* and the *release time*, which, in

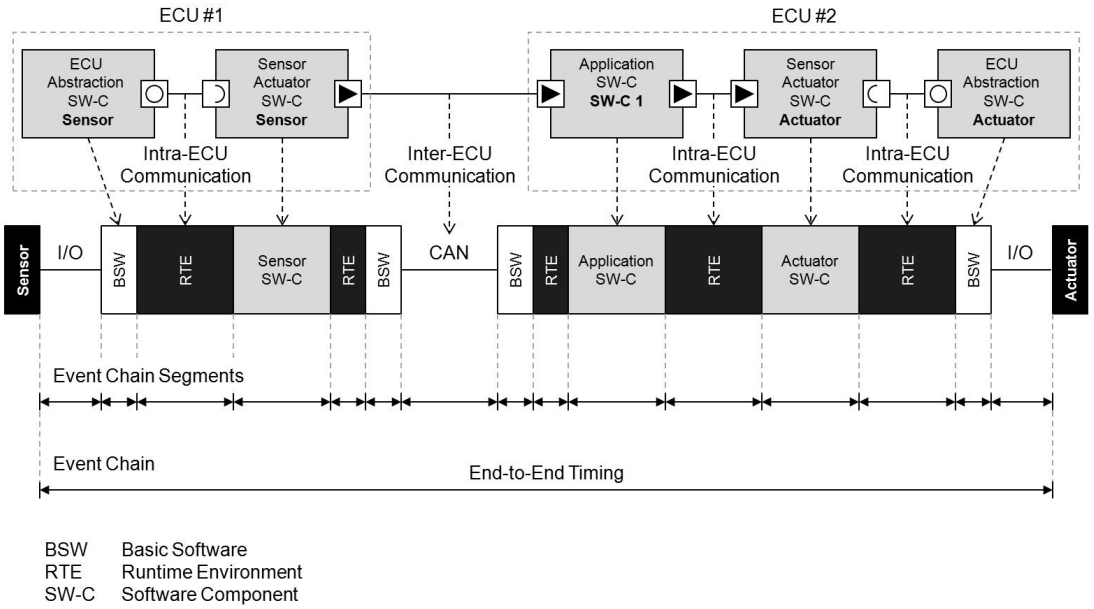


Figure 2.5: Example of an event chain and the related end-to-end timing (from AUTOSAR 4.2.1 [55]).

our case, describes the point in time an incoming event was received. Although an event arrived, processing might still be delayed due to unavailability of resources or the blocking by other events that arrived earlier. The actual time when the processing is started is called *start time*. After the start time, the component may get preempted again, for example if the execution slot is finished or a component with higher priority was triggered. The *execution time* is the overall time it needs to process an event, neglecting any interruptions by preemption or other means. In this thesis, execution time is frequently split into a lower and an upper bound, which is for example represented by the best-case execution time (BCET) and worst-case execution time (WCET) of a software component. The calculation of BCETs and WCETs is not part of this thesis, it is assumed that these values are known. Several methods and tools exist to calculate concrete values, for example [54] gives an overview.

2.4.2 Timing and constraints of event chains

Event chains contain multiple components, separated into event chain segments, which affect the timing [55]. An event chain has a start point, usually a port of a component, and an end point, usually also a port of a component or a component itself. Between the start and the end, a directed path of internal and external relations of the components has to exist in order to form a valid event chain. The typical property of interest for event chains is the maximum end-to-end delay (d_{e2e}), i.e., the maximum delay an event observes from the start point to the end point under the assumption it is processed at each involved component of the chain, see Fig. 2.5 for an example. In case of over-/under-sampling, the end-to-end delay is ambiguous. In the simplest case, it is then distinguished between

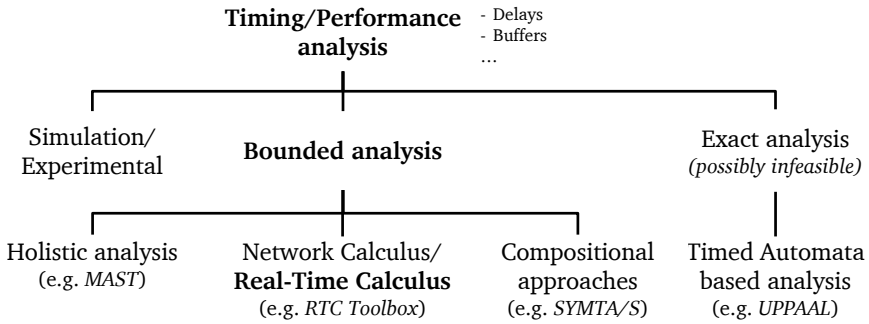


Figure 2.6: Structuring of timing analysis methods.

the response time, the time until an event causes a reaction, and the maximum age, the maximum time measured from the creation of the initial event. A more exact definition of these semantics is provided in [56]. In this thesis, we refer to the response time if not noted differently. Note that, in the case of under-sampling, events might get lost which are then not regarded for the calculation of the end-to-end delay. The concept of event chains does not match directly the loosely coupled character of data-centric systems, because data paths are not necessarily known during design time. A method is presented in Sec. 4.9 that combines both aspects for the specification of timing requirements.

2.5 Timing analysis methods

We distinguish three different kinds of analysis methods, each with different complexity and accuracy: Simulation/Experimental-based analysis, exact analysis and bounded analysis, where the latter two are formal methods. The structuring of the methods is visualized in Fig. 2.6 and their properties are summarized in Tab. 2.1.

For an **exact analysis**, the system has to be modeled in every detail. Appropriate representations are, e.g., timed automata [57] or time Petri nets [58]. Analysis tools in this area calculate all reachable states, or state classes, with the according timing parameters. Advantage of this method are the exact results, which are the outcome of the analysis process if the system is modeled in an appropriate detail. As a consequence, the calculated best- and worst-case timings match the real timings of the system. A drawback is the high computational complexity, which causes a long calculation time and may render this method infeasible [57, 59]. The scalability of this approach is limited, it is not always possible to connect models in an arbitrary way. It is further difficult to automatically find higher abstractions for the modeling to speed up the calculation process in favor of a reduced timing for analysis. **Simulation/Experimental-based analysis** is implemented frequently in the automotive industry. To check the feasibility of a system or to extract performance quantities, a system is either run in a virtual simulation [60, 61] or in the real-world with varying completeness [62, 63]. The outcome of this approach is the result of some specific instances of the system state and input signals. The extracted performance numbers may not capture all corner-cases [64], which means that the real system behavior might differ from the gained numbers, see also Fig. 2.7. It is easily possible to exchange result quality

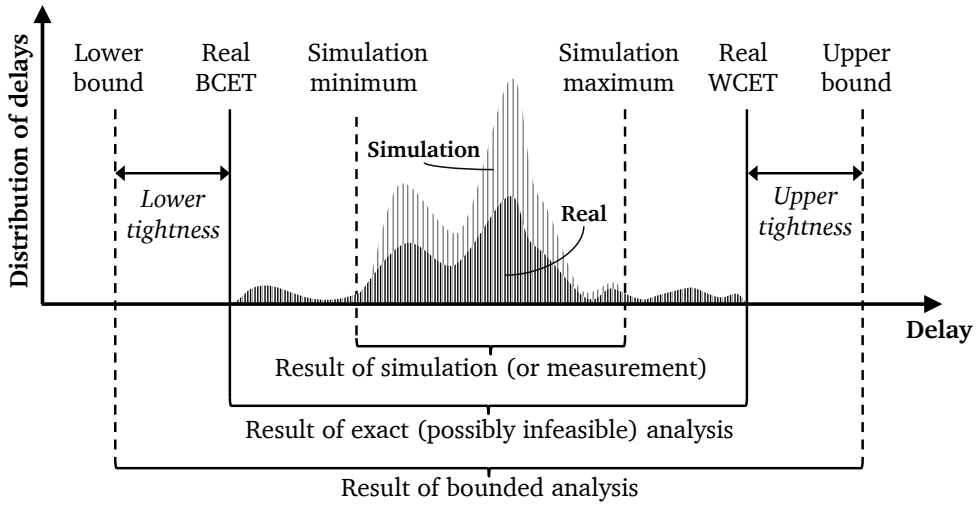


Figure 2.7: Comparison of results of bounded analysis to other analysis methods (adapted from [54]). Because the results of a simulation or measurement do not exactly capture the best-case and worst-case behavior (grey area), the goal is to approximate the exact behavior by lower and upper bounds. The exact calculation of the values is usually infeasible due to the needed computational effort. The tightness denotes the difference between the bounds and real values.

with the spent effort with this approach. The more often or longer a system is tested, the higher is the probability that analysis results are close to the real values. Composability of this approach is good, because limiting factors, in regards of analysis complexity, are not existing. **Bounded analysis** does not aim to calculate exact results for the worst and best cases. Instead, it calculates upper and lower bounds, i.e., a higher worst-case value than the real value (over-approximation) and a lower best-case value than the real value (under-approximation). One advantage is, that tightness of the results can be exchanged for computation time by a modification of the analysis process, which is further detailed in Ch. 7. Compared to an exact analysis, calculation effort can be heavily reduced [59]. Compared to simulation-based analysis, the calculated bounds are always valid. It is guaranteed that the system will not behave better than the calculated best-case and not worse than the calculated worst-case. Nevertheless, it is possible that the results of a bounded analysis indicate that a system is not runnable, but in reality it is. A comparison of the bounded analysis to other techniques is visualized in Fig. 2.7.

For this work, verification is based on a bounded analysis as it overcomes the computational complexity of an exact analysis. Simulation and experimental-based methods were not considered, because the results are not deterministic and are therefore not usable for a formal verification.

From the bounded analysis methods, an approach based on the *Real-Time Calculus (RTC)* [30] was chosen as the appropriate method, because it allows the analysis of distributed, heterogeneous, and event-based systems with hierarchical scheduling. A free implementation is available with the RTC toolbox [65], but as not all needed features were

Methodology	Exact analysis	Experimental analysis	Bounded analysis
Bound of results	Exact	Low - High	Low - High
Deterministic	Yes	No	Yes
Computation time	Very high	Low - High	Low - High
Composability	Poor	Good	Medium - Good
Trade time vs. accuracy	Limited	Yes	Yes

Table 2.1: Properties of timing analysis methods.

available for the approach of this thesis and the toolbox is closed-source, a customized tool was implemented, see Ch. 6. *Holistic analysis* methods, which are an extension of classical scheduling and response time analysis to distributed systems [66], was not considered as it limits the scope of the system to analyze to a set of technologies. An exemplary tool for the holistic analysis is the MAST framework [67], which implements a variety of techniques [68]. *Compositional-based approaches* process event streams locally with classic scheduling methods, but offer a global representation for the connection of components. Details of this concept are for example available in [69, 70]. These methods lack the ability to model hierarchical scheduling principles [71], for which reason they were not considered for the approach presented in this work. An industrial tool in this area is the SYMTA/S framework[1]. *Timed-automata based* approaches are a classic method for the performance analysis of systems [72], but are subject to the state-space explosion problem, which renders these methods unsuitable if used exclusively [73]. One prominent tool in this area is the UPPAAL framework [74]. On the other hand, timed automata-based methods can handle state-based representations, which are not thoroughly covered by the other approaches, why it is combined, e.g., with the RTC [73, 75].

The evaluation of the various methods is not consistent in the literature, especially regarding the analysis time and tightness of the results [57, 59, 73, 76, 77, 78, 79]. A short resume is, that timed automata-based methods are still regarded to be prone to the state-space explosion problem while they have potential to deliver exact results, bounded methods in comparison offer a better composability but might be overly pessimistic, depending on the concrete use case.

Another possibility to guarantee the timing behavior of systems is the construction following architectural guidelines. These systems are modularized into parts that allow an addition of components without the need for an analysis and verification step. The system is guaranteed to operate according to the desired specification regardless of the combination of installed components, as long as these components comply to certain properties. This can be achieved by a system strictly partitioned into memory blocks and timing slots, where each block and slot can be occupied by a software component. Disadvantages are the demand to conform to the predefined component properties and the limited system size on the other hand. It is also hard to guarantee correct behavior of the interactions of components, especially for multiple dependencies and long or branched event chains. Individual solutions have to be found for implementations that exceed the predefined slot sizes or memory limits. These configurations are commonly based on time-triggered frameworks

[1] http://www.symtavision.com, accessed 30-10-2015

that introduce additional disadvantages like the possibly delayed processing of high-priority messages. One example for this kind of systems is the proposed approach of the RACE project [27]. Because of the limitations, especially the limited flexibility, that approach is not suitable for the plug-and-play method proposed in this thesis.

2.6 Real-Time Calculus and Modular Performance Analysis

Modular Performance Analysis (MPA) [80] is a framework for best-case and worst-case timing analysis of heterogeneous, event- or time-triggered embedded systems, based on the mathematical background of the Real-Time Calculus (RTC) [30, 81], which itself is based on the Network Calculus (NC) [82, 83, 84] and extends its ideas to embedded systems. The basic concepts of the framework are explained in the following, because it builds the basis for the analysis approach in this work.

The basic building blocks of the Real-Time Calculus are service curves for the modeling of resources, arrival curves for the modeling of event streams and stream filters for the transformation of service and arrival curves as they are processed according to a certain strategy. Filters are abstract representations of real-world processing, communication or manipulation entities. For most stream filters, the semantics include that the incoming data is buffered in queues of infinite size if a processing is not immediately possible. Filters can be composed by a linking with data streams and remaining service can be used as an input for another filter element, enabling the modeling of hierarchical scheduling strategies. A comparison between Real-Time Calculus and Network Calculus was conducted by [85], with the result *"that the main alternative to Network Calculus (NC), Real-Time Calculus (RTC), is in fact very similar to NC"* (with respect of expressiveness and result tightness). Contrary, [80] emphasized in a previous work that Real-Time Calculus may provide tighter results, enables modular composability, and extends the modeling scope due to its definition of arrival and service curves.

Fig. 2.8 shows an example of the Real-Time Calculus framework. The modeled system consists of two processing units that are connected via a time-triggered bus and two data streams that traverse several entities of the system. Stream A is first processed by proc. unit 1, sent via the time-triggered bus, and then processed by proc. unit 2. Stream B is once processed at proc. unit 2 and traverses the system in the opposite direction, via the bus and proc. unit 1. Processing elements need resources to process data flows, these can be, for example, cycles of a processing unit or available time slots of a time-triggered bus. The service element FS in this case stands for a resource offering full service, i.e., a CPU that is completely available to the attached processing elements, while the service element TD models the availability of a time-triggered resource. Stream sources are labeled with PJD, which stands for event streams according to the period, jitter, and minimum distance model. The event stream and service models are detailed later in this chapter, see Eq. 2.5 and Eq. 2.7 with the according examples and definitions. The stream filters are of type GPC [77], which stands for greedy processing component, and FIFO [71], which stands for first-in-first-out processing component. The semantics and equations of diverse stream filters are presented in Appx. A.2. With the RTC framework, it is possible to derive the end-

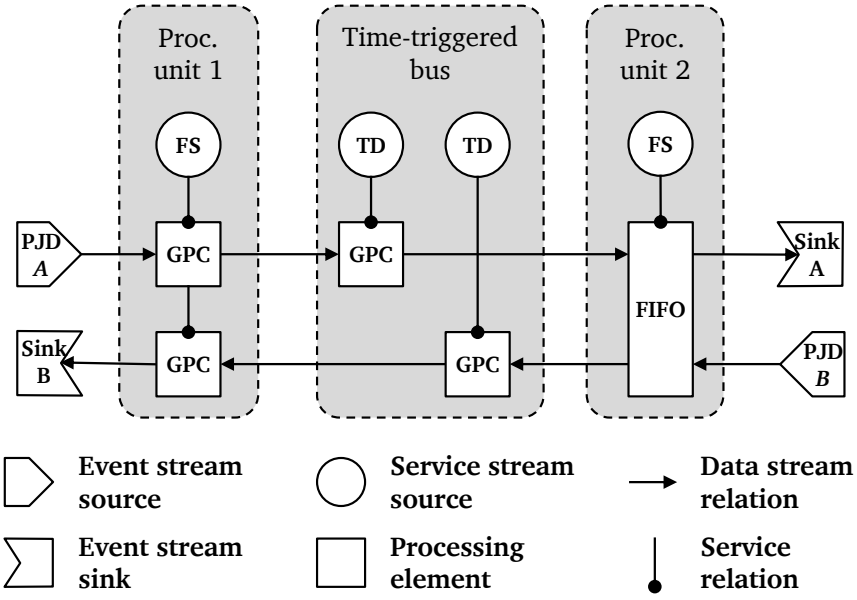


Figure 2.8: Example system model of the Real-Time Calculus framework, showing two event streams that traverse two processing units and a time-triggered bus.

to-end delays of the involved event streams. In the following, basic notions and semantics of curves and other entities of the Real-Time Calculus framework are introduced.

A **differential arrival function** abstracts properties of one particular trace of an event stream. They are an extension of the classic, cumulative arrival curves introduced by Cruz [82] to an interval-based representation. Similar, **differential service functions** capture the properties of computation or communication resources.

Definition 2.1 *A differential arrival function $R[r,s)$ describes the sum of events in a time interval $r \leq t < s$ and a differential service function $C[r,s)$ denotes the sum of resources in a time interval $r \leq t < s$, where $r,s \in \mathbb{R}$ [77].*

It follows, $R[r,r) = C[r,r) = 0$, because no events can occur or service can be available in an infinite small amount of time, and $R[r,s) \geq 0$, $C[r,s) \geq 0$. Curves to represent event streams and resources in the Real-Time Calculus fall into the class of **wide-sense increasing functions**, which are defined as:

Definition 2.2 *A function f belongs into the class of wide-sense increasing functions $\mathbb{F}$ if:*

$$\begin{aligned} f(r) \leq f(s) \;\forall\; r < s;\; r,s \in \mathbb{R} \\ \text{and } f(r) = 0 \;\forall\; r \leq 0 \end{aligned} \tag{2.1}$$

The mathematics of the Real-Time Calculus rely on the **min-plus algebra**. The min-plus algebra forms a Dioid $(\mathbb{R} \cup \{+\infty\}, \wedge, +)$, where $\wedge$ is the infimum or minimum operator. For details about the definition of the Dioid and involved axioms, refer to [84, 86]. The min-plus convolution operator $(\otimes)$ is equal to the convolution in traditional algebra, but addition becomes the infimum and multiplication becomes the addition.

Definition 2.3 *Min-plus convolution ($\otimes$) and deconvolution ($\oslash$) of two functions $f, g \in \mathbb{F}$:*

$$\begin{aligned}(f \otimes g)(\Delta) &= \inf_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \\ (f \oslash g)(\Delta) &= \sup_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}\end{aligned} \tag{2.2}$$

Similar to the min-plus algebra, the **max-plus algebra** forms a Dioid $(\mathbb{R} \cup \{-\infty\}, \vee, +)$, where $\vee$ is the supremum or maximum operator.

Definition 2.4 *Max-plus convolution ($\overline{\otimes}$) and deconvolution ($\overline{\oslash}$) for two functions $f, g \in \mathbb{F}$:*

$$\begin{aligned}(f \,\overline{\otimes}\, g)(\Delta) &= \sup_{0 \le \lambda \le \Delta} \{f(\Delta - \lambda) + g(\lambda)\} \\ (f \,\overline{\oslash}\, g)(\Delta) &= \inf_{\lambda \ge 0} \{f(\Delta + \lambda) - g(\lambda)\}\end{aligned} \tag{2.3}$$

The convolution and deconvolution operators of the min-plus and max-plus algebra are the backbone for filter operations to transform event streams and resource descriptions according to a certain processing strategy.

Arrival curves describe classes of event streams by bounding differential arrival functions. A pair of arrival curves gives an upper and lower bound for the occurrence of events in an interval at any point in time, for a whole class of traces. They can be either constructed if the parameters of an event source are known or extracted from an event trace.

Definition 2.5 *Let $\alpha(\Delta)$ denote a pair of arrival curves, where Δ is a timing interval:*

$$\begin{aligned}\alpha(\Delta) &= [\alpha^u(\Delta), \alpha^l(\Delta)] \\ \text{with } &\alpha^u(\Delta), \alpha^l(\Delta) \in \mathbb{F}\end{aligned} \tag{2.4}$$

The superscripts ^{u} and ^{l} in this context stand for upper and lower arrival curves. The curves bound the differential arrival functions as follows [77]:

$$\alpha^l(\Delta) \le R[s, s+\Delta) \le \alpha^u(\Delta) \quad \forall\, s \in \mathbb{R}, \Delta \in \mathbb{R}^{\ge 0} \tag{2.5}$$

Commonly, the tightest bound of the trace is of interest, which is [87]:

$$\alpha^l(\Delta) = \inf_{s \in \mathbb{R}} R[s, s+\Delta] \qquad \alpha^u(\Delta) = \sup_{s \in \mathbb{R}} R[s, s+\Delta] \tag{2.6}$$

The construction of tight upper and lower arrival curves from an event trace can be interpreted as follows: A window of fixed size is moved over the trace from negative to positive infinity. At any position, the number of events inside that window are counted. The minimum and maximum number of events at any point in time for a specific window size form one point of the lower or upper arrival curves, respectively. Upper arrival curves are subadditive [87]. If at maximum one event occurs during one time unit, then no more than two events can occur during two time units. Lower arrival curves are superadditive. If a

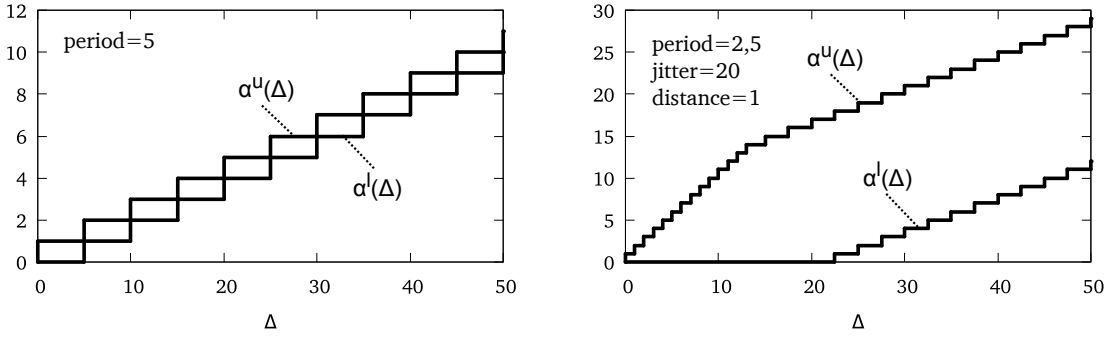


Figure 2.9: Examples of arrival curves. On the left side based on a completely periodic event stream model, on the right side based on an event stream model with jitter and a minimum distance between the events. The part with the higher slope of the upper curve on the right side reflects the bursty behavior caused by the jitter.

minimum of two events occur during one time unit, then at least four events have to occur within two time units. These *implicit properties* hold for all interval combinations:

$$\begin{aligned} \alpha^u(\Delta_1 + \Delta_2) &\leq \alpha^u(\Delta_1) + \alpha^u(\Delta_2) \qquad \text{(subadditivity)} \\ \alpha^l(\Delta_1 + \Delta_2) &\geq \alpha^l(\Delta_1) + \alpha^l(\Delta_2) \qquad \text{(superadditivity)} \end{aligned} \tag{2.7}$$

Because lower arrival curves define the minimum number of events in a certain interval and upper arrival curves define the maximum, the following *explicit property* holds:

$$\alpha^l(\Delta) \leq \alpha^u(\Delta) \tag{2.8}$$

Another explicit property of arrival curves is the dependency between the upper and lower parts. The upper curve must not contradict to the description of the lower part and vice versa. For example, if the minimum amount of events is defined to be two within two time units, then also the maximum amount of events has to grow with at least two events per two time units. As analyzed and pointed out by [88], the upper and lower parts of arrival curves have to fulfill the following equations in order to be *causal*:

$$\begin{aligned} \alpha^l(\Delta) &= \alpha^l(\Delta) \oslash \alpha^u(\Delta) \\ \alpha^u(\Delta) &= \alpha^u(\Delta) \mathbin{\overline{\oslash}} \alpha^l(\Delta) \end{aligned} \tag{2.9}$$

Arrival curves are always causal if extracted from event streams. But approximations, rounding errors and transformations to other representations may lead to the construction of non-causal curves, i.e., an arrival curve that describes a stream of events that is not possible in reality. To mitigate this issue, methods were presented in [88, 89] to transform any pair of curves into a causal representation.

Selected types of arrival curves. Fig. 2.9 shows two examples for arrival curves that were generated according to the period, jitter, and minimum distance (PJD) model. Within this model, the events occur periodically according to the period parameter P. In addition, events can deviate from the strictly periodic position by a jitter, denoted as J. The jitter

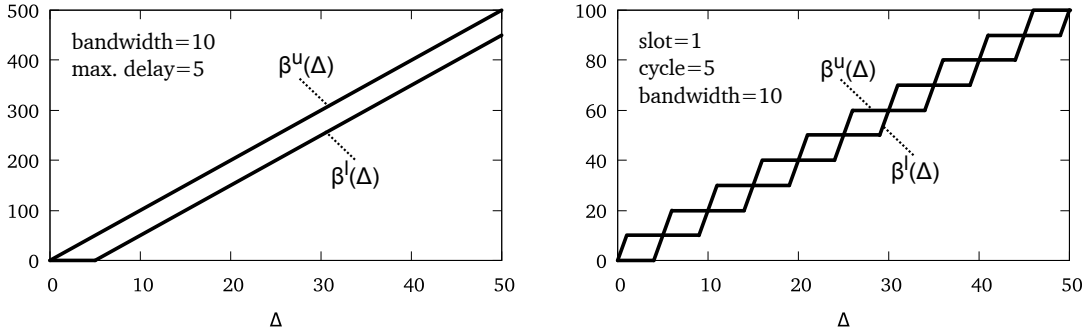


Figure 2.10: Examples of service curves. On the left side the model of a resource with a maximum delay (MD) and on the right the model of a resource according to a time-division multiple access (TD) schemata.

can be larger than the period, leading to overlapping ranges of the possible occurrences of events. The third parameter, D, defines the minimum distance between any two events, also in the overlapping areas caused by the jitter.

Definition 2.6 *Tight arrival curves according to the PJD model can be directly constructed out from the parameters [80]:*

$$\alpha^u(\Delta) = \min\left\{\left\lceil\frac{\Delta+J}{P}\right\rceil, \left\lceil\frac{\Delta}{D}\right\rceil\right\} \qquad \alpha^l(\Delta) = \left\lfloor\frac{\Delta-J}{P}\right\rfloor \tag{2.10}$$

Service curves describe classes of resources by bounding differential service functions. Similar to arrival curves, the modeling is conducted in an interval-based domain. A pair of service curves models the upper and lower amount of resources, which are available during a certain time interval.

Definition 2.7 *Let $\beta(\Delta)$ denote a pair of service curves:*

$$\begin{aligned}\beta(\Delta) &= [\beta^u(\Delta), \beta^l(\Delta)] \\ \text{with } &\beta^u(\Delta), \beta^l(\Delta) \in \mathbb{F}\end{aligned} \tag{2.11}$$

The properties of service curves are similar to arrival curves, except the unit: While arrival curves represent the number of events, a service curve represents resources like available cycles of a processing unit or available data rate of a network link.

Selected types of service curves. Two examples for service curves are shown in Fig. 2.10, a formal definition of selected curves is presented in Tab. 2.2. One shown example is a maximum delay (MD) service curve with the parameters B for available bandwidth and D for maximum delay. This curve models a resource with a fixed bandwidth that is available latest after a delay of D. The other example is a curve of a time division multiple access (TD) resource [90] with the parameters B, C, and S, where S is the duration of the active time slot, C is the cycle time, and B is the available bandwidth during the time slot. Intuitively, the curves are constructed as follows: For the upper service curve (β^u), assume that the time slot just started when looking at the system, which leads to an increase of the

Resource type	Lower service curve $\beta^l(\Delta)$	Upper service curve $\beta^u(\Delta)$
FS (full service)	$B \cdot \Delta$	$B \cdot \Delta$
MD (max. delay)	$\max\{B \cdot (\Delta - D), 0\}$	$B \cdot \Delta$
TD (time-division) [90]	$B \cdot \max\{\lfloor \frac{\Delta}{C} \rfloor \cdot S,$ $\Delta - \lceil \frac{\Delta}{C} \rceil \cdot (C-S)\}$	$B \cdot \min\{\lceil \frac{\Delta}{C} \rceil \cdot S,$ $\Delta - \lfloor \frac{\Delta}{C} \rfloor \cdot (C-S)\}$

Table 2.2: Definitions of selected types of service curves. Parameters denote: B – bandwidth, D – delay, C – cycle length, and S – slot length.

sum of the available resources with bandwidth B up to the slot length S. Afterwards, the available resources do not change for $C-S$ time units, because the slot is inactive. Then the process repeats. For the lower service curve (β^l) the process is equal but we assume that we start looking at the system when the active slot has just passed. Therefore the sum of available resources stays at zero for $C-S$ time units, before ascending with a rate of B for the slot time S. Not shown in the figure is a curve modeling a resource with full service (FS), which provides resources at a constant bandwidth without delay. A single parameter for the available bandwidth is sufficient for the modeling. The upper and lower curves of such resources are equal.

Workload curves [91] relate event-based representations of arrival curves with resource-based representations of service curves. For example, an event might consume a certain number of processing cycles or demand a certain packet size if transferred via a network. As arrival and service curves do not contain any phase information, i.e., an event is not directly identifiable, only relative specifications for the transformation can be made. Those transformations are expressed by workload curves that form an upper ($\gamma(k)^u$) and a lower ($\gamma(k)^l$) bound for the resource demand of k consecutive events.

Definition 2.8 *Workload relations [64] of a pair of workload curves $\gamma(k)^l, \gamma(k)^u \in \mathbb{F}$:*

$$\begin{aligned} \widehat{\alpha}^l(\Delta) &= \gamma^l(\alpha^l(\Delta)) \qquad \widehat{\alpha}^u(\Delta) = \gamma^u(\alpha^u(\Delta)) \\ \widehat{\beta}^l(\Delta) &= \gamma^{u-1}(\beta^l(\Delta)) \quad \widehat{\beta}^u(\Delta) = \gamma^{l-1}(\beta^u(\Delta)) \end{aligned} \tag{2.12}$$

$$\textit{with} \qquad \gamma^{u-1}(r) = \sup\{e : \gamma^u(e) \leq r\} \qquad \gamma^{l-1}(r) = \inf\{e : \gamma^l(e) \geq r\} \tag{2.13}$$

Where α are event-based and $\widehat{\alpha}$ are resource-based arrival curves, and β are resource-based and $\widehat{\beta}$ are event-based service curves (the hat symbol ($\widehat{\ }$) marks a transformation into the "non-standard" representation.) In the simplest case, all event instances have a constant factor for the transformation to the resource-based representation. For example, all events may have an equal worst-case and best-case execution time (WCET/BCET). Then, the workload curves degrade to a linear function and the relations can be expressed by:

$$\begin{aligned} \widehat{\alpha}^l(\Delta) &= \alpha^l(\Delta) \cdot \text{BCET} \qquad \widehat{\alpha}^u(\Delta) = \alpha^u(\Delta) \cdot \text{WCET} \\ \widehat{\beta}^l(\Delta) &= \beta^l(\Delta) \cdot \frac{1}{\text{WCET}} \quad \widehat{\beta}^u(\Delta) = \beta^u(\Delta) \cdot \frac{1}{\text{BCET}} \end{aligned} \tag{2.14}$$

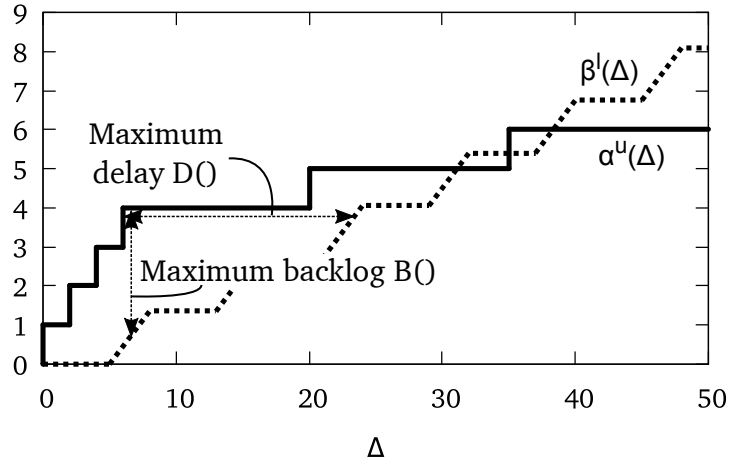


Figure 2.11: Calculation of maximum delay and backlog for greedy processing components.

The performance parameters of interest include the **maximum delay** d an event experiences while traversing a processing entity and the **maximum backlog** b, which is the maximum number of events stored in the input buffer of the processing entity at any point in time. In case of a greedy processing component (GPC), these bounds are derived as follows [92]:

$$b \leq B(\alpha^u_{in}, \beta^l_{in}) \qquad d \leq D(\alpha^u_{in}, \beta^l_{in}) \tag{2.15}$$

Definition 2.9 *The backlog-bound function $B()$ and the delay-bound function $D()$ are defined as [92]:*

$$\begin{aligned} B(\alpha^u, \beta^l) &= \sup_{\lambda \geq 0} \left\{ \alpha^u(\lambda) - \beta^l(\lambda) \right\} \\ D(\alpha^u, \beta^l) &= \sup_{\Delta \geq 0} \left\{ \inf\{\tau \geq 0 : \alpha^u(\Delta) \leq \beta^l(\Delta + \tau)\} \right\} \end{aligned} \tag{2.16}$$

The calculation of maximum delay and backlog depends on the concrete strategy of the processing entity and is given for several examples in Appx. A.2. Service and arrival curves have to be expressed in the same units for these equations. The backlog-bound function calculates the maximum vertical distance between its parameters, while the delay-bound function calculates the maximum horizontal distance, see also Fig. 2.11 for clarification.

Stream filters model the behavior of processing or communication entities in the real system that processes event flows under usage of resources according to a certain strategy. Stream filters are composable and form the basis for the system analysis with the Real-Time Calculus, a visualization is presented in Fig. 2.12.

Definition 2.10 *A stream filter is a function f_T that maps vectors of incoming service curves $\overrightarrow{\beta_{in}}$ and incoming arrival curves $\overrightarrow{\alpha_{in}}$ to vectors of outgoing service curves $\overrightarrow{\beta_{out}}$ and outgoing arrival curves $\overrightarrow{\alpha_{out}}$ according to its type T and parameter set $\mathbb{P}$. In addition, a result set $\mathbb{E}$ is obtained that captures performance properties:*

$$(\overrightarrow{\alpha_{out}}, \overrightarrow{\beta_{out}}, \mathbb{E}) = f_T(\overrightarrow{\alpha_{in}}, \overrightarrow{\beta_{in}}, \mathbb{P}) \tag{2.17}$$

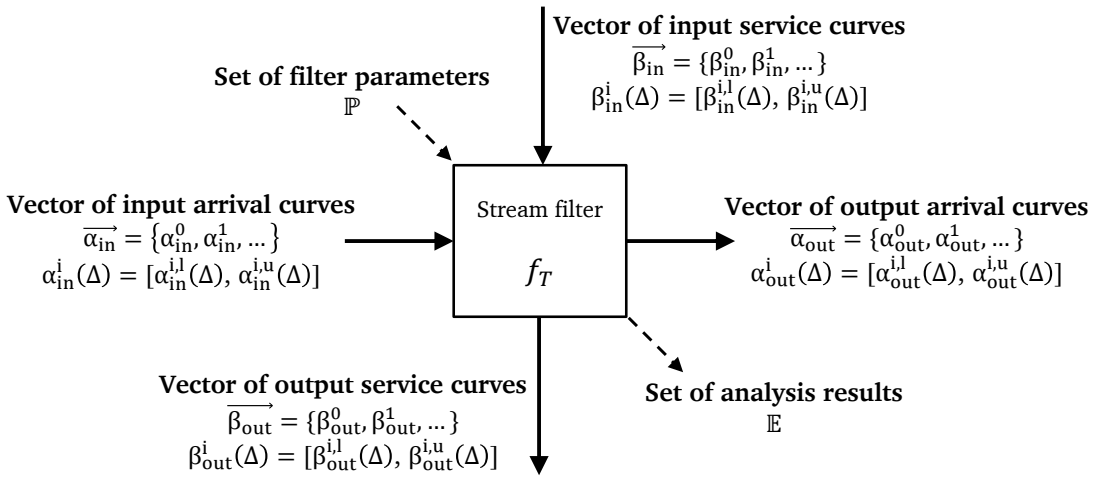


Figure 2.12: Stream filter element for system analysis with the Real-Time Calculus that transforms arrival and service curves according to a processing strategy, e.g., greedy or first-in first-out processing. Stream filters are an abstract representation of real processing entities and can be composed with each other by linking the arrival and service relations, forming a complete model of a system.

The quantity of curves in the vectors $\overrightarrow{\alpha}$, $\overrightarrow{\beta}$ depend on the type T of the stream filter and its configuration, the result $\mathbb{E}$ usually includes information about scheduling feasibility, delays, and buffer demands for the individual event streams. $\mathbb{P}$ normally contains data for the transformation of service-based streams to event-based streams and vice-versa.

The exact semantics and equations of stream filters are presented in Appx. A.2 for the greedy processing component (GPC), fixed-priority non-preemptive component (FPNP), first-in-first-out processing component (FIFO), OR component (OR), AND component (AND) and bounded greedy processing component (BGPC). The transfer functions are no contribution of this thesis, but were collected from various sources [71, 77, 92, 93].

2.7 Modeling and verification background summary

This chapter introduced the basic background needed for the following developments and discussions of this work. Component-based development, data-centric communication, and model-driven engineering were introduced as those are a prerequisite for the system-wide plug-and-play approach. Because this work focuses on the verification of constraints related to time, the different notions and semantics of timing definitions and properties were listed. A classification of various analysis methods was provided and completed with an introduction of the background on the Real-Time Calculus, which is the foundation for the developed timing analysis and verification tool as part of this thesis.

Chapter 3

Combining plug-and-play and timing guarantees

This chapter describes the proposed approach for the combination of plug-and-play behavior with the guarantee of timing requirements. It highlights requirements that led to the approach by an analysis of scenarios for a reconfiguration of the vehicle's setup, gives an overview of the actual approach and its steps, and introduces an example, which is the base for further refinements of the method in the next chapters.

3.1 Requirements for a flexible verification approach

For a better understanding, different scenarios are analyzed in this section. They reflect varying requirements and resulting strategies for the reconfiguration of a vehicle. Afterwards, the demand for a flexible verification approach is concretized, which is able to give reliable results for the scenarios.

3.1.1 Verification scenarios

As a vehicle has non-functional requirements such as real-time and safety constraints, the plug-and-play process has to consider those. We will focus on the timing requirements, i.e., the guarantee that all data in the system is delivered from the sender to the receiver within a specific time bound or according to a particular pattern. Depending on the point in the lifetime and current state of a vehicle, several verification scenarios are distinguished, see also Tab. 3.1.

Scenario 1: Verification during design time. Over the lifetime of a vehicle, a set of functions exists that will never change and are shared by many vehicle series. This includes the basic infrastructure and systems like the braking system, which is unlikely to be modified or extended because of its deep integration. Verification of these components is conducted by the manufacturer, using all necessary resources. In this scenario, a system designer benefits from a flexible approach for the analysis by the possibility to explore the performance of design choices with varying approximation levels. In the end, a tight result of the analysis is demanded in order to not waste system resources because of an overly pessimistic dimensioning.

Scenario	1	2	3	4
Point of reconfiguration	Design time	Customization before delivery	Customization after sale	In operation
Desired analysis tightness	Tight	Tight	Medium	Loose
Allowed calculation time	Long	Long - Medium	Medium - Short	Very short
No. of changed functions	Many	Medium	Few	Medium
Addition of safety functions	Yes	Yes	Yes	No
Scope	Many series	One vehicle	One vehicle	One vehicle
Lifetime of functions	Complete vehicle life	Complete vehicle life	Complete vehicle life	One journey

Table 3.1: Scenarios for the reconfiguration of a vehicle

Scenario 2: Verification of customization before delivery. Depending on the wishes of a costumer, the vehicle is customized before it is handed out. This may include the addition of sensors, screens, or other equipment. An integrated and flexible verification approach enables an individual alignment of resources according to a specific configuration of the vehicle. Requirements on the analysis time of this process are moderate; it should not delay delivery for too long to avoid additional costs. Ultimately, tight bounds are to be derived, which mark as much resources as possible available for further extensions. Compared to the previous scenario, verification is conducted on a per-vehicle basis.

Scenario 3: Verification of customization after sale. Whenever the costumer decides to change the function set of the vehicle after sale, two different situations are possible: Either the functionality of the vehicle is changed by the costumer directly (e.g., software download) or at a workshop (e.g., installation of an ultrasonic sensor). In both cases the person conducting the change cannot be regarded as an expert of the electric and electronic infrastructure of the vehicle. Therefore, the system should support the desired changes by an automatic integration and verification. Constraints on analysis time are strict, because the personal in the workshop as well as the costumer do not want to wait too long for the system reconfiguration. The lifetime of the changed functions usually extend to the complete vehicle life.

Scenario 4: Verification during operation. The system configuration may change while the vehicle is in operation, i.e., while it is driving or persons are in it. The lifespan of the reconfiguration in this scenario is bounded as it covers one journey, which is the range the first person enters the vehicle and the last one exits it again. For example, a mobile phone might connect to the system or a passenger starts an audio/video stream to a screen for entertainment. These kind of requests should be answered almost immediately. The user experience suffers if the decision process takes too long. However, it is valid for the system to deny certain requests if it is out of resources. Compared to the other scenarios, a change of safety-relevant functions is not permitted.

3.1.2 Properties of a flexible verification approach

The scenarios show that, depending on the vehicle state and point in the lifetime, varying constraints for plug-and-play and verification have to be fulfilled: Depending on the scenario, it can be necessary to reduce the *tightness* of the bounds in favor of a reduced *analysis time*. A solution should be able to trade analysis time for the exactness of the results while still guaranteeing a valid solution. *Lifetime and scope* of functions depend heavily on their types. It is very likely that basic functions, e.g., the control of the braking system, will stay constant over several series with the same platform. These systems are designed once and hence the time spent for analyzing them is allowed to be very long. Customizations before delivery are also usually installed once, e.g., a navigation system, and remain in the car for its complete lifetime. Compared to those, devices which are connected during the operation are relevant for one journey only. Those devices include smartphones or video streams within the infotainment system. The approach should allow the *extension* of the function set after the point of sale to enable further customization. Depending on the functionality, various *timing constraints* are required. If an automatic verification system is utilized, it is important that it understands these constraints and checks if the runtime environment and hardware platforms can accommodate the software components according to the specific requirements. This should happen in a *system-wide* fashion, i.e., considering the already existing components and infrastructure of the system. Further, the approach should be *homogeneous*, which means it can be used during the design and operation time of a vehicle to effortlessly transfer and reuse analysis results. In the following, the system-wide plug-and-play approach is developed that covers these points and offers a flexible framework during the design and operation time of a vehicle.

3.2 Method for system-wide plug-and-play

This section introduces the system-wide plug-and-play approach, which combines an automatic integration process of vehicle functions with the guarantee of global timing requirements.

3.2.1 Plug-and-play process and artifacts

The plug-and-play process is based on the *model-at-runtime* [94] principle, which utilizes models at the design time and also at runtime to enable dynamic adaptation. The process itself and involved artifacts are sketched in Fig. 3.1. The system state is a combination of the current configuration of a system, captured in a system model, and a snapshot of the analysis results, if available. New functionality can be added to the system by a *feature package*, consisting of a model describing the feature's functionality and requirements, the runnable code of the feature and the hardware items that are packed with it. Once a feature is added to the system, the plug-and-play process is triggered. To check if all timing requirements of the newly added feature are met, the feature model and the current system model are combined in a model-to-model transformation step, yielding a combined model. The combined model includes all the information of the current configuration of the system plus the information from the newly added feature. With an additional model-to-model transformation step, the combined model is transferred into a representation suitable for

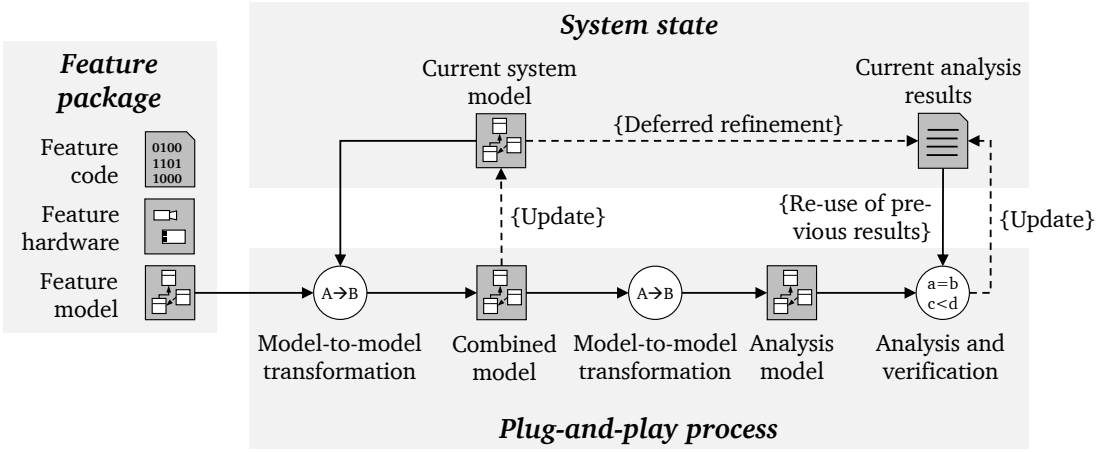


Figure 3.1: System-wide plug-and-play process and involved artifacts. Additional hardware and software is added via *feature packages*, which are automatically integrated and verified. The process is based on models that capture the technical and logical behavior, including a description of requirements. Solid lines reference the automatic verification process, while dashed lines indicate actions and data flows after a specific modification was accepted for integration.

analysis. This model is then analyzed and it is verified that all constraints are met and the feature can be safely integrated, which involves an update of the current system model. Results from the previous analysis are used to speed up the process and are refreshed according to the new results. As the tightness of the analysis step can be parameterized in our approach, a deferred refinement of the analysis results is possible. For example, a quick decision can be made with an analysis based on a rough approximation with loose tightness. Later on, after the decision has already been made and resources of the system are available, tightness of the results can be increased to allow a more exact representation of the system's performance. This approach extends concepts concerned about an integrated verification possibility for component-based systems, e.g., [33, 34, 35], by providing plug-and-play abilities and an adaptive verification, suitable for the design and operation phases of vehicles. On the other side, it extends concepts focusing on the plug-and-play ability, e.g., [94, 95, 96, 23], by an integrated, system-wide, and formal verification capability.

3.2.2 Development and adaption timeline

The development and adaption timeline of the proposed method changes compared to the traditional process, which is shown in Fig. 3.2. In the traditional process, a feature set of a vehicle is unchangeable after the design and implementation phase. Verification is based on a superset variant of the vehicle, which combines all possible features. In this hypothetical variant, excluding feature combinations might be considered that are actually not compatible with each other. The functionality of each shipped variant of the vehicle forms an according subset. After sale, adaptivity of the vehicle is limited, because only features can be added that were previously defined in the design and implementation phase.

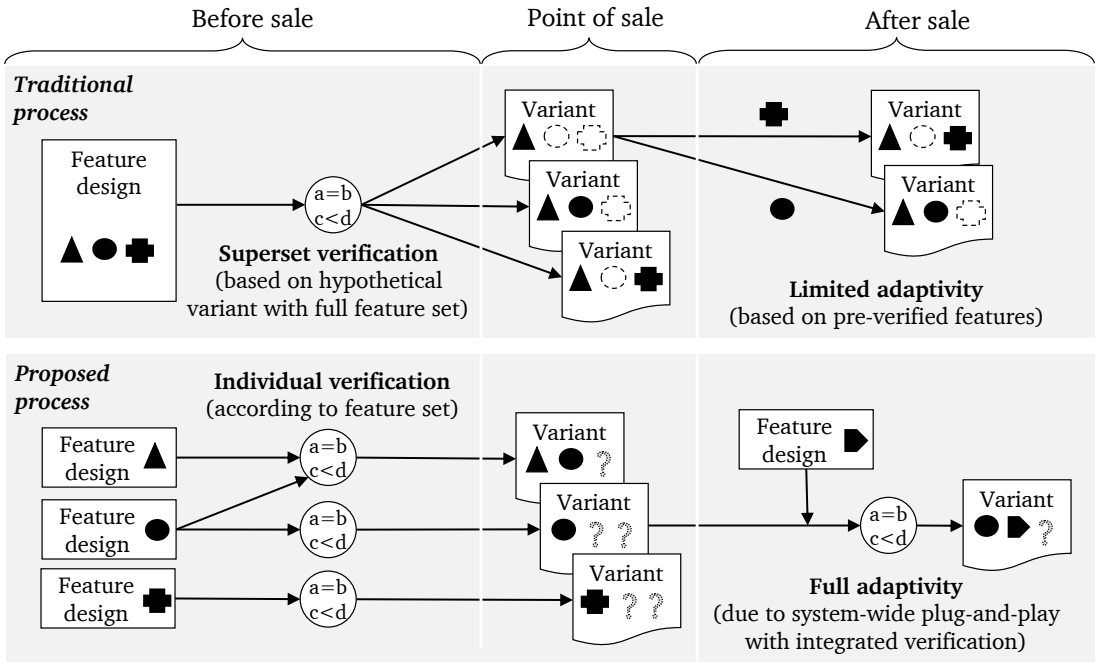


Figure 3.2: Development and adaption timeline. In the traditional process, a configuration with all possible functions is verified and subsets of it are sold. A modification after sale is hardly possible. In the proposed process, only the relevant set of functions is verified, possibly allowing the choice of a less powerful execution platform. Further, addition of features is possible after sale via an integrated verification process.

The proposed process differs from the traditional approach. Design and implementation phases are not strictly coupled with the resulting set of features of a vehicle. Especially, possible variants are not fixed to predefined sets of features during the production phase. To achieve this, each variant is verified individually, which is possible due to a deep integration of the verification process within the development and production process. Each variant – or even each vehicle – can be individually equipped with features this way and resources for processing and communication can be trimmed to the actual needs. Additional features can be easily integrated as they become available, even after the initial production phase started. Furthermore, full adaptivity is achieved due to a system-wide verification process, which can be triggered even after sale. This enables the addition of features that were unavailable during the development or production of a vehicle. The approach relies on a possibility to easily extend or modify the system capabilities in the sense of aggregates, software, resources, and communication to enable the integration of previously unknown functionality. This can be achieved by a centralized system architecture as the integration base, see Sec. 1.2.2.

3.2.3 Plug-and-play phases

The phases of the proposed plug-and-play process are shown in Fig. 3.3. The representation is a refinement of the steps already introduced in Fig. 3.1 with a focus on possible points

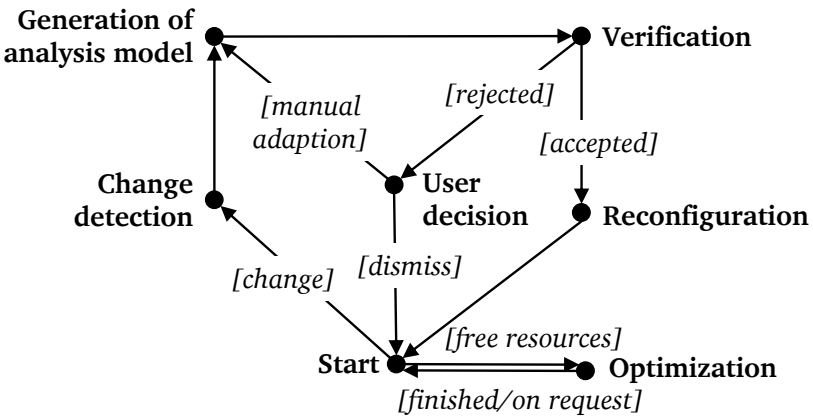


Figure 3.3: Phases of the system-wide plug-and-play approach.

for the interaction with a user and derived actions while omitting details of the involved artifacts. The proposed approach shares similarities with the concept of the CHROMOSOME middleware [95], where basically a "Plug" (detection and reconfiguration) and a "Play" (execution) phase are distinguished. The "Plug" phase is extended in our approach by an integrated verification step to formally guarantee the timing behavior of the system.

Phase 0: Start. The start (or idle) state is the initial phase of the plug-and-play process. If no changes of the system are conducted and no deferred work items have to be processed, the system will stay in this state. *Phase 1: Detection.* After a change of the system was triggered, either by an addition, removal, or change of a feature, a detection of the changed set of features is performed. All relevant information is collected that is necessary for the verification process. *Phase 2: Generation of the analysis model.* After all available information is collected, it is combined and an analysis model is derived. This model directly reflects the physical configuration of the system, including transmitted messages, instances of the software components, execution times, processor speeds of the electronic control units etc. This phase mainly consists of a mapping of the logical system representation to the physical entities considering the relations with each other. *Phase 3: Analysis and verification.* Based on the concrete model, analysis and verification are conducted. The feasibility of the setup is checked and end-to-end timings, resource utilizations and event patterns are calculated. Once these metrics are available, they are verified against the constraints of the features. If all constraints are met, the setup is considered as feasible and the reconfiguration is triggered. If at least one constraint is not met, the verification failed and the user is informed about the problem. In that case, no immediate changes are made to the system. *Phase 4: Reconfiguration.* If the verification of the system yields that the set of features is feasible, the reconfiguration of the system starts. This phase includes the configuration of the communication system, network infrastructure, aggregates, and electronic control units. The routes for the messages are adjusted, schedules are updated and the necessary software components are installed on the according platforms. After everything is setup, the new configuration is executed. All necessary information to apply the changes is available in the system model. The analysis results and system model are updated to reflect the current

configuration. After this phase completed, the plug-and-play process is finished. *Phase 3a: User decision.* If the verification phase fails because some constraints are not met, the user is asked how to proceed. In that case the user can either chose to disable features in order to free up resources and re-trigger the verification process, or the user can dismiss the changed setup, which means that a reconfiguration of the system is not applied and the system remains in its configuration. Although the setup is not feasible at that point, it can become feasible by an addition of resources (memory, processing power) later on. In that case, the plug-and-play process is triggered again and the verification is based on the new setup with the added capabilities. *Phase 0a: Optimization.* The verification process can be approximated, which still leads to valid bounds but with a looser tightness compared to the non-approximated case. If a decision was based on such an approximation, the tightness is later increased in the optimization step. The advantage is, that the utilization of the system is captured more exactly and therefore further features added to the system are more likely to be accepted. Whenever a change is triggered or the system has no free resources available, this state is left again. It can be regarded as a background task to optimize the analysis results. This step does not necessarily have to be conducted on the same resources as the approximated verification process. For example, it could be transferred from the infrastructure of a vehicle to external servers, where it is optimized.

3.2.4 System-wide and local plug-and-play principles

The method presented in this thesis is based on the system-wide plug-and-play principle that considers relations of multiple functions across several execution platforms and their dependencies for the plug-and-play process. In the following, a comparison between the system-wide and local plug-and-play principle is conducted. These are also presented in Fig. 3.4, as two possible integration variants beyond a manual and a pre-defined approach.

The idea of **system-wide plug-and-play** is the consideration of a whole system for the plug-and-play process, based on a model-driven and component-based development process with data-centric communication semantics. The approach applies an end-to-end analysis of event chains considering the mutual influence of functions of the system. For example, a collision avoidance feature, which is integrated to the existing vehicle infrastructure, usually consists of and interacts with several components: A camera may get installed that sends its data to a software component on a (already existing) central processing unit that in turn distributes a breaking command to the brake actuators. In this example, it is not enough to just look at the pairwise interaction of the involved components (sensor – processing unit, and processing unit – brake actuators). Instead, the important characteristic of this example is the involved end-to-end latency between the image capture of the camera and the execution of the braking command by the actuators. The system-wide plug-and-play approach always considers the complete processing chain – from the sensors to the actuators, including all processing stages in between and possible data dependencies – to decide if a certain setup is feasible in operation. In contrast to system-wide plug-and-play, **local plug-and-play methods** can appear as a client-server setup or in a contract-based manner. In the client-server scenario, a plug-and-play entity of the system is orchestrated by a master entity that manages the available resources and handles the configuration of those. The Universal Serial Bus (USB) [97] can be mentioned as a prominent example for

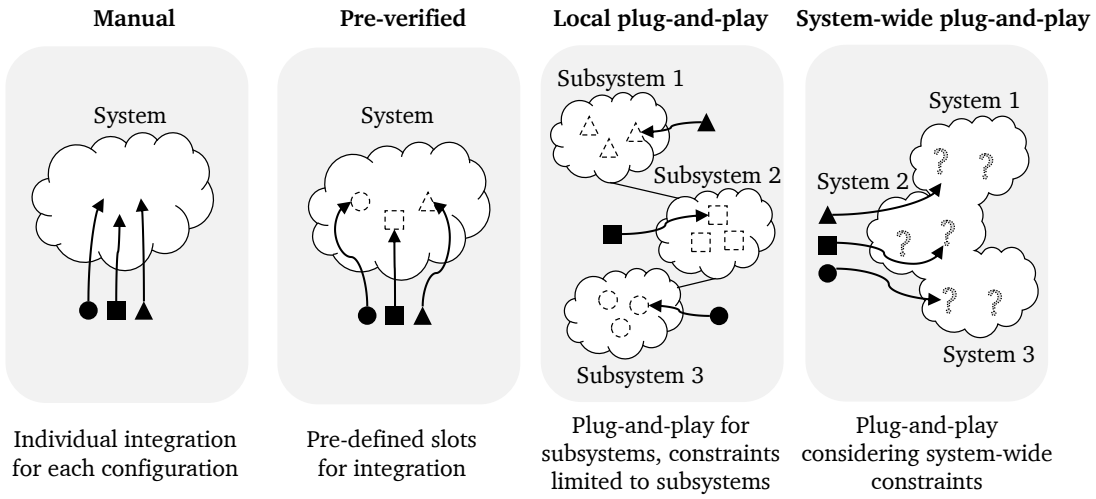


Figure 3.4: Comparison of integration variants. The manual variant cannot handle changes during the lifetime of the system, the pre-verified variant only allows changes of previously known and verified entities, the local plug-and-play variant only considers the relations within a limited area of the overall system, and the system-wide plug-and-play variant, as proposed in this work, holistically considers the entities in the system and is able to guarantee system-wide constraints.

the client-server method. The USB master in this case is responsible to handle the resources data rate and energy according to the requests of the attached devices. Notice that this is a local approach as only the bus itself is managed – no guarantees are given with respect to the chaining of processing stages. For example, consider two computers that are connected to each other via an USB-to-Ethernet adapter. While each USB master can handle the local resources, the overall latency between the computers (from a software component on one computer, via the USB and the adapters to a software component on the other computer) is not manageable. Beyond the client-server based method, we consider contract-based methods, e.g. [98, 99], to belong to the class of local plug-and-play approaches. In these methods, integration and refinement of components depends on a matching of assumptions and guarantees about their behavior [100]. In difference to the system-wide approach, the decomposition of guarantees is in the responsibility of a system designer. Correctness of the system is guaranteed as long as all local contracts are fulfilled. However, the decomposition is fixed, which limits flexibility. For example, it is not possible to automatically distribute a timing budget on a chain of components.

3.2.5 User roles of the system-wide plug-and-play approach

User roles define the tasks and responsibilities of a certain person or group within a project. As the scope of the user roles changes with the proposed approach compared to the classic procedure, the differences and similarities are highlighted in the following. As the approach is mainly focused on the timing behavior of the considered systems, only user roles affected by this topic are chosen for examination.

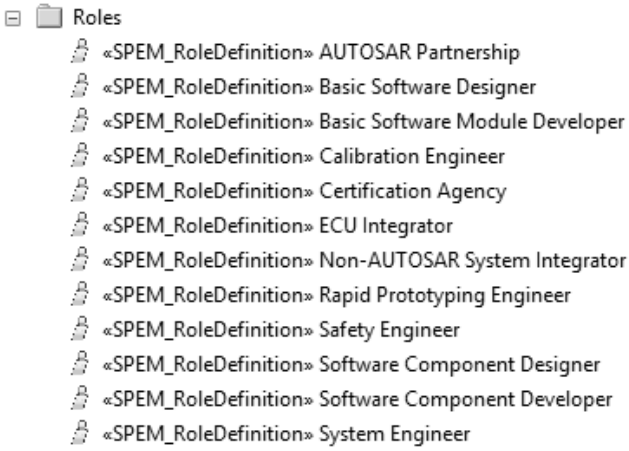


Figure 3.5: Roles defined in AUTOSAR 4.2.1 (Screenshot[2])

User roles according to the AUTOSAR [101] methodology are shown in Fig. 3.5. Considering the timing behavior of the system, related characteristics are specified and influenced by the roles basic software designer, basic software module developer, ECU integrator, software component designer, software component developer and system engineer. Especially the ECU integrator and the system engineer are involved in several tasks in parallel according to the AUTOSAR methodology, because the ECU integrator has to handle all aspects of the software running on a particular ECU, while the system engineer has to handle the system-wide interaction of the individual ECUs. Both roles are involved in the specification of timing requirements and the guarantee of those by test, simulation, or analysis of the ECUs and/or complete system. In practice, user roles are frequently not occupied by a single person but by a mixture of groups. Each of these groups is associated with a certain functionality and takes care of all aspects of it, i.e., all roles involved with this functionality. To the author's knowledge, AUTOSAR does not specify the interaction of those groups with each other, but treats the user roles as singular instances. In consequence, individual groups compete on the system resources, e.g., the available data rate on a communication entity. The **user roles in the system-wide plug-and-play approach** change compared to the classic methodology, which is sketched in Fig. 3.6. Due to the automatic verification process, the roles of the ECU integrator and system engineer are not responsible for the integration anymore. Instead, the system handles the integration automatically and gives feedback whether the process succeeded or failed. In the latter case, another configuration can be chosen or resources can be added to the system by a technician, which, in turn, are again detected automatically. To make this approach work, feature developers responsible for the roles of the basic software and software component designer and developer have to annotate the software components and feature bundles with timing information and constraints. Without these annotations, an automatic integration is not possible.

[2]Screenshot of Enterprise Architect, (c) Sparx Systems

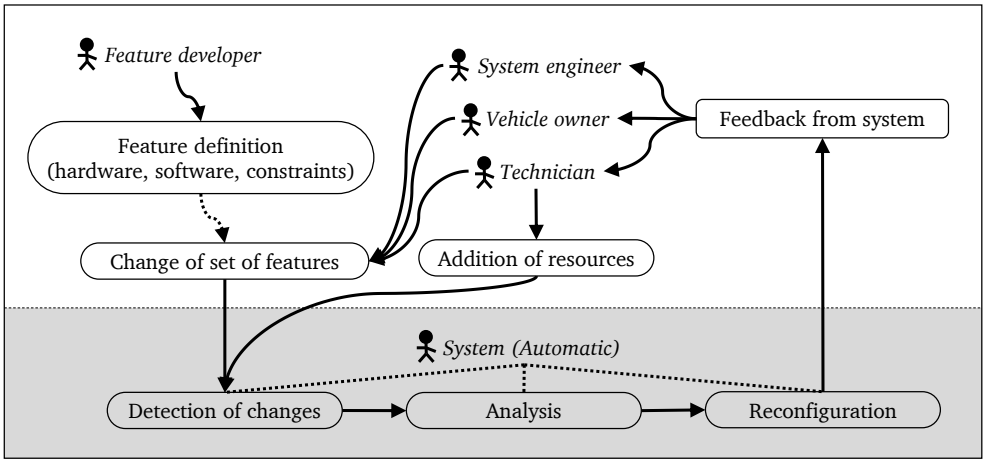


Figure 3.6: Roles within the system-wide plug-and-play approach. The changes compared to the classic roles are induced by the automatic verification process of the system.

3.2.6 Possible causes for a refused reconfiguration

The system may refuse a reconfiguration for several reasons. In this work, resource utilization and timing behavior is the focus. A configuration is refused if the analysis results indicate an over-utilization of the physical resources or if the results show that the timing requirements are not met by the system. An over-utilization is defined as the case where resources like processing time or communication data rate exceeds the limits of the underlying hardware. Timing requirements are defined to be not met if they exceed the specified bounds for the behavior in at least one case, e.g., the end-to-end timing is more than the specified maximum or the jitter of a data stream exceeds the maximum.

Beyond that, other measurements are also important to guarantee the correct execution of the system, but are not within the scope of this work. For example, the handling of safety requirements has to be an integrated part of a flexible approach to mitigate possible failures of the system [20, 102]. Also energy consumption of the electronic control units might play a role or in general the interface compatibility on a mechanical, electrical, and data level.

3.2.7 Comparison to other plug-and-play and reconfiguration approaches

As far as the author knows, the presented approach is unique in its combination of system-wide plug-and-play possibilities with integrated verification of hard real-time constraints for automotive systems, which are developed according to the data-centric communication paradigm in a component-based fashion. Other approaches [27, 95, 96, 98, 103, 104, 105] do not take into account an end-to-end analysis of data flows, are limited in their flexibility by only supporting certain types of hardware, implement a different execution or communication model, or do not take into account plug-and-play abilities. A comparison of the proposed to other selected approaches is given in Appx. A.1.4.

Figure 3.7: Rendering of the eCar demonstrator (adopted from [106]). On each corner, an *eCorner* is installed – an integrated component for steering, acceleration, and deceleration. Each eCorner can be controlled individually.

3.3 Introduction of the running example

A configuration based on the eCar demonstrator [106] (Fig. 3.7) is chosen to show the feasibility of the approach. The eCar demonstrator is an experimental platform for evaluation of information and communication technology (ICT) architectures in the automotive domain [4, 107]. The setup features four eCorners, which are integrated, electromechanical components for acceleration, deceleration, and turning of the vehicle. Each eCorner is controlled individually and includes an in-hub traction motor and a steering motor on the top. The eCar can transport one person and is controlled with a sidestick for the movement of the vehicle and a touch-screen for changing the drive modes and further interactions with the driver. To show the feasibility of the approach, the system will be constructed in several steps that show the possibility to iteratively grow a system in a plug-and-play manner. The following visualizations capture only a certain subset of the explained features, a complete introduction of the utilized models is given in Ch. 4, including models for the logical system and requirements. The running example starts with the *base* feature of the eCar, which can be extended by the *control and movement* feature, and further by the *camera* feature, which realizes a collision avoidance functionality. The example is used throughout this thesis to illustrate certain aspects of the approach.

The **base feature** of the eCar, as shown in Fig. 3.8, consists of ECUs and communication systems that form the basic infrastructure of the eCar. In this configuration, the eCar cannot be driven as no data flows and functions are defined yet. The base system consists of three ECUs, one for the vehicle center (*ECU Center*), one for the front axle (*ECU Front*), and one for the rear axle (*ECU Back*). The ECUs are connected with each other via a com-

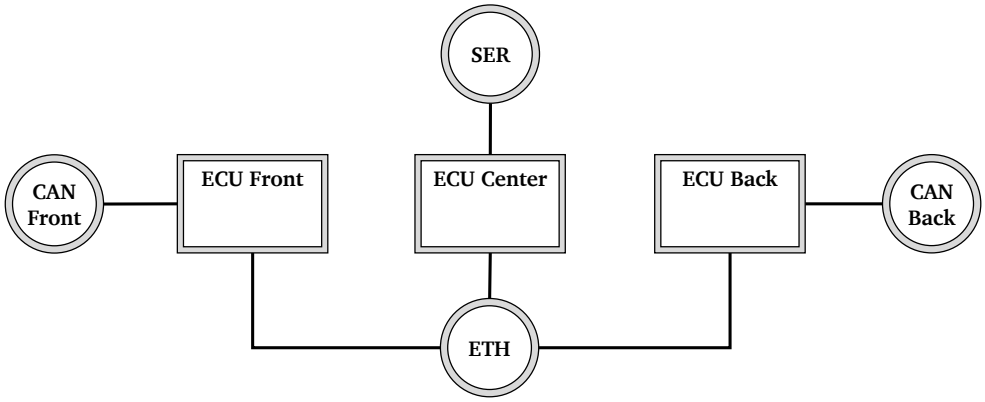


Figure 3.8: The *base* feature of the eCar, consisting of three ECUs that are interconnected via an Ethernet network. The central ECU has in addition a serial communication bus, while the front and back ECUs each have a CAN bus.

munication system based on switched Ethernet (*ETH*). In addition, *ECU Center* has a serial bus (*SER*) as a further communication possibility and the *ECU Front* and *ECU Back* are each connected to a CAN bus, named *CAN Front* and *CAN Back*, respectively. The **movement and control feature** of the eCar (Fig. 3.9) adds components and functionality to actually drive with the eCar. This feature requires the base feature and cannot work without it. The base system is extended by four eCorner aggregates, named *Agg. eCornerFR/-FL/-BR and -BL* according to their position in the eCar. Aggregates are black-box systems, which means that internal relations of the software components are not completely known. Each eCorner is equipped with a virtual software component (*Control eCornerFR/-FL/-BR and -BL*) that represents the data flow to the eCorners. The two front eCorners are connected to *CAN Front* and the other two to *CAN Back*. Three software components are part of the feature: *Control Front* and *Control Back* bundle information to eCorners on the according axle, and *Control Central* is responsible for the system-wide control and synchronization of the eCorner modules. The human-machine interface is connected to the system via the serial bus *SER*. It is modeled as an aggregate *Agg. HMI* with a virtual component *Sidestick* that abstracts the input characteristics from the sidestick used to control the vehicle. The human-machine interface is modeled as an aggregate with black-box behavior, because the internal relations are assumed to be unknown. The **camera feature** adds a collision warning system to the vehicle, based on a camera that observes traffic. The feature is built on the movement and control feature and requires it to work. The camera is modeled as an aggregate *Agg. Camera* with a virtual software component *Camera*, which abstracts the flow of objects from the camera. It is assumed that the camera preprocesses data and only transmits descriptions of recognized objects, like pedestrians or other vehicles. The camera is connected to *CAN Front* and the objects are processed in *ECU Front* in the software component *Camera Process*. This software component is connected to *Control Front* in order to have a prediction of the future movement of the vehicle. The data is forwarded by the software component *Camera Forwarder* on *ECU Center* to the aggregate *HMI*, where an additional virtual software component *Camera Signaler* is installed. This software component

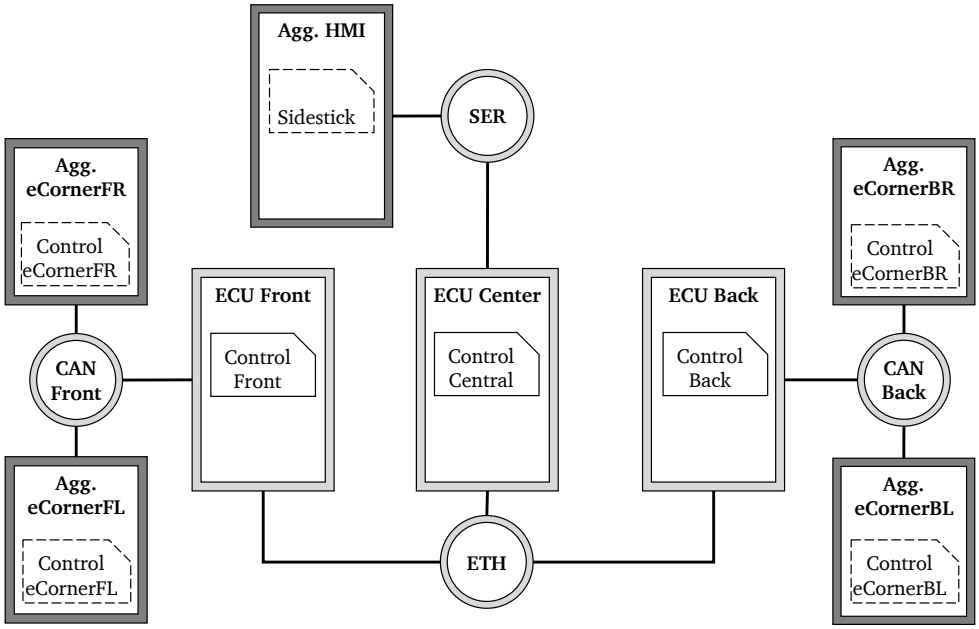


Figure 3.9: The *movement and control* feature adds driving capabilities to the eCar. It consists of four eCorner aggregates that abstract the behavior of the eCorner modules, several software components for the control and synchronization of the eCorners and an additional aggregate that abstracts the human-machine interface.

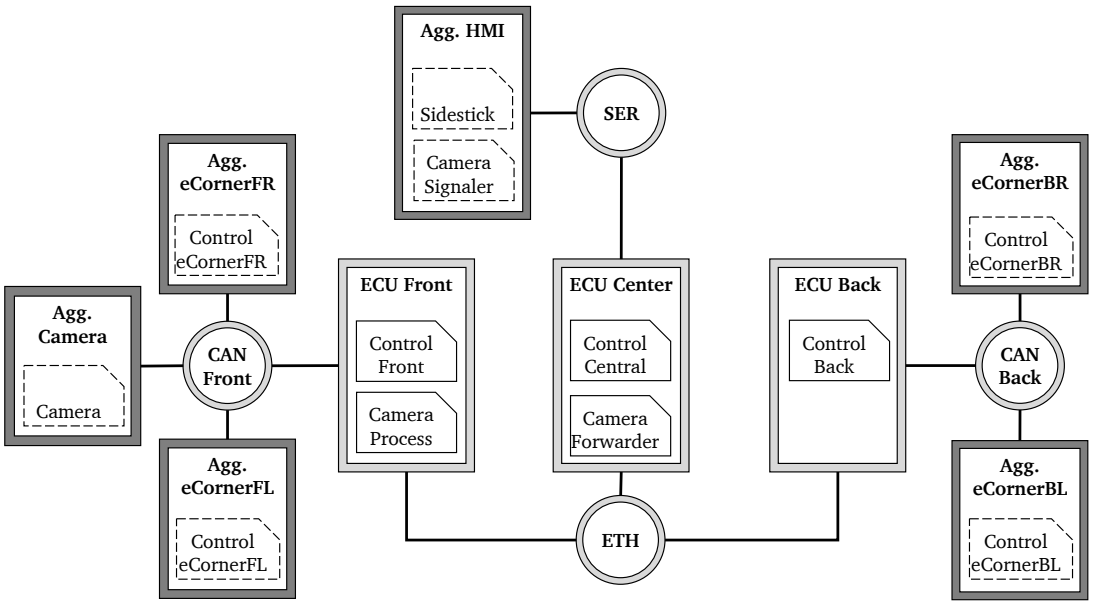


Figure 3.10: The *camera* feature adds a camera to the eCar infrastructure to implement a collision warning functionality. The detected objects of the camera are processed and possible collisions are shown on the human-machine interface.

represents an indicator on the human-machine interface, which warns the driver in case a possible collision is detected by the system.

In the next chapters, it is shown how the eCar example can be modeled by the proposed approach and how the features can be added in a plug-and-play manner (Ch. 4 and Ch. 5). Requirements are specified based on a fuzzy knowledge of the system's topology, which are automatically checked by our verification framework (Ch. 6). The verification framework is capable to exchange computation time for tightness, which is evaluated by experiments based on the introduced eCar example and other examples (Ch. 7).

3.4 Combining plug-and-play and timing guarantees summary

In this chapter, the proposed system-wide plug-and-play approach was introduced. The approach utilizes models to represent system behavior, structure, and requirements, which are the basis for an automatic integration process. It was pointed out that a flexible approach is needed where the verification tightness can be parameterized in exchange for analysis computation time to fit several scenarios. Processing phases and artifacts of the plug-and-play method were described and details of the involved development and adaption timeline were pointed out. The approach enables the addition of features into the vehicle, which were not integrated or not yet developed during the point of sale. The individual phases of the plug-and-play process, including the possible interaction points with the user, were highlighted, after which the method was differentiated against local plug-and-play methods. The advantage of the system-wide plug-and-play process is that end-to-end paths are considered even across heterogeneous system boundaries and a manual, static allocation of resources is not necessary. The system is always handled as a whole during the plug-and-play and verification processes, therefore leaving possibilities for a flexible adaption. Furthermore, the impacts on the classic user roles were discussed as the task of integration is now part of the system and not a manual process anymore. The chapter ended with the introduction of a running example based on the eCar demonstrator with its three different features (base, movement and control, camera), which will be used in the following chapters to show the feasibility of the system-wide plug-and-play method and to refine it.

Chapter 4

Adequate meta-models

This chapter develops meta-models for the system-wide plug-and-play approach. The goal is to define a minimal set of adequate models that capture the necessary information of a system for timing verification and that are extensible in a plug-and-play manner to show the feasibility of the approach. Apart from the specification of the according meta-models, which can be used as a blueprint for similar approaches, contributions include the definition of a set of timing requirements for loosely coupled systems according to the data-centric communication paradigm. Seven different model types are utilized for the approach, the relations between them are visualized in Fig. 4.1. A brief overview is given in the following.

A **LOGICAL model** describes software and external components including their ports, communication with certain topics, and captures properties of these components. A **DATA model** is a dictionary of available data topics in the system, forming the foundation for the communication specification. A dictionary abstracts from the direct communication relations and implements the data-centric principle [23]. A **TIMING REQUIREMENTS model** extends a LOGICAL model and captures timing constraints of software components as well as data chains with known or unknown sources or sinks. A **SYSTEM model** represents an electronic architecture of a vehicle with electronic control units, aggregates, and network elements. It captures specific properties of the hardware like processing speed or device latency. A **DEPLOYMENT model** defines a mapping of the LOGICAL model and DATA model to elements of the SYSTEM model. It covers instantiations of software and external components and includes properties of the mapping regarding the timing, e.g., execution times of a software component on a certain device. A **FEATURE model** combines a LOGICAL, DATA, TIMING REQUIREMENTS, SYSTEM, and DEPLOYMENT model. A feature represents a certain functionality of the vehicle with its sensors, actuators, and data processing. Features form a package that includes all information for the integration of a certain functionality and are an important concept of the system-wide plug-and-play approach. A **FEATURE-SET model** is a set of FEATURE models that represents a certain configuration of a vehicle. The union of all referenced FEATURE models yields the complete system for verification. The proposed structure of models implements the ideas of the Model Driven Architecture (MDA) [39] and can be regarded as an extension of the approach presented in [95] with a feature concept and the consideration of timing requirements.

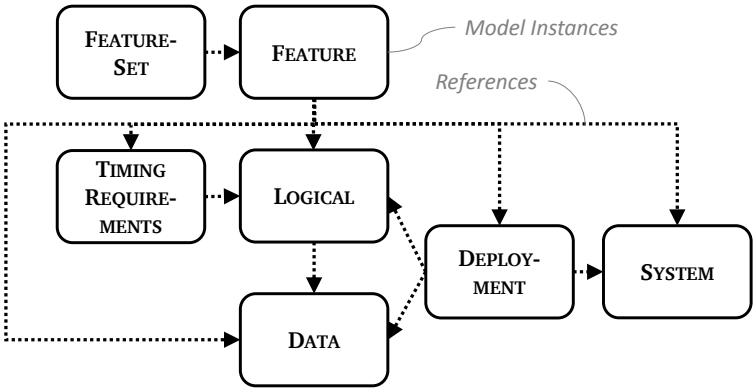


Figure 4.1: Relations of the models. Arrows represent references. A FEATURE-SET model combines several FEATURE models and stands for a certain configuration of a vehicle. Each FEATURE model can include a TIMING REQUIREMENTS, LOGICAL, DATA, DEPLOYMENT and SYSTEM model. A LOGICAL model is a component-based representation of the real-world system with data-centric communication based on a DATA model. TIMING REQUIREMENTS can be annotated to a LOGICAL model. A DEPLOYMENT model maps a LOGICAL and DATA model to a concrete hardware representation, captured by a SYSTEM model.

4.1 Requirements for adequate meta-models

Before the meta-models are described in detail, the requirements, which led to the actual representation, and applied solutions are briefly explained:

Requirement: The system described by the models is expandable. This requirement directly reflects the plug-and-play behavior of the approach. Starting from a certain system, it must be possible to correctly capture the addition of further functionality by the extension of the models representing the system. This is achieved in the approach by FEATURE models that capture constraints, components, and behavior of a certain functionality with their sub-models. The FEATURE models can be freely combined in FEATURE-SET models to specify the actual configuration of a vehicle. **Requirement: Functionality and technical realization is independent.** The functionality should be described in a technology-independent way in order to be re-usable for different setups of the hardware and communication systems. This guarantees a fast adaption of a functionality to various hardware platforms. In the proposed approach, a LOGICAL model captures the setup of the system in an abstract, platform independent way. This representation can be mapped via a DEPLOYMENT to a SYSTEM model in order to link it to a certain hardware configuration. **Requirement: Communication between entities is abstracted.** Instead of a manual, message-wise integration, a data-centric integration should be utilized. The communication must not be modeled by direct references between the components, but by references to an abstract data representation that can either be read or written by the components without knowing each other. This enables an extension of the communication behavior of the system, because components can read or write data without knowing the actual sender or receiver. In the approach, this is achieved by a DATA model, which represents a data dictionary – a set of available topics in the system. Features can add topics to this dictionary or access the

already existing topics. The communication of components within a LOGICAL model is restricted to publish or subscribe data to or from these topics. This representation is mapped automatically to messages during the model transformation steps. **Requirement: Timing requirements can be annotated.** As the approach integrates an automatic verification of a system configuration, the timing requirements have to be captured by the models. In addition, it has to be possible to describe timing constraints even if the sender or receiver of certain data instances is unknown, which might be the case in a data-centric communication abstraction. This is implemented by a TIMING REQUIREMENTS model, which captures all constraints of a certain functionality and is part of a FEATURE model. Constraints are available that directly reference component ports within a data chain, but also constraints that reference the source or sink of a data chain. **Requirement: Timing requirements are technology-independent.** During the design of a function, the actual mapping to a certain technology might be unknown. Hence, it should be possible to define constraints that are independent of the actual hardware mapping. This is achieved in the proposed approach because a TIMING REQUIREMENTS model exclusively references artifacts from a LOGICAL model. This way, a hardware-independent specification of constraints is possible. **Requirement: The models allow formal analysis.** The system-wide plug-and-play approach with automatic verification only works if the models allow a formal analysis in the end. This may not be directly possible with the input models, but at least after some further processing like model-to-model transformations. This is achieved in the approach because the different modeling elements have attributes that allow a concrete instantiation of the system, suitable for a formal analysis. For example, a DEPLOYMENT model includes execution times of all mapped software components and the size of topic items on several communication media.

4.2 Meta-model representation

Without loss of generality, the meta-models derived in this chapter were implemented and visualized with the Eclipse Modeling Framework (EMF) [40], which is based on the Ecore meta-meta-model. No exclusive features were used and therefore the approach can be transferred to other modeling concepts as well. Graphical representations are employed as a formal definition of the meta-models. The elements are visualized in Fig. 4.2 and are explained in the following:

Classes are templates for abstract representations of real-world or virtual objects. They can include functions and attributes that stand for operations, queries, and states of an object. *Abstract classes* do not directly represent an object but form the foundation for classes sharing certain properties. Classes can become *super-classes* of other entities that inherit all their properties. *Attributes* model states or properties of an object. Each attribute has a name associated to it and a data type. With *enumerations*, data types can be user-defined and applied within attributes. *Containments* are an inclusion relation between objects, while *references* are a knowledge relation. Usually, objects can only be part of one containment but may have several references to it. References are allowed to span across multiple meta-models, visualized by the indicator for *external elements*. *Cardinalities* form constraints or requirements for the number of objects that can be contained or referenced,

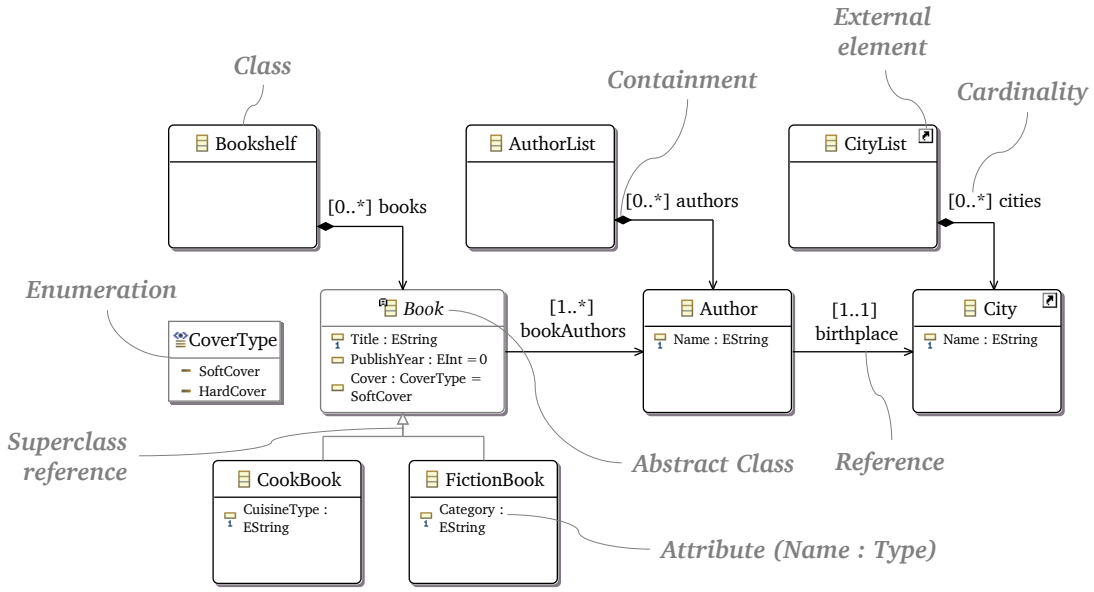


Figure 4.2: Visualization of the used modeling elements from the Ecore meta-meta-model [40], based on a book database example.

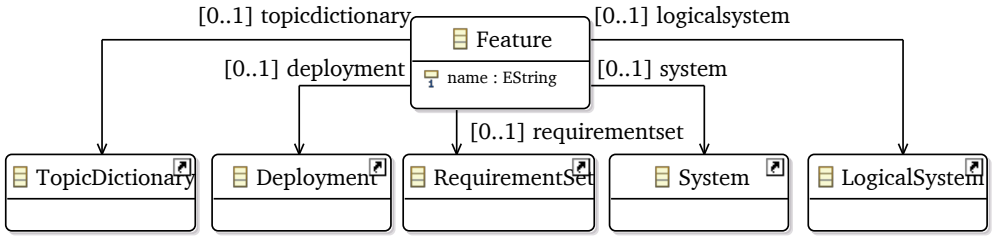


Figure 4.3: FEATURE meta-model. It models building blocks of a system, which can be composed in a plug-and-play manner.

and for the quantity of attribute instances that can be part of a class instance. Lower and upper bounds can be specified, where an asterisk (*) represents an arbitrary quantity. For example, [1..∗] means a cardinality of at least one and undefined maximum. If the lower and upper bounds are equal, only one number may be shown.

4.3 FEATURE meta-model

The FEATURE meta-model, as shown in Fig. 4.3, references zero or one of each DATA, DEPLOYMENT, TIMING REQUIREMENTS, SYSTEM and LOGICAL models. A feature is the basic building block of a concrete system and models one particular functionality. The idea is to be able to combine features with each other in order to define or change the functionality of a system. It is not necessary that a feature is self-contained. Elements of a feature might reference elements of another one. As restriction, it is assumed that the dependencies between features form no cycles.

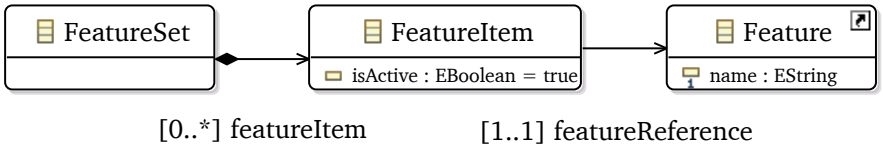


Figure 4.4: FEATURE-SET meta-model.

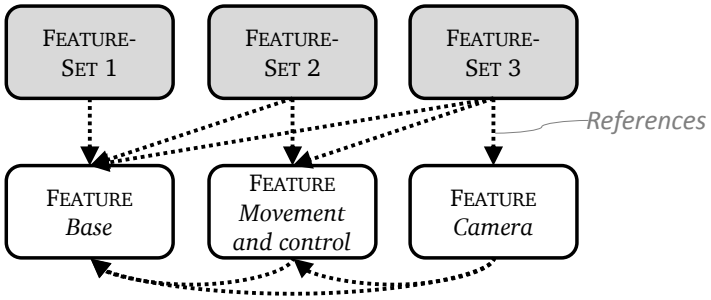


Figure 4.5: The three FEATURE-SET models of the eCar example. Each FEATURE-SET model represents a certain configuration.

The eCar [106] example consists of three different FEATURE models, according to the description in Sec. 3.3: Base FEATURE model, movement and control FEATURE model, and camera FEATURE model.

4.4 FEATURE-SET meta-model

The FEATURE-SET meta-model is shown in Fig. 4.4. Instances serve as a container to describe active features of a system. A FEATURE-SET model therefore forms a snapshot of a current system configuration. The referenced FEATURE instances may have cross-references to each other. A FEATURE-SET model represents one particular system configuration in a one-to-one relationship.

The three FEATURE-SET models of the eCar example, as shown in Fig. 4.5, represent the three possible configurations. FEATURE-SET 1 has only one reference to the base FEATURE model. In comparison, FEATURE-SET 2 has references to the base FEATURE and movement and control FEATURE models. The FEATURE-SET 2 is a configuration of the eCar that makes it driveable. FEATURE-SET 3 adds the camera feature to the system, while keeping the other two features. Each FEATURE model has several sub-models attached to it that are not visualized and are explained in the following sections.

4.5 SYSTEM meta-model

The SYSTEM meta-model, as shown in Fig. 4.6, represents the technical realization of a system and hence belongs to the platform-specific modeling view in the model-driven development process [39]. It describes processing units, aggregates, and communication infrastructure. Hence, it is compatible with a centralized architecture approach [26, 27] (see also Sec. 1.2.2). The SYSTEM model of one feature may reference and extend the SYSTEM model of another feature, e.g., to represent addition of hardware. Elements of a SYSTEM

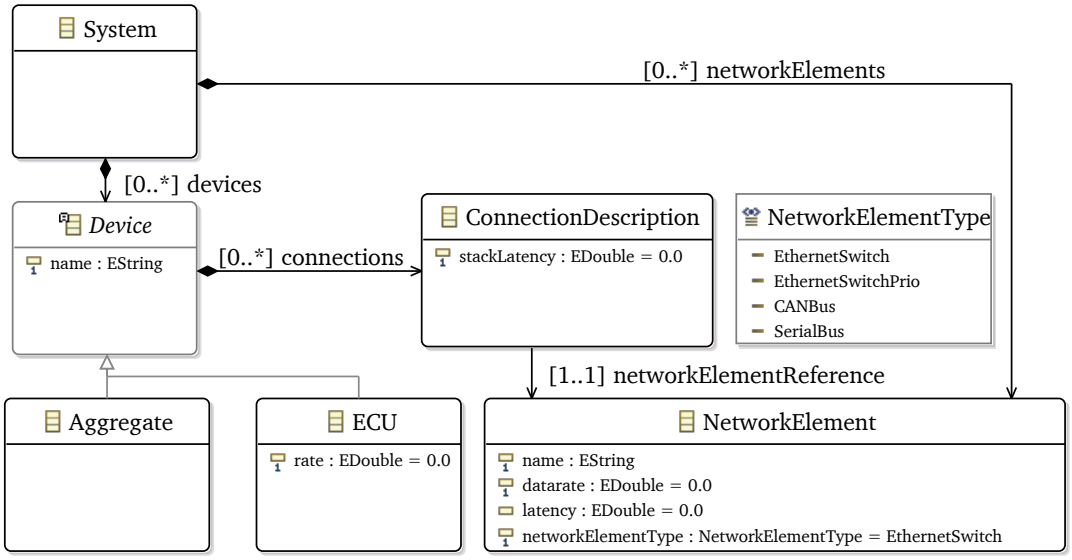


Figure 4.6: SYSTEM meta-model. It represents the technical realization of a system, including electronic control units, aggregates, communication relations, and network infrastructure.

model are annotated with parameters for technical properties (e.g., processor speed and network data rate). The SYSTEM meta-model consists of the following entities: *Electronic control units (ECUs)* model processing units of the system. ECUs can execute instances of software components and may have an undefined number of connections to network elements. For our approach, it is assumed that every ECU has one processing core and the parameters of all executed instances on the processing core are known. *Aggregates* are containers for sensors and/or actuators of the system. Only the external behavior of aggregates is known, e.g., parameters of outgoing data streams, which makes aggregates black-box entities of the system (compared to the white-box behavior of ECUs, where dependencies are known). *Network elements* define the protocol of the communication relations and may in extension represent network infrastructure elements. The parameters include available data rate and a possible latency during the processing in infrastructure or other elements. *Connections* stand for physical connections between network elements and devices, where devices include ECUs and aggregates. To limit the scope for an analysis, a direct connection of network elements is not considered in our approach, e.g., connections between Ethernet switches. Each connection includes a parameter to model the possible latency of the communication stack on the devices.

Each feature of the eCar example includes an individual SYSTEM model. For the base feature, the SYSTEM model describes the basic electronic system of the eCar consisting of three electronic control units, two controller area network (CAN) buses [108], a serial bus and a switched Ethernet. The movement and control feature adds four eCorners and a human-machine interface to steer the vehicle, all modeled as an aggregate. The camera feature adds a camera aggregate to the SYSTEM model. This is visualized in the Figs. 3.8, 3.9 and 3.10 if the software and external components are omitted.

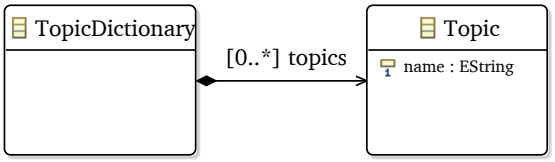


Figure 4.7: DATA meta-model. A topic defines data type and semantic to enable a decoupled communication. All available topics are combined in a dictionary.

Topic Name	Description
Movement and control feature	
MovementVector	Desired velocity and direction of the vehicle.
ControllerFront, ControllerBack	Control of the front or back axle.
WheelFrontLeft, WheelFrontRight, WheelBackLeft, WheelBackRight	Control of one of the four eCorners.
Camera feature	
CameraRaw	Raw objects from the camera.
CameraProcessed	Processed objects containing collision information.
CameraSignalHMI	Collision warning indicator.

Table 4.1: Dictionaries of the eCar example.

4.6 DATA meta-model

Communication relations are implemented according to the data-centric paradigm in our approach. Concrete dependencies between senders and receivers are abstracted. Instead, a communication is based on so-called topics, where each topic includes a definition of a data structure and a unique name. Since the concrete structure is not of interest for our approach, the only available attribute for topics is the name, see Fig. 4.7. The active dictionary can be extended by the addition of features and features can access already defined topics. In practice, the access rights to the dictionary have to be managed, which was not considered in this work. The topics can further be extended by quality-of-service attributes. For example, the Data Distribution Service (DDS) [36] defines 13 attributes, including durability, liveliness, lifespan, ownership, and history window of data samples.

For the base feature of the eCar example, a dictionary is not defined. The dictionaries for the movement and control, and camera features are combined in Tab. 4.1.

4.7 LOGICAL meta-model

The LOGICAL meta-model describes relations between software and external components in an abstracted, technology independent manner. The LOGICAL model itself does not describe the functionality but the type, characteristics, and dependencies of data relations. A representation is shown in Fig. 4.8. Note that no references to hardware modeling elements exists, which corresponds to the characteristics of a platform-independent model in the model-driven development process [39]. We employ the component-based design

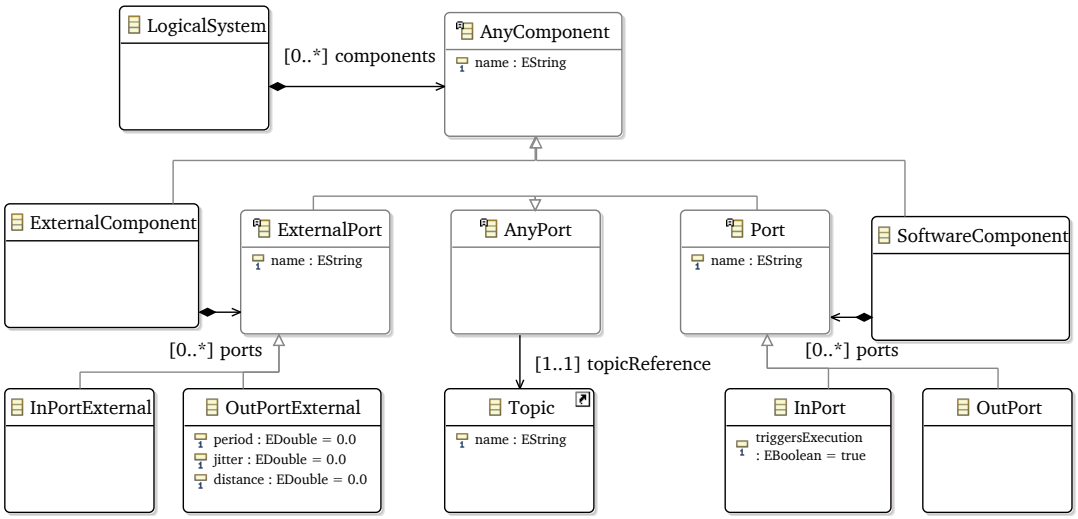


Figure 4.8: LOGICAL meta-model. It defines communication relations of software and external components to and from topics.

principle [31] in our approach; software instances are isolated from their environment and may only communicate via dedicated ports with each other. A direct communication or alternation of and between software instances is not allowed. This leads to the following elements in the LOGICAL meta-model: *Components* are entities that can produce, consume, and process data. We distinguish *software components*, which can be mapped onto electronic control units, and *external components*, which can be mapped onto aggregates. The internal behavior of software components is known, while only a specification of the incoming and outgoing data is available for external components. *Ports* form interfaces between components and topic instances. A communication of components is only possible via ports, which limits the behavior of the system and makes an analysis feasible. Ports have an associated direction (inbound or outbound) and topic. Only data corresponding to the associated topic might be received or transmitted. Outbound ports of external components can be annotated by information describing characteristics of the data stream according to the PJD model [80] (see Sec. 2.6). Furthermore, it can be chosen if an inbound port of a software component triggers its execution. For simplicity, we assume that all triggering inbound ports cause an equal behavior.

The LOGICAL model of the movement and control feature of the eCar example is shown in Fig. 4.9. It is visible that the ports of components are not directly connected to each other, but reference topics. The actual data flows are calculated later. The topics were previously defined in the topic dictionary and correspond to the definition of Tab. 4.1. The LOGICAL model of the camera feature is visualized in Fig. 4.10. It seems that the topic *ControllerFront* is referenced for input but no publisher for it is defined. This is not an issue, because it is already served within the movement and control feature. The example shows that topics are a possibility to decouple the functionality. It is not of interest within the camera feature, where the data actually comes from.

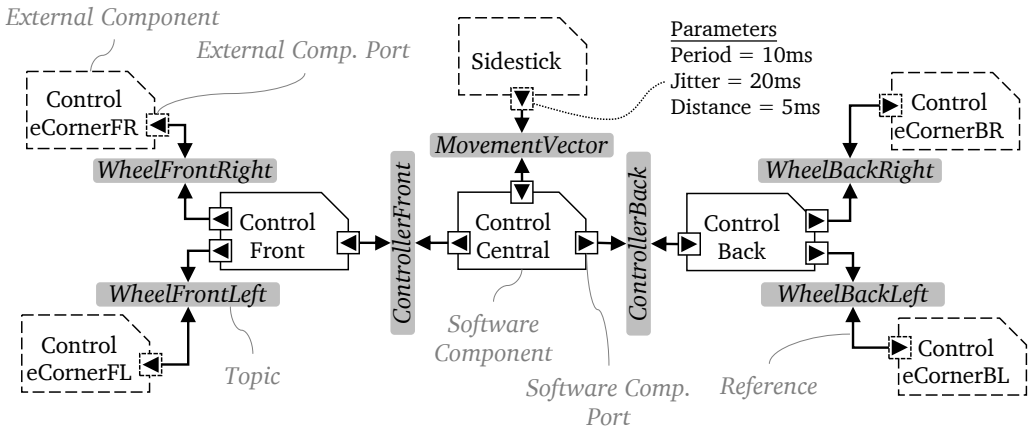


Figure 4.9: LOGICAL model of the movement and control feature of the eCar example. The components are not directly coupled. Instead, topics are referenced and the actual data flows are calculated later – this explains the counter-intuitive direction of the edges for input ports.

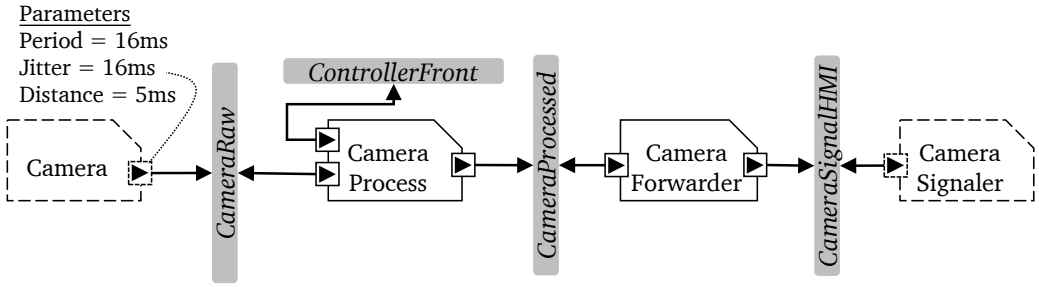


Figure 4.10: LOGICAL model of the camera feature of the eCar example.

4.8 DEPLOYMENT meta-model

The DEPLOYMENT meta-model, as shown in Fig. 4.11, models the mapping of elements from a LOGICAL model to a SYSTEM model. It is possible to map elements multiple times if necessary, e.g., for redundancy reasons or if software instances are just used several times. The DEPLOYMENT model together with the SYSTEM, LOGICAL and DATA models describe a functionality completely. The DEPLOYMENT meta-model consists of the following elements: *Software mappings* represent instantiations of logical software components onto system elements, in our case electronic control units. Software mappings have to be annotated with the best-case and worst-case execution times to make the system analyzable. Exemplary, we distinguish four different kinds of software mappings in our approach: Highest-priority interrupt service routines mappings, high-priority event-triggered mappings, medium-priority time-triggered mappings, and low-priority event-triggered mappings. All of the mappings are annotated with a priority valid for the individual group, except the medium-priority time-triggered mappings, which are annotated with a period for the triggering frequency. *External mappings* instantiate external software components onto aggregates. Dependencies between instances on aggregates are usually not known,

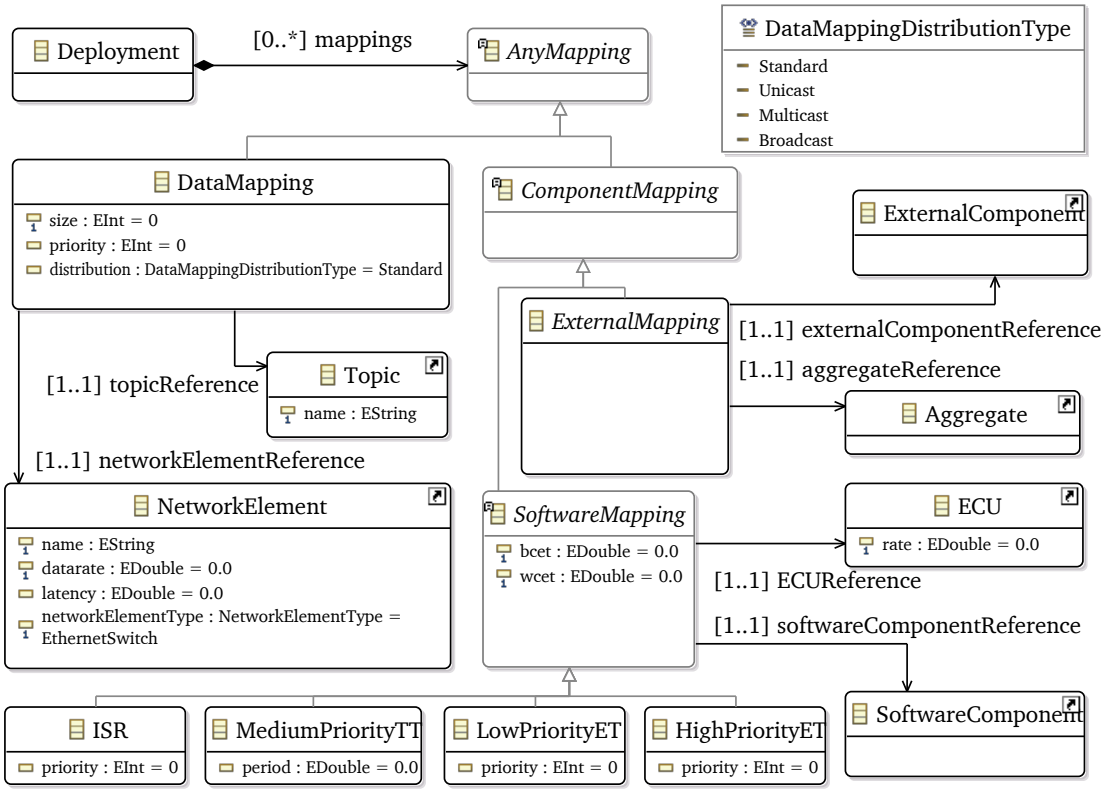


Figure 4.11: DEPLOYMENT meta-model. Software and external components are mapped to electronic control units and aggregates with a DEPLOYMENT model. It includes quantitative, technology-dependent annotations needed for a timing analysis of the system.

hence no annotations are available. It directly corresponds to the black-box behavior of aggregates. *Data mappings* describe technology-dependent parameters for topics that are sent via a specific network element. They include the size of a message, priority and distribution policy. The distribution policy can be unicast, multicast, or broadcast behavior, but the choice is limited by the underlying technology. For example, the distribution policy for controller area networks is always multicast, while for a serial bus it is always broadcast.

Four different kinds of software mappings are distinguished. These categories should be regarded as an example configuration to cover the most important use cases. The chosen software mappings are as follows: *ISR mappings (ISR)* correspond to highest priority, non-preemptive software instances. ISR routines are executed asynchronously to the program flow and handle external events approaching the processing unit. Compared to the other mappings, this is the only mapping that cannot be preempted. *High-priority event-triggered mappings (HighPriorityET)* are used for high-priority data, especially related to safety-relevant communication. For example, the triggering of an airbag-system falls into this category. *Medium-priority time-triggered mappings (MediumPriorityTT)* are employed to model periodic tasks, like those involved in control applications. *Low-priority event-triggered mappings (LowPriorityET)* model tasks that are related to background-traffic, e.g.,

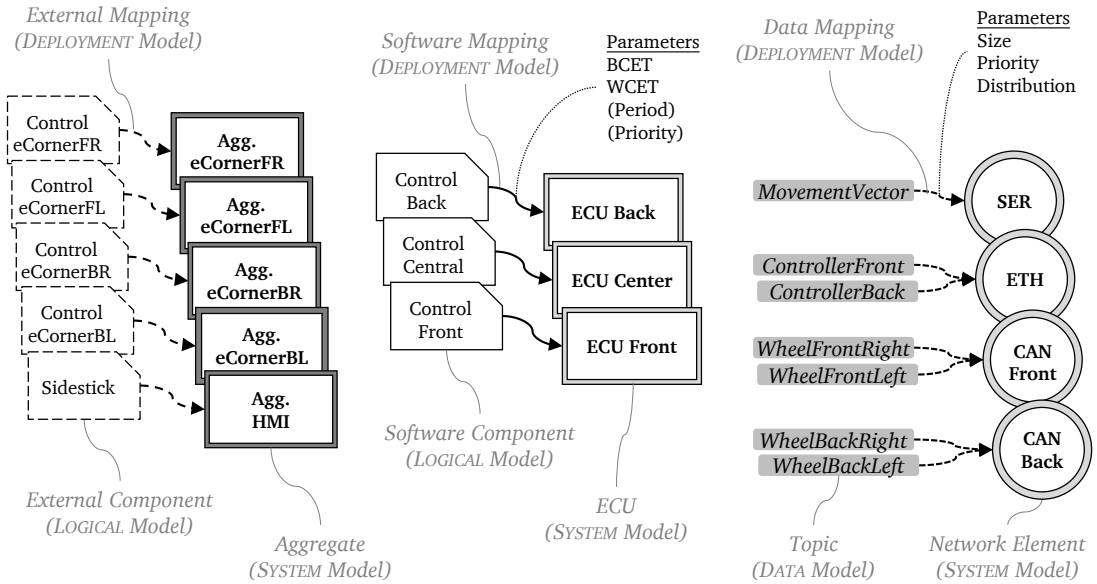


Figure 4.12: DEPLOYMENT model of the movement and control feature of the eCar example. The DEPLOYMENT model maps the LOGICAL and DATA models to the SYSTEM model and annotates it with all required parameters for an analysis.

firmware update, internet and diagnosis traffic. The four software mapping categories have also advantages for the analysis: Ad-hoc changes of the system, e.g., a connection of a smartphone to the internet via the infrastructure of the vehicle, fall usually into the last category, the low-priority mapping. Consequential, the timing of the other three categories is not affected and has not to be re-evaluated for this kind of changes.

The DEPLOYMENT models for the eCar example are visualized in Fig. 4.12 for the movement and control feature, and in Fig. 4.13 for the camera feature. Both figures show the mapping of the LOGICAL model to the SYSTEM model via external, software, and data mappings, stemming from the DEPLOYMENT model. The deployment is unambiguous in this example. However, several mappings from the logical model to the system model could exist, enabling the possibility of a design-space exploration as part of future work.

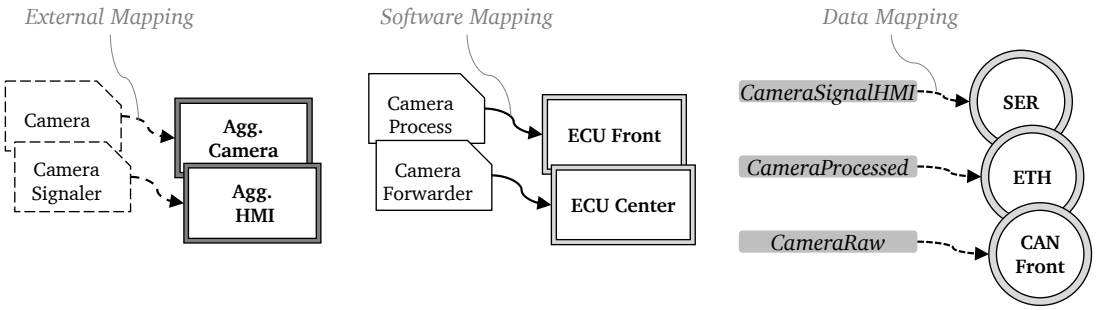


Figure 4.13: DEPLOYMENT model of the camera feature of the eCar example.

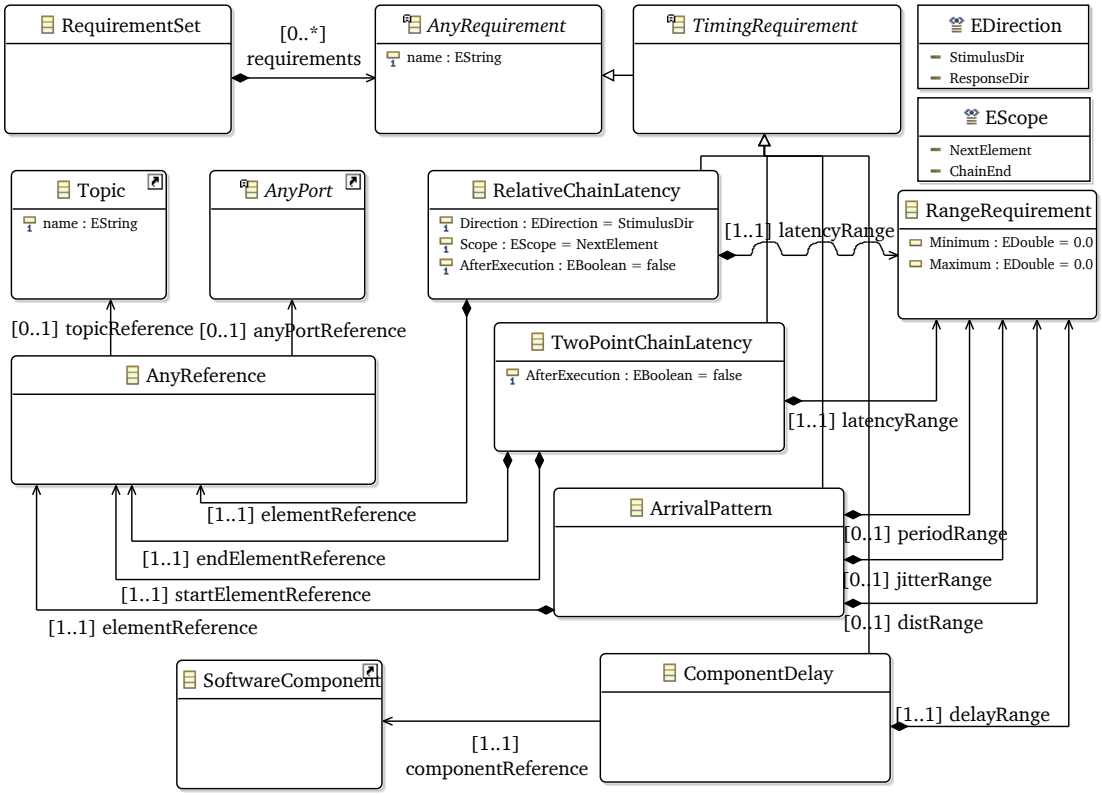


Figure 4.14: TIMING REQUIREMENTS meta-model, showing the relations of the four requirements *RelativeChainLatency*, *TwoPointChainLatency*, *ArrivalPattern* and *ComponentDelay*.

4.9 TIMING REQUIREMENTS meta-model

To complete our approach, properties of the TIMING REQUIREMENTS meta-model, shown in Fig. 4.14, are discussed. Its purpose is to annotate a LOGICAL model by descriptions for timing requirements. After the analysis of a system, it is verified that all timing requirements are met and the configuration is valid. Because communication between software instances is abstracted by the data-centric principle, a direct annotation of communication relations with timing requirements is not always possible. For this reason, we define a method in the following to specify timing requirements with unknown senders or receivers. The ideas of static concepts [46, 55, 109, 110] are hereby extended to support timing requirements within the data-centric, system-wide plug-and-play approach. Start and end points of event chains can be referenced, which makes it possible, for example, to specify the maximum delay until a certain data sample in the system causes a reaction in the physical world. Timing requirements can be annotated to ports, topics, or software components. If annotated to ports or software components, requirements apply directly to the referenced element. For topics, the behavior is different as multiple publishers and subscribers can exist. If a requirement applies to a topic, it automatically applies to all its subscribers or publishers, depending on its type. Each requirement bounds the range of certain attributes of the ana-

Name	References	Parameters	Description
Timing requirement: RelativeChainLatency			
RCL1	Topic	Scope: NextElement, Direction: Any	Latency between all publishers and all subscribers of the topic (see Fig. 4.15).
RCL2		Scope: ChainEnd, Direction: Stimulus	Latency between all sources of event chain and all subscribers of topic (see Fig. 4.15).
RCL3		Scope: ChainEnd, Direction: Response	Latency between all publishers of topic and sinks of event chain (see Fig. 4.15).
RCL4	Port	Scope: NextElement, Direction: Response	Latency between all direct successor ports and referenced port.
RCL5		Scope: NextElement, Direction: Stimulus	Latency between all direct predecessor ports and referenced port.
RCL6		Scope: ChainEnd, Direction: Response	Latency between all sinks of event chain and referenced port.
RCL6		Scope: ChainEnd, Direction: Stimulus	Latency between all sources of the event chain and referenced port.
Timing requirement: TwoPointChainLatency			
TPCL1	Two topics		Latency between all publishers of 1st and all subscribers of 2nd topic (see Fig. 4.16).
TPCL2	Two ports		Latency between the two referenced ports.
Timing requirement: ArrivalPattern			
AP1	Topic		Pattern valid for all subscribers of topic.
AP2	Port		Pattern valid for referenced port.
Timing requirement: ComponentDelay			
CD1	Software Component		Delay experienced by any input event of component.

Table 4.2: List of timing requirements, associated parameters and descriptions. The timing requirements are annotated to a LOGICAL model and can reference either topics, ports, or software components. For simplicity, the parameter *AfterExecution* was not included in this list.

lyzed system. For example, the minimum as well as the maximum jitter of an event chain at a certain component port can be specified or the maximum delay. In general, we distinguish between four types of timing requirements in our approach, named *RelaitveChainLatency*, *TwoPointChainLatency*, *ArrivalPattern* and *ComponentDelay*. These requirements are an exemplary set to show the applicability. A summary of the requirements with the according parameters, elements they apply to, and description is given in Tab. 4.2. Details are elaborated in the following.

The timing requirement **RelativeChainLatency** bounds the minimum and maximum latency between the referenced element and a relative point before or after this element in the event chain. The requirement applies either to a topic or to a port of a software or external component. The relative point in the event chain can be configured by the parameters *Scope*, *Direction* and *AfterExecution*. In case the requirement is applied to a port and *Scope* is set to *NextElement*, the latency between the referenced port and its direct successors or predecessors is meant, according to the chosen *Direction*. *Direction* may either be *Stimulus*, in the direction of the event sources, or *Response*, in the direction of the event

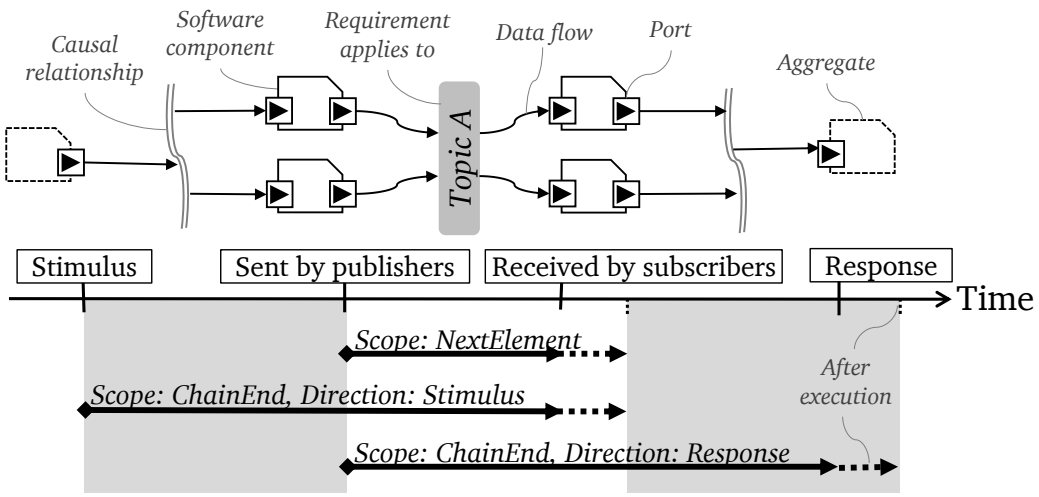


Figure 4.15: Elaboration of the different configurations of **RelativeChainLatency** for *Topic A*. The requirement is always valid for all publishers or subscribers of the topic under consideration or for all possible paths to and from chain ends. The concrete number of publishers and subscribers as well as the number of chain ends is possibly unknown during design time.

sinks. As the referenced port might be linked to several other ports, the requirement is applied to all possible paths. For example, if the requirement is applied to the outgoing port of a certain software component, it may describe the minimum or maximum allowed latency until an event of the outgoing port is transferred to all direct receiving ports. If the Boolean parameter *AfterExecution* is set to true, the maximum processing duration of the receiving component is added to the latency. In that case, not only the receiving of data is constrained, but the constraint also applies to its processing. When *Scope* is set to *ChainEnd* all existing sources or sinks of the event chain are considered, depending on the *Direction* parameter. For example, it can be specified that a brake signal has to cause a physical reaction within a certain latency by all actuators at the end of the event chain. Depending on the setup of the system, multiple chain ends may exist. In case *RelativeChainLatency* is applied to a topic, it has to be specified what timings are meant exactly, because samples of topics might exist at several locations simultaneously in the running system and hence the requirement is fuzzy at first sight. In our definition, *RelativeChainLatency* applied to topics constraints the transmission latency between all publishers and subscribers of that topic if *Scope* is set to *NextElement*. The parameter *Direction* is ignored in that case. In that configuration, the requirement guarantees that the transmission latency of any topic sample is within a certain range. If *Scope* is set to *ChainEnd*, all possible sinks or sources of all event chains related to the topic are considered. If *Direction* is set to *Stimulus*, all delays between event sources and subscribers of the referenced topic are constrained. If *Direction* is set to *Response*, delays are measured from the publishers of the referenced topic until all sinks of the related event chains. For example, a maximum delay between the occurrence of an external event and its receiving by all relevant components can be defined, without a need to consider intermediate components for conversion and processing. The different

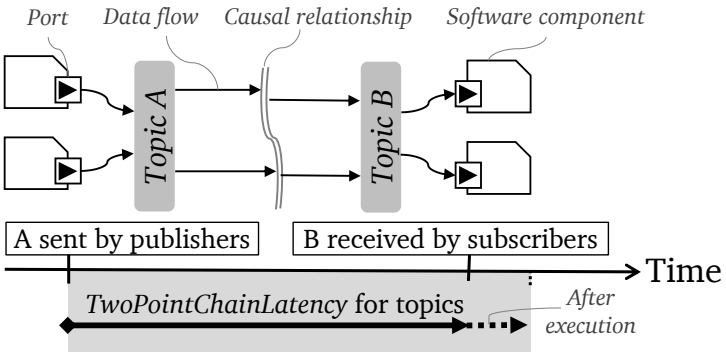


Figure 4.16: Elaboration of **TwoPointChainLatency** for *Topic A* and *Topic B*. The requirement is valid between all publishers of *Topic A* and all subscribers of *Topic B*, assuming a causal relationship exists.

configurations of *RelativeChainLatency* for topics are visualized in Fig. 4.15. The timing requirement **TwoPointChainLatency** bounds the minimum and maximum latency between two referenced elements. These elements might be either ports or topics and we assume, without loss of generality, that they are of the same type. If the requirement is applied to ports, all possible event paths between the first and second port are constrained. In case of topics, all possible paths between the publishers of the first topic and the subscribers of the second topic are considered, as shown in Fig. 4.16. In both cases, a causal relationship between the two entities has to exist. If a causal relationship is not present but a requirement is defined, the analysis will always reject the configuration. This is useful to verify that two topics are related with each other, e.g., it can be checked that way that a steering wheel causes a movement of the wheels with a certain system configuration. The parameter *AfterExecution* defines if the execution duration of the receiving component is added to the delay. The timing requirement **ArrivalPattern** applies constraints on the characteristics of an event stream for ports or topics. The characteristics are the minimum and maximum period, jitter, and inter-arrival time within an event stream. If *ArrivalPattern* is applied to topics, it refers to all subscribers of the referenced topic. The timing requirement **ComponentDelay** constraints the minimum or maximum processing delay of a specific software component. For the calculation of the delay, the preemption of the component is considered, see Fig. 2.4. *ComponentDelay* applies to all possible combinations between input and output ports according to the internal relations of a component. It is not possible to map this constraint to external components because the internal behavior is not available during the analysis.

A comparison to the AUTOSAR Timing Specification [55] is summarized in Tab. 4.3 and detailed in the following. A main difference of the proposed approach is that it enables the specification of requirements based on topics. In AUTOSAR, timing constraints in the logical view ("*Virtual Function Bus Timing*") are always bound to ports. The proposed *ArrivalPattern* requirement is comparable to AUTOSAR's *EventTriggeringConstraint*. It further allows the specification of timings for individual events, which is not directly possible in our proposed framework because of the lack of phase information during the analysis with the

AUTOSAR concept	Comparable concept in proposed approach	Differences of our approach
EventTriggeringConstraint	ArrivalPattern	Can also be attached to topics.
LatencyTimingConstraint	TwoPointChainLatency	Event chain automatically calculated. Can be attached to topics. No multirate possible.
AgeConstraint	RelativeChainLatency	Specification in response and stimulus direction, but reaction semantics. Can be attached to topics.
SynchronizationTimingConstraint, OffsetTimingConstraint	n/a	Hardly possible because of lack of phase information.
ExecutionOrderConstraint, ExecutionTimeConstrain	n/a (but possible)	Was not implemented.
n/a (indirectly possible)	ComponentDelay	All port relations covered. (AUTOSAR: can be mapped to LatencyTimingConstraints.)

Table 4.3: Comparison of proposed specification possibilities for timing requirements with concepts of the AUTOSAR Timing Extensions [55].

Real-Time Calculus [30] framework. The proposed *TwoPointChainLatency* is comparable to AUTOSAR's *LatencyTimingConstraint*, but the latter needs a complete specification of the event chain and cannot be mapped to topics. In our approach, the event chain calculation is an integrated part, which enables the specification of latencies between entities without direct knowledge of their relationship. On the other hand, this can lead to ambiguities in multi-path scenarios. AUTOSAR considers scenarios with over- and undersampling behavior for the *LatencyTimingConstraint*, which is only partly captured in our approach, i.e., no maximum-age semantics are provided. The proposed *RelativeChainLatency* has similarities to AUTOSAR's *AgeConstraint*, because both provide a possibility to specify relative requirements. In our approach, we provide reaction instead of age semantics and the specification is possible in both directions of the event chain, while AUTOSAR limits it to the stimulus direction. AUTOSAR's *SynchronizationTimingConstraint* and *OffsetTimingConstraint*, used for the specification of tolerances for offsets between events in one or different event chains with or without functional dependencies, cannot directly be analyzed with the Real-Time Calculus and thus no mapping is possible. Modeling of constraints on the execution order of entities is possible with the *ExecutionOrderConstraint* in AUTOSAR and could be mapped to an analysis of the components' relations in a LOGICAL model in the proposed approach. For simplicity reasons, it was not considered as it does not directly reference the timing behavior. AUTOSAR's *ExecutionTimeConstraint* constraints the worst-case execution time of components. The intention is to formulate requirements for the implementation of software components. Actual latencies of components are modeled with the specification of an event chain between the input and output ports of the components in connection with a *LatencyTimingConstraint*. Although this is also possible with our approach and the

Name	Type	Description	Value
Feature: Movement and control			
MovVecResp	RCL3	Latency until *MovementVector* topic causes physical reaction.	$d \leq 30ms$
CContrIn	RCL5	Transmission latency for data to *Control Central* input port.	$d \leq 5ms$
MovVecPer	AP1	Minimum and maximum period for *MovementVector* topic.	$8ms \leq p \leq 12ms$
Feature: Camera			
CamSigIn	RCL7	Maximum latency for data of *Camera Signaler* input port.	$d \leq 75ms$
CamProSig	TPCL2	Latency between *Camera Process* output port and *Camera Signaler* input port.	$d \leq 10ms$
CamRate	AP2	Minimum/maximum message rate for camera input stream of *Camera Process*.	$16ms \leq p \leq 34ms$
CamProDel	CD1	Maximum delay for *Camera Process* component.	$d \leq 40ms$

Table 4.4: List of timing requirements of the eCar example. Types are explained in Tab. 4.2.

TwoPointChainLatency requirement, the additional *ComponentDelay* requirement helps to specify constraints for all possible combinations of events on input and output ports simultaneously. A direct constraint of the worst-case or best-case execution time according to the *ExecutionTimeConstraint* was not added in our approach for simplicity reasons.

A summary of the TIMING REQUIREMENTS models of the eCar example is given in Tab. 4.4 and is explained in the following. The first three requirements are part of the movement and control feature, the last four are part of the camera feature. The requirements were chosen to cover a mixture of the available types while be based on the eCar example and should be regarded as an exemplary set for demonstration purposes of the approach rather than real-world examples. The **Movement Vector Response Latency (MovVecResp)** defines the maximum latency until any event associated with the *MovementVector* topic is processed and has caused a physical reaction. It is modeled by a *RelativeChainLatency (RCL3)* requirement that references the topic, has the chain end as scope, and is directed to the response direction. The requirement was attached to the topic and not a port as other software components might publish data for this topic if the car is reconfigured, e.g., if it is equipped with an autonomous driving ability that provides the *MovementVector*. The value was chosen to be $d \leq 30ms$, which is equal to the requirement defined in the examples for the AUTOSAR timing extension [55]. There, a maximum delay of 30ms was proposed between a change of the accelerator paddle and the reaction of the actuator. In contrast, it was specified in a use case of the TIMMO2-USE project that the delay of a brake-by-wire system should be in the range of 70ms to 120ms [111]. The **Central Control Input Latency (CContrIn)** limits the transmission latency of events approaching the input port of the *Control Central* software component and the according senders. It is realized with a *RelativeChainLatency (RCL5)* requirement that references the port, has next element as scope and directs to the stimulus of the event chain. The requirement was not attached to the topic to not affect other entities listening to the same topic. Exemplary,

the value was chosen to be $d \leq 5ms$. The **Movement Vector Period (MovVecPer)** is an *ArrivalPattern (AP1)* requirement and constraints the minimum and maximum period of all subscribers of the *MovementVector* topic. This ensures that the topic is updated frequently. Exemplary, the value was chosen to be $8ms \leq p \leq 12ms$. The **Camera Signaler Stimulus Latency (CamSigIn)** constraints the maximum allowed latency between any stimulus and the input port of the *Camera Signaler* component. It is realized as a *RelativeChainLatency (RCL7)* requirement in direction of the stimulus with the chain end as scope. The execution latency of the last component in the chain is not considered. This requirement ensures that the driver is warned in time if a critical situation appears on the road. Exemplary, the constraint was chosen to be $d \leq 75ms$. The **Camera Process to Camera Signaler Latency (CamProSig)** bounds the transmission latency between the *Camera Process* component's output port and the *Camera Signaler* component's input port. The requirement is of type *TwoPointChainLatency (TPCL2)* without considering the execution time of the receiving component. As an example, the value was chosen to be $d \leq 10ms$. The **Camera Message Rate at Camera Process Input Constraint (CamRate)** constraints the message rate at the input port of the *Camera Process* component. It is an *ArrivalPattern (AP2)* requirement and the constraint was defined to be $16ms \leq p \leq 34ms$, which roughly corresponds to a rate between 30 and 60 messages per second, reflecting the accepted frame rate of a camera system. The **Camera Process Delay (CamProDel)** bounds the maximum processing latency of the *Camera Process* component. It is of type *ComponentDelay* requirement that constraints all possible combinations of input and output ports. The value was chosen to be $d \leq 40ms$.

4.10 Adequate meta-models summary

In this chapter, a minimal set of meta-models was described, which are suitable to implement the system-wide plug-and-play approach with timing verification. After the requirements were developed, each meta-model was discussed in detail and the feasibility was shown with examples based on the eCar test case. The LOGICAL, DATA, SYSTEM and DEPLOYMENT models are utilized to describe interactions, topology, and dependencies of logical and physical entities of the vehicle. The LOGICAL description can be annotated by elements of the TIMING REQUIREMENTS model to define requirements relative or absolute to topics or ports in the event chains of components. The relative approach makes it possible to define requirements if the concrete event chains are unknown during design time. The five mentioned meta-models are combined to a FEATURE model, which represents one functionality of the vehicle, composable in a plug-and-play manner. A concrete configuration of the vehicle is represented with a FEATURE-SET model, which combines all active features.

Chapter 5

Model transformation and platform mapping

This chapter describes transformation and mapping patterns to convert a FEATURE-SET model into a representation that can be analyzed and eventually deployed. After the analysis step, the timing requirements are verified based on the results of the analysis and a decision of the feasibility of the system is drawn. The transformation is conducted in several steps as visualized in Fig. 5.1:

1. **Combination of FEATURES:** The individual FEATURE models of a system are combined to a single feature, represented by the combined FEATURE model.
2. **Transformation to an INSTANCE model:** The combined FEATURE model is transformed to an INSTANCE model. An INSTANCE model is a deployable representation where all software, hardware, and communication entities are instantiated and the data flows through the system are known.
3. **Transformation to an ANALYSIS model:** The INSTANCE model is further transformed to yield the analyzable model. An ANALYSIS model consists of entities that represent

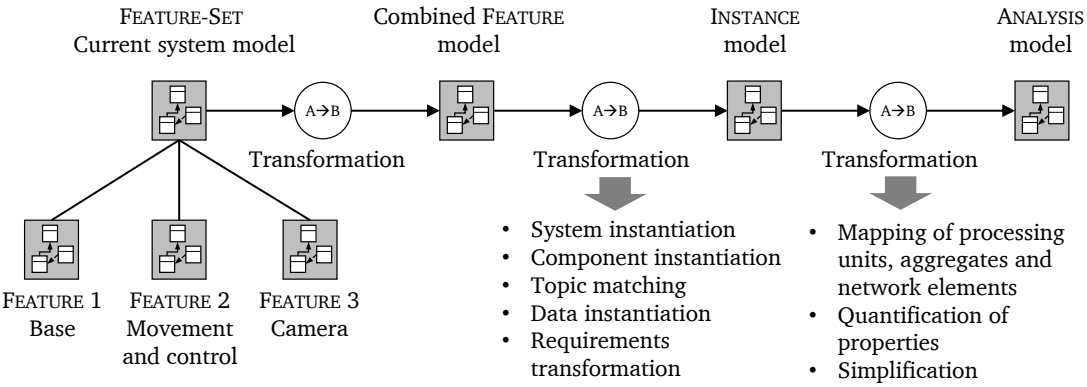


Figure 5.1: Transformation steps from a FEATURE-SET model to an ANALYSIS model.

components of the Real-Time Calcululs framework. The model is self-contained, i.e., no reference to external models exist, and all quantities from the DEPLOYMENT models are integrated for the analysis.

The result of the complete transformation and mapping process is available in Appx. A.4 for the eCar example. The set presented in Appx. A.3 was used as exemplary values for the parameters.

5.1 Transformation from a FEATURE-SET model to a combined FEATURE model

The first step in the transformation chain is the combination of individual FEATURE models into a combined FEATURE model. A FEATURE-SET model references all FEATURE models under consideration and is the starting point for this transformation. The goal of the transformation is to generate a holistic model, which combines all unique model elements of the active FEATURE models. A feature is called active if it is part of the FEATURE-SET model and the active attribute is set to true. Because FEATURE models might have cross-references between them, it is important in this step to resolve these references and to guarantee that each element in the combined FEATURE model is unique. However, collisions might still occur, e.g., if model elements have identical names, but this aspect is subject to future work. The combination of homogeneous models is a basic operation in model-driven development and frequently referred to as a *composition* or *merge* of models [38, 50, 112].

Definition 5.1 *Transformation step from a* FEATURE-SET *model to a combined* FEATURE *model.* ***Input:*** *A* FEATURE-SET *model with references to an arbitrary number of* FEATURE *models. Each* FEATURE *model may reference zero or one of each* SYSTEM*,* LOGICAL*,* DEPLOYMENT*,* TOPIC *and* TIMING REQUIREMENTS *models.* ***Output:*** *One* FEATURE *model with zero to one of each* SYSTEM*,* LOGICAL*,* DEPLOYMENT*,* TOPIC *and* TIMING REQUIREMENTS *models.* ***Constraints:*** *Each input* FEATURE *model is a subset of the resulting combined* FEATURE *model, i.e., all model elements of the input are represented in the output. The output model consists only of unique model elements.*

5.2 Transformation from a combined FEATURE model to an INSTANCE model

The second step in the chain is the transformation of the combined FEATURE model to an INSTANCE model. An INSTANCE model consists of all instantiated software, hardware, and communication elements including data flows. These instances are independent from numeric properties of the underlying technology, like data rates or processing speeds, but topology and data distribution schemata are integrated to correctly calculate data flows within the system. Relative timing requirements are resolved and mapped to absolute references. An INSTANCE model is a deployable representation of a configuration, which is utilized to reconfigure the setup after a successful verification.

Definition 5.2 *Transformation step from a combined* FEATURE *model to an* INSTANCE *model.* ***Input:*** *A combined* FEATURE *model representing a complete system.* ***Output:*** *An* INSTANCE

model. ***Constraints:*** *The* INSTANCE *model contains all software and hardware instances of the combined* FEATURE *model and data flows between them. The* INSTANCE *model may still contain references to the combined* FEATURE *model. All timing requirements are absolute.*

This transformation step is divided into three sub-steps. *1. System and component instantiation:* Relations between network elements and containers for aggregates and processing units are built, defining the system topology. All software and external components with the according ports are instantiated according to the DEPLOYMENT model. During the instantiation, the execution priorities of the components are calculated and for each container, the published and subscribed topics are collected. *2. Topic matching and data instantiation:* Correspondent to published and subscribed topics of each component within a container, reference connections to the network elements are setup for further processing. Depending on the availability and requests of certain topic instances at network elements, data flows are initiated. Quantity and characteristics of data flows depend on the amount of senders and receivers as well as the underlying distribution strategy for a certain message and the capabilities of the network element. *3. Requirements transformation:* All relative requirement specifications are resolved to absolute references only, because data flows are known at this point and absolute references are required for the analysis.

INSTANCE meta-model and transformation sub-steps

The INSTANCE meta-model, as shown in Fig. 5.2, is the basis for the deployment of a configuration and for the final transformation into a model representation that can be analyzed. Compared to the meta-models introduced in the previous chapter, the INSTANCE meta-model does not implement measurements for clarification or to enforce constraints as it is machine generated and processed, and hence is not affected by human modeling mistakes. An INSTANCE model is not self-contained as it includes references to the combined FEATURE model. For example, performance metrics from a DEPLOYMENT model are not represented directly in an INSTANCE model. In the following, the introduced sub-steps to transform a combined FEATURE model into an INSTANCE model are explained in detail.

1. System and component instantiation. Each device (aggregate or electronic control unit) and network element of the combined SYSTEM model is mapped to a generic container. Links between devices and network elements are transformed to *external connections*, one for each direction, and terminated by additional ports of the containers. This defines the complete topology of the system. Virtual network elements, e.g. for a CAN bus, are also transformed to containers, which unifies the following transformations. Software and external mappings of the combined DEPLOYMENT model are both mapped to component instances. Those stemming from ECUs are sorted and linked according to their priorities. Software components of the combined LOGICAL model might be mapped to several processing units, e.g., because of duplication for safety reasons. For each container a virtual inbound port, called *internal inbound collector*, and a virtual outbound port, called *internal outbound collector*, is instantiated. The inbound collector represents the distribution of incoming topic samples to the components, the outbound collector stands for the combination of outgoing topic samples from component instances. For each subscribed topic per component, a *reference connection* is created between the internal inbound collector and

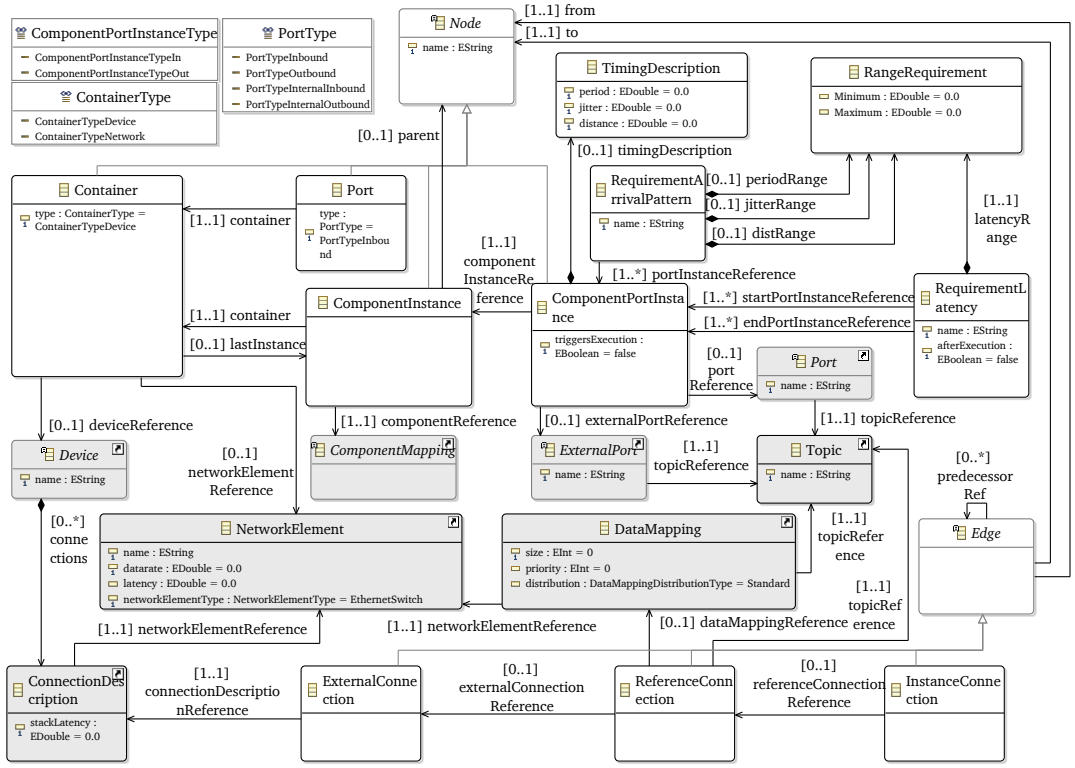


Figure 5.2: INSTANCE meta-model. External model elements are marked with shaded boxes.

the according component port instance. For each published topic per component, a reference connection is created between the component port instance and the internal outbound collector. An exemplary result is shown on the left side of Fig. 5.3.

2. Topic matching and data instantiation. In the second sub-step, topics are matched across network elements and message instances are derived. A topic is matched if a publisher and a subscriber of the topic under consideration are connected to the same network element and are not part of the same container. In that case, *reference connections* are established between containers and network elements to model the availability and request of topic types. Reference connections are the base for the further refinement into *instance connections*, which model transmitted messages. A topic is further matched via the internal inbound and outbound collectors if the publisher and subscriber reside on the same container. An exemplary result is shown in the center of Fig. 5.3.

Subsequently, each reference connection is mapped to one or more instance connections according to the number of subscribers and publishers and the desired distribution method. An exemplary result is shown on the right side of Fig. 5.3. Each instance connection has a reference to its predecessor; the complete chain between data sources, processing entities, and data sinks is captured in the graph of data instance elements. The distribution variants considered in this work are visualized in Fig. 5.4 and detailed in the following:

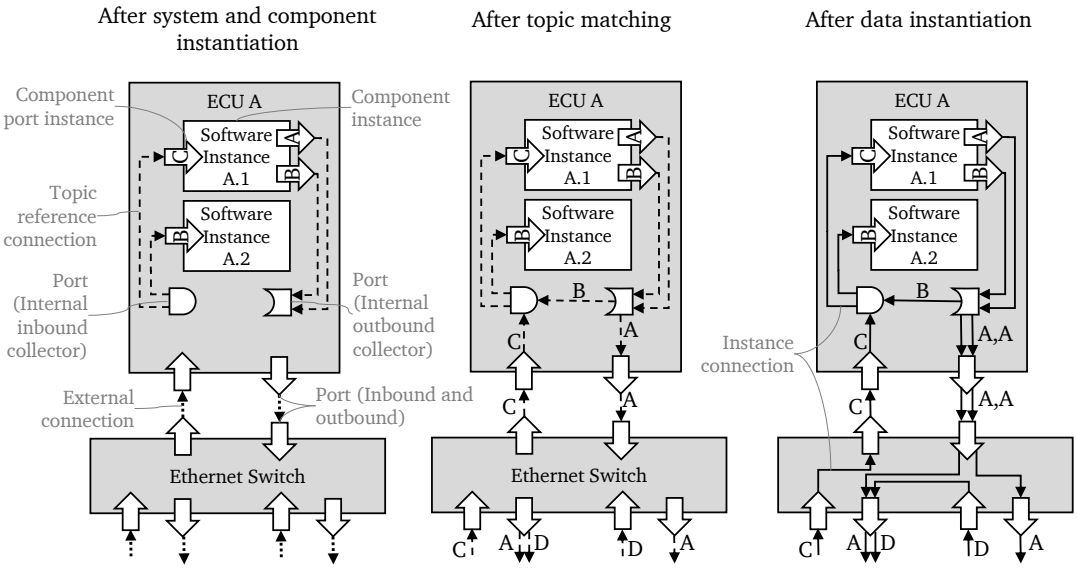


Figure 5.3: Exemplary visualization of transformation steps from a combined FEATURE to an INSTANCE model. After system and component instantiation (left figure) system topology and software instance relations are known. Internal inbound and outbound collectors represent points for the branching and joining of topic instances. After topic matching (center figure) and data instantiation (right figure), all topics are matched across network devices and messages are instantiated according to the network media and distribution strategy.

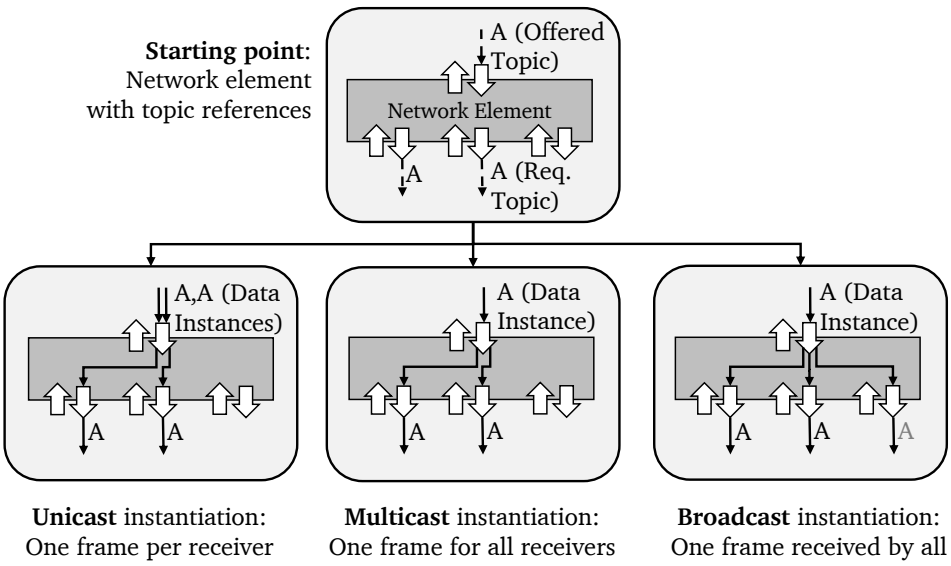


Figure 5.4: Distribution variants and impact on the data instantiation process.

Technology	Distribution Variants	Default
Switched Ethernet	Unicast, multicast, broadcast	Unicast
CAN	Multicast	Multicast
Serial bus	Broadcast	Broadcast

Table 5.1: Choices of distribution variants for network element types.

In the **unicast** case, an individual frame is created for each external receiver of the topic. Each frame is described by a chain of data instance relations that start at an internal outbound collector and end at an internal inbound collector. It is important that these instances are already modeled in the devices to account for the queuing delay and precedence of frames in the networking stack later on correctly. An example is visible in the left of Fig. 5.4, where one topic reference connection (visible in the top figure) was mapped to two data instance connections, because two external subscribers exist. In the **multicast** case, one frame instance serves a multitude of receivers. Compared to unicast, a device only sends a single frame, which gets replicated as necessary within the network infrastructure. Consequently, the networking stack on the device only has to process one single instance, represented by a single instance relation between the internal outbound collector and network element. An example is shown in the center of Fig. 5.4. In the **broadcast** case, the frame is transmitted to all receivers that are connected to a network element – independent from whether a subscription to the topic exists. The instance relation is multiplied within the network element, with each relation ending at the internal inbound collector of all connected devices. The right part of Fig. 5.4 shows an example for the broadcast case. Notice that the device attached to the rightmost port does not subscribe to the topic but an instance relation exists because of the broadcast behavior. The set of choices of the distribution variant is technology dependent. If several variants are possible, the developer can decide on one. Tab. 5.1 shows the different possibilities for the communication technologies available in the introduced examples of our approach.

3. Requirements transformation. Because data and software instances are known at this point, all requirements can be transformed to a representation with absolute references. The objective is to have only requirements regarding the arrival pattern at certain component port instances or requirements regarding the delay between two component port instances. Topics no longer exist in an INSTANCE model and hence requirements attached to topics have to be resolved as well. The transformation is explained below:

A **RelativeChainLatency** requirement involves the most complex transformation compared to the other requirements, because references can be relative and it can be attached to ports and topics. Further, scope can be the next element or the end of the event chain. A sketch of the transformation algorithm is shown in Alg. 5.1. It transforms all requirements of the type *RelativeChainLatency* in the combined TIMING REQUIREMENTS model to elements of the type *RequirementLatency* in the INSTANCE model. The function *toCompPortInstances* resolves directly the referenced port instance if the requirement is valid for a port, or resolves all input or output port instances that handle data of a certain topic if a requirement is attached to a topic. If a topic is resolved, the direction influences the result and has to be provided as a parameter, because either all subscribing or all publishing ports

Algorithm 5.1: Mapping of *RelativeChainLatency* of a combined TIMING REQUIREMENTS model to *RequirementLatency* elements of an INSTANCE model (sketch).

Input: TIMINGREQUIREMENTSPackageMM::RelativeChainLatency *self*
Result: INSTANCEPackageMM::RequirementLatency (startPortInstRef, endPortInstRef: *Sequence*(INSTANCEPackageMM::ComponentPortInstance))

```
1  if self.Direction = ResponseDir then
2  |  startPortInstRef ← self.elementRef.toCompPortInstances(dirResponse=false);
3  |  if self.Scope = NextElement then
4  |  |  endPortInstRef ← startPortInstRef.findRelPortsOneStep(dirResponse=true);
5  |  else                                          /* self.Scope = ChainEnd */
6  |  |  endPortInstRef ← startPortInstRef.findRelChainEndPorts(dirResponse=true);
7  |  end
8  else                                        /* self.Direction = StimulusDir */
9  |  endPortInstRef ← self.elementRef.toCompPortInstances(dirResponse=true);
10 |  if self.Scope = NextElement then
11 |  |  startPortInstRef ← endPortInstRef.findRelPortsOneStep(dirResponse=false);
12 |  else                                          /* self.Scope = ChainEnd */
13 |  |  startPortInstRef ← endPortInstRef.findRelChainEndPorts(dirResponse=false);
14 |  end
15 end
```

are of interest. The functions *findRelPortsOneStep* and *findRelChainEndPorts* backtrack the event chains according to the data instance relations that were created previously. Direction has to be provided as a parameter (stimulus or response). Depending on the respective function, event chains are backtracked until the next component port instance or until the end of the event chain. As event chains may join or branch, the functions can return a set of results. The end of an event chain is defined as a software component instance without any output port instances in case the backtracking is performed in the response direction, or without any input port instances in case of the other direction. As the backtracking only stops at component port instances, all intermediate ports and data relations introduced by the communication infrastructure are included in the resulting requirement *RequirementLatency*. The transformation of a requirement of type **TwoPointChainLatency** is similar to the previous mapping of *RelativeChainLatency*, but without the backtracking step. If it is attached to topics, according ports have to be resolved, where senders are always selected for the starting topic and receivers are always selected for the ending topic. In case topics are referenced, the relevant component port instances may form a set after the transformation. The result of the transformation is a requirement of the type *RequirementLatency* in the INSTANCE model. Requirements of type **ArrivalPattern** are transformed to *RequirementArrivalPattern* elements. In case component ports are referenced, the transformation is straightforward. In case topics are referenced, the requirement is applied to receiving component port instances. A **ComponentDelay** requirement is transformed to a *RequirementLatency*, where the resulting requirement is applied to all meaningful combinations of input and output component port instances that are derived from the referenced software component.

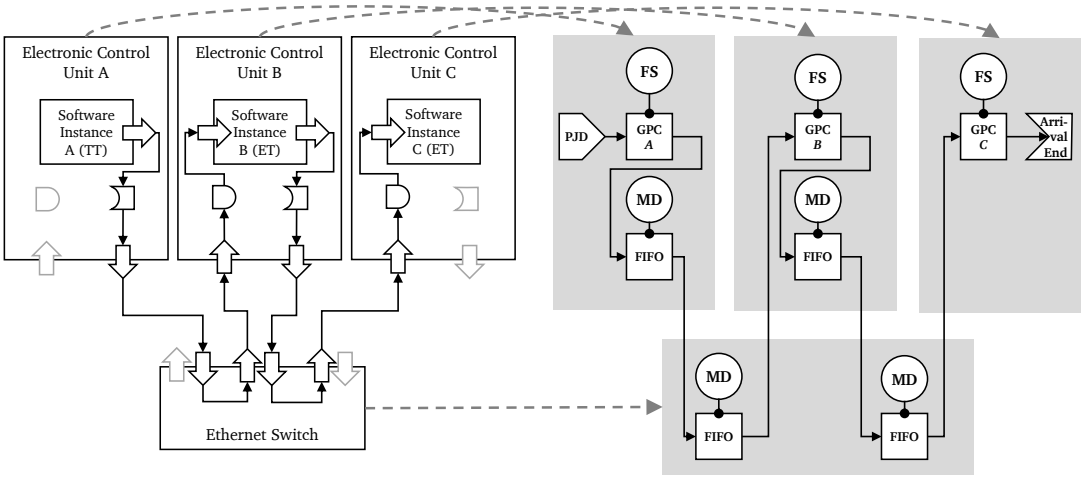


Figure 5.5: Example for the transformation from an INSTANCE model (left side) to an ANALYSIS model (right side). The transformation includes the handling of the execution orders of software component instances and the latency and order introduced by the communication stacks on the devices. Delay within network elements is also considered.

5.3 Platform mapping: From an INSTANCE model to an ANALYSIS model

After a FEATURE-SET model was transformed to an INSTANCE model, the final transformation to an ANALYSIS model is executed. The result is a tool dependent representation to derive real-time properties of the system and verify those against a set of requirements. In the following sections, it is shown how a transformation to elements of the Real-Time Calculus framework [30] is conducted. An exemplary transformation is shown in Fig. 5.5.

Definition 5.3 INSTANCE *model to* ANALYSIS *model transformation.*
Input: *An* INSTANCE *model.* ***Output:*** *An* ANALYSIS *model.* ***Constraints:*** *The* ANALYSIS *model is self-contained and includes all information necessary for a timing analysis and verification, including the requirements. The elements can be directly mapped to elements of the Real-Time Calculus framework. The transformation preserves the system semantics.*

5.3.1 ANALYSIS meta-model

An ANALYSIS model is a graph that captures stream filters with arrival and service relations. Arrival relations describe the progress of data streams through a system and service relations represent the progress of available resources. Resources may refer to computational capacity (e.g., available cycles of a processing unit) or communication capacity (e.g., available data rate of a bus). In the following, a semi-formal definition of the ANALYSIS meta-model elements is given; a more rigorous definition is provided in Sec. 6.1. A simplified representation of the ANALYSIS meta-model is shown in Fig. 5.6. **Arrival sources** provide one outgoing arrival stream. An arrival stream (α) consists of an upper and a lower curve ($\alpha = [\alpha^l, \alpha^u]$), representing the upper and lower bound of a data stream. A typical

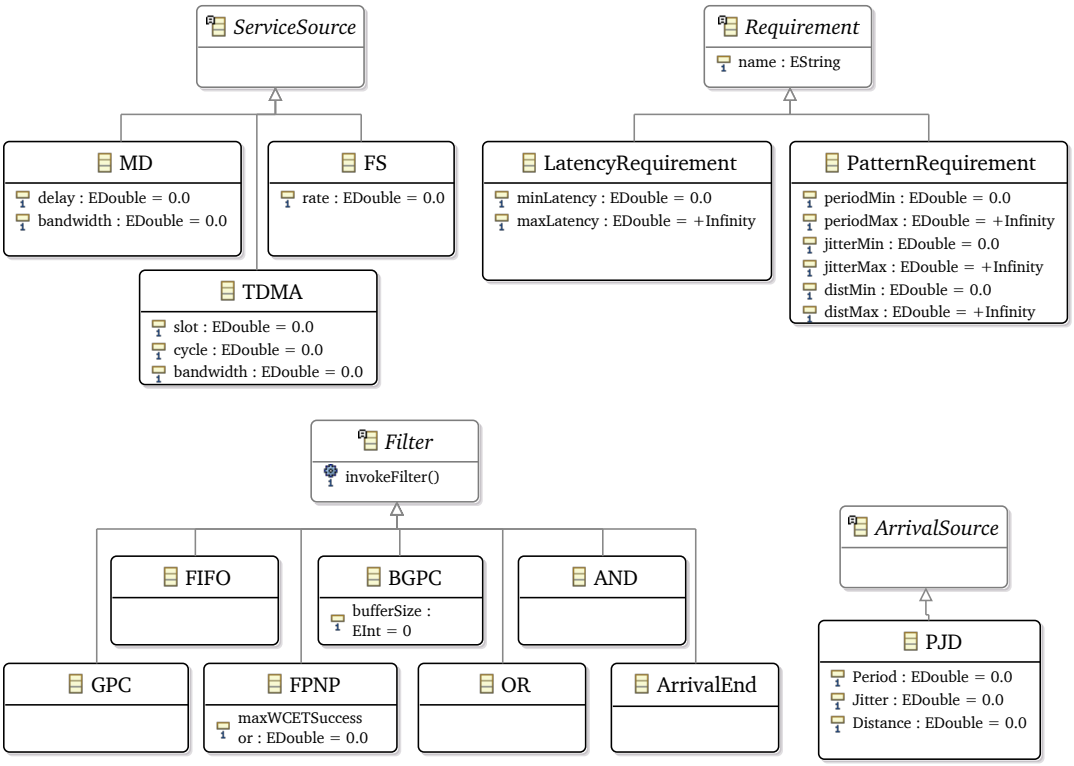


Figure 5.6: Simplified representation of the ANALYSIS meta-model. Edges representing arrival and service relations are omitted from this representation.

specification of an arrival stream, captured by this model element, is a stream according to period, jitter, and minimum distance parameters (PJD). Beyond, arbitrary data streams can be modeled. **Service sources** have exactly one outgoing service stream. A service source models available resources for stream filters. A service stream (β) consists of an upper and a lower curve ($\beta = [\beta^l, \beta^u]$), representing bounds of the described resource. **Stream filters** (f_T) are entities that manipulate data streams. This corresponds to processing components, e.g., greedy (GPC) or FIFO processing components, or logical components, e.g., interleaving of data streams with an OR component [92]. Depending on the stream filter type, a filter has a specific number of incoming service streams. If it has an incoming service stream, it can also have an outgoing one. The amount and relations of incoming and outgoing arrival streams depend on the filter type as well. Possible connection points for arrival streams are referred to as slots. Each inbound slot of a filter can be connected to at most one arrival stream. Outbound slots on the other hand can be connected to an arbitrary number of successor elements. For each stream filter, a set of internal relations ($\mathbb{I}$) is defined. This set describes the relations of incoming to outgoing slots. For example, each outgoing slot of a FIFO filter is influenced by one incoming slot and the available service. For an AND filter the behavior is different – the outgoing slot is influenced by all incoming slots. A set of parameters ($\mathbb{P}$) is associated with each filter that describes parameters

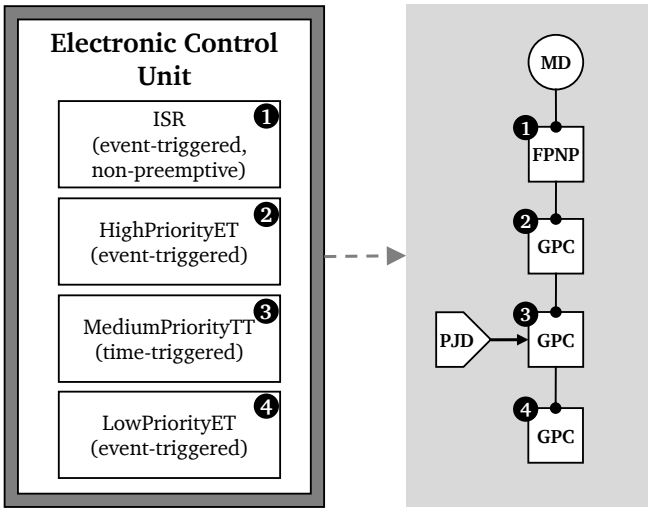


Figure 5.7: Mapping of software component instances from an INSTANCE model (left side) to elements of an ANALYSIS model (right side).

like worst- and best-case execution times for a certain slot, and a set of results ($\mathbb{E}$) that holds the performance metrics after the analysis, as depicted in Fig. 2.12. **Edges** connect arrival sources, service sources and stream filters and represent the progress of event or service streams. All edges are unidirectional within an ANALYSIS model with two possible types: *Arrival edges* are utilized exclusively to propagate arrival relations, and *service edges* to propagate service relations.

5.3.2 Mapping of processing units

A processing unit offers computational, memory, and communication resources to execute software component instances. Amount and properties of these units were initially defined in the SYSTEM models of features, and software components of the LOGICAL models were instantiated via the DEPLOYMENT models on these units. The mapping of processing units uses information from the INSTANCE and combined SYSTEM, LOGICAL, and DEPLOYMENT models.

Mapping of containers. Each ECU is represented as a generic container in the INSTANCE model. For each of these containers, a FS (full service) service source is created to model the available computational resources.

Mapping of software component instances. Four exemplary mapping types are defined in the DEPLOYMENT meta-model. Each software component instance of the mapping type *ISR* is transformed to a FPNP (fixed-priority non-preemptive) stream filter [92]. All instances of types *HighPriorityET*, *MediumPriorityTT* and *LowPriorityET* are transformed to GPC (greedy) filters. For all instances of type *MediumPriorityTT*, a PJD arrival source is instantiated in addition and linked to the filter input to model the periodic activation. In the end, the highest-priority element of the sorted instances of type *ISR* is connected to the FS service source. The chain is completed in order by the instances of type *HighPriorityET*,

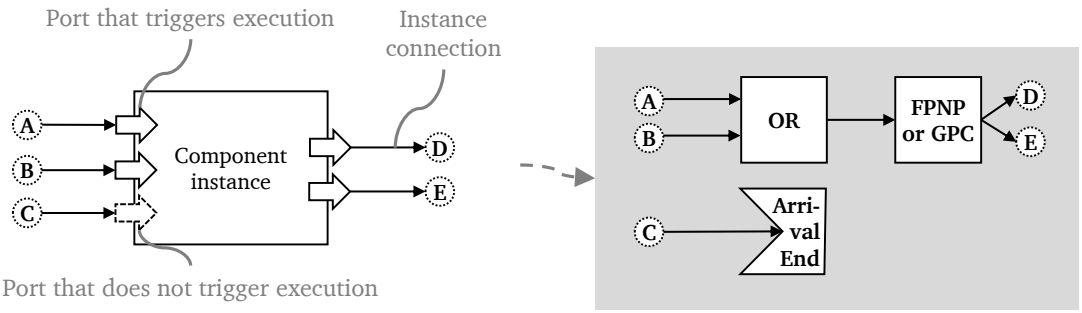


Figure 5.8: Example for the handling of execution triggers and multiple outputs. On the left side, an extract of an INSTANCE model is shown with ingoing and outgoing instance relations derived from topic publications and subscriptions. This representation is transformed to elements of an ANALYSIS model, shown on the right side. All incoming subscriptions are combined with an OR filter if these trigger an execution. All other subscriptions are mapped to *ArrivalEnd* filters.

MediumPriorityTT and *LowPriorityET*, an example for this mapping is shown in Fig. 5.7. The filters are parametrized with the worst- and best-case execution times from the combined DEPLOYMENT model.

Mapping of software component instance subscriptions and publications. Each software component instance in the INSTANCE model can have several input and output port instances. These instances were derived from the subscriptions and publications to or from topics in LOGICAL models. For each software component instance, an OR filter is instantiated where all subscriptions are connected to that trigger an execution. The OR filter is connected to the input slot of the filter representing the processing to model the combination of arriving data streams. All subscriptions that do not trigger an execution according to the combined LOGICAL model are connected to an *ArrivalEnd* stream filter, modeling the sink of an arrival stream. Outgoing slots of the filter are connected to all following filter elements according to the instance relations. This behavior is shown in Fig. 5.8.

Mapping of internal inbound and outbound collectors. The incoming and outgoing data streams of a device are modeled by internal inbound and outbound collectors in the INSTANCE model. For each topic at those collectors, an OR filter is instantiated that combines the data streams of a certain topic. This models the combination of all data streams before these are distributed to the software component instances as well as the combination before the instances are forwarded to the communication stacks. It re-assembles the data-centric handling of data, where samples from different sources, or to different sinks, are not distinguished.

Mapping of incoming and outgoing data streams from and to the network. For outgoing data streams to the network, the ordering of messages, the communication stack delay and properties introduced by the communication technology have to be considered. As this is technology dependent, mappings are presented for the available communication methods of our approach: For all outgoing ports connected to a **switched Ethernet**, an MD service

Property influencing timing	ANALYSIS element
computational capacity	full service (FS) source element
software components	greedy processing components (GPC)
component precedence	graph of service streams
interrupts	fixed-priority non-preemptive (FPNP) components
transmission rate, comm. stack delay	maximum-delay (MD) service source element
queuing delay	first-in-first-out (FIFO) elements
communication priority	chain of FIFO elements
multi-core architectures	n/a (not covered)

Table 5.2: Properties that influence the timing behavior of processing units and their according elements in an ANALYSIS model.

source is instantiated with the available data rate and a parameter to model the communication stack delay. All outgoing instances are connected to a FIFO filter to account for the queuing delay of the messages. The filter itself is connected to the MD service source. The transformation for **switched Ethernet with priority** is almost similar, except that one FIFO filter is instantiated for each priority and the filters are chained according to the priorities. The delay parameter of the MD service source is increased by the maximum blocking time of a packet to compensate the preemptive semantics of the FIFO filter chaining. This modification compensates the semantics of the model: the preemption of a low-priority message by a high-priority message, which is usually not possible in practice. If the outgoing port is connected to a **CAN bus**, a transformation is not performed as the complete behavior is captured later during the transformation of the corresponding network element. On the devices, hardware usually handles transmission and reception of messages, which means no additional delays have to be taken into account. For a communication via a **serial bus**, an MD service source to model the data rate restriction and stack delay is instantiated as well as a FIFO filter to account for the queuing delay of the messages. This is the same transformation as for the switched Ethernet. However, the transformation of the network element is different for both cases, which is explained later (Sec. 5.3.4).

Discussion of modeling properties, assumptions and possible extensions. The shown mappings are based on some implicit assumptions of the technical architecture of the platform and the behavior of the communications stacks and operating systems. In the following, the captured and open points are discussed and possible variations in the mapping to adapt to other architecture concepts. An overview of properties that influence the timing behavior is given in Tab. 5.2. For our approach, we only consider processing units with *one available core* at a *fixed speed*. It is possible to extend the approach to also cover delays introduced by shared resource access on multi-core platforms. For example, [113] and [114] have implemented this within the Real-Time Calculus framework. The consideration of adaptive processing speeds was introduced by [115]. However, as the best-case and worst-case execution times model the resource demand in means of processing cycles, execution times can be given independently from the actual processing speed of the devices in the proposed approach. For simplicity reasons, *interrupts* are always mapped to non-preemptive filters in our approach, which does not always model the reality correctly.

Interrupt service routines might preempt each other and have another counterpart within the common program flow, which gets activated by the initial interrupt service routine. The *system tick* behavior of an operating system can influence the execution point of components. Especially for periodic tasks, it might be the case that the start time is affected by the granularity of the system tick frequency. This introduces a possible additional delay that can be accounted for by a change of the FS service source element stemming from the transformation of the ECU modeling element to an MD service source, which can reflect an additional delay. For simplicity, this was omitted in the presentation.

5.3.3 Mapping of aggregates

Compared to processing units, only external behavior is specified and modeled for aggregates. For the mapping, aggregates are handled like processing units, but internal relations are not resolved. In the current implementation of our approach, it is not possible to model the dependencies of the virtual components inside aggregates, this could be a possible extension. The exact best- and worst-case execution-times of the internal event chains of aggregates are usually not known but could be given either as a possible range representing an assumption or simply as a causal relationship.

5.3.4 Mapping of network elements

Network elements abstract the data transfer between processing units and/or aggregates. Primarily, network elements define the access strategy to the communication medium but may also represent infrastructure elements (e.g., for switched Ethernet). The following points of network elements are considered for an ANALYSIS model: In case the communication is based on infrastructure elements, the *processing delay* influences the timing of messages. Depending on the technology, *arbitration* schemata have to be considered. The *transmission delay* of network elements describes time needed to push all bits into the wire. *Queuing delay* defines the time the transmission is blocked by messages of an equal priority and the *communication precedence* accounts for the preference of messages according to their priorities. These definitions are adapted from [116, 117]. The concrete mapping is given below for the network elements available in our approach.

Mapping of switched Ethernet. Simplified, packets processed by an Ethernet switch are subject to a delay caused by the input buffer, the switching fabric and the output buffer [118]. In practice, input buffer delay is not relevant as the processing capacity of a network switch is equal or higher to the maximum possible load caused by the input connections. In our approach, a generic latency can be given per network element that combines possible delays. To capture the transmission rate and the generic latency, an MD service source element is instantiated per outgoing port of the switched Ethernet model. A FIFO stream filter element is attached to this service source and all packets that are transmitted via the output port pass the filter, which models the queuing of the packets according to the first-in-first-out semantic. The suggested approach follows the modeling of [119] (omitting priorities), which itself was derived from a comparison of modeling variants in [118] and the adaption to the Real-Time Calculus. An example for the mapping is visualized in Fig. 5.9 and a summary of the modeling artifacts to represent the timing effects is given in Tab. 5.3.

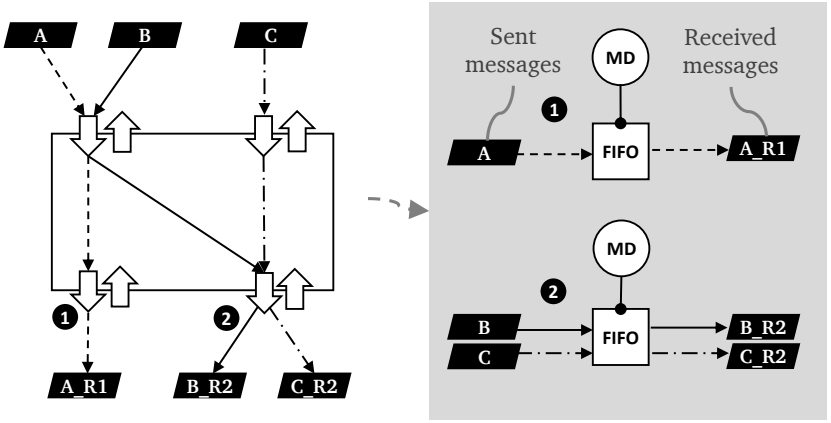


Figure 5.9: Mapping of a switched Ethernet network element of an INSTANCE model to elements of an ANALYSIS model. Messages queue up in the outgoing ports if several are sent at the same time. This behavior is modeled with FIFO elements.

Property influencing timing	ANALYSIS element
processing delay, transmission delay	maximum-delay (MD) element
queuing delay	first-in-first-out (FIFO) element

Table 5.3: Properties that influence the timing behavior of switched Ethernet infrastructure elements and their according elements in an ANALYSIS model.

Mapping of priority-based switched Ethernet. Compared to the modeling of a switched Ethernet element without priorities, the following differences exist in the mapping: For each priority and output port, one FIFO filter element is instantiated and all packets of the according priority traverse this element. The filter elements are chained according to their priorities to model the precedence of messages. The delay parameter of the MD service source element is increased to account for the possible blocking by non-preemptable packets with lower priority (head-of-line blocking) [119]. An example for the mapping is shown in Fig. 5.10 and the artifacts influencing the timing are listed in Tab. 5.4.

Mapping of controller area network. Messages sent via a controller area network (CAN) bus [108] are arbitrated according to a fixed-priority non-preemptive strategy. Each message is assigned an identifier (ID) that defines the priority of the message. A value of 0 for the identifier corresponds to the highest possible priority. No two senders may send the same identifier. The arbitration is performed locally for each transceiver individually, which means that no network infrastructure exists. The CAN bus implements a multicast behavior, because all nodes attached to the bus receive all sent messages. We do not regard it as a broadcast bus for the analysis, since the hardware is usually able to filter incoming messages according to the identifiers and therefore only lets required messages pass. Messages with identifiers that are not of interest for a particular node are not forwarded to the communication stack and do not occupy processing capacity. The concrete mapping is defined follows: A maximum-delay (MD) service source element is instantiated per CAN

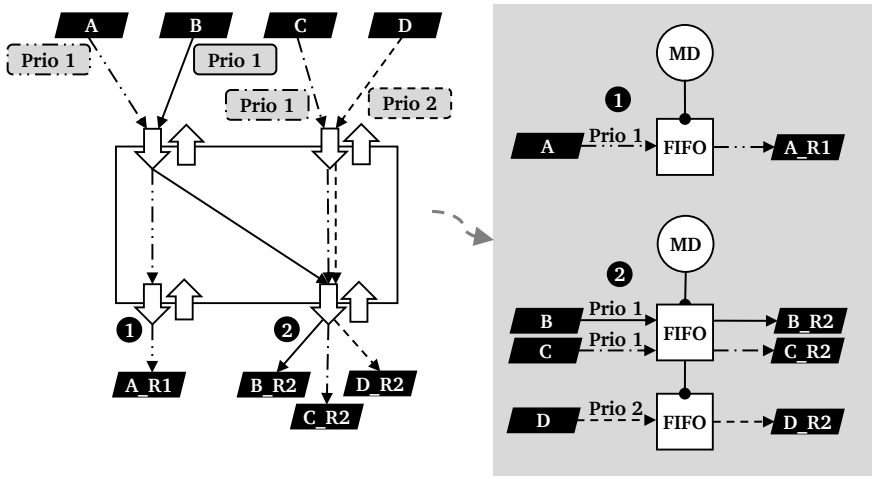


Figure 5.10: Mapping of a priority-based switched Ethernet network element from an INSTANCE model (left side) to elements of an ANALYSIS model.

Property influencing timing	ANALYSIS element
processing delay, transmission delay, head-of-line blocking	maximum-delay (MD) element
queuing delay	first-in-first-out (FIFO) elements
communication precedence	chain of FIFO elements

Table 5.4: Properties that influence the timing behavior of priority-based switched Ethernet elements and their according elements in an ANALYSIS model.

bus to represent the transmission rate and possible delays. For each sent message of the bus, one fixed-priority non-preemptive (FPNP) filter is instantiated. The filters are chained according to the priorities of the messages. Each FNPN filter has one input connection (the sending entity) and possibly multiple output connections, according to the amount of receivers. This follows from the finding, that CAN buses can be modeled by FPNP filters, pointed out in [120] and [121]. An example of the mapping is shown in Fig. 5.11 and the applied ANALYSIS elements are given in Tab. 5.5.

Mapping of serial bus. In our context, a serial bus is a bidirectional point-to-point connection between devices. It is assumed that messages are queued within a node and sent in a first-come-first-serve manner, which is already handled by the communication stack mapping within the processing units transformation. For a serial bus, a network infrastructure does not exist and hence does not influence the communication timing. Data instance connections are simply mapped to arrival connections in the ANALYSIS model as shown in Fig. 5.12.

5.3.5 Mappings of further communication technologies

Certainly, the shown mappings cover only a few technologies, which can be modeled and analyzed within the Real-Time Calculus framework. In the following, a selection of fur-

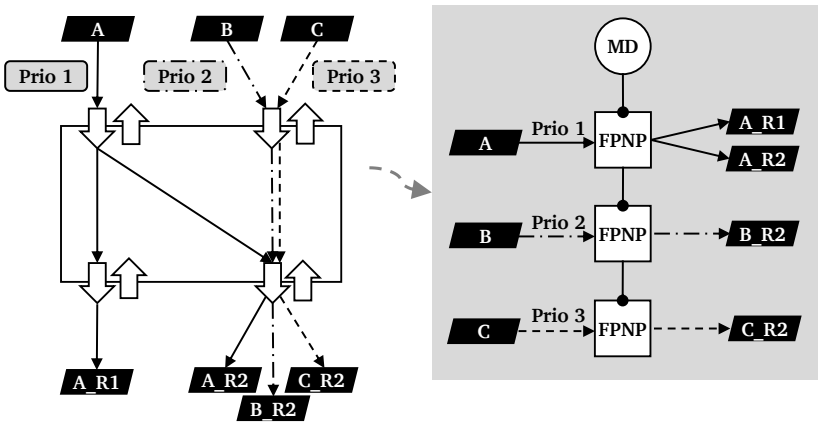


Figure 5.11: Mapping of a CAN bus from an INSTANCE model (left side) to elements of an ANALYSIS model.

Property influencing timing	ANALYSIS element
transmission delay, other delays	maximum-delay (MD) element
processing delay	not applicable
arbitration	fixed-priority non-preemptive (FPNP) elements
communication precedence	chain of FPNP elements

Table 5.5: Properties that influence the timing behavior of a CAN bus and their according elements in an ANALYSIS model.

ther transformations is given that could be integrated into the approach. As pointed out in [122], the Audio-Video Bridging (AVB) [123] standard is a candidate for a future communication technology within vehicles. A communication system according to the AVB standard offers possibilities to guarantee rate and latency of data streams across several network elements. Queck [124] has shown how the Network Calculus can be used to derive the performance properties of such a network in the automotive context and in [125] a similar analysis was conducted with the Real-Time Calculus framework. The parameters needed can directly be extracted from the presented models of our approach. For the successor of AVB, called Time-Sensitive Networking (TSN) according to the IEEE task group[1], a formal analysis within the Real-Time Calculus or Network Calculus frameworks is not yet known to the author of this work. FlexRay is another protocol deployed in vehicles that guarantees fixed delays by a time-triggered transfer of messages. The analysis within the Real-Time Calculus framework was shown in [126] and later refined in [127]. Although FlexRay is available as an ISO standard[2], the future is fuzzy as the consortium disbanded in 2009. For the protocols Media-Oriented Systems Transport[3] (MOST) and Local Interconnect Network (LIN), no approaches for a formal analysis within the Network Calculus or Real-Time Cal-

[1] http://www.ieee802.org/1/pages/tsn.html, accessed 30-10-2015
[2] ISO standards 17458-1 to 17458-5
[3] http://www.mostcooperation.com, accessed 30-10-2015

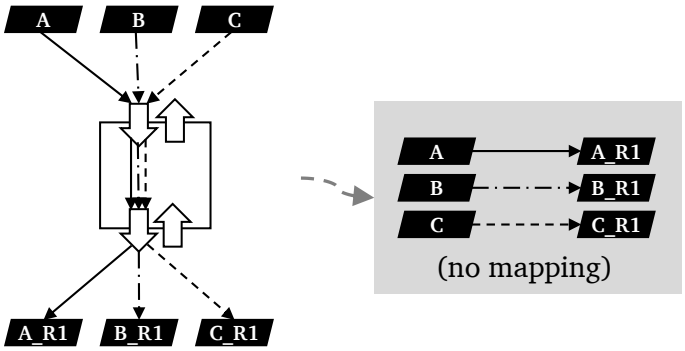


Figure 5.12: Mapping of a serial bus from an INSTANCE model (left side) to elements of an ANALYSIS model.

culus frameworks are known to the author of this work. While LIN is not further developed and under transfer into an international standard[1], the consortium of MOST is still active.

5.3.6 Mapping of timing requirements

The resolving of relative references, which is the most complex part of the timing requirements mapping, was already conducted during the transformation of a combined FEATURE model to an INSTANCE model, see Sec. 5.2. The mapping of these timing requirements from an INSTANCE model to an ANALYSIS model is a straightforward process and the artifacts are simply copied into the resulting model.

5.4 Model transformation and platform mapping summary

The transformations and mappings shown in this chapter are an important step to enable the deployment and timing verification of systems that are capable of system-wide plug-and-play. It enables the composition and verification of loosely-coupled systems with initially fuzzy timing requirements. It was described how a system representation according to a FEATURE-SET model can be step-wise transformed to a combined FEATURE model, an INSTANCE model and finally an ANALYSIS model, where the latter one is a representation suitable for an analysis within the Real-Time Calculus framework. For the transformation to an INSTANCE model, several sub-steps were introduced that map the loose coupling of senders and receivers according to the data-centric approach to a concrete representation. It was also discussed how relative timing requirements are mapped to absolute references during the transformation. For the transformation to an ANALYSIS model, several strategies were given how the abstract representation for the processing units and communication variants are mapped to concrete technologies.

[1] ISO 17987 Part 1-7, as stated at http://www.lin-subbus.org, accessed 30-10-2015

Chapter 6

Timing verification framework

This chapter introduces the timing verification framework and how it is applied in the system-wide plug-and-play approach. The verification framework analyzes the effects of filters, representing processing entities, on data streams through a system. It considers the interaction of processing elements and is able to derive delay bounds on data instances of particular streams. The verification itself is based on the Real-Time Calculus [30, 77] and extends it by an automatic analysis and verification process. The objective is, starting from a system description and timing requirements specification according to an ANALYSIS model, to check if timing bounds are met by the running system. The verification framework forms an essential part of the system-wide plug-and-play approach and makes a modular configuration possible while guaranteeing timing requirements. The RTC was chosen as the mathematical backbone because it allows an analysis of distributed, event-based, heterogeneous systems with hierarchical scheduling strategies. It also allows an adaption of result tightness for analysis runtime, which is further detailed in Ch. 7. The main contributions of this chapter include the specification of the automatic verification process, a novel handling of cyclic resource dependencies in the system graph, a simplified derivation of a closed-form solution of the subadditive closure for bounded buffer handling, and the evaluation of different heuristics for the combination of intermediate results of the subadditive closure.

The verification framework developed in this thesis presents novel approaches to handle cyclic resource dependencies, closures, and approximations. A selected overview and comparison to other implementations is presented in Appx. A.1.5.

6.1 Refined ANALYSIS meta-model ($\mathfrak{M}$)

Parts of the ANALYSIS meta-model were already introduced in Sec. 5.3. Now, their description is refined and constraints of the model are explained.

Definition 6.1 *An* ANALYSIS *model* $\mathfrak{M} = (\mathfrak{A}, \mathfrak{B}, \mathfrak{F}, \mathfrak{E}, \mathfrak{R})$ *consists of a set of arrival sources* $\mathfrak{A}$*, a set of service sources* $\mathfrak{B}$*, a set of stream filters* $\mathfrak{F}$*, a set of edges* $\mathfrak{E}$*, and a set of timing requirements* $\mathfrak{R}$*. Each edge* $e \in \mathfrak{E}$ *is unidirectional and described by* $e = (e_{from}, e_{\#from}, e_{to}, e_{\#to}, e_T)$*, where the type of edges* $e_T \in \{Arrival, Service\}$ *comprises arrival or service relations. The slot numbers* $e_{\#from}$ *and* $e_{\#to}$ *define the input or output position at a filter if multiple positions*

are possible, e.g., for FIFO filter elements, which can have an unlimited amount of input and output streams. Each filter $f = (f_T, f_{\mathbb{P}}, f_{\mathbb{I}}) \in \mathfrak{F}$*, arrival source* $a = (a_T, a_{\mathbb{P}}) \in \mathfrak{A}$ *and service source* $b = (b_T, b_{\mathbb{P}}) \in \mathfrak{B}$ *is described by its type (*a_T*,* b_T*,* f_T*) and a set of parameters (*$a_{\mathbb{P}}$*,* $b_{\mathbb{P}}$*,* $f_{\mathbb{P}}$*) that depend on the type. In addition, each filter* f *has a set of internal relations (*$f_{\mathbb{I}}$*). The set of all nodes* $\mathfrak{V}$ *in the model* $\mathfrak{M}$ *is defined as* $\mathfrak{V} = \mathfrak{A} \cup \mathfrak{B} \cup \mathfrak{F}$*. A requirement* $u \in \mathfrak{R}$ *is defined as* $u = (u_{from}, u_{\#from}, u_{to}, u_{\#to}, u_T, u_{\mathbb{P}})$ *with the available types* $u_T \in \{Latency, Pattern\}$ *and the parameter set* $u_{\mathbb{P}}$*. The other variables of u are similar to those in the definition of an edge e.*

For the constraints, we define the following operator: The number $|v|^{\{in,out\}}_{\{\alpha,\beta\}} : v \in \mathfrak{V} \mapsto \mathbb{N}$ represents the number of incoming (in) or outgoing (out) arrival (α) or service (β) edges for a node v. If the description of the direction or type is omitted, v refers to the union of options. The model $\mathfrak{M}$ has to fulfill the following properties: All service sources and all arrival sources are connected ($|a|^{out}_{\alpha} \geq 1 \; \forall a \in \mathfrak{A}$ and $|b|^{out}_{\beta} \geq 1 \; \forall b \in \mathfrak{B}$). Arrival edges must not start from service source nodes and vice versa ($|a|^{out}_{\beta} = 0 \; \forall a \in \mathfrak{A}$ and $|b|^{out}_{\alpha} = 0 \; \forall b \in \mathfrak{B}$). No incoming connections to service and arrival sources are allowed ($|v|^{in} = 0 \; \forall v \in \mathfrak{A} \cup \mathfrak{B}$). Each stream filter f has at least one incoming arrival relation ($|f|^{in}_{\alpha} \geq 1 \; \forall f \in \mathfrak{F}$) and zero to one incoming service relations ($|f|^{in}_{\beta} = \{0, 1\} \; \forall f \in \mathfrak{F}$, depends on filter type). No two edges must share the same incoming slot of a filter ($\forall (x, y) \in \mathfrak{E}, x \neq y : x_{\text{to}} = y_{\text{to}} \implies x_{\#\text{to}} \neq y_{\#\text{to}}$).

6.2 Analysis and verification procedure

The analysis and verification engine processes a system specification according to an ANALYSIS model ($\mathfrak{M}$). It calculates the end-to-end timing behavior of the modeled system, compares the results with the given requirements and generates a report for further processing. This is performed in several steps: Detection of cycles in the system graph, execution order calculation, analysis execution, result collection, and requirements checking. These steps are detailed below.

1. Cycle detection. To analyze a model $\mathfrak{M}$ with the Real-Time Calculus, an execution order of the stream filters has to be defined. A filter can be executed once the characteristics of all input arrival and input service streams are known. If the dependencies between the filters form a directed acyclic graph (DAG), an order can be calculated and one filter after each other is evaluated. A different approach is needed if the input of a specific filter depends on its output and hence the dependencies form a cycle. In that case, the analysis strategy is changed to a fixed point calculation. We distinguish three cycle configurations:

- *Resource (arrival-service) cycles* are cycles that partly consist of one or more arrival connections and partly of one or more service connections. This is a common situation, for example, if two tasks share the same processing resource and a lower-priority task forwards data to a higher-priority task. In that case, the available resources of the lower-priority task are only known after the evaluation of the higher-priority task. But, that can only be calculated if the incoming event characteristics of the higher-priority task are known, which in turn is dependent from the lower-priority task. This forms a cycle in the analysis process that has to be resolved.

- *Data flow (arrival-arrival) cycles* are cycles within the arrival connections of the analysis graph. If the number of events is limited or joined, for example by an AND-filter or a filter with bounded input buffer, such constructs are analyzable, otherwise the number of events might grow to infinity. These kind of cycles were not further considered for the implementation of the verification framework; a model $\mathfrak{M}$ with such a cycle is considered invalid.

- *Service-service cycles* refer to a cycle of service connections within a graph. This concept was applied in [81] to model proportional share scheduling within one stream filter with multiple inputs, where remaining resources after processing of one event stream are transfered to another stream. A generic relevance across filter boundaries is doubtful to the author of this work. In our framework, those cycles are detected and the system is considered as an invalid configuration.

In order to detect cycles, the strongly connected components [128] of the model $\mathfrak{M}$ are calculated. The strongly connected components are a partition of the vertices $\mathfrak{V}$ of the original graph $\mathfrak{V} \cup \mathfrak{E}$ into strongly connected subgraphs $\mathfrak{S}_i$, where $\bigcup_i \mathfrak{S}_i = \mathfrak{V}$. A subgraph is strongly connected if all vertices are reachable from each other one. This especially means that all cycles are within the subgraphs in the end and that the relations between subgraphs themselves form a DAG, as visible in the examples of Figs. 6.4 and 7.9. For implementation, the classic algorithm proposed by Tarjan [128] is applied.

2. Analysis order. After the system graph was partitioned into a DAG consisting of subgraphs $\mathfrak{S}_i$, the execution order can be determined. This is done by topological sorting [129] of the subgraphs $\mathfrak{S}_i$, with all edges of $\mathfrak{E}$ that connect the subgraphs. Depending on the concrete graph, multiple solutions may exist. This is not a problem as it will not affect the output of the analysis process nor change the analysis complexity. For implementation, the classic algorithm according to Kahn [129] is utilized. This processing step always succeeds as the partitioning step guarantees that the graph is a directed acyclic graph.

3. Analysis execution. The analysis of the subgraphs $\mathfrak{S}_i$ is conducted in the order of the topological sort. This guarantees that all incoming arrival and service streams are known before the subgraph is processed. If only a single filter is part of the subgraph, i.e., $|\mathfrak{S}_i| = 1$, then the analysis is as follows:

(a) *Schedulability check:* Based on the input service and arrival curves, the schedulability of the filter is checked. Filters that do not depend on service resources (e.g., logical OR/AND of data streams) always have a positive schedulability result. If the schedulability check fails, the outgoing service and arrival curves are marked as invalid.

(b) *Filter process:* In this step, the actual filtering of the arrival streams considering the available service takes place. If the schedulability check fails, this step is not executed. Depending on the concrete execution semantic of the filter, the maximum delays and backlogs of the streams are calculated.

(c) *Result propagation:* The derived results are forwarded to the successor elements of the current filter for further processing. This includes the arrival as well as the service curves.

If the current subgraph $\mathfrak{S}_i$ contains cyclic dependencies, i.e., $|\mathfrak{S}_i| > 1$, the execution is different and explained further in section 6.4.

4. Collection of results. After all filters are processed, the results are collected. Starting from each arrival source, the network graph is traversed and the latencies introduced by each filter are accumulated. The results are stream descriptions that include the latency introduced at each filter and the overall latency of the stream. The traversal of a stream stops if the outgoing slot of a filter is not connected or the filter itself forms a sink. As streams might be forwarded to several elements, the descriptions may branch and form a tree in the end; see Fig. 7.5 for an example representation for the eCar instance.

5. Requirements checking. As the last step, the requirements are matched against the collected results. The requirements define either bounds for latencies or characteristics of the event streams at certain points that can be compared to the results. The result of the requirements checking is binary: Either a model $\mathfrak{M}$ fulfills a requirement according to the analysis or it fails to do so. However, because the analysis is an approximation, requirements might still be met by the real-world system even if the analysis contradicts.

6.3 Implementation

The verification concepts were implemented in a Java-based tool, a screenshot is shown in Fig. 6.1. This section elaborates the implementation details.

6.3.1 Curve representation

Representation of curves in the verification tool is analogous to the approach used in [130] and [131]. The representation is based on segment-wise defined curves that belong to the class of wide-sense increasing functions ($\mathbb{F}$) with ultimately pseudo-periodic behavior. The description of curves is split into an aperiodic and a periodic part, an exemplary visualization is provided in Fig. 6.2.

Definition 6.2 *Formally, an ultimately pseudo-periodic curve C is a tuple*

$$C = (\mathbb{S}_a, \mathbb{S}_p, c_{x0}, c_{y0}, c_{\Delta x}, c_{\Delta y}) \tag{6.1}$$

with two parts: A set of segments $\mathbb{S}_a$ defining the aperiodic part of the curve from 0 up to c_{x0}, and a set of segments $\mathbb{S}_p$ for the periodic part that defines a pattern, which is repeated from the point (c_{x0}, c_{y0}) on. Parameters of the pattern include the translation in x- and y-direction $(c_{\Delta x}, c_{\Delta y})$ during each repetition step. The periodic or aperiodic part can be omitted ($\mathbb{S}_a = \emptyset$ or $\mathbb{S}_p = \emptyset$).

Definition 6.3 *Each segment $s \in \mathbb{S}_a, \mathbb{S}_p$ is described as a four-tuple $s = (s_m, s_n, s_{min}, s_{max})$ that reflects the parameters of the standard line equation, where the inclusion or exclusion of the starting and ending point depends on the type of the curve (upper or lower):*

$$s(\Delta) = \begin{cases} s_m \cdot \Delta + s_n & \forall s_{min} \leq \Delta < s_{max} \quad \text{for upper curves } (\alpha^u, \beta^u) \\ s_m \cdot \Delta + s_n & \forall s_{min} < \Delta \leq s_{max} \quad \text{for lower curves } (\alpha^l, \beta^l) \end{cases} \tag{6.2}$$

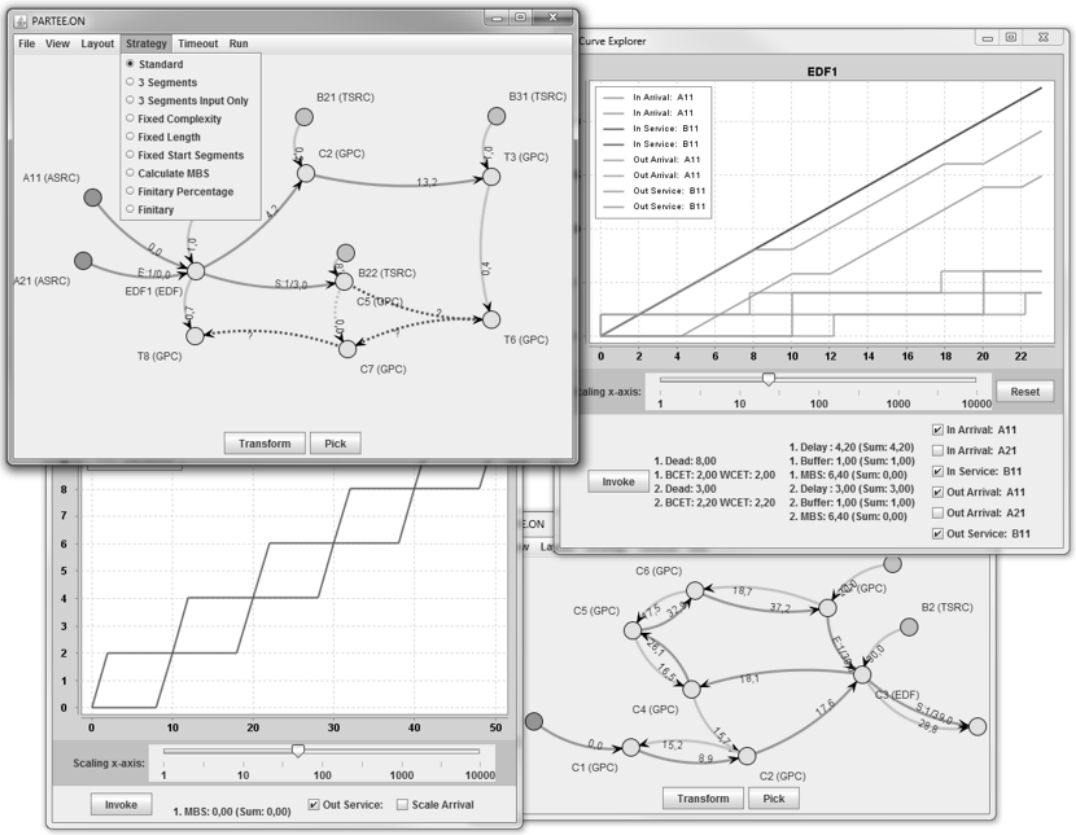


Figure 6.1: Screenshot of our verification tool, showing representations of graphs of ANALYSIS models (top left and bottom right) and arrival/service curve representations at stream filter inputs and outputs (top right and bottom left).

Given a set of segments $\mathbb{S}$, the functions $c(\Delta)_{\{a/p\}}$, which represent the aperiodic and periodic parts of C in the time interval domain, are defined as the union of all relevant segments:

$$c_{\{a/p\}}(\Delta) = \bigcup_i s_i(\Delta) \quad \forall s_i \in \mathbb{S}_{\{a/p\}} \tag{6.3}$$

Definition 6.4 *Given a curve description C, the complete curve can be reconstructed in the time interval domain with the* unfolding *operation, where the aperiodic part is kept and the periodic part is constructed by the union of shifted segment sequences of the periodic part:*

$$\begin{aligned} c(\Delta) &= c_a(\Delta) && 0 \leq \Delta < c_{x0} \\ c(\Delta) &= \bigcup_{i \in \mathbb{N}_{\geq 0}} \left(c_p(\Delta - c_{x0} - i \cdot c_{\Delta x}) + c_{y0} + i \cdot c_{\Delta y} \right) && \Delta \geq c_{x0} \end{aligned} \tag{6.4}$$

The process is analogous to the building method in [77], but adapted to the segment definition in this work. Because the unfolding until infinity is not feasible for processing, a

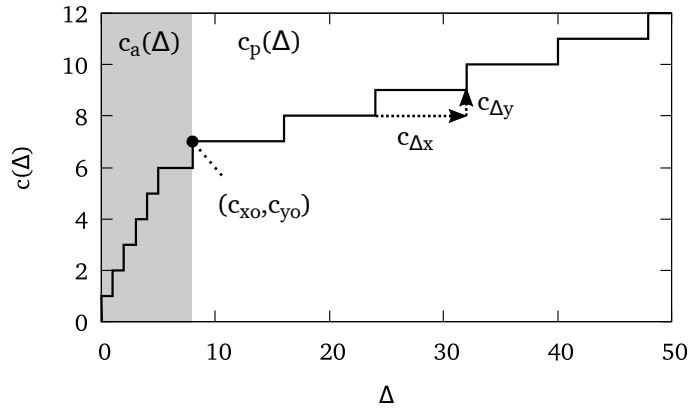


Figure 6.2: Unfolded representation of an ultimately pseudo-periodic curve C with the aperiodic part $c_a(\Delta)$ and periodic part $c_p(\Delta)$. The periodic part starts at point (c_{x0}, c_{y0}) and the repetitions have the offset $(c_{\Delta x}, c_{\Delta y})$.

certain point Φ is determined up to which the unfolding is carried out. This point depends on the operation and on the parameters of the curve(s).

Definition 6.5 *The unfolding of a curve C up to a point Φ is given by the function*

$$c(\Delta)_\Phi = \begin{cases} c(\Delta) & \forall \Delta \leq \Phi \\ \text{undefined} & \forall \Delta > \Phi \end{cases} \tag{6.5}$$

An important property of a curve C is its long-term slope C_ρ, which describes the rate of events or the availability of resources for an infinite-long observation [73] and always exist for super- or subadditive functions [87]. The long-term slope is required to check the schedulability of stream filters and to execute curve operations.

Definition 6.6 *The long-term slope C_ρ of a curve C equals*

$$C_\rho = \lim_{\Delta \to \infty} \left(\frac{c(\Delta)}{\Delta} \right) \tag{6.6}$$

In case $\mathbb{S}_p \neq \emptyset$, the long-term slope is $C_\rho = \frac{c_{\Delta y}}{c_{\Delta x}}$. In case $\mathbb{S}_p = \emptyset$, the long-term slope equals the slope of the last segment of the aperiodic part: $C_\rho = s_{m,i}$, where $i = |\mathbb{S}_a|$ and $s_i \in \mathbb{S}_a$.

Comparison to curve representations in related works. Wandeler [77] defines a different representation for segments. Instead of four parameters $s = (s_m, s_n, s_{min}, s_{max})$, only three parameters are used: A start point of the segment in x- and y-direction (x_0, y_0) and a parameter s_w for the slope of the curve, giving the tuple (x_0, y_0, s_w). The length of a segment is defined by the start of a successor element in the ordered list of segments for the aperiodic and periodic parts. We did not use this concept, because it cannot describe finite curves or gaps in the definition. Gaps do usually not appear in the results of the basic operations, but may exist in intermediate representations, for example during the calculation of the min-plus convolution. Except for the representation of individual segments, a

Unary Operators	Binary Operators
Ceiling/Floor ($\lceil\ \rceil$,$\lfloor\ \rfloor$)	Min-/Max-plus convolution ($\otimes$, $\overline{\otimes}$)
Subadditive/Superadditive closure ($x^\star$,$x^{\overline{\star}}$)	Min-/Max-plus deconvolution ($\oslash$, $\overline{\oslash}$)
Scaling ($\cdot$)	Minimum/Maximum (min/$\wedge$, max/$\vee$)
Shifting ($(c(\Delta - x_0) + y_0)$	Addition/Subtraction ($+$, $-$)
	Vertical/Horizontal distance ($B()$, $D()$)

Table 6.1: Basic operations for curves within the Real-Time Calculus.

curve is described in [77] similar to Eq. 6.1. On the other hand, [131] utilizes also four parameters to represent a segment: $t_i = (x_i, f(x_i), f(x_i+), \rho_i)$, where x_i and $f(x_i)$ define the starting point, $f(x_i+)$ the y-coordinate directly right of the starting point and ρ_i the slope of the segment. Consequently, it is possible to define a gap between the starting point and the actual segment. We did not use this representation, because the interpretation if a segment is right- or left-continuous is implicitly applied in our approach. An ultimately pseudo-periodic curve is described in [131] by three additional parameters for an ordered list of segments: The x-coordinate, where the periodic part starts, and the offset in x- and y-direction of the periodic part. This definition was re-defined in [132], where a curve is represented as $c = v_a \wedge (v_p \otimes r^\star)$, which is inspired from [86]. We will use this representation later on for the calculation of the subadditive closure because it simplifies the implementation (Sec. 6.5.1).

6.3.2 Basic operations

The basic operations of the Real-Time Calculus, as implemented in our verification tool, are shown in Tab. 6.1. These operators form the mathematical backbone for the calculations and are needed to transform streams traversing filters and to derive real-time properties. Because curves are represented with an aperiodic and a periodic part, they are unfolded before the actual operation is executed. Unfolding generates a connected set of segments that represents the curve up to a certain point. The basic operations of the curves were already presented and partly analyzed in [77] and [131], also with the calculation of the corresponding unfolding points. For clarity, the basic algorithm is shown on the example of the min-plus convolution below.

Min-plus convolution of curves (adapted from [77]). Given two curves C_1, C_2 as input, the process of min-plus convolution $C_r = C_1 \otimes C_2$ is divided into five steps, which are described in the following.

1. Calculation of parameters for unfolding of curves. Operations are usually not directly performed on ultimately pseudo-periodic curves because of their infinite definition. A finite representation has to be found for the calculation. Based on the parameters of both curves, the characteristics of the resulting curve are calculated or approximated and are the foundation for the definition of the limit Φ for unfolding. Depending on the operation, the limit Φ depends on the long-term slope C_ρ, periodicity $c_{\Delta x}$ and $c_{\Delta y}$, starting point of the periodic part (c_{x0}, c_{y0}), cross point of both curves Δ_x, or hyper-period hp() of the periods in

x-direction ($\text{hp}(C_1, C_2) = \text{lcm}(c_{\Delta x,1}, c_{\Delta x,2})$, where *lcm* refers to the least common multiple of the arguments.) The parameters in the case of the min-plus convolution can be obtained as follows:

$$c_{\Delta x,r} = \begin{cases} c_{\Delta x,1} & \text{if } C_{\rho,1} < C_{\rho,2} \\ c_{\Delta x,2} & \text{if } C_{\rho,1} > C_{\rho,2} \\ \text{hp}(C_1, C_2) & \text{otherwise} \end{cases}$$
$$\Phi = \begin{cases} \max(c_{x0,1} + c_{x0,2} + \text{hp}(C_1, C_2), \Delta_x + c_{\Delta x,r}) & \text{if } C_{1,\rho} \neq C_{2,\rho} \\ c_{x0,1} + c_{x0,2} + \text{hp}(C_1, C_2) & \text{otherwise} \end{cases} \tag{6.7}$$

In case a curve C does not have a periodic part, i.e., $\mathbb{S}_p = \emptyset$, we set $c_{\Delta x} = 0$ and c_{x0} is equal to the start of the definition range of the last aperiodic segment: $c_{x0} = s_{min,i}$, where $i = |\mathbb{S}_a|$ and $s_i \in \mathbb{S}_a$.

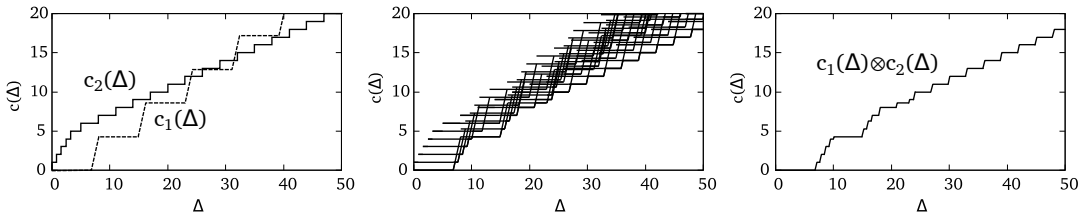
2. Unfolding of curves. In this step, the curves to be processed are unfolded according to Eq. 6.4 up to the previously calculated unfolding limit Φ (Eq. 6.7).

3. Execution of the operation. Usually, the operations can be further divided so that they can be applied segment-wise or range-wise. The min-plus convolution is calculated segment-pair-wise with an additional step of calculating the envelope of the results: Given the segment sets $\mathbb{S} = s_0 \wedge s_1 \wedge ... \wedge s_n$ and $\mathbb{V} = q_0 \wedge q_1 \wedge ... \wedge q_m$ from the unfolding step of both curves C_1, C_2, then the min-plus convolution $\mathbb{S} \otimes \mathbb{V}$ is calculated as:

$$\begin{aligned} r(\Delta) &= \mathbb{S} \otimes \mathbb{V} \\ &= (s_0 \wedge s_1 \wedge ... \wedge s_n) \otimes (q_0 \wedge q_1 \wedge ... \wedge q_m) \\ &= (s_0 \otimes q_0) \wedge (s_0 \otimes q_1) \wedge ... \wedge (s_0 \otimes q_m) \\ &\quad \wedge (s_1 \otimes q_0) \wedge (s_1 \otimes q_1) \wedge ... \wedge (s_1 \otimes q_m) \\ &\quad ... \\ &\quad \wedge (s_n \otimes q_0) \wedge (s_n \otimes q_1) \wedge ... \wedge (s_n \otimes q_m) \\ &= \inf_{\substack{i \leq n \\ j \leq m}} (s_i \otimes q_j) \end{aligned} \tag{6.8}$$

This transformation is possible because of the distributivity property within the min-plus algebra, see [84] and [133] for details. According to Eq. 6.8, all segments of set $\mathbb{S}$ are convoluted with each of set $\mathbb{V}$ and the envelope of all segment-pair-convolutions equals the overall result. In this case, the envelope is the infimum of all piecewise linear segments after the pair-wise convolution operation. The envelope operation is detailed in the next section, an exemplary visualization is shown in Fig. 6.3.

4. Construction of a new curve from the intermediate result. The intermediate result $r(\Delta)$ has to be transformed into a representation C_r according to an ultimately pseudo-periodic



(a) Two exemplary ultimately pseudo-periodic, piecewise linear curves, $c_1(\Delta)$ and $c_2(\Delta)$, as input for the min-plus operator ($\otimes$).

(b) An intermediate result of the min-plus convolution, showing the pair-wise convolution of all relevant segments of c_1 and c_2.

(c) The final result of the min-plus convolution after the envelope calculation of the intermediate result.

Figure 6.3: Example for the envelope calculation during the min-plus convolution.

curve. For this step, the previously calculated parameters are used to divide the result into an aperiodic and a periodic part. For the min-plus convolution, these parameters are:

$$\begin{aligned} c_{\Delta x,r} &= \text{(see Eq. 6.7)} \\ c_{\Delta y,r} &= \begin{cases} c_{\Delta x,r} \cdot C_{\rho,1} & \text{if } C_{\rho,1} \leq C_{\rho,2} \\ c_{\Delta x,r} \cdot C_{\rho,2} & \text{otherwise} \end{cases} \\ c_{x0,r} &= \Phi - c_{\Delta x,r} \\ c_{y0,r} &= r(c_{x0,r}) \\ c(\Delta)_a &= r(\Delta) \qquad\qquad \forall\ 0 \leq \Delta < c_{x0,r} \\ c(\Delta)_p &= r(\Delta + c_{x0,r}) \quad \forall\ 0 \leq \Delta \leq c_{\Delta x,r} \end{aligned} \tag{6.9}$$

The functions $c(\Delta)_a$ and $c(\Delta)_p$ stand for the according set of segments $\mathbb{S}_a$ and $\mathbb{S}_p$ of the result, and $r(\Delta)$ refers to the set of segments of the intermediate pair-wise convolution.

5. Optimization of the result. After each operation in the analysis, the curves are post-processed to remove artifacts caused by numerical issues and to minimize the amount of segments needed for the representation. As the results of this step directly influence the processing time of the following operations, details are deferred to Sec. 7.2.3.

6.3.3 Envelope calculation

The calculation of the envelope is a crucial part of many operations within the min-plus calculus if linear piece-wise segments are deployed as a representation for the curves like in our case. The envelope operation is needed for the convolution, deconvolution, minimum, maximum, and indirectly for the subadditive closure. Depending on the operation, the envelope refers to either the infimum or supremum of a set of segments. For example, the min-plus convolution needs an infimum envelope, an example is shown in Fig. 6.3, and the min-plus deconvolution needs a supremum envelope. Algorithms are known that can find the envelope of n linear piece-wise segments in $O(n \log n)$ time in the single-processor case [134], and in $O(\log n)$ using $O(n)$ processors in the parallel case [135].

The following two sections introduce two concepts that were integrated into our verification tool: The automatic handling of resource cycles in an ANALYSIS model ($\mathfrak{M}$) and the simplified calculation of the subadditive closure for ultimately pseudo-periodic curves defined as segment-wise linear functions, which is needed for the analysis of stream filters with bounded buffer semantics.

6.4 Automatic handling of resource cycles in the system graph

Cycles in an ANALYSIS model ($\mathfrak{M}$) are handled with a special analysis approach. For cycles, a fixed point iteration is conducted. This approach was introduced by [87] for cyclic resource dependencies and in [136, 137] for cyclic data flow dependencies. However, both works are not concerned with an automatic analysis processing and are founded on a manual derivation of the formulas for the calculation. In [138], a framework was presented that is able to handle cyclic resource dependencies by a greedy and recursive construction of the according filter operations by code generation. The approach presented here differs in its automatic, non-recursive processing character, novel strategies for the initialization and iteration steps, and in its deep integration into the system-wide plug-and-play approach for automotive systems. In the following, the handling of cyclic resource dependencies for an automatic processing within the verification tool is described. In general, the fixed point iteration consists of the following steps, where $\Sigma_{\mathfrak{S}_i}$ represents all parameters of a certain processing step for subgraph $\mathfrak{S}_i$, especially, the state of all arrival and service curves:

(a) *Definition of the starting point* $\Sigma^0_{\mathfrak{S}_i}$*:* The starting point $\Sigma^0_{\mathfrak{S}_i}$ defines the initial values of the parameters of the equation. In our case, it refers to the definition of an initial set of arrival and service streams, which are used for the first iteration.

(b) *Definition of the iteration step* $\xi(\Sigma_{\mathfrak{S}_i})$ *:* The iteration step transforms one state of the subgraph into its successor state: $\xi(\Sigma^n_{\mathfrak{S}_i}) \Rightarrow \Sigma^{(n+1)}_{\mathfrak{S}_i}$. Mapped to our case, all filters involved in a cycle are executed. During each iteration, the state $\Sigma_{\mathfrak{S}_i}$ converges closer to the final solution $\Sigma^*_{\mathfrak{S}_i}$.

(c) *Definition of the stop condition* $\chi(\Sigma^n_{\mathfrak{S}_i}, \Sigma^{n-1}_{\mathfrak{S}_i})$*:* The stop condition defines the termination of the iteration; $\chi(\Sigma^n_{\mathfrak{S}_i}, \Sigma^{n-1}_{\mathfrak{S}_i}) \mapsto \mathbb{B}$, where $\Sigma^n_{\mathfrak{S}_i}$ is the current state of the subgraph $\mathfrak{S}_i$ and $\Sigma^{n-1}_{\mathfrak{S}_i}$ refers to the previous iteration state. In the verification tool, the termination is reached once the output curves do not change anymore.

This approach works only if the system converges during the iteration step towards a fixed point Σ^*. As shown by [87], this is always the case for cyclic resource dependencies in the Real-Time Calculus framework. In the following, the individual steps are more elaborated.

Starting point $\Sigma^0_{\mathfrak{S}_i}$. Before the calculation of a subgraph $\mathfrak{S}_i$ begins, all inbound stream references have to be known. Then, we derive the starting point $\Sigma^0_{\mathfrak{S}_i}$ for a certain subgraph $\mathfrak{S}_i$ according to the following strategy: The arrival and service curves available for filters with incoming edges from outside the subgraph are propagated. Depending on the filter type, curves are either directly copied from the input slots to the output slots, or, if an approximation of an outgoing stream is possible, the according operation is executed before

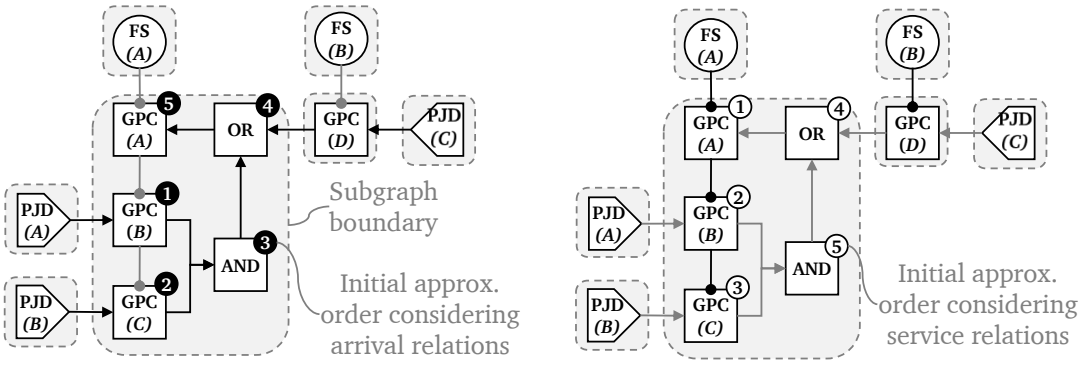


Figure 6.4: Approximation order for the starting point of a fixed point calculation for an example with cyclic resource dependencies. The order is derived by a topological sort of the nodes of the subgraph considering arrival and omitting service relations (left side) and vice-versa (right side). This example is referred to as *complex loop example*.

Node	Parameters	Node	Parameters	Node	Parameters
FS (A)	rate=r_{loop}	PJD (B)	p=10,j=6,d=0.1	GPC (B)	et=4
FS (B)	rate=r_{loop}	PJD (C)	p=12,j=7,d=0.2	GPC (C)	et=2
PJD (A)	p=10,j=5,d=0.1	GPC (A)	et=2	GPC (D)	et=5

Table 6.2: Parameters for the complex loop example (Fig. 6.4). $r_{loop} = \{1.0, 2.0\}$, depending on test case.

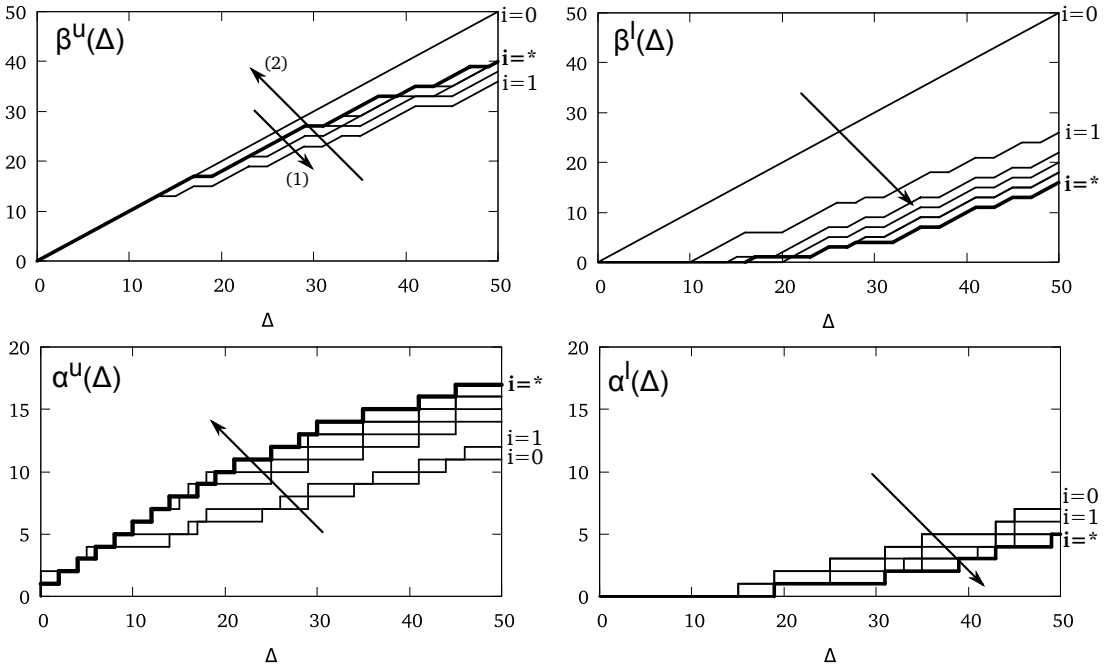


Figure 6.5: Development of arrival and service curves of node *GPC (A)* in example shown in Fig 6.4 with parameters of Tab. 6.2 ($r_{loop} = 1.0$). The indexes right of the graphs refer to the iteration step, where i=* defines the final result. The arrow corresponds to the direction of convergence.

propagation. The last point is valid for OR and AND filters, because they do not depend on service curves. To derive the order for the initialization, two topological sortings with the nodes of the subgraph are conducted. First, sorting is performed considering all arrival relations in the subgraph, second, sorting is performed considering all service relations in the subgraph. Afterwards, both results are combined. Because we require the graph to be free of cyclic data flows and cyclic service streams, an order can always be derived in the proposed way. The corresponding algorithm is sketched in Alg. 6.1 and the initialization order of an exemplary system is presented in Fig. 6.4. With this strategy, long-term rates of the initial arrival curves match the long-term rates of the fixed point solution, but are tighter:

$$\begin{aligned} \alpha^{0,u}_{\rho} &= \alpha^{*,u}_{\rho} \qquad \alpha^{0,u}(\Delta) \leq \alpha^{*,u}(\Delta) \\ \alpha^{0,l}_{\rho} &= \alpha^{*,l}_{\rho} \qquad \alpha^{0,l}(\Delta) \geq \alpha^{*,l}(\Delta) \end{aligned} \tag{6.10}$$

Here, α^0 refers to the initial approximation of the arrival curve, α^* to the fixed point solution and the sub-index ρ to the long-term slope (Eq. 6.6). Service curves are initialized with higher values than the fixed point solution and the long-term slope is different:

$$\begin{aligned} \beta^{0,u}_{\rho} &\neq \beta^{*,u}_{\rho} \qquad \beta^{0,u}(\Delta) \geq \beta^{*,u}(\Delta) \\ \beta^{0,l}_{\rho} &\neq \beta^{*,l}_{\rho} \qquad \beta^{0,l}(\Delta) \geq \beta^{*,l}(\Delta) \end{aligned} \tag{6.11}$$

β^0 refers to the initial approximation of the service curve and β^* to the fixed point solution. The proposed initialization schemata provided reasonable results in our experiments and led to a quick convergence. In contrast, [87] proposed an initialization either by a simulation trace or by an analytical derivation based on the long-term rates, where the latter one is an adaption of the work in [139]. Compared to [138], our approach is non-recursive and approximates filter operations where possible during the initialization, which leads to a tighter starting point, depending on the system configuration.

Iteration step $\Sigma^{(n+1)}_{\mathfrak{S}_i} = \xi(\Sigma^n_{\mathfrak{S}_i})$. After the initialization was done, the iteration phase of the subgraph starts. During each step, filters are invoked in topological order according to the arrival stream and service stream relations, similar to the initialization step. After each step, it is checked if the stop condition is fulfilled. The overall algorithm is sketched in Alg. 6.2. An example for the development of arrival and service streams during the iterations is presented in Fig. 6.5.

Stop condition $\chi(\Sigma^n_{\mathfrak{S}}, \Sigma^{n-1}_{\mathfrak{S}})$. The iteration stops once all arrival and service streams, referred to as $\Sigma^n_{\mathfrak{S}_i}$ for a certain subgraph $\mathfrak{S}_i$ and iteration step n, do not change between iteration steps anymore, expressed by $\chi(\Sigma^n_{\mathfrak{S}_i}, \Sigma^{n-1}_{\mathfrak{S}_i}) = \text{true}$. Two curves are defined to be equal, if their canonical representation has an equal number of segments for the aperiodic and the periodic part, the parameters defining the start and periodicity of the periodic part are equal and all segments of the aperiodic and periodic part match each other. The definition of the canonical representation is deferred, see Eq. 7.1.

6.5 Bounded buffer handling

The semantics of classic Real-Time Calculus include the assumption that processing elements are equipped with infinite buffers. In case of bursts or unavailable service resources,

Algorithm 6.1: Initialization of a fixed point calculation for cyclic resource dependencies (sketch).

Input: SubGraph $\mathfrak{S}_i$
Result: Initialized SubGraph state $\Sigma^0_{\mathfrak{S}_i}$

```
1 FilterList flSorted ← topologicalSortArrivalEdges(S_i);
2 flSorted ← flSorted ∪ topologicalSortServiceEdges(S_i);
3 forall the Filters f ∈ flSorted do
4 |   f.invokeCycleApproximation();          /* updates Σ^0_{S_i} */
5 |   f.propagateStreamsOneStep();           /* updates Σ^0_{S_i} */
6 end
```

Algorithm 6.2: Algorithm for iteration step of fixed point calculation (sketch).

Input: SubGraph $\mathfrak{S}_i$, initial SubGraph state $\Sigma^0_{\mathfrak{S}_i}$
Result: SubGraph fixpoint state $\Sigma^*_{\mathfrak{S}_i}$

```
1 FilterList flSorted ← topologicalSortArrivalEdges(S_i);
2 flSorted ← flSorted ∪ topologicalSortServiceEdges(S_i);
3 Σ^n_{S_i} ← Σ^0_{S_i};
4 repeat
5 |   Σ^{n-1}_{S_i} ← Σ^n_{S_i};
6 |   forall the filters f ∈ flSorted do
7 |   |   f.invokeFilter();                  /* updates Σ^n_{S_i} */
8 |   |   f.propagateStreamsOneStep();       /* updates Σ^n_{S_i} */
9 |   end
10 until χ(Σ^n_{S_i}, Σ^{n-1}_{S_i}) = true;   /* check for fixpoint */
```

incoming events are enqueued before processing. In reality, bounded buffers are deployed frequently in automotive systems for several reasons: Considering real-time control flows, regularly only the most current sample is relevant. For example, if a function requires the current velocity of the vehicle, only the most up-to-date sample is of interest. However, it may be the case that several samples are kept to calculate an average or median of a certain value to mitigate outliers. Due to the heterogeneous structure of the electronic architecture, individual systems work with different processing frequencies. If a sender provides data with a higher rate than the receiver can process it, the buffer of the receiving processing element will eventually overflow if no counter-measurements are provided. Further, the memory of the electronic control units is limited and thus simply cannot handle an infinite amount of data samples. The imprecise modeling of this facts leads to overly conservative timing estimates [140]. In the following, we consider the correct modeling of systems with under-sampling behavior; a receiver processes the data at a slower rate than they are produced by the sender. As a result, packets will be dropped due to the bounded buffer. To handle this issue, the Real-Time Calculus was extended by two approaches:

- *Transformation to stateful analysis tools.* One option is to convert arrival and service curves in a representation suitable for the input of a state-based analysis tool and to re-convert the results of that tool. For example, an interface to connect the Real-Time Calculus with timed automata was described in [73] and [141], and a bridge between RTC and the programming language Lustre was constructed in [142].
- *Integrated approach.* An integrated approach was developed by [93], which natively works in the Real-Time Calculus framework, without the need to transform between different tools. They succeeded to derive a mathematical concept to model the behavior of a bounded buffer. These equations are a mapping of the results in [84, "Losses in a Finite Storage Element"] to the Real-Time Calculus.

The advantage of the approach by conversion is that it can handle complex buffer semantics and over- as well as under-sampling. For example, time variant buffers are possible that drop samples after a certain amount of time. Drawbacks are the expensive transformations of arrival curves and the application of a reachability analysis, which is prone to the state-explosion problem [73]. Further, the transformation does not fit into the proposed verification framework. Especially the approximation approach, which will be introduced in the next chapter, does not fit seamlessly. The integrated approach matches smoothly into the Real-Time Calculus framework but relies on the expensive calculation of the sub-/superadditive closure of curves. It is an approximation of the behavior of finite buffer semantics and was only derived for under-sampling scenarios.

Because the integrated approach is compatible with the approximation strategies and is based on the mathematical background of the min-plus algebra, we have chosen to use it in our verification tool. However, the subadditive closure of an ultimately pseudo-periodic, piecewise linear defined curve is an operation that is, to our knowledge, not available yet in any free tool (see Appx. A.1.5). Hence, we derive the concrete implementation of the subadditive closure in detail in the following. The superadditive closure can be calculated analogously.

6.5.1 Calculation of the subadditive closure

The subadditive closure is an expensive operation, but a closed-form solution exists. The basic algorithm to calculate it was explained in [131], which is based on the decomposition of a curve into segments and iterated segments, and the calculation of the subadditive closure of these parts and a combination of the results. Segments and iterated segments are handled separately in the approach and the calculation of the resulting closures are subject to distinction of many cases. We are going to show that, by another curve representation (adapted from [86]), a distinction between segments and iterated segments is not necessary anymore, which reduces the complexity of an implementation. This concept was already mentioned in [132], but a completion of the concept or an implementation has not been presented in prior works. In contrast, the following derivation shows the complete process how the subadditive closure is calculated for ultimately pseudo-periodic curves based on a compact curve representation, which can directly be implemented in according frameworks. It simplifies the method from [131] because the concept of iterated segments is not necessary and because the derivation is based on a more simple representation of

linear segments. Let us represent an alternative description of ultimately pseudo-periodic functions [132]:

$$v(\Delta) = v_a(\Delta) \wedge [v_p(\Delta) \otimes r^\star(\Delta)] \tag{6.12}$$

Here, v_a is the aperiodic part of the curve, v_p is the periodic part, r describes the offset of repetition of the periodic parts, and $r^\star$ is the union of all possible offsets. The equation can express the identical class of curves to those of Eq. 6.4. The transformation between both representations is given by:

$$\begin{aligned} v_a(\Delta) &= c_a(\Delta) && \forall\ 0 \le \Delta \le c_{x0} \\ v_p(\Delta) &= c_p(\Delta - c_{x0}) + c_{y0} && \forall\ c_{x0} \le \Delta \le c_{x0} + c_{\Delta x} \end{aligned} \tag{6.13}$$

The aperiodic part $v_a(\Delta)$ is equal to Eq. 6.3, but the periodic part $v_p(\Delta)$ is shifted by the offsets. The function for repetition is a single point at the offsets for the periodic part:

$$r(\Delta) = \begin{cases} c_{\Delta y} & \text{if } \Delta = c_{\Delta x} \\ +\infty & \text{otherwise} \end{cases} \tag{6.14}$$

The closure $r^\star$ is effectively a repetition of the point $r(\Delta)$ with an offset of $(c_{\Delta x}, c_{\Delta y})$:

$$r^\star(\Delta) = \begin{cases} i \cdot c_{\Delta y} & \text{if } \Delta = i \cdot c_{\Delta x} \quad \forall i \in \mathbb{N}_{\ge 0} \\ +\infty & \text{otherwise} \end{cases} \tag{6.15}$$

The min-plus convolution of the periodic part with the repeated points $(v_p(\Delta) \otimes r^\star(\Delta))$ re-assembles in the end the complete periodic part of the curve. The minimum operation $(\wedge)$ connects the aperiodic and periodic parts. The subadditive closure of Eq. 6.12 can be transformed [133] to:

$$\begin{aligned} v^\star &= [v_a \wedge (v_p \otimes r^\star)]^\star \\ &= v_a^\star \otimes (v_p \otimes r^\star)^\star \end{aligned} \tag{6.16}$$

In Eq. 6.16, the term v_a already has a finite number of segments and therefore the closure $v_a^\star$ can be calculated. However, $v_p \otimes r^\star$ contains an infinite amount of segments and has to be further refined. According to the definition [133] of the closure operator $(^\star)$, the term equals

$$(v_p \otimes r^\star)^\star = (v_p \otimes r^\star)^{(0)} \wedge (v_p \otimes r^\star)^{(1)} \wedge (v_p \otimes r^\star)^{(2)} \wedge \ldots \tag{6.17}$$

where the upper number in brackets refers to the number of self-convolutions, i.e., $f^{(1)} = f$, $f^{(2)} = f \otimes f$, etc. Knowing that $r^\star \otimes r^\star = r^\star$, the individual summands of Eq. 6.17 become:

$$\begin{aligned} (v_p \otimes r^\star)^{(0)} &= \delta_0 \\ (v_p \otimes r^\star)^{(1)} &= v_p \otimes r^\star \\ (v_p \otimes r^\star)^{(2)} &= (v_p \otimes r^\star) \otimes (v_p \otimes r^\star) = v_p^{(2)} \otimes r^\star \\ &\ldots \\ (v_p \otimes r^\star)^{(n)} &= v_p^{(n)} \otimes r^\star \end{aligned} \tag{6.18}$$

where δ_0 is the identity element ($\delta_0(\Delta) = 0\ \forall \Delta \leq 0$ and $\delta_0(\Delta) = +\infty$ otherwise). This results in:

$$\begin{aligned}(\nu_p \otimes r^\star)^\star &= \delta_0 \wedge (\nu_p^{(1)} \otimes r^\star) \wedge (\nu_p^{(2)} \otimes r^\star) \wedge ... \wedge (\nu_p^{(n)} \otimes r^\star) \\ &= \delta_0 \wedge [r^\star \otimes (\nu_p^{(1)} \wedge \nu_p^{(2)} \wedge ... \wedge \nu_p^{(n)})] \\ &= \delta_0 \wedge \nu_p \otimes [r^\star \otimes (\delta_0 \wedge \nu_p^{(1)} \wedge ... \wedge \nu_p^{(n)})] \\ &= \delta_0 \wedge \nu_p \otimes [r^\star \otimes \nu_p^\star]\end{aligned} \tag{6.19}$$

Inserting Eq. 6.19 into Eq. 6.16 yields our final result:

$$\nu^\star = \nu_a^\star \otimes [\delta_0 \wedge \nu_p \otimes (r^\star \otimes \nu_p^\star)] \tag{6.20}$$

Compared to Eq. 6.16, this representation does only contain the closure of a finite amount of segments ($\nu_a^\star, \nu_p^\star$) and spots ($r^\star$). We will now show how these closures are calculated. The presented equations for the closure of a point and a single segment are analogous to the description of [131], but based on our definition of segments. Assume a segment set $\mathbb{S} = s_0 \wedge s_1 \wedge s_2 \wedge ... \wedge s_n$ with n segments s of the form of Eq. 6.2. Then, the closure of the set $\mathbb{S}$ can be written as:

$$\begin{aligned}\mathbb{S}^\star &= (s_0 \wedge s_1 \wedge s_2 \wedge ... \wedge s_n)^\star \\ &= s_0^\star \otimes s_1^\star \otimes s_2^\star \otimes ... \otimes s_n^\star\end{aligned} \tag{6.21}$$

It is enough to calculate the closure of each segment individually and then combine the results via the min-plus convolution. Equal to spots, the result of a closure of a segment is itself an ultimately pseudo-periodic function, i.e., the definition range of the result spans to infinity. Analog to Eq. 6.17, the closure of a single segment $s \in \mathbb{S}$ is defined as:

$$\begin{aligned}s^\star &= s^{(0)} \wedge s^{(1)} \wedge s^{(2)} \wedge s^{(3)} \wedge ... \\ &= \inf_{k \geq 0} \{s^{(k)}\}\end{aligned} \tag{6.22}$$

Using the definition of the min-plus convolution (Eq. 2.2), the terms $s^{(k)}$ of the closure of a segment s can be directly given as:

$$\begin{aligned}s^{(0)} &= \delta_0 \\ s^{(1)} &= s_m \cdot \Delta + 1 \cdot s_n \quad \forall \Delta \in [1 \cdot s_{min}, 1 \cdot s_{max}] \\ s^{(2)} &= s_m \cdot \Delta + 2 \cdot s_n \quad \forall \Delta \in [2 \cdot s_{min}, 2 \cdot s_{max}] \\ &... \\ s^{(k)} &= s_m \cdot \Delta + k \cdot s_n \quad \forall \Delta \in [k \cdot s_{min}, k \cdot s_{max}]\end{aligned} \tag{6.23}$$

The range intervals grow with each summand by $s_{max} - s_{min}$, which will eventually lead to an overlapping of the definition intervals. To calculate the number of segments until they overlap, it is checked when the starting range of the next segment ($s_{min} \cdot (k+1)$) is lower than the ending range of the current segment ($s_{max} \cdot k$) to get the number of the first overlapping segment k_0 (this is similar to [131]):

$$\begin{aligned}s_{min} \cdot (k+1) &< s_{max} \cdot k \\ k_0 &= \left\lceil \frac{s_{min}}{s_{max} - s_{min}} \right\rceil + 1\end{aligned} \tag{6.24}$$

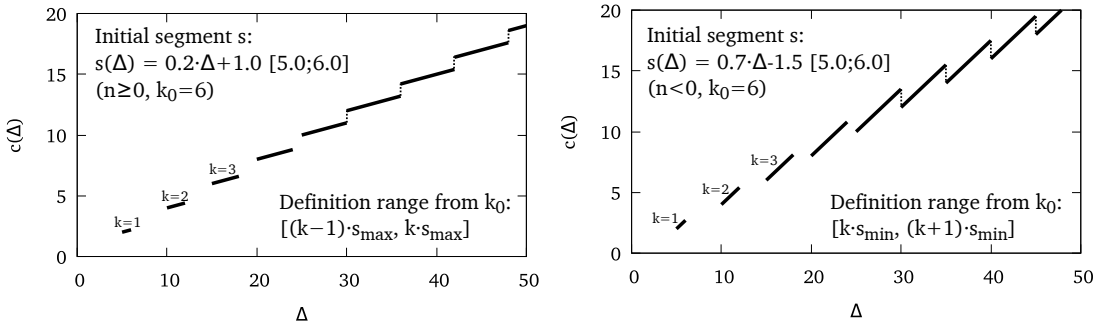


Figure 6.6: Two examples for the subadditive closure of a single segment s. On the left side with $s_n \geq 0$ and on the right side with $s_n < 0$, where s_n refers to the offset of the initial linear segment s for the subadditive closure operation.

With the knowledge of the starting point k_0 for the overlapping of the segments, we derive a representation of the complete curve from that point on. We will see that from k_0 on, the elements are similar and only differ by a translation. The difference of two consecutive summands of the closure is an offset equal to s_n:

$$s^{(k+1)} - s^{(k)} = s_m \cdot \Delta + (k+1) \cdot s_n - (s_m \cdot \Delta + k \cdot s_n) = s_n \tag{6.25}$$

Therefore, neglecting the definition range, the segments $s^{(k)}$ are all parallel lines, where s_n defines if these lines are above or below each other with increasing k. Depending on s_n, two cases are distinguished to construct the complete closure of a segment s (see Fig. 6.6 for an example):

$$s^\star(\Delta) = \bigwedge_{k \in \mathbb{N}_{>0}} \begin{cases} s_m \cdot \Delta + k \cdot s_n & \forall \Delta \in [k \cdot s_{min}, k \cdot s_{max}] \quad \text{if } k < k_0 \\ s_m \cdot \Delta + k \cdot s_n & \forall \Delta \in [k \cdot s_{min}, (k+1) \cdot s_{min}] \quad \text{if } k \geq k_0 \text{ and } s_n < 0 \\ s_m \cdot \Delta + k \cdot s_n & \forall \Delta \in [(k-1) \cdot s_{max}, k \cdot s_{max}] \quad \text{if } k \geq k_0 \text{ and } s_n \geq 0 \\ +\infty & \text{otherwise} \end{cases} \tag{6.26}$$

The closure of the segment ($s^\star$) has gaps for $k < k_0$ and is periodic from $k \geq k_0$. Thus, it is possible to represent the closure as a ultimately pseudo-periodic function similar to Eq. 6.1. The parameters are presented in Tab. 6.3.

6.5.2 Order of artifact combination for the subadditive closure

For the subadditive closure of a finite set of segments, the min-plus convolution of the individual segment closures is necessary (Eq. 6.21). Due to the associativity of the convolution operation, the result does not depend on the order of the operations, but the computation time is affected by the ordering. To the author's knowledge, up to now, no methods have been examined that consider a speedup of the calculations by a re-ordering of the individual results.

A comparison of different heuristics is given in the following, which were empirically evaluated with our verification tool. For evaluation, random curves C_r were generated with the following properties: Each curve contains exactly five segments, where the first

$s_n \geq 0$	$s_n < 0$
$\mathbb{S}_a =$ (see Eq. 6.26, $k < k_0$)	$\mathbb{S}_a =$ (see Eq. 6.26, $k < k_0$)
$c_{\Delta x} = s_{max}$	$c_{\Delta x} = s_{min}$
$c_{\Delta y} = s_m \cdot c_{\Delta x} + s_n$	
$\mathbb{S}_p = \{s(\Delta) = s_m \cdot \Delta \quad \forall \Delta \in [0; c_{\Delta x}]\}$	
$c_{x0} = (k_0 - 1) \cdot c_{\Delta x}$	$c_{x0} = k_0 \cdot c_{\Delta x}$
$c_{y0} = s_m \cdot c_{x0} + k_0 \cdot s_n$	

Table 6.3: Parameters for the construction of an ultimately pseudo-periodic curve for the closure $s^\star$ of a single segment $s(\Delta) = s_m \cdot \Delta + s_n \quad \forall \Delta \in [s_{min}; s_{max}]$.

segment's (s_r^0) parameters are: $s_{r,m}^0 = \kappa()$, $s_{r,n}^0 = \kappa()$, $s_{r,min}^0 = 0.0$ and $s_{r,max}^0 = \kappa()$, where $\kappa()$ refers to a function choosing pseudo-randomly a number of the set {0.0, 0.1, 0.2, ..., 0.9, 1.0}. The other segments (s_r^i, $1 \leq i \leq 4$) are constructed as follows: $s_{r,min}^i = s_{r,max}^{(i-1)}$, $s_{r,max}^i = s_{r,min}^i + \kappa()$, $s_{r,m}^i = \kappa()$, $s_{r,n}^i = (s_{r,m}^{(i-1)} - s_{r,m}^i) \cdot s_{r,max}^{(i-1)} + s_{r,n}^{(i-1)} + \kappa()$, i.e., no definition gap along the x-axis exist, the starting point is higher or equal to the ending point of the previous segment and the slope is randomly chosen.

The following heuristics were compared: *Standard* means that the individual closures of the segment were convoluted in direction of increasing index: $C_r^\star = (((s^{0,\star} \otimes s^{1,\star}) \otimes s^{2,\star}) \ldots \otimes s^{4,\star}$, and *Reverse* refers to a convolution in the opposite direction. The *Random* heuristic selects two curves for convolution randomly until only one remains. *LeastHyperPeriod* choses those two curves with the smallest hyper-period, and *MinMax* selects those for convolution that have the highest and lowest long-term slope until only one curve remains. A graph of the normalized computation times of the experiments is shown in Fig. 6.7 and the results are summarized in Tab. 6.4. Each point in the graph stands for the mean of one test run, where each test run consists of 50 subadditive closures of pseudo-randomly generated curves according to the above definition. 100 test runs were conducted and the measured computation times were normalized in the end correspondent to the *Standard* heuristic. It is visible that the *MinMax* heuristic outperforms all other heuristics for curves fulfilling the above mentioned construction properties – it was faster than the *Standard* approach in 99% of the cases, and faster than any of the other heuristics in 97% of the cases. Over the average of all test runs, the *MinMax* heuristic is 40% faster than the *Standard* approach. This is caused by the fact that this strategy decreases the unfolding point for the convolution in many cases; very differing long-term slopes lead to a cross-point with a small x-coordinate, which is part of the calculation of the range for unfolding (see Eq. 6.7). The range for unfolding influences the number of segments needed for the convolution and hence has a direct effect on the complexity. However, the calculation of the unfolding point can be dominated by the hyper-period of the involved curves for the convolution. In that case, a heuristic following the *LeastHyperPeriod* strategy is significantly faster.

6.6 Discussion of the verification framework

This section discusses the possibilities and limitations in the application of the proposed verification approach based on the Real-Time Calculus (RTC). Because the RTC describes

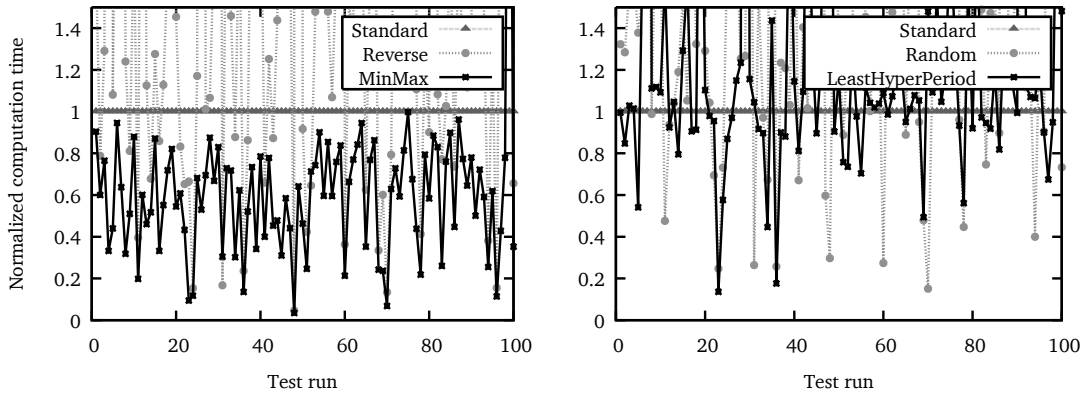


Figure 6.7: Comparison of computation times for the subadditive closure of a curve with a finite amount of segments with various heuristics. Each point represents the average of 50 closure operations.

	Norm. comp. times per test run		
Strategy	Mean	Maximum	Minimum
Standard	1	1	1
Reverse	8.6	243.6	0.05
Randomized	5.8	247.3	0.15
LeastHyperPeriod	1.5	26.1	0.14
MinMax	0.6	4	0.03

Table 6.4: Comparison of the normalized computation times for the calculation of the subadditive closure of a curve with a finite number of segments with several heuristics, see also Fig. 6.7

event and service streams in an *interval-based domain* [81], restrictions to the system under consideration apply. The proposed verification framework is in general suitable for a flexible analysis of systems-of-systems as required for the system-wide plug-and-play approach. The strong points include the composability of the system to analyze and the versatility of the framework to model characteristics of various platforms and technologies. The precision of results is appropriate to draw a decision about the feasibility of a certain setup in reasonable time. In contrast, a detailed analysis of a subsystem is subject to other methods, because the bounded analysis strategy of the RTC can lead to pessimistic results, which might not be suitable for certain use cases. In the Real-Time Calculus, data and events are *directly coupled*. Whenever an event triggers a stream filter, it is assumed that data needed for the processing is available when it is consumed. By filters with bounded buffer semantics [93], situations with under-sampling can be modeled, although only with the computational expensive operation of the subadditive closure. On the other hand, the handling of over-sampling in a native way has not been considered yet. A direct mapping is not possible, because events are consumed, i.e., semantically disappear at the input buffer after first processing. In real systems, both sampling types might occur [56], e.g., for data that is sent via a CAN bus and updated asynchronously. In that case, depending on the frequency

of the processing task, an over- and under-sampling might occur. Because event and service streams are represented in the interval domain, *phase information* is not available for a refinement of the calculations. Although worst- and best-case patterns are modeled, it is not possible to express at what exact time they occur. The point in time might be between negative and positive infinity, this directly follows from the definition of the differential arrival and service functions. Consequently, results might be overly pessimistic compared to approaches that handle phase-correct relations between events [80, 76]. This fact is accepted because the abstraction from a phase-correct calculation reduces analysis time and enhances composability. The *complexity* of each basic operation is directly influenced by the number of segments that are used to perform the operation after the unfolding of curves. This complexity is usually in the class *P*, i.e., the complexity is bounded by a polynomial with the number of curve segments as parameter. This fact was pointed out by [131] for several operations. Because the analysis process of a whole system involves many basic calculations with unknown parameters of the resulting curves, the complete process is part of the complexity class *NP*, i.e., nondeterministic polynomial time, where a polynomial as bound for the complexity exists but is not known in advance. In practice, this can lead to long analysis times that not only depend on the size of the modeled system but also on its parameters, eventually limiting the scalability of the approach. Therefore, approximation methods are introduced in Ch. 7, which can significantly speed up the analysis process.

6.7 Timing verification framework summary

In this chapter, we have introduced our verification framework based on foundations of the Real-Time Calculus. It is capable of an automatic verification process given a description according to the ANALYSIS meta-model ($\mathfrak{M}$). Compared to other approaches, our proposed solution features a unique concept to handle resource loops in the model graph and can automatically check the gained results according to the given requirements. A method to simplify the calculation of the subadditive closure was presented and various heuristics to speed up the process were evaluated. The framework can handle several representation of curves, including those based on a finite amount of linear segments and ultimately pseudo-periodic functions. It enables the implementation of the system-wide plug-and-play principle with an integrated, automatic verification process. However, so far the problem of computational complexity of the analysis has not fully been solved. Depending on size, topology, and involved parameters of the concrete graph for the verification process, the analysis may take up a considerable amount of time. In the next chapter, we will introduce methods to control the tightness of the approximation in trade for computation time.

Chapter 7

Adaptive approximate analysis

The analysis of systems with the Real-Time Calculus can take an extensive amount of computation time. This chapter introduces approaches to reduce the calculation time in exchange for result tightness and memory utilization. The selection of the tightness in this context does not have to be a global property of the analysis process. It is possible to switch between different tightness levels as desired, even during processing. The main contributions of this chapter include a method to perform an analysis with restricted memory resources, the specification of the three-segment approximation for a fast analysis, the extension of the Finitary Real-Time Calculus approach [143] to be controllable in computation time and tightness, and experimental evaluations of the proposed methods.

The demand for an adaptive approximate analysis was motivated in Sec. 3.1 and summarized in Tab. 3.1. The discussed scenarios show that the requirements for the flexibility of the analysis are multi-folded. First, it should be possible to seamlessly switch between computational effort and tightness, even during the analysis of one particular system. Second, it should be possible to choose the desired tightness during the design time to speed up the development process. Third, it should be possible to re-use existing results for further calculations. Not yet mentioned in the scenarios, is an adaption of the analysis process to platform abilities, i.e., the amount of available processing and memory resources may be different depending on the platform for the analysis. This feature becomes relevant when the analysis is conducted completely or in parts on the vehicle.

7.1 Effects on the computation time of the analysis

The computation time of the basic Real-Time Calculus operations (Tab. 6.1) mainly depends on the number of segments after the unfolding process, the parameters of ultimately pseudo-periodic curves, the application of ceiling and floor operations, and the available amount of memory. These four influencing factors are elaborated in the following.

The *number of segments* refers to the segment count after the unfolding of a curve, before the actual execution of a basic operation. In case of the convolution, the segment count directly influences the calculation time as all segments of the two involved curves are convoluted with each other. Each single convolution step might result in up to two segments that are part of the following envelope calculation, which forms the bottleneck

	System fulfills constraints	System violates constraints
Feasibility analysis positive	Quantity reduced after approximation	(False positive) Not affected, guaranteed to be avoided
Feasibility analysis negative	(False negative) More likely after approximation	Not affected, guaranteed to be detected

Table 7.1: Effect on verification results caused by approximation. The likeliness of false negatives is increased by the approximation and the number of correctly as feasible considered setups is decreased.

in the process. Let $c_a(\Delta)$ and $c_b(\Delta)$ be the functions of two unfolded curves, with an equal segment count of $|c_a| = |c_b| = n$. The first step of the min-plus convolution produces an intermediate result with a maximum of $2 \cdot n^2$ segments as input for the envelope calculation. The best known envelope algorithms for single-core processors are in the complexity class of $O(n \cdot \log n)$ (Sec. 6.3.3), thus the overall complexity lies in the class $O(n^2 \cdot \log n)$. However, specialized algorithms exist for certain classes of curves. For example, the convolution of convex functions can be constructed by a sort of the involved segments by increasing slope ([84, Theorem 3.1.5, Rule 9]). In our case, as the arrival curves are usually step-wise curves, these specializations cannot be used. The *parameters of ultimately pseudo-periodic functions* include the start of the periodic part (c_{x0}, c_{y0}), and an according translation for repetitions $(c_{\Delta x}, c_{\Delta y})$. In certain cases, the calculations depend on the hyper-period of the periodic length of the involved curves. This causes two effects: The hyper-period might lead to a long length estimation for the unfolding process and the periodic length of the resulting curve of the operation can be the hyper-period. Especially the last point can occur at every processing step, hence leading to an exponential growth of the periodic lengths of the curves, known as the hyper-period explosion [143]. This particularly takes place if the curves have the same long-term slope and are interleaved with each other. Many stream filters include *ceiling and floor operations* in their calculations, because a fractional processing of events is not useful for a timing analysis. It semantically guarantees that only completed events are forwarded to successor filters. The complexity of the ceiling and floor operations are directly coupled with the slope of the curves under consideration; the higher the slope, the more segments the resulting curve will have. The *available memory* during the basic operations has an indirect effect on the computation time. As the memory may run low during the segment-wise calculation of the convolution and deconvolution, counter-measurements have to be considered that monitor the memory utilization during the calculation and react on situations with low memory. These measurements have a negative effect on the computation time but will prevent out-of-memory exceptions.

These four effects – segment count, curve parameters, ceiling/floor operations, and available memory, are mitigated in the following sections.

7.2 Balancing computation time, tightness, and memory

This section introduces the approaches to speed up the analysis process and to handle situations with limited available memory.

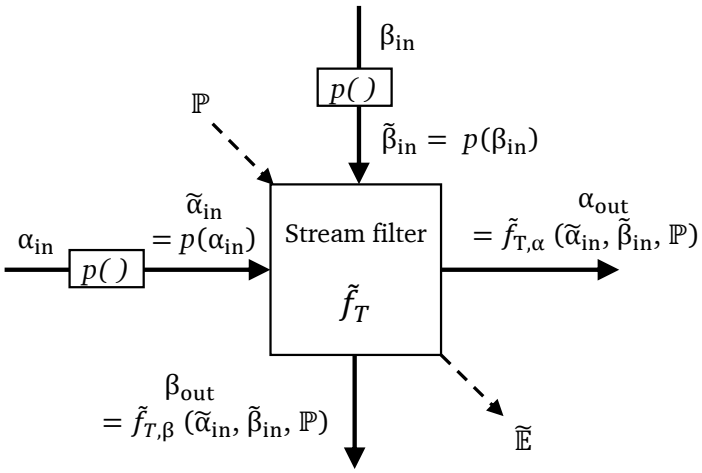


Figure 7.1: Visualization of arrival and service curve approximations before filtering, the indication of vectors is omitted from this representation for clarity. Approximated arrival and service curves $(\tilde{\alpha}, \tilde{\beta})$ are derived from the original arrival and service curves (α, β) with an approximation function $p()$. The internals of the filters f_T are also partly approximated, e.g., the floor/ceiling functions, which is denoted as $\tilde{f}_T$. The analysis result $\widetilde{\mathbb{E}}$ includes an over-approximation of the worst-case timing and an under-approximation of the best-case timing.

7.2.1 Approximation approach

The approximation has to guarantee that the results of the analysis are still valid bounds. To achieve this, upper curves are over-approximated and lower curves are under-approximated, which is called a *safe* approximation [144]. This makes sure that the verification does not produce false positives, i.e., the requirements are accidentally regarded as fulfilled by the verification while the real-world system does not stick to the constrains. This situation has to be avoided under all circumstances to prevent system failures because of timing problems. In contrast, an approximation increases the likeliness of false negatives, i.e., the results of the verification indicate that the system does not fulfill the requirements, but in reality all constraints are met. These facts are summarized in Tab. 7.1.

Our proposed approximation process includes additional steps in the generic analysis strategy. The basic idea is to approximate curves before entering stream filters in order to reduce the segment count and to decrease the computation time, see Fig. 7.1. The degree and strategy of approximation can be selected individually for each stream filter, leading to a flexible approach with configurable tightness and computation time. Besides, the approximation impacts the kernel level, where floor and ceiling operations are abstracted by counterparts that simply shift the curves along the y-axis. This contributes to a reduction of the tightness of the results but reduces computation complexity. The replacement of the floor and ceiling operations is binary in our approach, i.e., the degree of approximation cannot be chosen in this case. The advantage of the proposed approximation approach is that the filtering functions do not have to be modified in any way. The degree of approximation is not visible and its knowledge is not necessary during the processing of one filter. This makes the approach flexible as the degree of approximation is orthogonal to

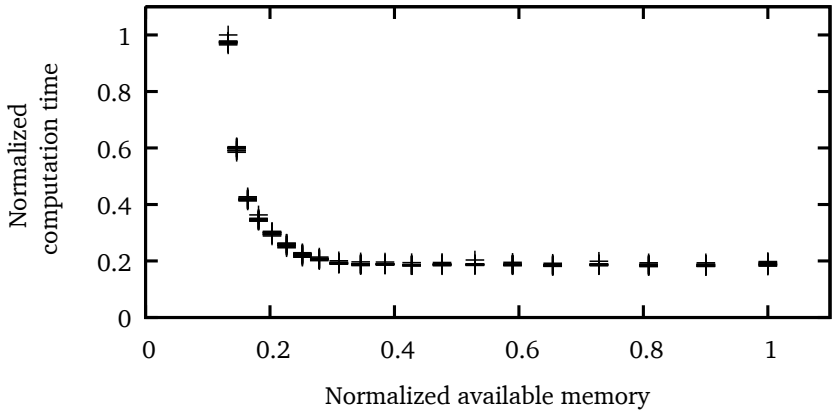


Figure 7.2: Analysis of available memory versus computation time. Each cross stands for the result of one complete analysis process of an exemplary system. The obtained analysis results are equal, but the amount of available memory influences the computation time. The vertical asymptote on the left is the minimum amount of memory that has to be available so that the results fit into memory. The horizontal asymptote marks the minimum achievable computation time.

the actual analysis process. Our approach shares similarities with the general procedure of [144], where arrival curves are approximated before entering a filter. But our concepts extends it by an approximation of service curves, an adaption of floor and ceiling operations at kernel level, and more powerful approximation strategies for curves, which are detailed in Sec. 7.3.

7.2.2 Effect of memory size

The pair-wise convolution of segments within the min-plus convolution operation (Eq. 6.8) can cause an enormous amount of segments for the subsequent envelope calculation. Because the memory is limited, it is likely possible to reach an out-of-memory exception in this step. To avoid this situation, the occupied memory is constantly observed in our approach. When memory runs low, the envelope calculation is triggered to reduce the amount of segments. Consequently, the envelope operation might be calculated several times according to the progress of the memory utilization. From a timing point of view, the shortest computation time can be achieved if all segments are first collected and then the envelope calculation is carried out in a singular step. However, this is not always possible because of the finite amount of available memory.

The approach does not help to prevent situations where the resulting curve of an operation does not fit into memory. In this case, the approximation level of the curves has to be increased. It only helps to mitigate situations, where the intermediate results might lead to a memory overflow. Note that this does not affect the tightness of the results in any way. Independent of the available memory, the results are equal, but the computation time changes. An exemplary result for the exchange of available memory for computation time is presented in Fig. 7.2, the exact algorithm is depicted in Alg. 7.1. In the algorithm,

the parameter T_{Mem} defines a threshold when the envelope calculation is executed. In our experiments, we found a value of $T_{Mem} = 0.75$ appropriate. The double-check of the available memory and the amount of intermediate segments (line 6) takes care of the fact that the garbage collection in Java might be deferred.

Algorithm 7.1: Memory observation and shortage handling during min-plus convolution operation (sketch).

```
   Input: SegmentList 𝕊1, 𝕊2, Memory threshold T_Mem
   Result: SegmentList 𝕍 = 𝕊1 ⊗ 𝕊2
   Data: Segment threshold T_Seg
 1 T_Seg ← 0, 𝕍 ← ∅;
 2 forall the Segment s1 ∈ 𝕊1 do
 3 |   forall the Segment s2 ∈ 𝕊2 do
 4 |   |   𝕍 = 𝕍 ∪ (s1 ⊗ s2);          /* s1 ⊗ s2 can be a set of Segments */
 5 |   end
 6 |   if (usedMemory() > maxMemory() · T_Mem) and (|𝕍| > T_Seg) then
 7 |   |   T_Seg = |𝕍|;
 8 |   |   𝕍 = envelope(𝕍);
 9 |   end
10 end
11 𝕍 = envelope(𝕍);
```

7.2.3 Reduction of curves

Since the calculations to derive the parameters for the extraction of curves after the basic operations are based on worst-case assumptions, reduced representations of curves may exist than the direct output of the basic operations provide. For example, the start of the periodic part might be shifted further to the origin without changing its actual expressiveness as shown in Fig. 7.3. In general, a reduction step is executed after each filter operation in our verification framework and has two purposes: First, removal of redundant artifacts and gaps caused by numerical issues. Second, reduction of the curve representation to a minimal form.

Artifacts in the results of an operation may appear because of numerical errors within the calculations. To mitigate them, the resulting curve is processed according to Alg. 7.2. The algorithm removes small gaps in the definition of curves and joins neighboring curve segments if possible. This algorithm is applicable if the curve is ultimately affine, i.e., has a finite set of segments. This is usually the case for the intermediate results of the basic operations. Because the number of segments is reduced, the computation time of further curve operations is shortened.

An ultimately pseudo-periodic curve has multiple representations. This is due to the fact that no constraints for the transition point from the aperiodic to the periodic part were defined up to now. Because the periodic part is repetitive, any unfolded range of it can

Algorithm 7.2: Removal of artifacts of curves (sketch).

Input: SegmentList $\mathbb{S}$
Result: SegmentList $\mathbb{V}$ (equals $\mathbb{S}$, but artifacts are removed)
Data: Segment s,o

```
1  V ← ∅;
2  o ← S.getAndRemoveFirst();
3  V ← V ∪ o;
4  while |S| > 0 do
5  |  s ← S.getAndRemoveFirst();
6  |  if (s_min ≈ s_max) and (!s.isPoint()) then
7  |  |  continue;                    /* Remove point-like segments */
8  |  end
9  |  if (s_m ≈ o_m) and (s_n ≈ o_n) and (s_min ≈ o_max) then
10 |  |  o_max ← s_max;        /* Combine previous and current segment */
11 |  |  continue;
12 |  end
13 |  if s_min ≈ o_max then
14 |  |  s_min ← o_max;              /* Remove gaps in definition range */
15 |  end
16 |  V ← V ∪ s;
17 |  o ← s;
18 end
```

be added to the aperiodic part without changing the overall curve. For an unambiguous representation of the transition point, the canonical definition of curves is introduced.

Definition 7.1 *Let $C = (\mathbb{S}_a, \mathbb{S}_p, c_{x0}, c_{y0}, c_{\Delta x}, c_{\Delta y})$ be a curve description (Eq. 6.1), then we define its canonical representation C^C as:*

$$C^C = \min_{c^i_{x0}} \{C^i \mid c^i(\Delta)_\infty = c(\Delta)_\infty\} \tag{7.1}$$

From all equivalent curve representations C^i the one with the smallest starting point c^i_{x0} is chosen. The condition $c^i(\Delta)_\infty = c(\Delta)_\infty$ guarantees that the curves are equal if unfolded until infinity (Eq. 6.4). Beyond the starting point for the periodic part c_{x0}, it is possible to further try to find a representation with also a minimal period $c_{\Delta x}$. However, this was not required for our use cases and is part of future work. The algorithm to reduce a curve to its canonical form is detailed in Alg. 7.3 and an example is presented in Fig. 7.3.

7.3 Approximation strategies

This section introduces the approximation strategies for curves applied in our verification tool. The overall goal is to trade computation time for analysis tightness. The proposed strategies come with individual advantages and disadvantages that are discussed in the respective subsections.

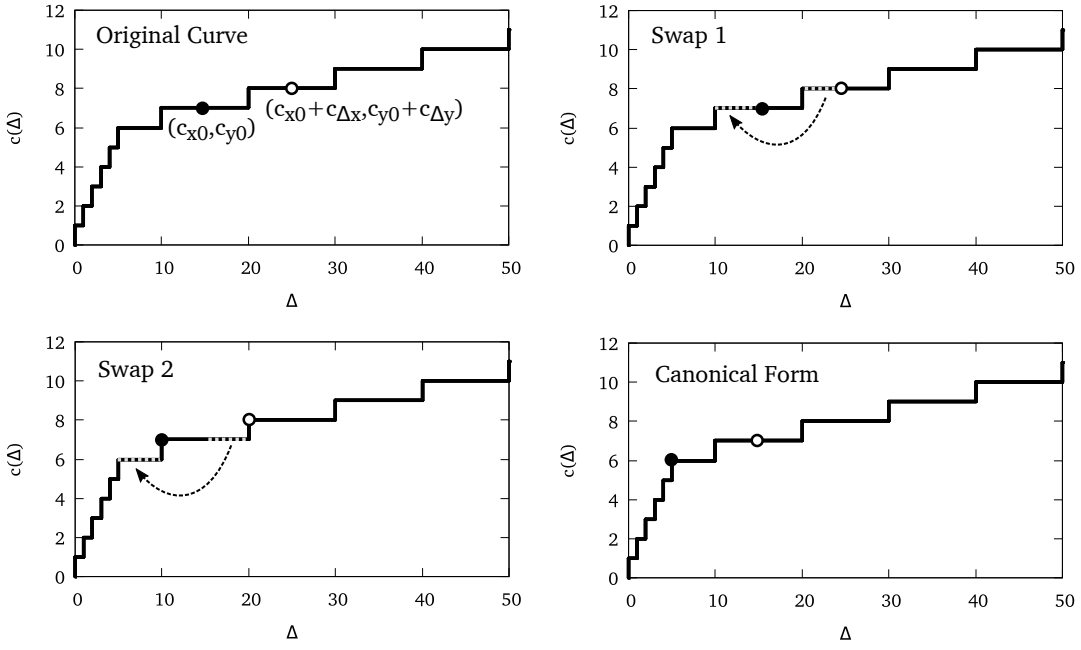


Figure 7.3: Example for the transformation of a curve into its canonical representation. Segments of the periodic part (between the two points) are swapped into the aperiodic part of the curve. The goal is to minimize the starting coordinate c_{x0} of the periodic part.

7.3.1 Three-segment approximation

The three-segment approximation reduces the number of segments to represent arrival and service curves to a maximum of three. This reduces computational complexity but weakens the tightness of the results. We consider the three-segment approach as one of the fastest but most inaccurate analysis methods. An approximation of the Real-Time Calculus by curves based on three segments was also presented in [145], but without the possibility to switch between approximation levels. Instead, the mathematical framework was adapted to handle three-segment based curves directly – as only option for the representation of curves. This gives advantages in sense of analysis performance, but lacks the seamless integration into a framework with different approximation levels. Therefore, we present a strategy to convert an arbitrary sub- or superadditive curve into a three-segment based representation. As the curves are still described according to Eq. 6.1, it can be applied at any point in time within the verification framework without a modification of the underlying mathematical relations. Furthermore, the operators for ceiling and floor are changed to simple shifts of the curves during the processing. These measurements efficiently bound the computational complexity as the number of segments involved in the operations is limited.

The approximation of upper and lower curves has to be handled differently. As an example, we consider the approximation of an arrival curve $\alpha(\Delta) = [\alpha^u(\Delta), \alpha^l(\Delta)]$. Then, the approximation $\tilde{\alpha}(\Delta) = [\tilde{\alpha}^u(\Delta), \tilde{\alpha}^l(\Delta)]$ of the curve has to fulfill [144]

$$\tilde{\alpha}^u(\Delta) \geq \alpha^u(\Delta) \quad \text{and} \quad \tilde{\alpha}^l(\Delta) \leq \alpha^l(\Delta) \tag{7.2}$$

Algorithm 7.3: Transformation of a curve to its canonical form (sketch).

Input: Curve $C = (\mathbb{S}_a, \mathbb{S}_p \in \text{SegmentList}; c_{x0}, c_{y0}, c_{\Delta x}, c_{\Delta y} \in \mathbb{R})$
`/* 𝕊a,𝕊p must be free of artifacts according to Alg. 7.2 */`
Result: Curve C^C (canonical form of C)
Data: Segment a,p

```
1  while 𝕊a ≠ ∅ do
2      p ← 𝕊p.getLastCopy();        /* Get last segment of periodic...  */
       a ← 𝕊a.getLastCopy();                  /* ...and aperiodic part.  */
       dx ← min(a_max − a_min, p_max − p_min);
3      p_min ← p_max − dx;              /* Equalize definition range...  */
       a_min ← a_max − dx                 /* ...to minimum of segments.  */
       a ← a.shift(c_Δx − c_x0, c_Δy − c_y0); /* Aper. segm. to per. part.  */
4      if p ≠ a then break;
5      𝕊p ← 𝕊p.extract(0.0, c_Δx − dx);/* Swap seg. from end to start...  */
6      𝕊p ← p.shift(dx − c_Δx, −c_Δy) ∪ 𝕊p.shift(dx, 0.0); /* ...of per. part.  */
7      c_x0 ← c_x0 − dx;                  /* Adjust per. start point.  */
8      𝕊a ← 𝕊a.extract(0.0, c_x0);              /* Adjust aper. part.  */
9      dy ← 𝕊p.first().n();                      /* Handle possibly...  */
10     𝕊p ← 𝕊p.shift(0.0, dy);                       /* ...negative...  */
11     c_y0 ← c_y0 − dy;                             /* ...per. part.  */
12 end
```

	$s_n^0 \neq 0$ **or** $s_m^0 \neq 0$	$s_n^0 = 0$ **and** $s_m^0 = 0$
$\lim_{\Delta\to\infty} c(\Delta) \geq 0$	$m_x = 0.0\ m_y = s_n^0$	Invalid
$\lim_{\Delta\to\infty} c(\Delta) < 0$	Invalid	$m_x = s_{max}^0\ m_y = 0.0$

Table 7.2: Point (m_x, m_y) of the middle segment that is used for slope calculation. s_n^o refers to the first segment of the curve C that is to be approximated.

in order to be safe. To reduce the number of cases we have to distinguish, the calculation of the lower approximation is mapped to the upper as follows: $\tilde{\alpha}^l(\Delta) = -\zeta^u(-\alpha^l(\Delta))$, where $\zeta^u()$ is the approximation function for the upper curve. For the calculation of the upper three-segment approximation $\zeta^u()$, we calculate the first ($s^{\zeta 0}$), middle ($s^{\zeta 1}$) and last segment ($s^{\zeta 2}$) with different strategies.

If the long-term slope C_ρ is positive then the *first segment* $s^{\zeta 0}$ is not considered, because it only exists virtually, see Fig. 7.4 (left side). If the long-term slope is negative, then the first segment is defined as $s^{\zeta 0}(\Delta) = 0.0\ \forall\ \Delta \in \mathbb{R}_{\geq 0}$. One point of the *middle segment* $s^{\zeta 1}$ is defined by the end of the first segment. Tab. 7.2 shows the parameters of this point (m_x, m_y). The segment with the minimum possible slope that goes through this point and does not cut the original curve is the tightest bound that can be achieved. The slope is defined as follows: $s_m^{\zeta 1} = \min(s_m \mid s_m \cdot (\Delta - m_x) + m_y \geq c(\Delta)\ \forall\ \Delta \geq m_x)$. The calculation of the offset $s_n^{\zeta 1}$ is then straight-forward by inserting the known point (m_x, m_y) and the

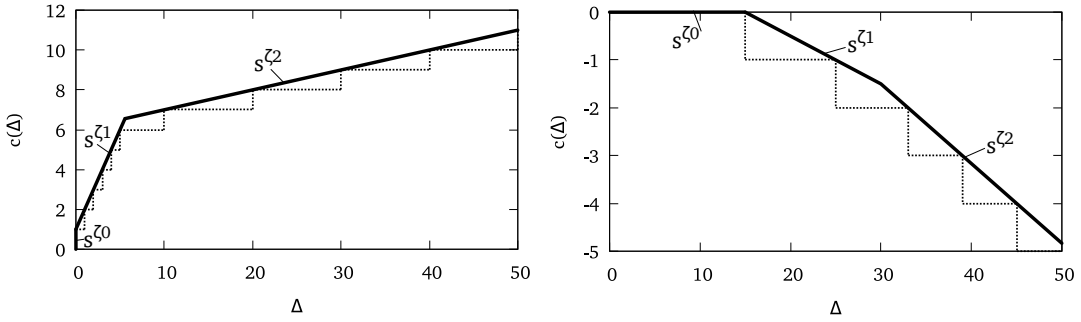


Figure 7.4: Example of a three-segment approximation for a positive curve (left side) and a negative curve (right side). Both curves are subadditive.

slope $s_m^{\zeta 1}$ into the segment equation (Eq. 6.2). For the algorithmic construction of $s_m^{\zeta 1}$, the curve is unfolded until $\phi = c_{x0} + c_{\Delta x}$. Let $\mathbb{S}_\phi$ contain all segments of the unfolded part of the curve, then a slope candidate $s_{m,c}^i$ for the segment $s^i \in \mathbb{S}_\phi$ is calculated as $s_{m,c}^i = (s_m^i \cdot s_x^i + s_n^i - m_y)/(s_x^i - m_x)$, where $s_x^i = s_{min}^i$ for positive and $s_x^i = s_{max}^i$ for negative curves. The slope is then the maximum candidate: $s_m^{\zeta 1} = \max_{\forall i}(s_{m,c}^i)$. For the *last segment* $s^{\zeta 2}$ we know the slope as it has to be equal to the long-term slope of the original curve, i.e., $s_m^{\zeta 2} = C_\rho$ (Eq. 6.6). The offset $s_n^{\zeta 2}$ of the last segment is then defined as the minimum offset s_n, which is still above the original curve for a segment with slope $s_m^{\zeta 2}$, i.e., $s_n^{\zeta 2} = \min(s_n \mid s_m^{\zeta 2} \cdot \Delta + s_n \geq c(\Delta)\ \forall\ \Delta \in \mathbb{R}_{\geq 0})$. For the algorithmic construction in this case, we take the same set of segments $\mathbb{S}_\phi$ and definition of s_x^i as for the middle segment and select the smallest candidate for the offset $s_n^{\zeta 2} = \min_{\forall i}(s_{n,c}^i)$, where each candidate is calculated as $s_{n,c}^i = (s_m^i - s_m^{\zeta 2}) \cdot s_x^i + s_n^i$. To derive the final result, the envelope is calculated: $\zeta^u = s^{\zeta 0} \wedge s^{\zeta 1} \wedge s^{\zeta 2}$. As desired, the result has a maximum of three segments and is subadditive.

Examples and discussion of the three-segment approximation. To make the results comparable, two metrics are defined to work with approximated results. We will refer to a latency derived with the standard approach by d_{e2e}, where $e2e$ stands for end-to-end, and for a latency derived with an approximate approach by $\tilde{d}_{e2e}$. Then, we define the normalized delays d^N to be: $d_{e2e}^N = d_{e2e}/d_{e2e} = 1.0$ and $\tilde{d}_{e2e}^N = \tilde{d}_{e2e}/d_{e2e}$, i.e., the ratio between the results of the approximated and the standard approach. Further, we define the average normalized delay to be $\tilde{d}_{e2e}^{\varnothing N} = \sum_{\forall i} \tilde{d}_{e2e,i}^N / i_{max}$, where i is the stream number and i_{max} the number of streams of the test case under consideration. The same definition holds for the computation time t_{ct} of the different approaches, where the normalized computation time is $t_{ct}^N = 1.0$ and $\tilde{t}_{ct}^N = \tilde{t}_{ct}/t_{ct}$.

Tab. 7.3 shows the results of the analysis of the eCar example in the standard and approximate case. On average, the end-to-end delay of each stream is 285% higher when computed with the approximation, while the computation time is ≈ 31 times faster. The introduced approximation error of the delays causes some of the requirements to fail the verification, see Tab. 7.4. Tab. 7.5 depicts the results for the complex loop example (Fig. 6.4). The method for the fixed point calculation was not altered, the curves were only approxi-

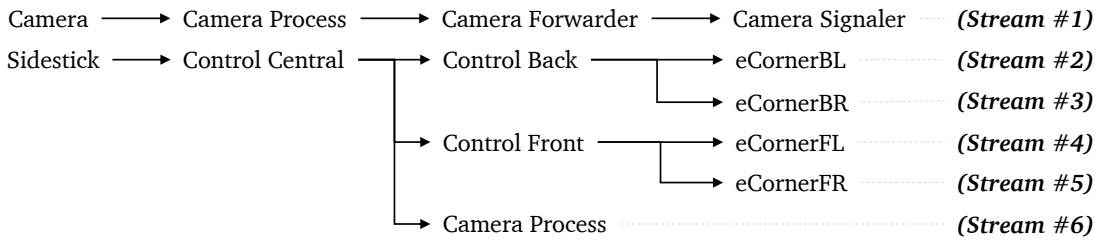


Figure 7.5: Event streams of the eCar example.

		Worst-case end-to-end timings for stream						
Strategy	**Comp. time**	**#1**	**#2**	**#3**	**#4**	**#5**	**#6**	
	Absolute values							
Standard	0.332s	28.28	9.14	9.14	9.56	9.82	5.13	
Finitary	0.141s	28.28	9.14	9.14	9.56	9.82	5.13	
Three-segment	0.011s	75.52	42.16	42.28	38.27	38.83	16.59	
	Normalized values							$\tilde{d}_{e2e}^{\varnothing N}$
Standard	1.0	1.0	1.0	1.0	1.0	1.0	1.0	
Finitary	0.42	1.0	1.0	1.0	1.0	1.0	1.0	
Three-segment	0.03	2.67	4.61	4.63	4.00	3.95	3.24	3.85

Table 7.3: Comparison of analysis results with standard processing, three-segment approximation and Finitary processing for the eCar example. The verification of some requirements fails with the three-segment approach due to the over-approximation. A description of the streams is available in Fig. 7.5.

Req. / Strategy	Cam-ProDel	CamRate	CContr-In	CamSigIn	Cam-ProSig	Mov-VecPer
Standard	OK	OK	OK	OK	OK	OK
Finitary	OK	OK	OK	OK	OK	OK
Three-segment	FAIL	OK	OK	OK	FAIL	OK
	MovVecResp					
	BackLeft	Back-Right	FrontLeft	Front-Right	CamProc	
Standard	OK	OK	OK	OK	OK	
Finitary	OK	OK	OK	OK	OK	
Three-segment	FAIL	FAIL	FAIL	FAIL	OK	

Table 7.4: Verification results of the eCar example with standard approach, three-segment approximation and Finitary processing.

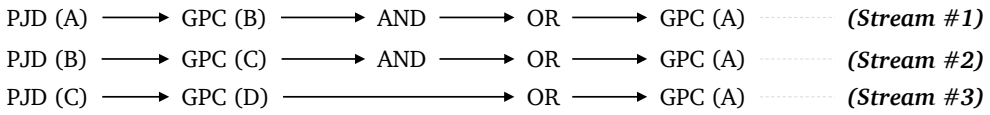


Figure 7.6: Event streams of the complex loop example (Fig. 6.4).

		Worst-case end-to-end timings for stream			
Strategy	**Comp. time**	**#1**	**#2**	**#3**	
	Absolute values				
Standard	0.26s	40.00	45.00	4.50	
Finitary	0.12s	40.00	45.00	4.50	
Three-segment	0.02s	92.88	98.77	9.57	
	Normalized values				$\tilde{d}_{e2e}^{\varnothing N}$
Standard	1.00	1.00	1.00	1.00	
Finitary	0.46	1.00	1.00	1.00	
Three-segment	0.10	2.32	2.19	2.13	2.21

Table 7.5: Comparison of analysis results with standard processing, Finitary processing and three-segment approximation for the complex loop example (Fig. 6.4) with parameters of Tab. 6.2 ($r_{loop} = 2.0$). The classification of the stream numbers is available in Fig. 7.6.

mated before entering a filter (Fig. 7.1). In this case, the average approximation error per stream is 121%, while the computation is ≈ 13 times faster. In Tab. 7.7, the results for the cyclic mesh example (see Fig. 7.9) are presented. While the average approximation error for the three-segment approach in this case is 115%, the computation time could be reduced by a factor of ≈ 1200. The three examples show, that the gained speedup and approximation error can vary heavily depending on the test case. No suitable method is known to the author of this work to predict the approximation error or computational speedup exactly in advance. As it depends on the modeled system and its many parameters, it is hard to give a meaningful prediction.

7.3.2 Integration of Finitary Real-Time Calculus

Finitary Real-Time Calculus was proposed by [143] as a method to speed up the analysis within the RTC framework. The approach exploits the fact that curves are only needed up to a certain length to calculate the relevant parameters maximum delay and backlog at a specific stream filter. This also explains the name "Finitary" as curves are only kept up to a certain threshold. Compared to standard RTC, Finitary RTC is a three-step approach: In the first step, the system is analyzed with a high abstraction level. The second step is an accumulation of derived bounds, which are later used to limit the size of curves. In the third and final step, the system is analyzed according to standard RTC, but curves are only considered up to the bounds previously calculated. These steps are visualized in Fig. 7.8. In the following, the Finitary Real-Time Calculus idea from [143] is summarized. We extend the concept with the ability to process cyclic resource dependencies, show how it is integrated into our verification framework and provide experimental results. In addition, we extend the method in the next section by a flexible approximation concept.

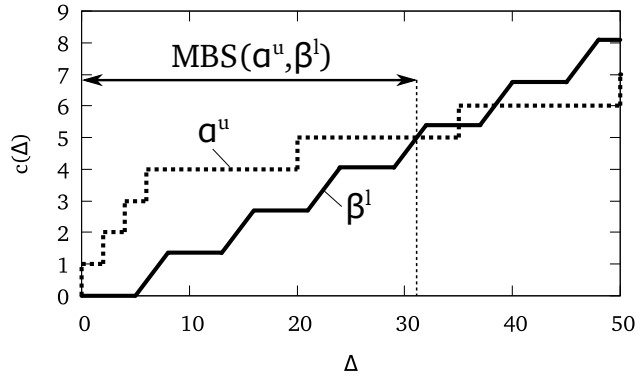


Figure 7.7: Illustration of the maximal busy-period size (MBS, [143]). The MBS is the first point on the interval axis, where the upper arrival curve (α^u) is above the lower service curve (β^l). To calculate the maximum delay and backlog at a component, only those ranges of the curves are relevant that are smaller or equal to the MBS.

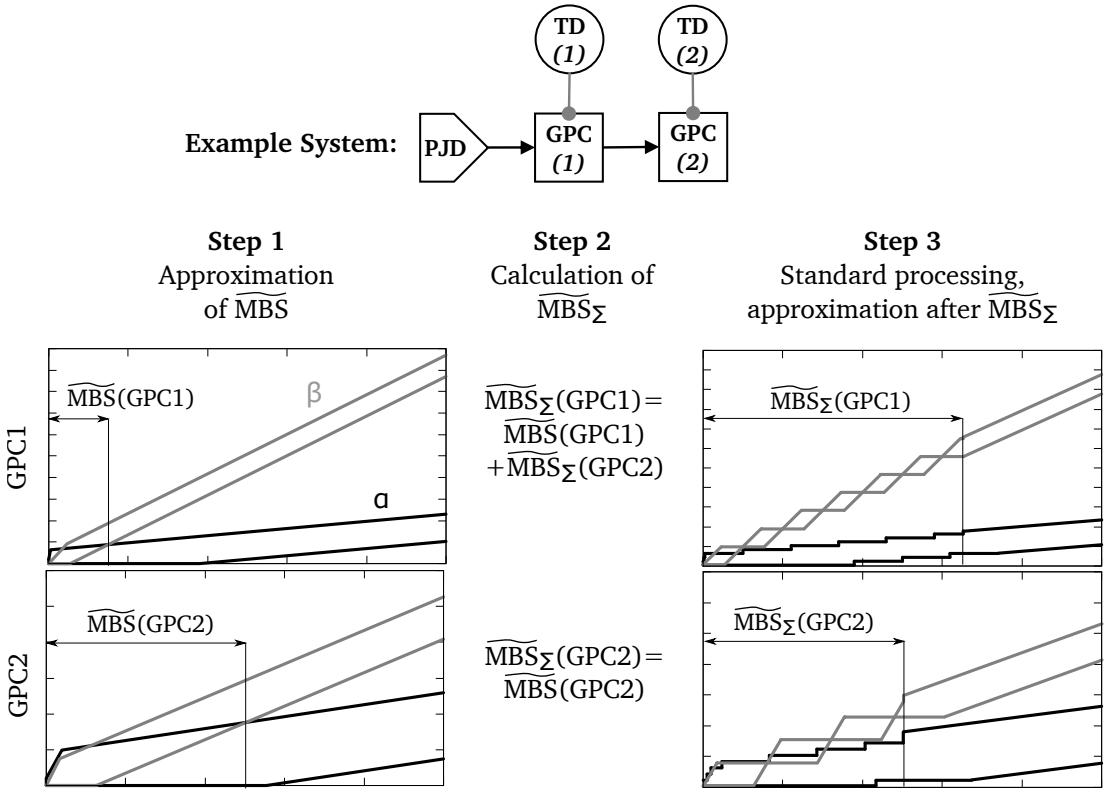


Figure 7.8: Exemplary visualization of Finitary Real-Time Calculus. In the 1st step, the MBS (see Fig. 7.7) is approximated with help of the three-segment approximation (see Fig. 7.4). The 2nd step recursively sums up the maximum MBS of all successor components. In the 3rd step, the standard processing is executed, but input curves are approximated after the summed up MBS. The result is equal to the standard approach, but computational complexity is reduced.

1. Analysis with high abstraction level, calculation of $\widetilde{MBS}$. The maximal busy-period size (MBS) of GPC stream filters is the first point, where the lower input service curve β^l_{in} is above the upper input arrival curve α^u_{in}: $MBS(\alpha^u_{\text{in}}, \beta^l_{\text{in}}) = \min(\Delta > 0 : \alpha^u_{\text{in}}(\Delta) \geq \beta^l_{\text{in}}(\Delta))$, see Fig. 7.7 for an example. Only the parts of these curves with $\Delta \leq MBS$ are needed to calculate the maximum delay (as it is the maximal horizontal deviation between $\alpha^u_{\text{in}}, \beta^l_{\text{in}}$), and the maximum buffer utilization (as it is the maximal vertical deviation). Hence, the MBS defines a bound up to which the curves are relevant. For other filter types, the MBS is calculated analogously, but depends on the concrete filter equations. For a determination of the MBS of the filters, the system is first analyzed with a high abstraction level to calculate $\widetilde{MBS}$ – an over-approximation of the MBS. We do this in our approach by analyzing the system with the three-segment approximation as described in the previous section.

2. Calculation of $\widetilde{MBS}_\Sigma$ *from* $\widetilde{MBS}$. The sums of the $\widetilde{MBS}$ are recursively defined as follows: $\widetilde{MBS^i}_\Sigma = \widetilde{MBS^i} + \max(\widetilde{MBS^j} : j \in f^i_{\text{succesor}})$, where f^i_{succesor} refers to the indices of relevant successors of the filter with index i. $\widetilde{MBS}_\Sigma$ defines the bound until which the curves are needed at the input of a certain filter considering the successor filters, see [143] for details. For systems with cyclic resource dependencies, the recursive calculation of $\widetilde{MBS}_\Sigma$ does not work as it would lead to infinite loops. Therefore, we derive it after the pre-analysis as follows: $\widetilde{MBS}_\Sigma$ is constructed in the reverse order of the topological sorting of the subgraphs $\mathfrak{S}_i$. For each stream filter of a subgraph with a cyclic dependency, an recursive algorithm is executed while keeping track to not calculate $\widetilde{MBS}_\Sigma$ for any filter twice, which breaks the cyclic dependency. The results are stored, all the filters of the subgraph are reset and the process is repeated with the next filter of the subgraph. The recursion compromises all arrival and service relations to successor elements, but stops in case a filter is referenced that is not part of the processed subgraph. In the end, $\widetilde{MBS}_\Sigma$ for each filter is the maximum of the individual results.

3. Standard processing with bounding of curves by $\widetilde{MBS}_\Sigma$. For the actual analysis, the standard steps of the verification framework are executed, but each input curve of a filter is approximated after $\Delta \geq \widetilde{MBS}_\Sigma$ with a linear (possibly shifted) segment with the overall slope of the curve. The used approximation of a curve by a single segment from a certain point on is equal to the approach presented in [144]. With the shift of the last segment, it is guaranteed that the curve is always strictly above or strictly under the original curve. For systems with cyclic resource dependencies, the algorithm as introduced in the previous chapter is executed until a fixed point is found.

Examples and discussion of the Finitary Real-Time Calculus. The analysis results with the Finitary Real-Time Calculus are equal to the standard approach. Because the curves are bounded and so are the number of segments, the computational complexity is usually lower. This effect can change if the calculation of the size of the maximal busy-period size is too pessimistic or if the curves' parameters allow an immense reduction (Sec. 7.2.3) in the standard approach.

Tab. 7.3 shows the results of the eCar example. The processing according to the Finitary RTC approach is ≈ 2.4 times faster compared to the standard approach. The results of the complex loop example (Tab. 7.5) show a speedup of factor ≈ 2.2. Not shown in the results, if a rate of $r_{loop} = 1.0$ is chosen for the service sources of the same example, the

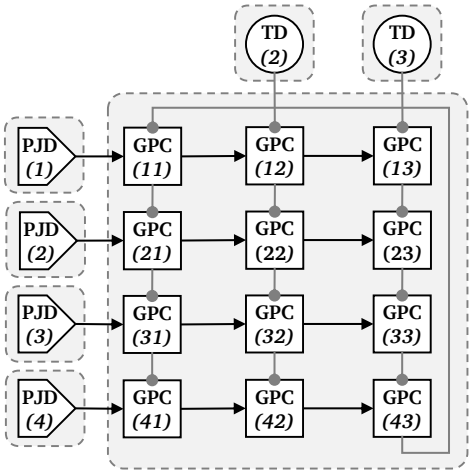


Figure 7.9: *Cyclic mesh example* with cyclic resource dependencies for the evaluation of approximation approaches, adapted from [143]. See Tab. 7.6 for parameters and Tab. 7.7 for results.

Node	Parameters	Node	Parameters	Node	Parameters
GPC (all)	et=1	PJD (1)	p=10,j=2,d=4	PJD (3)	p=14,j=5,d=8
TDMA (2)	s=6,c=8,b=1	PJD (2)	p=12,j=3,d=6	PJD (4)	p=21,j=6,d=4
TDMA (3)	s=8,c=10,b=10	s=slot,c=cycle,b=bandw./p=period,j=jitter,d=min. dist.			

Table 7.6: Parameters for cyclic mesh example (Fig. 7.9).

PJD (1) → GPC (11) → GPC (12) → GPC (13) ····· ***(Stream #1)***
PJD (2) → GPC (21) → GPC (22) → GPC (23) ····· ***(Stream #2)***
PJD (3) → GPC (31) → GPC (32) → GPC (33) ····· ***(Stream #3)***
PJD (4) → GPC (41) → GPC (42) → GPC (43) ····· ***(Stream #4)***

Figure 7.10: Event chains of the cyclic mesh example (Fig. 7.9).

		Worst-case end-to-end timings for stream				
Strategy	**Comp. time**	**#1**	**#2**	**#3**	**#4**	
	Absolute values					
Standard	47.5s	7.80	9.00	10.20	16.50	
Finitary	3.0s	7.80	9.00	10.20	16.50	
Three-segment	0.04s	13.49	19.24	26.56	35.14	
	Normalized values					$\tilde{d}_{e2e}^{\varnothing N}$
Standard	1.0	1.0	1.0	1.0	1.0	
Finitary	0.06	1.0	1.0	1.0	1.0	
Three-segment	0.001	1.73	2.14	2.60	2.13	2.15

Table 7.7: Comparison of analysis results with standard processing, Finitary processing and three-segment approximation for cyclic mesh example with cyclic resource dependencies (Fig. 7.9). The meaning of the stream numbers is shown in Fig. 7.10.

Req. / Strategy	Cam-ProDel	CamRate	CContr-In	CamSigIn	Cam-ProSig	Mov-VecPer
Finitary 100%	OK	OK	OK	OK	OK	OK
Finitary 17.8%	OK	OK	OK	OK	FAIL	OK
Finitary 5.6%	FAIL	OK	OK	OK	FAIL	OK
	MovVecResp					
	BackLeft	Back-Right	FrontLeft	Fron-tRight	CamProc	
Finitary 100%	OK	OK	OK	OK	OK	
Finitary 10.0%	OK	FAIL	OK	OK	OK	
Finitary 5.6%	FAIL	FAIL	OK	OK	OK	
Finitary 2.4%	FAIL	FAIL	FAIL	FAIL	OK	

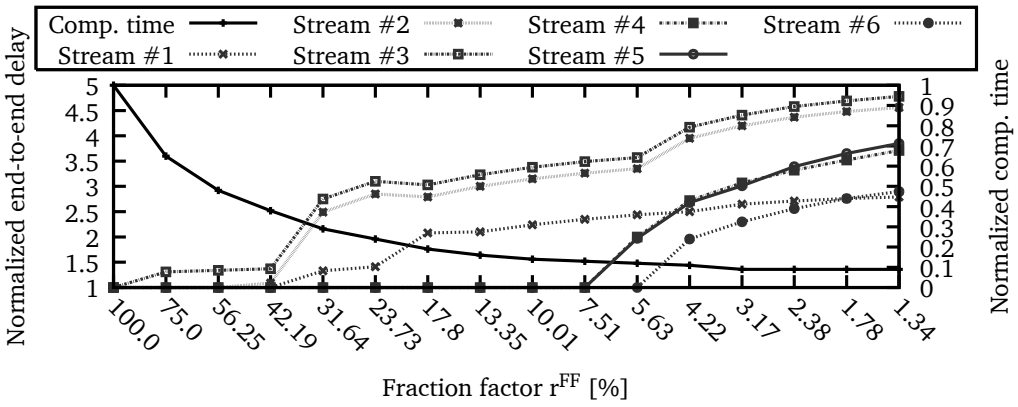
Table 7.8: Verification results of the eCar example with Fractional Finitary analysis. The analysis results of the approximation are less tight than with the standard approach, leading to negative verification results. The requirements are summarized in Tab. 4.4.

calculation with the Finitary Real-Time Calculus is substantially slower than with the standard approach. This is due to the high utilization (partly more than 93%) of the filters in the loop, causing high values for $\widetilde{MBS}$ and $\widetilde{MBS}_\Sigma$. In the cyclic mesh example (Tab. 7.7), the measured computation times were ≈ 16 times faster. Compared to the three-segment approach, the Finitary RTC is not always faster as the standard approach. However, if the utilization is low and the hyper-periods are large, the computational time can be reduced by several orders of magnitude (see also [143]).

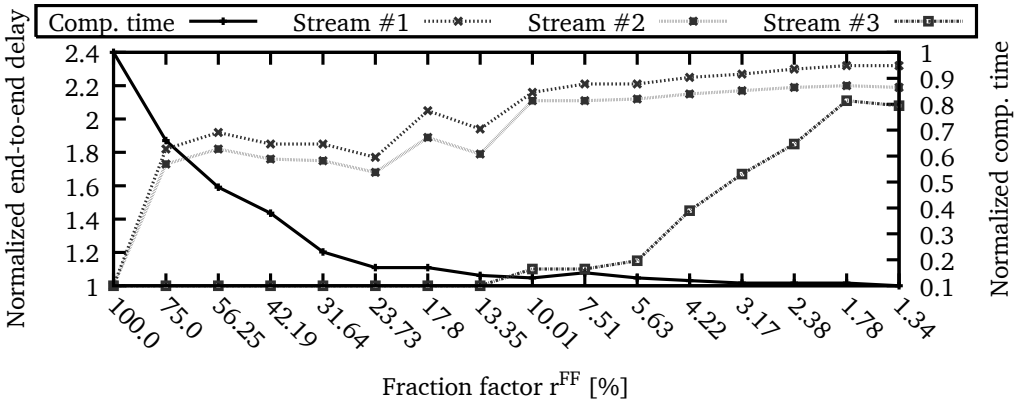
7.3.3 Fractional Finitary Real-Time Calculus

We now introduce Fractional Finitary Real-Time Calculus, which is a modification of the Finitary approach to allow seamless tightness adjustments. The bounds calculated by the first two phases of the Finitary approach are scaled by a user-adjustable factor. The parameter can directly control the tightness of the results in exchange for the computation time of the analysis. Our approach is only possible due to the fact that an upper bound $\widetilde{MBS}_\Sigma$ exists after the first two steps of the Finitary RTC, which can be utilized for the scaling. This is an advantage to approaches that have a fixed value for bounding of curves or a bound derived from the curve's parameters (e.g., [144]), because these values depend on the concrete system parameters and have to be provided by an expert. Formally, given a fraction factor (FF) r^{FF} with $\{r^{FF} \in \mathbb{R} \mid 0\% \leq r^{FF} \leq 100\%\}$, then the fraction factor maximal busy-period size sum $\widetilde{MBS}_\Sigma^{FF}$ is used to bound the input curves of a filter, where $\widetilde{MBS}_\Sigma^{FF} = \widetilde{MBS}_\Sigma \cdot r^{FF}$. While a fraction factor of $r^{FF} = 100\%$ corresponds to the standard Finitary RTC, a factor of $r^{FF} \to 0\%$ corresponds to an approximation with two segments.

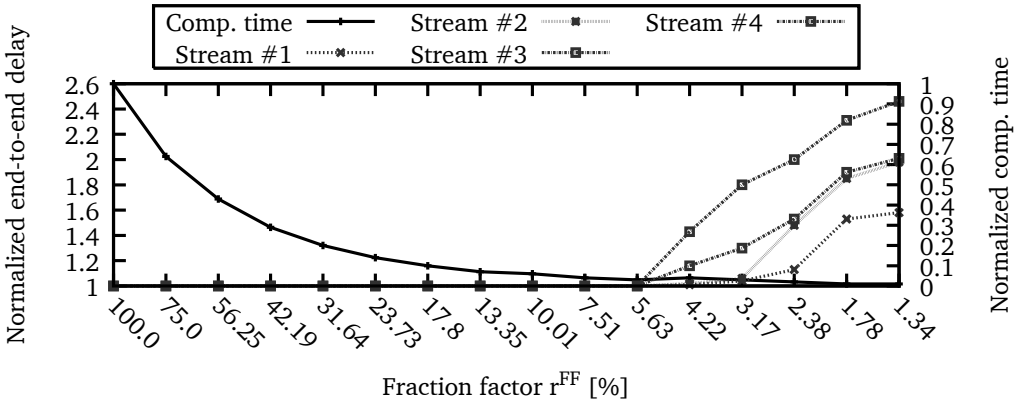
Examples and discussion of Fractional Finitary Real-Time Calculus. Results for several examples that were processed according to the Fractional Finitary RTC approach are shown in Fig. 7.11, the according verification results for the eCar example with different approximation levels are presented in Tab. 7.8. Although hard to see in the graphs of Fig. 7.11, because of the logarithmic scaling of the fraction factor r^{FF}, the computation time is reduced almost linearly with the fraction factor. The computation time is bounded on the



(a) eCar example



(b) Complex loop example



(c) Cyclic mesh example

Figure 7.11: Normalized end-to-end delays and normalized computation times for various examples, which are processed according to the Fractional Finitary Real-Time Calculus approach. The scaling of the fraction factor r^{FF} is logarithmic. The results indicate that a trade of computation time for result tightness is possible with the proposed approach.

left side ($r^{FF} = 100\%$) by the timing of the standard Finitary approach, and on the right side ($r^{FF} \to 0\%$) by the processing time caused with a two-segment approximation. It is noticeable that the calculated end-to-end delays do not change from the tightest possible analysis ($r^{FF} = 100\%$) a certain range with decreasing r^{FF}. For example, in the cyclic mesh setup (7.11c), the delays do not change much for $r^{FF} \geq 6\%$. This is a direct effect of the over-approximation of the maximal busy-period size $\widetilde{MBS}_{\Sigma}$. Because the approximation errors are summed up, the overall error of the over-approximation increases with the number of filters that have data- or service-dependencies with each other. The approximated curves are not necessarily strictly sub-/superadditive. As the curves are approximated from an arbitrary point on as a single segment, which has to be strictly above/under the original curve (see Eq. 7.2), inconsistencies might be introduced regarding the sub-/superadditive property (Eq. 2.7) and the causality of the curves (Eq. 2.9). In practice, we did not encounter any instabilities during the processing with the proposed approximation methods. However, a formal proof that the calculations are always stable remains open. Another possibility to ensure the sub-/superadditive property and causality is the application of the closure operation or specialized algorithms [88, 89] on the resulting curves. But these options introduce excessive overhead and increase the number of segments of the approximated curves significantly, why we do not regard it as an option. Further, we have chosen a linear curve approximation approach, i.e., the approximation point is a constant fraction from $\widetilde{MBS}_{\Sigma}$. Because the approximation error increases with the quantity of stream filters traversed by a particular arrival or service stream, other (non-linear) functions can be chosen for the calculation of the approximation point. Overall, the provided results show that the seamless trade of tightness and computation time is possible.

7.4 Adaptive approximate analysis summary

In this chapter, it was shown which parameters affect the computational complexity of the operations involved in the Real-Time Calculus and how this complexity can be influenced, with and without affecting the tightness of the results. An algorithm was presented that adapts analysis calculations to scenarios with different available memory sizes. Experimental data indicate that the computation time of the analysis is slower if less memory is available. A canonical form of segment-wise defined, ultimately pseudo-periodic curves was introduced that helps to minimize the computational complexity by a compact representation without altering the results. A three-segment based approximation was developed that can be seamlessly integrated into the analysis framework at any point. The Finitary Real-Time Calculus approach [143] was extended to handle cyclic resource dependencies and to the Fractional Finitary approach, which allows a fine-granular exchange of result tightness for analysis computation time. The performance of the approximation strategies was verified by three experimental setups, including the dataset from the eCar example. In the discussions of the results, limiting factors were pointed out. The reduction of computational complexity helps to make the system-wide plug-and-play approach feasible. Without our proposed methods, the analysis can take a considerable amount of time, which makes it unusable for a quick verification process of a changed system.

Chapter 8

Conclusion

The thesis in hand proposed an approach to combine system-wide plug-and-play with automatic timing verification for automotive systems that are developed in a model-driven manner according to the component-based and data-centric paradigms. This directly addresses the increasing complexity and diversity of automotive electric and electronic systems caused by the permanent addition of functionality, with a focus on the domain inherent real-time constraints. The approach helps to automate the development process of vehicles and makes it possible to deeply integrate software and hardware components even after sale by non-experts. The basic idea is to modularize the functionality into features, where each feature provides its own specification of software, hardware, and requirements. The proposed and implemented framework automatically combines a set of features and verifies that all timing requirements are met in the final system, even across feature borders. The analysis process itself can be controlled to trade analysis computation time for accuracy of the approximation to adapt it to miscellaneous scenarios.

8.1 Summary of contributions

The idea of system-wide plug-and-play was introduced, which holistically considers the setup of a system and applies an end-to-end analysis of event chains, considering the mutual influence of functions and communication relations. It was compared to the existing solutions in the automotive domain and a minimal set of meta-models was developed, suitable to implement the proposed system-wide plug-and-play approach. In this context, a concept was described that allows the specification of timing requirements for data-centric systems, where the relations of senders and receivers of data are not known at the point of development. A chain of transformations was defined that combines a given set of features for a vehicle, instantiates it for deployment, and refines it for the analysis process. Examples were given for the mapping of the abstract, platform-independent representations to concrete information and communication technologies. The feasibility of the transformations was shown on the example of the eCar demonstrator. A verification framework was implemented, based on the Real-Time Calculus theory, which is suitable to calculate performance metrics of event-based, distributed systems. It includes a unique approach to automatically handle cyclic resource dependencies in the analysis model and a simplified method

to calculate the closure for segment-wise defined curves. The verification framework can determine delays and characteristics of event and service streams and verify these against a given set of requirements. The analysis methods were enhanced with various computation and approximation strategies, which allow a fine-grained trade of analysis tightness for computation time and that enable the analysis with limited memory capabilities. The proposed strategies were evaluated on a set of well-defined example systems, including the eCar demonstrator.

Applications in other domains

Although this thesis originated from the context of automotive systems, the approach is not limited to the automotive domain. In general, it is suited for all cases where system-wide plug-and-play and hard real-time requirements have to be combined. For example, it is applicable in the domain of industrial automation, where the quick adaptivity of manufacturing sites becomes a requirement [146, 147] and where communication according to the data-centric paradigm is an emerging topic [148]. The approach can further be utilized to enhance the composability of audio and video studio applications. The Audio Video Bridging (AVB) standard [123] specifies a plug-and-play network protocol for the real-time distribution of audio and video streams, but does not consider the chained interactions of attached devices. With the proposed framework, system-wide end-to-end guarantees could be given, with respect to the complete processing chains. Other fields of application include test systems, where the demand of a fast and adaptable setup stands in contrast to challenging requirements on data quality and synchronization. The proposed approach could be used to automatically integrate the individual test components and to check the characteristics and feasibility of a certain setup.

8.2 Future work

The scope of this thesis was the proof-of-concept for the applicability of a development process according to the system-wide plug-and-play principle with automatic timing verification. The following list summarizes selected directions for a further extension of the proposed approach:

- **Alignment with AUTOSAR.** Although the impact to a development process according to the AUTOSAR methodology was discussed, a complete mapping of the approach to AUTOSAR artifacts was not conducted and the impact on the involved development roles was not elaborated in detail.

- **Extended timing requirements for data-centric communication.** We have shown basic possibilities for the specification of timing requirements when data senders or receivers are not known during design time. For example, it is possible to specify requirements relative to the begin or end of event chains. This approach has its limitations, because relative points cannot be further constrained. For example, a diagnostic component at the begin or end of an event chain is considered for the verification process. As this behavior is not always intended, future work can generalize this behavior with additional constraints.

- **Mapping to new technologies.** The mappings from the platform-independent representation to specific technologies during the transformation steps towards the analysis model can be enhanced by further technologies. For example, Time-Sensitive Networking (TSN) is extensively discussed as a communication technology for future vehicle infrastructures. The correct modeling of this technology within the Real-Time Calculus is still a research question.
- **Analysis with multiple approximation levels.** Specific parts of the system might be analyzed at different approximation levels. For example, parts that seldom change can be analyzed with an approximation with tight results, while often-changing parts can be analyzed with a focus on a reduction of computation time. In general, a switch between approximation levels is possible at any point during the analysis process, but the details are not presented in this thesis.
- **Extension for parallel processing.** The algorithms for the calculations within the min-plus/max-plus algebra and the analysis steps themselves can be partly parallelized. The verification framework can be modified accordingly to benefit from analysis platforms with multi-core processing capabilities.

Appendix A

A.1 Related work

A.1.1 Selected modeling approaches of the automotive domain

This section introduces selected modeling approaches for the design and development of software and electronics in the automotive domain and highlights the relations with each other. Refer to [149] for a brief comparison of the approaches with a focus on consistency.

Automotive Open System Architecture (AUTOSAR). AUTOSAR [101, 150] is a standardized and open software architecture to provide the automotive manufacturers and suppliers a common specification and guideline for the development of vehicle software and electronics. It was developed by several automotive manufacturers, suppliers, tool creators and semi-conductor producers. The first version was published in 2005 and is continuously improved, leading to yearly or two-yearly releases. A discussion of the evolution is provided in [43]. AUTOSAR includes a standardization of the software architecture running on electronic control units, a set of basic services including the application programming interfaces, a communication abstraction and semantic definition, a definition of the development methodology of automotive electronic systems, and a specification of a set of standard signals present in the classic automotive domains. It includes a meta-model that supports the development according to the AUTOSAR architecture and methodology. Based on this meta-model, systems can be integrated and exchanged between different manufacturers and suppliers. The meta-model itself is a based on UML2 [42]. AUTOSAR can be regarded as the most important standard for the collaborative development of automotive software.

Electronics Architecture and Software Technology – Architecture Description Language (EAST-ADL). EAST-ADL was developed within two European research projects. Its design goal was the extension of the functionality of AUTOSAR, therefore a unidirectional mapping between EAST-ADL elements to AUTOSAR elements exists. An AUTOSAR model can still exist without the EAST-ADL extension, but usually not the other way around. The second release, **EAST-ADL2**, includes four abstraction layers [151]: Vehicle Level, Analysis Level, Design Level and Implementation Level. Each level refines its predecessor, including non-functional requirements. The bottom level, the implementation level, corresponds to a system description within AUTOSAR and renders the point of highest refinement. During the development of EAST-ADL, non-functional requirements like timing constraints were

not yet included in AUTOSAR. As the development of AUTOSAR continued, more and more functions previously exclusive to EAST-ADL were integrated into the AUTOSAR standard.

Architecture Analysis and Design Language (AADL). AADL [152] is a domain-specific language for the design of distributed, real-time embedded systems and was standardized by the Society of Automotive Engineers (SAE). AADL was initially developed and used in the field of avionics and was known as the Avionics Architecture Description Language before it was renamed. AADL is not an UML profile, but an own meta-language specification. It combines the modeling of functional and non-functional properties of embedded systems and focuses on a description close to the actual real-world system. Compared to AUTOSAR, the aspect of non-functional properties was always deeply integrated from the beginning on into AADL. On the other hand, more abstract views of the system, like a use case or feature specification, are missing. Compared to EAST-ADL, the focus of AADL is closer to the actual hardware [153].

Modeling and Analysis of Real Time and Embedded Systems (MARTE). MARTE [46] is a UML2 profile to support the development of real-time and embedded applications. Compared to AADL, MARTE is a more generic representation supporting a multitude of views of embedded systems and implementations. However, it was pointed out in [154] that a description of a system in MARTE can be simpler compared to AADL. The standard also includes a mapping of MARTE elements to AADL elements. This can be used to transform a MARTE model into an AADL representation and to analyze it with the according tools. Non-functional properties are an integrated part of MARTE, especially the notion and handling of time. Resources can be modeled in detail, e.g., scheduling strategies, buffer semantics and synchronization strategies. A harmonization to benefit from the advantages of MARTE, EAST-ADL and AUTOSAR was discussed in [155].

Systems Modeling Language (SysML). SysML [44] is a graphical modeling language based on UML2 to describe complex systems. The standard was developed by the Object Management Group (OMG) in conjunction with several industrial partners. The first standard was available in 2007 and subsequently refined in the next years. SysML partly directly adopts diagrams from UML and also introduces new types. The diagrams are categorized into four pillars: Structure, behavior, requirements, and parametrics, where the latter two describe new diagram types compared to UML. The aim of SysML is to provide a standardized mean to describe the behavior and requirements of systems. Those descriptions should be used for the communication between different partners that are involved in the design and development of a certain system, and should also be the base for the analysis and evaluation of design alternatives. Compared to AUTOSAR, SysML focuses on a higher level of abstraction and is comparable to EAST-ADL, while EAST-ADL features more domain-specific extensions, i.e., automotive related, while SysML is more generic and also suitable for avionic and other systems.

A.1.2 Comparison of time handling in selected frameworks

This section gives an overview of selected frameworks and standards that are applied to specify properties and constraints of the timing behavior of systems.

AUTOSAR Timing Extension (TIMEX). Since version 4.0, AUTOSAR includes means to express timing-related requirements and properties of automotive systems including a methodology to handle these requirements. This extension is included in the official AUTOSAR specification [55]. The Timing Extension covers different levels of the implementation, but lacks traceability across all these levels [156]. The different levels are [55]: *Virtual Functional Bus Timing (VfbTiming)* for the specification of constraints on a logical level, without regarding the distribution, internal behavior or execution platform of software components. *Software Component Timing (SwcTiming)* handles the internal behavior of software components, especially the behavior of runnable tasks that implement the functionality. Each software component can include several tasks and interactions. *System Timing (SystemTiming)* includes the deployment of the software components and therefore can map the communication between components either to a local communication process or an external communication technology. *Basic Software Module Timing (BswModuleTiming)* captures all constraints related to the modules of the basic software, i.e., the drivers and runtime system of a certain electronic control unit. The *BswModuleTiming* is similar to the *SwcTiming* but refers to basic software modules that are determined during the configuration process of an ECU. The *Electronic Control Unit Timing (EcuTiming)* is the most concrete representation level for timing constraints and includes all information that is necessary to determine the timing of one single electronic control unit, i.e., configuration and mapping of the basic software modules and software components, complete configuration of the ECU and mapping of messages and signals to internal and external communication means. Compared to the *SystemTiming*, this view also includes all the information about the basic software modules and their interactions. The timing constraints themselves are organized into eight groups: *Event triggering*, *latency timing*, *age*, *synchronization timing for events*, *synchronization timing for event chains*, *offset timing*, *execution order* and *execution time*. The *age* constraint abstracts from the concrete event chain and can be used to specify requirements on the freshness of data if the source is unknown. While in the *synchronization timing for events and event chains* it is assumed that either a common stimulus or response exists to formulate the constraint, the *offset timing* is utilized to relate events that do not have a common cause or reaction, for example when an event is processed by a cyclic execution of a software component. In that case, the processing is not related to the actual arriving event. The *execution order* does not bound timings, but allows the specification of constraints relating the execution order of either software components or events in a standard, hierarchical, or cyclic manner. A comparison to the proposed approach in this thesis is provided in Sec. 4.9.

Timing Augmented Description Language V2 (TADL2). TADL2 was adopted with version 2.1.11 (2013) into the EAST-ADL2 specification [45]. It was developed in the TIMMO-2-USE [109] project and replaced its predecessor TADL. TADL2 covers the feature, analysis, and design levels of a vehicle and thus is focused on higher levels of abstraction than AUTOSAR, which mainly captures the implementation level [157]. In fact, TADL2 models can reference events from AUTOSAR and allow a seamless traceability of the constraints. The other way around, a referencing of TADL2 model elements from AUTOSAR, is not possible. TADL2 allows the specification of probabilistic timings, constraints dependent from modes,

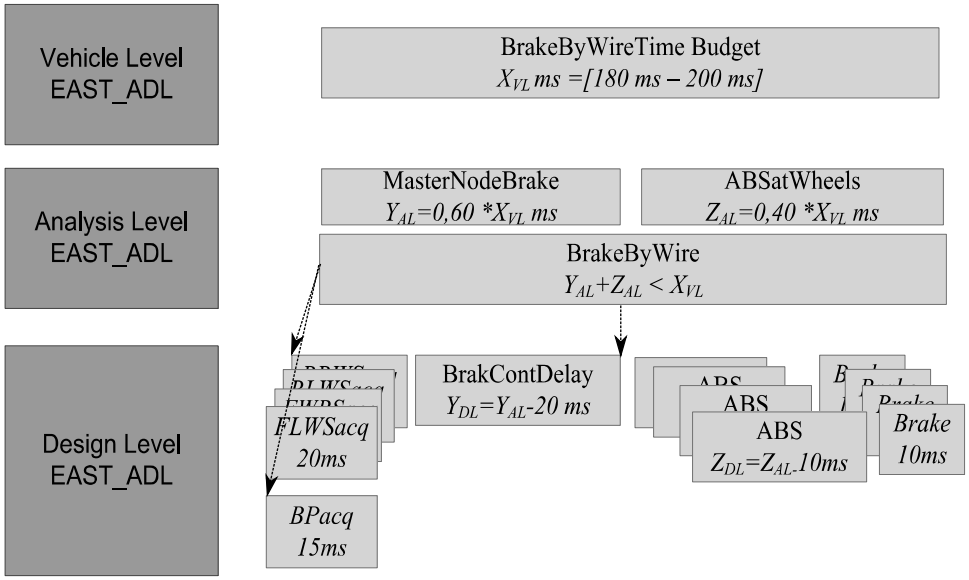


Figure A.1: Example of the abstraction levels of EAST-ADL and the according TADL2 annotations for a break-by-wire system (adapted from [109]). The analysis and design levels make usage of symbolic expressions to specify timing requirements.

and allows the usage of symbolic expressions for constraints, see Fig. A.1. The concrete constraints, like *SynchronizationConstraint*, *PatternConstraint* or *DelayConstraint*, form on the one hand a superset of the AUTOSAR constraints as they are more expressiveness, and are on the other hand aligned to AUTOSAR if possible. A concrete comparison of the relationships between the timing constraints available in AUTOSAR and TADL2 is presented in [109]. Compared to the proposed approach of this work, the refinement of timing constraints is a manual process in TADL2.

Architecture Analysis and Design Language (AADL). The AADL allows the specification of constraints on event flows. The property *Latency* can be specified for end-to-end flows, flow specifications and connections [110]. It represents the maximum allowed time difference for an event to enter and exit an event flow. The delays for the execution of communication and implementation processes can be bounded in the same manner. Semantics to describe constraints for the synchronization between events or for the pattern of events do not exist yet. A more expressiveness model for timing specification is designed within annexes that are under development [158].

SysML [44], a profile of UML [42] for systems engineering applications, has rather limited capabilities to express timing properties and constraints. However, an annotation *rate* is available, which can be attached to flows, ports, and blocks to specify the expected data, event, or block rate. For the rate specification also distributions are allowed. The UML elements *communication*, *interaction overview* and *timing diagram* were explicitly excluded from the SysML specification. This shows that the focus of SysML is not the detailed modeling of the timing behavior. UML on the other hand gives means to define interval-based,

absolute, or symbolic constraints on durations of certain actions. But the expressiveness of these constraints is limited and an end-to-end specification is not directly considered [42].

Clock Constraint and Specification Language (CCSL). MARTE [46] with its CCSL directly addresses the specification of timing properties and requirements. While SysML is suitable for the modeling at a system level, MARTE's focus is the complete opposite direction by enabling technology-dependent and very exact descriptions of concrete implementations. MARTE's capabilities are the most generic of the derivates introduced in this section and hence it is very complex [159]. The specification includes guidelines to map EAST-ADL2 and AADL models to the MARTE syntax for analysis. MARTE distinguishes between *chronometric clocks* that are bound to the real time and *logical clocks*. A specification can have multiple clock sources and the relations between individual clocks and events can be expressed with the CCSL, including relations like "clock A is finer than clock B", "clock A is faster than clock B", "clock A is periodic with clock B" or "clock A alternates with clock B". Many properties of digital physical clocks can be modeled like offsets, clock skew or drifting. Several arrival patterns for events are available including periodic, aperiodic, sporadic, and burst patterns. Constraints may be attached to processing elements, end-to-end flows or event streams, e.g., limitations for the maximum jitter or delay. Utilization of elements and data rates can also be modeled as well as the laxity of the constraints (hard real time, soft real time or user-defined). Compared to the proposed approach of this work, the adaptivity of a system is not in the focus of the CCSL.

A.1.3 Extensions and improvements to Real-Time Calculus

Since the first introduction of Real-Time Calculus in [30] several improvements and extensions to the framework have been proposed by the research community. In the following, some of the key results are highlighted: **Interface-based design** in the context of RTC was described in [130] and [77]. The idea is to extend the interfaces of components to also include information about the assumed and guaranteed properties of resource and event streams. This enables the automatic proof of compatibility if two components are linked to each other. It is comparable to a contract-based composition approach and thus does not include end-to-end requirement specification and analysis possibilities. The concept of interface-based design was refined to be able to **minimize the energy needs** for a single processing unit [160] and pipelined multiprocessor systems [115]. This is possible by defining the desired performance of the system and optimizing the processor speed(s) while obeying the constraints. A more complete discussion of this topic is available in [161]. The effect of **cyclic dependencies** between components was analyzed in [87] and a fixed point solution was shown. That work focused on cycles formed by resource dependencies, i.e., the available resource to a component depends on a component later in the event chain. This concept was further enhanced by [136] and [137] for component networks that can be represented by marked graphs and that can have data flow dependencies, i.e., the arrival pattern of events at a component depends on the arrival pattern at its output. Real-Time Calculus can capture the behavior of stateless systems. But often also a stateful analysis is demanded, e.g., to model special buffer access and overriding strategies. For this reason, an **interface between Real-Time Calculus and timed automata** was developed in [73] and

further refined in [71]. This method can be applied to transform arrival curves into a representation suitable for the analysis with timed automata. Beyond, a transformation of the results back into the framework of the Real-Time Calculus was shown. A similar approach was presented in [142], where the RTC was connected to the synchronous programming language Lustre. The assumption of infinite FIFO-buffers at the input of processing elements was weaken in [93] by considering **data refresh semantics**. The authors present a method to directly model finite buffers within the RTC framework. The work is an adaption of the analysis of systems with losses in [84]. The approach discussed in [162] focuses on another aspect: Instead of lossy systems, systems with **blocking-write semantics** and **state-based scheduling policies** are considered. Both results make the transformation to stateful analysis methods obsolete for certain cases as those can be directly expressed within the RTC framework. Some extensions to RTC add **phase information** to the analysis to increase the accuracy of the results. In [163], the correlation of events in a split-join scenario is exploited to increase the tightness. In the analyzed scenario, events are first split to several processing units and afterwards joined again to a common event stream. As the distribution to the processing units follows a round-robin fashion in the scenario and the processing time is equal for all units, the results have a well-known phase shift from each other, which is exploited for an improved analysis. The correlation of events to the workload was analyzed in [64] for cases where each event or group of events has different resource demands. With the help of **structured streams** the analysis of join-fork scenarios for event streams can be improved. For example, this could happen during the traversal of a communication stack where multiple event streams are joined before being processed by the stack and afterwards are separated again into individual streams. This concept was introduced in [164] and refined in [71]. A lot of efforts were put into the speedup of the analysis process, either by an **approximation approach** or by an **optimized computation**. An approximation based on a three-piece linear approximation was presented in [165, 166, 145], without the possibility of a seamless switch during an analysis. Another approximation with improved tightness, but also increased analysis time, was given in [144] by a linearization of the arrival and service curves from a dynamic point on. In [167] and [168], a hierarchical decomposition of arrival curves to adjust the analysis time and tightness was shown. However, the approach also needs a modification of the transfer functions and feasibility checks of the processing elements and therefore is not ad-hoc compatible with the mathematical background of the RTC. The work in [143] proposes a method for an optimized computation of component networks without the loss of any tightness. Some of these aspects are further discussed and developed in Ch. 6 and Ch. 7.

A.1.4 Comparison of selected plug-and-play and reconfiguration approaches

In the following, a brief comparison of the proposed system-wide plug-and-play concept to selected other approaches, mainly from research projects, is given.

CHROMOSOME Middleware and Run-Time Environment. CHROMOSOME is developed at the research institute fortiss (since 2011) and consists of a run-time environment and a configuration tool to build *"Plug&Play-Capable Embedded Real-Time Systems"* [95]. It utilizes a model-based development process for component-based applications with com-

munication semantics according to the data-centric paradigm. It distinguishes a "Plug" phase, where newly added components are detected and changed schedules and communication routes are established, and a "Play" phase, where the configuration is then executed. CHROMOSOME was applied in the RACE project [27] (described below) and in the AutoPnP project [146] to realize adaptable manufacturing systems. Compared to the approach presented in this work, the specification and verification of system-wide timing requirements was not in the focus and is not available as an integrated part of the plug-and-play procedure. However, similar are the component-based and data-centric development process and the ability to integrate new functions into an existing system based on a model of the currently running setup and a specification of properties of the newly added functions.

Robust and Reliant Automotive Computing Environment for Future eCars (RACE). The aim of the RACE project (2012-2015) was the development of a centralized ICT infrastructure for vehicles, which helps to reduce the design and implementation complexity. The resulting RACE architecture [27] is a synchronized, globally time-triggered execution framework with plug-and-play capabilities that support safety-critical functions [20]. Application logic is developed according to the data-centric and component-based design principles. The execution framework and development tools are an adaption of the CHROMOSOME middleware [95]. Because of its separation of communication and computation cycles, the deployment location of software components has no impact on the timing behavior. To check the feasibility of a setup on the node level, the worst-case execution times of individual components are summed up and it is verified that the result is still within the duration of a time slot. While this approach guarantees that a single node is within its limits, it does not handle the properties of data flows traversing multiple components on one or several nodes. In addition, a possible over-utilization of the communication infrastructure is not considered and the flexibility of the system is limited due to the globally time-triggered architecture.

Dynamically Self-Configuring Automotive Systems (DySCAS). The aim of the DySCAS [96] project (2006-2008) was the development of a middleware for adaptive and self-configurable systems in the automotive domain. It was part of the 6th framework program "Information Society Technologies" of the European Commission. Beyond the middleware, the results include a model-based approach to specify composable components of a system. From a first look, this is very similar to the approach presented here. For example, the use cases are analogous to the ones used in this work, as they compromise: The automatic integration of devices into the vehicle IT systems, the integration of new software functionality, and a closed reconfiguration, i.e., the degradation or migration of functions in case of failures [96]. In DySCAS, tools were analyzed and proposed to perform a verification of the system configuration automatically [169]. However, the verification approach was not completed and in the end limited to the verification of the single-node case. Furthermore, the flexible configuration of the analysis in terms of tightness and computation time was not scope of the project. The applied analysis technology was based on timed automata [170] rather than on the Real-Time Calculus as presented here. For the online case, a reconfiguration strategy was presented [171], which takes into account the utilization of different

electronic control units across a network and matches the input and output signals to each other. That work did not cover the timed interaction of tasks and the utilization handling is based on a simple approximation.

Framework for Real-time Embedded Systems based on COntRacts (FRESCOR). The FRESCOR project's (2006-2009) main objective was *"to develop the enabling technology and infrastructure required to effectively use the most advanced techniques developed for real-time applications with flexible scheduling requirements, in embedded systems design methodologies and tools, providing the necessary elements to target reconfigurable processing modules and reconfigurable distributed architectures."* [98] The result was a contract-based design model (called FRSH) in connection with an execution platform. The main difference to this work is that only local guarantees can be given with the contract-based approach. Global properties, like the timing of flows through several software components, are not captured. Although all components according to the FRESCOR methodology may pass a feasibility test, the behavior of the overall system might still not meet the assumed constraints. The analysis of the systems itself is based on the sporadic server principle and was presented in [172].

AUTOSAR does not directly support an equivalent mechanism like proposed in this work. The limiting point is that parts of the software are generated according to the specified communication demands of the software components. This configuration cannot be changed after the deployment of the system. To achieve a rudimentary amount of flexibility, it is standard to make the implementation of the software components themselves changeable. This means that the implementation of a software component can be replaced in the firmware of an electronic control unit. This works as long as the replaced image sticks to the same external interface and does not violate the previous negotiated properties like the timing behavior and memory demand. Therefore, this approach is suitable when the external interface is constant and when bug fixes or features should be added in this way. However, it is usually not possible to install new software components in the system or to alter the communication relations. To do so, the firmware has to be re-generated and it is very likely that it has an impact on several ECUs.

DREAMS (Distributed Real-time Architecture for Mixed Criticality Systems). The DREAMS project (2013-2017) aims at the development of an architecture that supports the distributed execution of mixed-criticality multicore systems, considering reconfiguration and security aspects. The project started in 2013 and is going to last until 2017. As of writing of this thesis, the deliverables for the requirements and building blocks specification are available [173] and a description of the architectural style [174]. It is visible from these documents that the architecture shall support a dynamic reconfiguration considering system-wide constraints like the end-to-end timing between certain functions across multiple nodes. According to the requirements document, a candidate for the specification of timing requirements is the Timing Augmented Description Language V2 (see Appx. A.1.2) and an analysis based on the principles of the Network Calculus is considered. For the resource management and dynamic reconfiguration, the results of the projects FRESCOR,

ACTORS, DIVA, and SCARLETT shall be integrated. The concrete realization is to be defined in the work packages during the remaining project duration.

DiVA (Dynamic Variability in complex, Adaptive systems). In the EU-ICT DiVA (2008-2011) project, software models are utilized at execution and design time to create dynamically adapting component-based applications (*models-at-runtime* approach) [94]. During runtime, the performance of the system is monitored and a reconfiguration can be triggered to adapt the system to changes. Several variants of the system's architecture are evaluated and the best fitting configuration is chosen. This process depends on a fuzzy description of the impact factors on the system, e.g., "high", "medium" and "low" priorities [103], which are mapped to numbers to calculate a score for a possible configuration candidate. The whole process is comparable with a design-space exploration method, but with a feedback loop from the running system and a model-centric approach that also allows a reconfiguration during runtime. Although the scope of the project does not cover hard real-time systems and is therefore different to the approach presented in this work, it certainly shows how a completely model-driven approach can be used for the adaption of systems. Work started to standardize the underlying syntax and semantics as the Common Variability Language (CVL) [175].

SCARLETT (SCAlable and ReconfigurabLe Electronics plaTforms and Tools). The focus of the SCARLETT project (2008-2012) was the development of the next generation of the Integrated Modular Avionic (IMA) platform to reduce development and maintenance cost and speed up the integration of functions in the avionic domain [104]. The proposed platform contains services to re-allocate functions in case of a node failure. This does not directly re-assemble the plug-and-play principle proposed in this thesis but already includes many of the steps as an altered configuration has to be validated and accepted before it is actually executed. This can either happen in a pre-calculated fashion or also online on the proposed platform of the SCARLETT project.

A.1.5 Comparison of performance analysis tools

This section gives an overview and comparison of selected tools and frameworks for the performance analysis of networked and embedded systems, which are based on the Network Calculus, Real-Time Calculus or similar concepts. The frameworks and tools differ in the representation of curves, available operations, and features to handle the systems to analyze; a summary is presented in Tab. A.1.

The closest match to the approach presented in this work is the **Real-Time Calculus Toolbox** [30] that was developed at the ETH Zürich. The toolbox consists of a Java library for the min-plus/max-plus calculus operations and a set of Matlab functions that implement filter operations. The Matlab part is called Modular Performance Analysis, because it implements a compositional systems analysis. The analysis with the Real-Time Calculus Toolbox itself is a manual process, an automatic interpretation of an adaptive system graph is not implemented. Compared to the proposed approach, the subadditive closure operation is limited, cycles are not handled automatically, the analysis of filters with bounded buffer semantics is not available, and an exchange of analysis tightness for computation time is not possible. The toolbox was integrated into a framework for the exploration of architectural

designs for MPSoCs (Multiprocessor System-on-Chip) [138], which is able to generate the according Matlab code out of a system model. It can also generate the code for resource cycles in the system graph in a greedy manner, but the utilized initialization and iteration strategies for the fixed point calculation differ from the approach presented in this work, refer to Sec. 6.4 for details. **chronVal** [176] is a commercial tool for the timing verification of real-time systems, based on a hierarchical event stream model introduced by [167]. The event stream model allows an approximation to reduce the complexity of feasibility tests and filter operations. The approach deploys the Real-Time Calculus framework for the calculation of response times in distributed systems, the interfacing between both worlds was shown in [177]. According to [176], *"The chronVAL analysis is mainly based on the Real-Time Calculus with the necessary supplements for a better support of realistic systems. These are for example analysis methods for cooperative scheduling, support of offset analysis and so on."* As the implementation or specification was not available to the author of this work, a detailed analysis was not conducted. The **Cyclic Network Calculus (CyNC)** [178] is a theory and tool to calculate the performance of systems with loops in the data flow. It was developed at the Aalborg University and the tool comes as a set of Matlab functions and Simulink modeling elements. With the Simulink modeling elements, it is possible to define a process network in a graphical manner. The approach is based on a discrete description of arrival and service curves and does not support ultimately pseudo-periodic functions. Periodic functions can be defined but are approximated from a configurable point on. In general, CyNC extends the Network Calculus theory by the ability to process loops. The **Deterministic Network Calculator (DiscoDNC)** [179] from the University of Kaiserslautern is a tool for the system-level performance analysis on the basis of the Network Calculus. It includes the possibility to specify a network graph with an automatic execution of the corresponding stream filters. The underlying curves are represented as a finite amount of linear segments. Ultimately pseudo-periodic functions are not supported. DiscoDNC comes as a set of commented Java source code files that can be easily extended with further functionality. Except the convolution and deconvolution of curves, most of the complex operations like subadditive closure and pseudo-inverses are missing. The **Computational Issues in Network Calculus (COINC)** [132] library offers basic operations to manipulate curves according to the min-plus algebra. It uses an alternative representation of curves according to the model presented in [86]. The representation allows partly a more convenient expression of the mathematical operations. Except for the one publication, no further mentioning of this tool are known to the author of this work. The implementation seems rudimentary. **NC-Maude** [180] is a specification of Network Calculus operations in the tool Maude. Maude itself is an implementation of rewriting logic, i.e., it is possible to specify algebraic systems with operations and relations. Given an algebraic expression, Maude can automatically reduce (rewrite) it until a more compact or normalized state is reached. As the operators and algebraic classes are in a certain order, the aim of the rewriting task is implicitly defined. NC-Maude is a set of files for Maude that implement an algebra and rules to operate on curves within the Network Calculus. As far as the author of this work knows, NC-Maude is based on curve representations with a finite set of linear segments. According to the author of NC-Maude, the amount of features is a subset of the features of the COINC library. The

Table A.1: Tools for system level performance analysis, based on Network Calculus, Real-Time Calculus or similar concepts.

Name	Organization/Company	Development	Comments	References
Tools based on Real-Time Calculus, Non-Commercial				
Real-Time Calculus Toolbox (Matlab)	ETH Zürich	2006 - 2011 (?)	Uses ultimately pseudo-periodic functions, limited closure operator	[65, 30, 92, 77, 71, 137]
Tools based on Real-Time Calculus, Commercial				
chronVAL (GUI)	Inchron	2012(?)-2015+	Approximation possible, own event stream model	[176, 167, 177]
Tools based on Network Calculus, Non-Commercial				
Cyclic Network Calculus (CyNC) (Matlab/Simulink)	Aalborg Universitet	2005 (?)	Uses ultimately affine functions	[139, 178]
Computational Issue in Network Calculus (COINC) (Min-Plus Interpreter)	INRIA	2009 (?)	Alternative curve representation, rudimentary implementation	[132]
Deterministic Network Calculator (DiscoDNC) (Java Sources)	University of Kaiserslautern	2006-2015+	Only concave/convex functions, finite amount of segments	[179, 184]
NC-Maude (Library for Maude interpreter)	ONERA	2010-2014(?)	Based on rewriting logic, ultimately affine functions only	[183, 180]
Delay Bound Rating Algorithm (DEBORAH) (C++ Sources)	University of Pisa	2008-2010	FIFO and Tandem configurations, limited scope	[181]
Tools based on Network Calculus, Commercial				
Min-Plus Console (Min-Plus Interpreter)	Real-Time at Work	2010-2012(?)	Ultimately pseudo-periodic functions, closure-operator limited	[182]
Further tools (neither NC nor RTC)				
Symbolic Timing Analysis for Systems (SymTA/S) (Commercial) (GUI)	Symtavision	2005-2015+	Classic local analysis with generic interfaces	[69, 70, 185]
Compositional Analysis of Real-Time Systems (CARTS) (Java GUI) (Research)	University of Pennsylvania	2009	Hierarchical scheduling analysis	[183, 186]

Delay Bound Rating Algorithm (DEBORAH) [181] by the University of Pisa is a tool for the evaluation of FIFO and tandem configurations with the Network Calculus. The curve representations are restricted, i.e., a convolution is only implemented for a curve with two segments, which makes it inapplicable in a generic context. The **Min-Plus Console** [182] was developed by the company Real-Time at Work within the research project PEGASE. The Min-Plus console is a command-line interpreter to execute operations within the min-plus algebra. It features two implementations for evaluation: One based on increasing convex or concave curves for a fast processing in trade for expressiveness of the available curves, and an implementation based on the class of ultimately pseudo-periodic functions. It further implements operators for the subadditive closure and pseudo-inverse. But it seems that these operators only work properly for the class of increasing convex and concave functions, but not for the more general class of ultimately pseudo periodic functions. The tool is free for research and educational purposes. Real-time at Work offers industry-grade libraries for calculations based on the min-plus algebra. These libraries were not evaluated in this work as they are not freely available. **Symbolic Timing Analysis for Systems (SymTA/S)** [69] is a commercial tool for the end-to-end timing analysis of heterogeneous architectures. Locally, it uses classic scheduling approaches to calculate the performance of components. The interconnection of components follows a standardized event model, which makes the approach composable. Because the event model between the components is based on a fixed amount of parameters, the expressiveness is limited. The approach does not support hierarchical scheduling analysis, because the remaining service of a component is not derived. The **Compositional Analysis of Real-Time Systems (CARTS)** [183] is a theory and tool for the performance calculation of hierarchical resource interfaces. It covers the calculation of deadlines, jitters, and utilization of task sets that are scheduled under various strategies like earliest deadline first (EDF) and rate-monotonic (RM). Although it is not possible to calculate end-to-end delays with CARTS, because a representation of events is not part of the analysis, this tool is useful for feasibility checks of heterogeneous scheduler configurations on one node, for example as experienced by the utilization of virtualization technologies.

A.2 Definitions and equations of stream filters

In the following, the definition and equations of the stream filters applied in the examples are given, based on the Real-Time Calculus framework. None of the equations were initially developed by the author of this work, the references to the according sources and modifications to it are given in the descriptions.

A.2.1 Greedy processing component (GPC)

A greedy processing component (GPC) is a stream filtering block of the Real-Time Calculus framework that models the processing of events in a greedy fashion. Whenever resources are available and at least one event arrives or is already in the input queue, the event is immediately processed. In case more events arrive than can be processed, the events are queued in an infinite-sized buffer.

Definition A.1 *The GPC interface is defined as*

$$(\alpha_{out}, \beta_{out}, d, b) = f_{GPC}(\alpha_{in}, \beta_{in}, \gamma) \tag{A.1}$$

where α_{in}, $\alpha_{out} \in \mathbb{F}$ are pairs of arrival curves, β_{in}, $\beta_{out} \in \mathbb{F}$ are pairs of service curves, $d, b \in \mathbb{R}$ are the scalars maximum delay and backlog, and $\gamma \in \mathbb{F}$ is a pair of workload curves.

Definition A.2 *The transfer function for the GPC is derived and proved in [77]:*

$$\begin{aligned}
\alpha^u_{out} &= \lceil \min\{(\alpha^u_{in} \otimes \widehat{\beta}^u_{in}) \oslash \widehat{\beta}^l_{in}, \widehat{\beta}^u_{in}\} \rceil \\
\alpha^l_{out} &= \lfloor \min\{(\alpha^l_{in} \oslash \widehat{\beta}^u_{in}) \otimes \widehat{\beta}^l_{in}, \widehat{\beta}^l_{in}\} \rfloor \\
\beta^u_{out} &= (\beta^u_{in} - \widehat{\alpha}^l_{in}) \mathbin{\overline{\oslash}} 0 \\
\beta^l_{out} &= (\beta^l_{in} - \widehat{\alpha}^u_{in}) \mathbin{\overline{\otimes}} 0 \\
d &\leq D(\alpha^u_{in}, \lfloor \widehat{\beta}^l_{in} \rfloor) \\
b &\leq B(\alpha^u_{in}, \lfloor \widehat{\beta}^l_{in} \rfloor)
\end{aligned} \tag{A.2}$$

with $\widehat{\beta}^l, \widehat{\beta}^u, \widehat{\alpha}^l, \widehat{\alpha}^u$ according to Eq. 2.12 or Eq. 2.14.

Note that in the older publication [187], a different definition was given for the GPC component. The difference are the time domains of consideration: While the here presented equations from [77] consider the whole time domain $t \in [-\infty, +\infty]$, the equations from [187] are valid for the positive time domain only [81].

A.2.2 Fixed-priority non-preemptive component (FPNP)

The fixed-priority non-preemptive processing component (FPNP) is a stream filtering block of the RTC framework that models the processing of events in a greedy-like fashion, but might get blocked for a certain amount of time by a non-preemptive component with lower priority. The FPNP component itself is not necessarily non-preemptive, but one or more components along the remaining service stream chain. This definition is different from those in [92], where a FPNP component has several streams as input, which are all non-preemptive. We have chosen the alternative setup as it enhances the modularity of the

analysis process. Another derivation of the equations was presented in [121] with better tightness according to the authors, but it was not applied as the processing is more complex.

Definition A.3 *The FPNP interface is defined as:*

$$(\alpha_{out}, \beta_{out}, d, b) = f_{FPNP}(\alpha_{in}, \beta_{in}, \gamma, \delta_{max}) \tag{A.3}$$

where α_{in}, $\alpha_{out} \in \mathbb{F}$ are pairs of arrival curves, β_{in}, $\beta_{out} \in \mathbb{F}$ are pairs of service curves, $d, b \in \mathbb{R}$ are the scalars maximum delay and backlog, $\gamma \in \mathbb{F}$ is a pair of workload curves and $\delta_{max} \in \mathbb{R}$ is the maximum quantity of resources demanded by a lower-priority non-preemptive task.

Definition A.4 *The transfer function of the FPNP is an adaption from [92]:*

$$\begin{aligned}
\alpha^u_{out} &= \lceil \min\{(\alpha^u_{in} \otimes \check{\beta}^u) \oslash \check{\beta}^l, \check{\beta}^u\} \rceil \\
\alpha^l_{out} &= \lfloor \min\{(\alpha^l_{in} \oslash \check{\beta}^u_i) \otimes \check{\beta}^l, \check{\beta}^l\} \rfloor \\
\beta^u_{out} &= (\beta^u_{in} - \widehat{\alpha}^l_{in}) \mathbin{\overline{\oslash}} 0 \\
\beta^l_{out} &= (\beta^l_{in} - \widehat{\alpha}^u_{in}) \mathbin{\overline{\otimes}} 0 \\
d &\leq D(\alpha^u_{in}, \lfloor \check{\beta}^l_{in} \rfloor) \\
b &\leq B(\alpha^u_{in}, \lfloor \check{\beta}^l_{in} \rfloor)
\end{aligned} \tag{A.4}$$

with

$$\begin{aligned}
\check{\beta}^u &= \gamma^{l-1}(\beta^u_{in}) \\
\check{\beta}^l &= \gamma^{u-1}\left(\max(\beta^l_{in} - \delta_{max}, 0)\right)
\end{aligned}$$

and $\widehat{\alpha}^l, \widehat{\alpha}^u$ according to Eq. 2.12 or Eq. 2.14.

A.2.3 First-in-first-out processing component (FIFO)

The first-in-first-out processing component (FIFO) is a stream filtering block of the RTC framework that models the processing of events from multiple event streams in a FIFO manner. Whenever resources are available, the incoming events are processed in the same order as they arrived. If the processing of events is deferred because of missing resources or because other events arrived before, the according events are queued in a buffer of infinite length. The deployed equations are taken from [71]. In [77], another realization based on a mapping to earliest deadline first filters is suggested, which was also implemented in [65]. The latter possibility was not considered in this work.

Definition A.5 *The FIFO interface is defined as:*

$$(\vec{\alpha}_{out}, \beta_{out}, \vec{d}, \vec{b}) = f_{FIFO}(\vec{\alpha}_{in}, \beta_{in}, \vec{\gamma}) \tag{A.5}$$

where $\vec{\alpha}_{in}$, $\vec{\alpha}_{out} \in \mathbb{F}$ are a vector of pairs of arrival curves, β_{in}, $\beta_{out} \in \mathbb{F}$ are pairs of service curves, $\vec{d}, \vec{b} \in \mathbb{R}$ are the maximum delay and backlog results, and $\vec{\gamma} \in \mathbb{F}$ is a vector of pair of workload curves.

Definition A.6 *Then the transfer functions of the FIFO are as follows [71]:*

$$\begin{aligned}
\alpha^u_{out,i} &= \lceil \min\{(\alpha^u_{in,i} \otimes \check{\beta}^u_i) \oslash \check{\beta}^l_i, \check{\beta}^u_i\} \rceil \\
\alpha^l_{out,i} &= \lfloor \min\{(\alpha^l_{in,i} \oslash \check{\beta}^u_i) \otimes \check{\beta}^l_i, \check{\beta}^l_i\} \rfloor \\
\beta^u_{out} &= (\beta^u_{in} - \sum_i \widehat{\alpha}^l_{in,i}) \,\overline{\oslash}\, 0 \\
\beta^l_{out} &= (\beta^l_{in} - \sum_i \widehat{\alpha}^u_{in,i}) \,\overline{\otimes}\, 0 \\
d_i &\leq D(\alpha^u_{in}, \lfloor \check{\beta}^l_{in} \rfloor) \\
b_i &\leq B(\alpha^u_{in}, \lfloor \check{\beta}^l_{in} \rfloor)
\end{aligned} \tag{A.6}$$

with

$$\begin{aligned}
\check{\beta}^u_i &= \gamma^{l-1}_i(\beta^u_{in}) \\
\check{\beta}^l_i &= \gamma^{u-1}_i\left((\beta^l_{in} - \sum_{j \neq i} \widehat{\alpha}^u_j) \,\overline{\otimes}\, 0 \right)
\end{aligned}$$

and $\widehat{\alpha}^l, \widehat{\alpha}^u$ according to Eq. 2.12 or Eq. 2.14.

A.2.4 OR component (OR)

The OR processing component is a stream filtering block of the RTC framework that joins a set of arrival streams. Whenever an event is available at any of the input streams, one event at the output is created. This component does not introduce delay nor backlog into the stream and does not depend on any service resources. The equations are available in [92] or [77].

Definition A.7 *The OR interface is defined as:*

$$\alpha_{out} = f_{OR}(\vec{\alpha}_{in}) \tag{A.7}$$

where $\vec{\alpha}_{in} \in \mathbb{F}$ is a vector of pairs of arrival curves and $\alpha_{out} \in \mathbb{F}$ is a pair of arrival curves.

Definition A.8 *The transfer functions of the OR is [92]:*

$$\begin{aligned}
\alpha^u_{out} &= \sum_i \alpha^u_{in,i} \\
\alpha^l_{out} &= \sum_i \alpha^l_{in,i}
\end{aligned} \tag{A.8}$$

A.2.5 AND component (AND)

The AND processing component is a stream filtering block of the RTC framework that processes a set of two incoming arrival streams. Whenever an event is available at both inputs, one event at the output is created and one event from each input queue is removed. It is possible that events sum up in the input queues. The AND component is feasible for setups

where both input streams have an equal long-term slope. Otherwise, the number of events in one of the input queues ultimately grows to infinity and bounds for the delay and backlog go to infinity, too. The equations are from [92] and [77] and are presented here without considering initial tokens in the input queues.

Definition A.9 *The AND interface is defined as:*

$$(\alpha_{out}, \vec{d}, \vec{b}) = f_{AND}(\vec{\alpha}_{in}) \tag{A.9}$$

where $\vec{\alpha}_{in} \in \mathbb{F}$ is a vector of two pairs of arrival curves, $\alpha_{out} \in \mathbb{F}$ is the resulting pair of arrival curves, and $\vec{d}, \vec{b} \in \mathbb{R}$ are the maximum delay and backlog results.

Definition A.10 *The transfer functions of the AND are [92]:*

$$\begin{aligned}
\alpha^u_{out} &= \max\left\{\min\{\alpha^u_1 \oslash \alpha^l_2, \alpha^u_2\}, \min\{\alpha^u_2 \oslash \alpha^l_1, \alpha^u_1\}\right\} \\
\alpha^l_{out} &= \min\left\{\max\{\alpha^l_1 \mathbin{\overline{\oslash}} \alpha^u_2, \alpha^l_2\}, \max\{\alpha^l_2 \mathbin{\overline{\oslash}} \alpha^u_1, \alpha^l_1\}\right\} \\
d_1 &\leq D(\alpha^u_1, \alpha^l_2) \\
d_2 &\leq D(\alpha^u_2, \alpha^l_1) \\
b_1 &\leq \max\left\{B(\alpha^u_1, \alpha^l_2), 0\right\} \\
b_2 &\leq \max\left\{B(\alpha^u_2, \alpha^l_1), 0\right\}
\end{aligned} \tag{A.10}$$

A.2.6 Bounded greedy processing component (BGPC)

The bounded greedy processing component (BGPC) is equal to the greedy processing component but implements the semantics of a bounded buffer at the input. If the buffer is full and another event approaches the component, the oldest event is deleted from the buffer and the arriving one is enqueued. Therefore, this component is useful in scenarios, where under-sampling occurs, i.e., the data is processed slower than received. The presented equations are adapted from [93].

Definition A.11 *The BGPC interface is defined as*

$$(\alpha_{out}, \beta_{out}, d, b) = f_{BGPC}(\alpha_{in}, \beta_{in}, \gamma, b_{max}) \tag{A.11}$$

where α_{in}, α_{out} are pairs of arrival curves, β_{in}, β_{out} are pairs of service curves, d and b are the scalars maximum delay and backlog, γ is a pair of workload curves and B_{max} is the maximum buffer size of the input queue.

Definition A.12 *Transfer function of the BGPC according to [93], but omitting the refinement of the results [93, Lemma 3.9]:*

$$\begin{aligned}
\alpha^u_{out} &= \lceil \min\{(\alpha^u_{in} \otimes \check{\beta}^u) \oslash \check{\beta}^l, \check{\beta}^u\} \rceil \\
\alpha^l_{out} &= \lfloor \min\{(\alpha^l \oslash \check{\beta}^u) \otimes \check{\beta}^l, \check{\beta}^l\} \rfloor \\
\beta^u_{out} &= (\beta^u_{in} - \check{\alpha}^l) \mathbin{\overline{\oslash}} 0 \\
\beta^l_{out} &= (\beta^l_{in} - \check{\alpha}^u) \mathbin{\overline{\otimes}} 0 \\
b &\leq B_{max} \\
d &\leq \min\left\{V(\alpha^l_{in}, B_{max}), V(\widehat{\beta}^l_{in}, B_{max}), D\left(\min(\alpha^u, \widehat{\beta}^u + B_{max}), \widehat{\beta}^l_{in}\right)\right\}
\end{aligned} \tag{A.12}$$

with

$$\breve{\alpha}^u = \min\{(\alpha^u_{in} \otimes \overline{\beta}^u_{in}) \oslash \overline{\beta}^l_{in}, \overline{\beta}^u_{in}\} \qquad \breve{\beta}^u = \widehat{\beta}^u_{in} \otimes \overline{\beta}^u$$
$$\breve{\alpha}^l = \min\{(\alpha^l_{in} \oslash \overline{\beta}^u_{in}) \otimes \overline{\beta}^l_{in}, \overline{\beta}^l_{in}\} \qquad \breve{\beta}^l = \widehat{\beta}^l_{in} \otimes \overline{\beta}^l$$
$$V(x(\Delta), k) = \min\{\Delta \geq 0 | x(\Delta) \geq k\} \qquad \overline{\beta}^u = \alpha^u_{in} \otimes (\alpha^u_{in} \otimes \widehat{\beta}^u_{in} + B_{max})^\star$$
$$\overline{\beta}^l = (\widehat{\beta}^l_{in} + B_{max})^\star$$

and $\widehat{\beta}^l, \widehat{\beta}^u, \widehat{\alpha}^l, \widehat{\alpha}^u$ *according to Eq. 2.12 or Eq. 2.14.*

Note that $\overline{b}$ refers to a service-curve in an event-based representation and $()^\star$ is the subadditive closure of a function.

A.3 Complete parameter set of the eCar example

The parameters in the Tabs. A.2, A.3, A.4, and A.5 are exemplary values that were applied for the eCar example to evaluate the verification framework with various approximation strategies.

Entity	Parameter	Value	Feature
ECU Center	Processor speed	30k cycles/ms	Base
ECU Center → HMI Serial	Stack latency	2 ms	Base
ECU Center → Ethernet Center	Stack latency	0.5 ms	Base
ECU Front	Processor speed	25k cycles/ms	Base
ECU Front → Ethernet Center	Stack latency	2 ms	Base
ECU Back	Processor speed	20k cycles/ms	Base
HMI Serial	Data rate	62.5 bytes/ms	Base
CAN Front/Back	Data rate	31.25 bytes/ms	Base
CAN Front/Back	Latency	0.5 ms	Base
Ethernet Center	Data rate	1250 bytes/ms	Base
Ethernet Center	Latency	1 ms	Base

Table A.2: Parameter set of the eCar example – SYSTEM model.

Component	WCET/BCET	Type	Mapped to	Feature
Control Central	90k cycles	HighPriorityET	ECU Center	Mov./Cntr.
Control Front	60k cycles	HighPriorityET	ECU Front	Mov./Cntr.
Control Back	60k cycles	HighPriorityET	ECU Back	Mov./Cntr.
Camera Process	120k cycles	LowPriorityET	ECU Front	Camera
Camera Forwarder	5k cycles	LowPriorityET	ECU Center	Camera

Table A.3: Parameter set of the eCar example – DEPLOYMENT model (software components).

Topic	Size	Mapped to	Priority	Feature
MovementVector	32 bytes	HMI Serial	n/a	Mov./Cntr.
ControllerFront	48 bytes	Ethernet Center	n/a	Mov./Cntr.
ControllerBack	48 bytes	Ethernet Center	n/a	Mov./Cntr.
WheelFrontLeft	8 bytes	CAN Front	0	Mov./Cntr.
WheelFrontRight	8 bytes	CAN Front	1	Mov./Cntr.
WheelBackLeft	8 bytes	CAN Back	0	Mov./Cntr.
WheelBackRight	8 bytes	CAN Back	1	Mov./Cntr.
CameraRaw	40 bytes	CAN Front	2	Camera
CameraProcessed	48 bytes	Ethernet Center	n/a	Camera
CameraSignalHMI	8 bytes	HMI Serial	n/a	Camera

Table A.4: Parameter set of the eCar example – DEPLOYMENT model (messages).

External source	Period	Jitter	Min. distance
Camera	16.0	16.0	5.0
Sidestick	10.0	20.0	5.0

Table A.5: Parameter set of the eCar example – LOGICAL model.

A.4 Complete ANALYSIS model ($\mathfrak{M}$) of the eCar example

The ANALYSIS model of the eCar example (Fig. A.2) is the derived model after the transformation and mapping process, which renders the input for the verification tool.

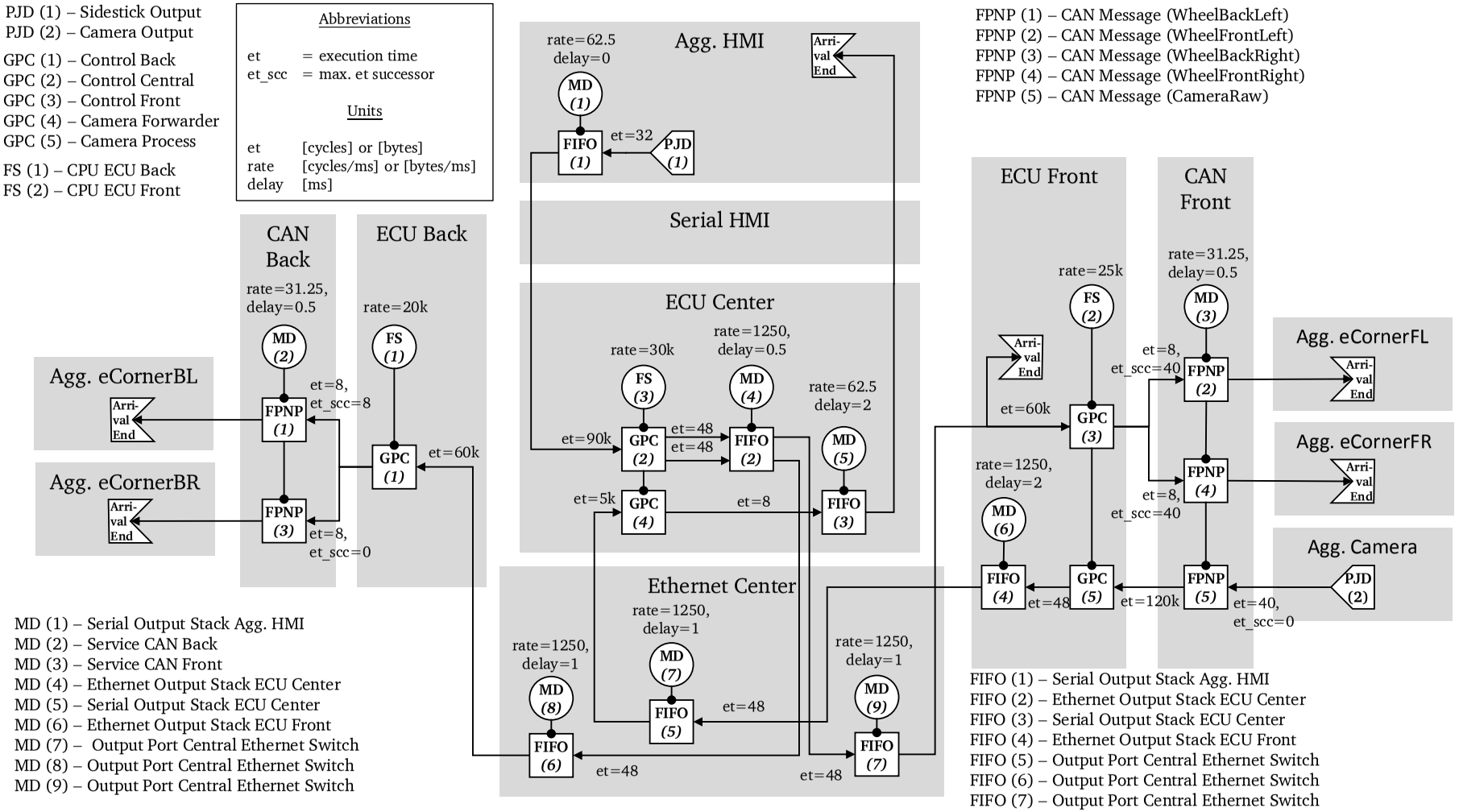


Figure A.2: Complete ANALYSIS model of the eCar example.

References

[1] Robert N. Chartte. **This Car Runs on Code**. *IEEE Spectrum*, 2009.

[2] Ulrich Abelein, Helmut Lochner, Daniel Hahn, and Stefan Straube. **Complexity, quality and robustness - the challenges of tomorrow's automotive electronics**. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 870–871, Dresden, March 2012. IEEE.

[3] Manuel Bernard, Christian Buckl, Volkmar Döricht, Marcus Fehling, Ludger Fiege, Helmuth von Grolman, Nicolas Ivandic, Christoph Janello, Cornel Klein, Karl-Josef Kuhn, Christian Platzlaff, Bettine Cassandra Riedl, Bernhard Schätz, and Christian Stanek. **Mehr Software (im) Wagen : Informations- und Kommunikationstechnik (IKT) als Motor der Elektromobilität der Zukunft**. Technical report, fortiss GmbH, 2011.

[4] Christian Buckl, Alexander Camek, Gerd Kainz, Carsten Simon, Ljubo Mercep, Hauke Stähle, and Alois Knoll. **The Software Car: Building ICT Architectures for Future Electric Vehicles**. In *2012 IEEE International Electric Vehicle Conference - IEVC'12*, pages 1–8, Greenville, SC, USA, March 2012. IEEE.

[5] Ljubo Mercep, Claudia Buitkamp, Hauke Stähle, Gernot Spiegelberg, Alois Knoll, and Markus Lienkamp. **The Innotruck Case Study on a Holistic Approach to Electric Mobility**. In *5th International Conference on Sustainable Automotive Technologies - ICSAT'13*, pages 277–287, Ingolstadt, Germany, September 2013. Springer International Publishing.

[6] Alois Knoll, Ljubo Mercep, and Hauke Stähle. **Electric Mobility: Chances and Technical Challenges**. In Hardo Bruhns, editor, *Energie - Technologien und Energiewirtschaft*, pages 47–62. Deutsche Physikalische Gesellschaft, Dresden, Germany, November 2013.

[7] Gitta Rohling, Gernot Spiegelberg, Hauke Stähle, Ljubo Mercep, and Claudia Buitkamp. **Das Fahrzeug der Zukunft**. *Faszination Forschung (ISSN 1865-3022)*, (10):76–84, June 2012.

[8] Sonja Haustein, Anu Siren, Elisabeth Framke, Daniel Bell, Eike Pokriefke, Aline Alauzet, Claude Marin-Lamellet, Jimmy Armoogum, and Desmong O'Neil. **CONcerns and SOLutions - Road Safety in the Ageing Societies: WP1 Demographic Change and Transport Final Report**. Technical Report February, CONSOL Partners, 2013.

[9] International Organization for Standardization (ISO); TC 22/SC 3. **ISO 26262:2011. Road vehicles - Functional safety**, 2010.

[10] Vladimir Rupanov, Christian Buckl, Ludger Fiege, Michael Armbruster, Alois Knoll, and Gernot Spiegelberg. **Early Safety Evaluation of Design Decisions in E/E Architecture according to ISO 26262**. In *Proceedings of the 3rd international ACM SIGSOFT*

symposium on Architecting Critical Systems - ISARCS '12, page 1, Bertinoro, Italy, 2012. ACM Press.

[11] EUROPEAN PARLIAMENT AND THE COUNCIL OF THE EUROPEAN UNION. **Decision No 585/2014/EU of the European Parliament and of the Council of 15 May 2014 on the deployment of the interoperable EU-wide eCall service**, 2014.

[12] TROY R. HAWKINS, OLA MOA GAUSEN, AND ANDERS HAMMER STRØMMAN. **Environmental impacts of hybrid and electric vehicles - a review**. *The International Journal of Life Cycle Assessment*, **17**(8):997–1014, September 2012.

[13] SAM DOECKE, ALEX GRANT, AND ROBERT W. G. ANDERSON. **The Real-World Safety Potential of Connected Vehicle Technology**. *Traffic Injury Prevention*, **16**(sup1):S31–S35, June 2015.

[14] PURNENDU SINHA. **Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives**. *Reliability Engineering & System Safety*, **96**(10):1349–1359, October 2011.

[15] LJUBO MERCEP. *Context-Centric Design of Automotive Human-Machine Interfaces*. Phd thesis, Technische Universität München, 2014.

[16] LJUBO MERCEP, GERNOT SPIEGELBERG, AND ALOIS KNOLL. **A Case Study on Implementing Future Human-Machine Interfaces**. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1077–1082, Gold Coast City, Australia, June 2013. IEEE.

[17] GERNOT SPIEGELBERG, ANDREAS SCHWARZHAUPT, OTTMAR GEHRING, ARMIN A. SULZMANN, AND OLIVER ROOKS. **Using drive-by-wire technology to design integrated powertrain modules - integration of the evaluation of surrounding variables**. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, **5**, pages 3719–3728. IEEE, American Automatic Control Council, 2002.

[18] MARCO DI NATALE AND ALBERTO L. SANGIOVANNI-VINCENTELLI. **Moving From Federated to Integrated Architectures in Automotive: The Role of Standards, Methods and Tools**. *Proceedings of the IEEE*, **98**(4):603–620, April 2010.

[19] SHANE TUOHY, MARTIN GLAVIN, CIARAN HUGHES, EDWARD JONES, MOHAN TRIVEDI, AND LIAM KILMARTIN. **Intra-Vehicle Networks: A Review**. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–12, 2014.

[20] JELENA FRTUNIKJ, VLADIMIR RUPANOV, ALEXANDER CAMEK, CHRISTIAN BUCKL, AND ALOIS KNOLL. **A Safety Aware Run-Time Environment for Adaptive Automotive Control Systems**. In *Embedded Real-Time Software and Systems (ERTS2)*, Toulouse, France, 2014.

[21] BERNHARD SCHÄTZ. **The Role of Models in Engineering of Cyber-Physical Systems - Challenges and Possibilities**. In *10. Dagstuhl-Workshop MBEES: Modellbasierte Entwicklung eingebetteter Systeme X*, pages 91–96, March 2014.

[22] EDWARD ASHFORD LEE AND SANJIT ARUNKUMAR SESHIA. *Introduction to Embedded Systems - A Cyber-Physical Systems Approach*. UC Berkeley, second edition, 2015.

[23] GERARDO PARDO-CASTELLOTE, BERT FARABAUGH, AND RICK WARREN. **An Introduction to DDS and Data-Centric Communications**. Technical report, Real-Time Innovations (RTI), 2005.

[24] ROMAN OBERMAISSER, CHRISTIAN EL SALLOUM, BERNHARD HUBER, AND HERMANN KOPETZ. **From a Federated to an Integrated Automotive Architecture**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **28**(7):956–965, July 2009.

[25] P. PETI, R. OBERMAISSER, F. TAGLIABO, A. MARINO, AND S. CERCHIO. **An Integrated Architecture for Future Car Generations**. In *Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 2–13, Seattle, May 2005. IEEE.

[26] HAUKE STÄHLE, LJUBO MERCEP, ALOIS KNOLL, AND GERNOT SPIEGELBERG. **Towards the Deployment of a Centralized ICT Architecture in the Automotive Domain**. In *2nd Mediterranean Conference on Embedded Computing - MECO'13*, pages 66–69, Budva, Montenegro, June 2013. IEEE.

[27] STEPHAN SOMMER, ALEXANDER CAMEK, KLAUS BECKER, CHRISTIAN BUCKL, ANDREAS ZIRKLER, LUDGER FIEGE, MICHAEL ARMBRUSTER, GERNOT SPIEGELBERG, AND ALOIS KNOLL. **RACE: A Centralized Platform Computer Based Architecture for Automotive Applications**. In *2013 IEEE International Electric Vehicle Conference (IEVC)*, pages 1–6, Santa Clara, CA, USA, October 2013. IEEE.

[28] FRÉDÉRIC HOLZMANN, MARIO BELLINO, SASCHA KOLSKI, ARMIN SULZMANN, GERNOT SPIEGELBERG, AND ROLAND SIEGWART. **Robots go automotive - the SPARC approach**. In *Intelligent Vehicles Symposium 2005*, pages 478–483, Las Vegas, NV, USA, June 2005. IEEE.

[29] MARIO BELLINO, YURI LOPEZ DE MENESES, PETER RYSER, AND JACQUES JACOT. **Lane detection algorithm for an onboard camera**. In THOMAS P. PEARSALL, editor, *SPIE proceedings of the first Workshop on Photonics in the Automobile*, pages 102–111, Geneva, Switzerland, February 2005.

[30] LOTHAR THIELE, SAMARJIT CHAKRABORTY, AND MARTIN NAEDELE. **Real-time calculus for scheduling hard real-time systems**. In *2000 IEEE International Symposium on Circuits and Systems*, pages 101–104, Geneva, Switzerland, May 2000. Presses Polytech. Univ. Romandes.

[31] CLEMENS SZYPERSKI, DOMINIK GRUNTZ, AND MURER STEPHAN. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, second edition, 2002.

[32] IVICA CRNKOVIC, JUDITH STAFFORD, AND CLEMENS SZYPERSKI. **Software Components beyond Programming: From Routines to Services**. *IEEE Software*, **28**(3):22–26, May 2011.

[33] JAN CARLSON, JOHN HÅKANSSON, AND PAUL PETTERSSON. **SaveCCM: An Analysable Component Model for Real-Time Systems**. *Electronic Notes in Theoretical Computer Science*, **160**:127–140, August 2006.

[34] ANETA VULGARAKIS, JAGADISH SURYADEVARA, JAN CARLSON, CRISTINA SECELEANU, AND PAUL PETTERSSON. **Formal Semantics of the ProCom Real-Time Component Model**. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 478–485, Patras, Greece, August 2009. IEEE.

[35] MARCO PANUNZIO AND TULLIO VARDANEGA. **A component-based process with separation of concerns for the development of embedded real-time software systems**. *Journal of Systems and Software*, **96**:105–121, October 2014.

[36] OBJECT MANAGEMENT GROUP (OMG). **Data Distribution Service for Real-time Systems Version 1.2**, 2007.

[37] JEAN BÉZIVIN. **In search of a basic principle for Model Driven Engineering**. *Special Novatica Issue - UML and Model Engineering*, **5**(2):21–24, 2004.

[38] THOMAS STAHL, MARKUS VÖLTER, JORN BETTIN, ARNO HAASE, SIMON HELSEN, AND KRZYSZTOF CZARNECKI. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons Ltd., 2006.

[39] OBJECT MANAGEMENT GROUP (OMG). **MDA Guide Version 1.0.1**, 2003.

[40] DAVE STEINBERG, FRANK BUDINSKY, ED MERKS, AND MARCELO PATERNOSTRO. *EMF: Eclipse Modeling Framework 2.0*. Addison-Wesley Professional, second edition, 2009.

[41] OBJECT MANAGEMENT GROUP (OMG). **OMG Meta Object Facility (MOF) Core Specification Version 2.4.1**, 2013.

[42] OBJECT MANAGEMENT GROUP (OMG). **OMG Unified Modeling Language (OMG UML) Version 2.5**, 2015.

[43] DARKO DURISIC, MIROSLAW STARON, MATTHIAS TICHY, AND JORGEN HANSSON. **Evolution of Long-Term Industrial Meta-Models – An Automotive Case Study of AUTOSAR**. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 141–148, Verona, Italy, August 2014. IEEE.

[44] OBJECT MANAGEMENT GROUP (OMG). **Systems Modeling Language (SysML)**, 2012.

[45] EAST-ADL ASSOCIATION. **EAST-ADL Domain Model Specification V2.1.12**, 2013.

[46] OBJECT MANAGEMENT GROUP (OMG). **UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems**, 2011.

[47] FRÉDÉRIC JOUAULT, FREDDY ALLILAIRE, JEAN BÉZIVIN, AND IVAN KURTEV. **ATL: A model transformation tool**. *Science of Computer Programming*, **72**(1-2):31–39, 2008.

[48] OBJECT MANAGEMENT GROUP (OMG). **Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification Version 1.2**, 2005.

[49] ALEXANDER KRAAS. **Realizing Model Simplifications with QVT Operational Mappings**. In *4th International Workshop on OCL and Textual Modelling co-located with 17th International Conference on Model Driven Engineering Languages and Systems (MODELS 2014)*, pages 53–62, Valencia, Spain, 2014.

[50] DIMITRIS KOLOVOS, LOUIS ROSE, ANTONIO GARCÍA-DOMÍNGUEZ, AND RICHARD PAIGE. *The Epsilon Book*. Eclipse Foundation, 2015.

[51] LESLIE LAMPORT. **Time, Clocks, and the Ordering of Events in a Distributed System**. *Communications of the ACM*, **21**(7):558–565, July 1978.

[52] HERMANN KOPETZ. *Real-Time Systems*. Real-Time Systems Series. Springer US, Boston, MA, 2011.

[53] M. RAYNAL AND M. SINGHAL. **Logical Time: Capturing Causality in Distributed Systems**. *Computer*, **29**(2):49–56, 1996.

[54] REINHARD WILHELM, TULIKA MITRA, FRANK MUELLER, ISABELLE PUAUT, PETER PUSCHNER, JAN STASCHULAT, PER STENSTRÖM, JAKOB ENGBLOM, ANDREAS ERMEDAHL, NIKLAS HOLSTI, STEPHAN THESING, DAVID WHALLEY, GUILLEM BERNAT, CHRISTIAN FERDINAND, AND REINHOLD HECKMANN. **The Worst-Case Execution Time Problem - Overview of Methods and Survey of Tools**. *ACM Transactions on Embedded Computing Systems*, **7**(3):1–53, April 2008.

[55] AUTOSAR 4.2.1. **Specification of Timing Extensions**, 2014.

[56] 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 *IEEE Real-Time System Symposium - Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*, Barcelona, Spain, 2008.

[57] MARTIJN HENDRIKS AND MARCEL VERHOEF. **Timed Automata Based Analysis of Embedded System Architectures**. In *Proceedings 20th IEEE International Parallel & Distributed Processing Symposium*, page 8 pp., Rhodes Island, Greece, April 2006. IEEE.

[58] BERNARD BERTHOMIEU AND MICHEL DIAZ. **Modeling and Verification of Time Dependent Systems Using Time Petri Nets**. *IEEE Transactions on Software Engineering*, **17**(3):259–273, March 1991.

[59] SIMON PERATHONER, ERNESTO WANDELER, LOTHAR THIELE, ARNE HAMANN, SIMON SCHLIECKER, RAFIK HENIA, RAZVAN RACU, ROLF ERNST, AND MICHAEL GONZÁLEZ HARBOUR. **Influence of different abstractions on the performance analysis of distributed hard real-time systems**. *Design Automation for Embedded Systems*, **13**(1-2):27–49, April 2008.

[60] SEBASTIAN REITER, ANDREAS BURGER, ALEXANDER VIEHL, OLIVER BRINGMANN, AND WOLFGANG ROSENSTIEL. **Virtual Prototyping Evaluation Framework for Automotive Embedded Systems Engineering**. In *Proceedings of the Seventh International Conference on Simulation Tools and Techniques*, Lisbon, Portugal, March 2014. ICST.

[61] ARQUIMEDES CANEDO, JIANG WAN, AND MOHAMMAD ABDULLAH AL FARUQUE. **Functional Modeling Compiler for System-Level Design of Automotive Cyber-Physical Systems**. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 39–46, San Jose, CA, USA, November 2014.

[62] HOSAM K. FATHY, ZORAN S. FILIPI, JONATHAN HAGENA, AND JEFFREY L. STEIN. **Review of Hardware-in-the-Loop Simulation and Its Prospects in the Automotive Area**. In KEVIN SCHUM AND ALEX F. SISTI, editors, *Modeling and Simulation for Military Applications*, **6228**, pages 1–20, May 2006.

[63] VEJLUPEK JOSEF, GREPL ROBERT, KREJCI PETR, LESAK FRANTISEK, AND MATOUS KAREL. **Hardware-In-the-Loop Simulation for Automotive Parking Assistant Control Units**. In *Proceedings of the 16th International Conference on Mechatronics - Mechatronika 2014*, pages 325–330, Brno, Czech Republic, December 2014. IEEE.

[64] ERNESTO WANDELER AND LOTHAR THIELE. **Characterizing Workload Correlations in Multi Processor Hard Real-Time Systems**. In *11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS 2005)*, pages 46–55, San Francisco, CA, USA, March 2005. IEEE.

[65] ERNESTO WANDELER AND LOTHAR THIELE. **Real-Time Calculus (RTC) Toolbox**, 2006.

[66] KEN TINDELL AND JOHN CLARK. **Holistic schedulability analysis for distributed hard real-time systems**. *Microprocessing and Microprogramming*, **40**(2-3):117–134, April 1994.

[67] M. GONZALEZ HARBOUR, J.J. GUTIERREZ GARCIA, J.C. PALENCIA GUTIERREZ, AND J.M. DRAKE MOYANO. **MAST: Modeling and Analysis Suite for Real Time Applications**. In *Proceedings 13th Euromicro Conference on Real-Time Systems (ECRTS01)*, pages 125–134, Delft, The Netherlands, June 2001. IEEE Comput. Soc.

[68] MICHAEL GONZÁLEZ HARBOUR. **Analysis Techniques used in MAST**. Technical report, Grupo de Ingeniería de Software y Tiempo Real, Universidad de Cantabria, 2014.

[69] R. HENIA, A. HAMANN, M. JERSAK, R. RACU, K. RICHTER, AND R. ERNST. **System level performance analysis - the SymTA/S approach**. *IEEE Proceedings - Computers and Digital Techniques*, **152**(2):148–166, 2005.

[70] KAI RICHTER. *Compositional Scheduling Analysis Using Standard Event Models - The SymTA/S Approach*. Phd thesis, Technical University of Braunschweig, 2005.

[71] SIMON PERATHONER. *Modular Performance Analysis of Embedded Real-Time Systems: Improving Modeling Scope and Accuracy*. Phd thesis, Swiss Federal Institute of Technology Zurich (ETH), 2011.

[72] RAJEEV ALUR AND DAVID L. DILL. **A theory of timed automata**. *Theoretical Computer Science*, **126**(2):183–235, April 1994.

[73] KAI LAMPKA, SIMON PERATHONER, AND LOTHAR THIELE. **Analytic Real-Time Analysis and Timed Automata: A Hybrid Methodology for the Performance Analysis of Embedded Real-Time Systems**. *Design Automation for Embedded Systems*, **14**(3):193–227, 2010.

[74] KIM G. LARSEN, PAUL PETTERSSON, AND WANG YI. **UPPAAL in a nutshell**. *International Journal on Software Tools for Technology Transfer*, **1**(1-2):134–152, February 1997.

[75] KAI LAMPKA AND LOTHAR THIELE. **Combining computational and analytic model descriptions for evaluating embedded real-time systems**. Technical report, Computer Engineering and Communication Networks Lab., ETH Zurich, Switzerland, 2008.

[76] STEFFEN KOLLMAN AND VICTOR POLLEX. **Comparative Application of Real-Time Verification Methods to an Automotive Architecture**. In *18th International Conference on Real-Time and Network Systems*, pages 89–98, Toulouse, November 2010.

[77] ERNESTO WANDELER. *Modular Performance Analysis and Interface-Based Design for Embedded Real-Time Systems*. Phd thesis, Swiss Federal Institute of Technology Zurich (ETH), 2006.

[78] GODOFREDO R. GARAY, JULIO ORTEGA, AND VICENTE ALARCON-AQUINO. **Comparing Real-Time Calculus with the Existing Analytical Approaches for the Performance Evaluation of Network Interfaces**. In *21st International Conference on Electrical Communications and Computers (CONIELECOMP 2011)*, pages 119–124, Cholula, Mexico, February 2011. IEEE.

[79] THILO STREICHERT AND MATTHIAS TRAUB. *Elektrik/Elektronik-Architekturen im Kraftfahrzeug - Modellierung und Bewertung von Echtzeitsystemen*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[80] ERNESTO WANDELER, LOTHAR THIELE, MARCEL VERHOEF, AND PAUL LIEVERSE. **System architecture evaluation using modular performance analysis: a case study**. *International Journal on Software Tools for Technology Transfer*, **8**(6):649–667, July 2006.

[81] SAMARJIT CHAKRABORTY, SIMON KUNZLI, AND LOTHAR THIELE. **A General Framework for Analysing System Properties in Platform-Based Embedded SystemDesigns**. In *2003 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 190–195, Munich, Germany, February 2003. IEEE Comput. Soc.

[82] RENE L. CRUZ. **A Calculus for Network Delay, Part I: Network Elements in Isolation**. *IEEE Transactions on Information Theory*, **37**(1):114–131, 1991.

[83] RENE L. CRUZ. **A Calculus for Network Delay, Part II: Network Analysis**. *IEEE Transactions on Information Theory*, **37**(1):132–141, 1991.

[84] JEAN-YVES LE BOUDEC AND PATRICK THIRAN. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet*, **2050** of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, January 2001.

[85] ANNE BOUILLARD, LAURENT JOUHET, AND ERIC THIERRY. **Service curves in Network Calculus: dos and don'ts**. Technical report, INRIA (French Institute for Research in Computer Science and Automation), 2009.

[86] FRANCOIS BACCELLI, GUY COHEN, GEERT J. OLSDER, AND JEAN-PIERRE QUADRAT. *Synchronization and Linearity - An Algebra for Discrete Event Systems*. École des Ponts ParisTech, web edition, 2001.

[87] BENGT JONSSON, SIMON PERATHONER, LOTHAR THIELE, AND WANG YI. **Cyclic Dependencies in Modular Performance Analysis**. In *Proceedings of the 8th ACM international conference on Embedded software - EMSOFT '08*, page 179, Atlanta, GA, USA, October 2008. ACM Press.

[88] MATTHIEU MOY AND KARINE ALTISEN. **Arrival Curves for Real-Time Calculus: The Causality Problem and Its Solutions**. In *Proceedings of the 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, **6015**, pages 358–372, Paphos, Cyprus, March 2010.

[89] KARINE ALTISEN AND MATTHIEU MOY. **Causality Closure for a New Class of Curves in Real-Time Calculus**. In *1st International Workshop on Worst-Case Traversal Time*, pages 3–10, Vienna, Austria, November 2011. ACM Press.

[90] ERNESTO WANDELER AND LOTHAR THIELE. **Optimal TDMA Time Slot and Cycle Length Allocation for Hard Real-Time Systems**. In *Proceedings of the 11th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 479–484, Yokohama, Japan, January 2006.

[91] ALEXANDER MAXIAGUINE, SIMON KÜNZLI, AND LOTHAR THIELE. **Workload Characterization Model for Tasks with Variable Execution Demand**. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE)*, **2**, pages 1040–1045, Paris, France, February 2004. IEEE.

[92] WOLFGANG HAID AND LOTHAR THIELE. **Complex Task Activation Schemes in System Level Performance Analysis**. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis - CODES+ISSS '07*, pages 173–178, Salzburg, Austria, October 2007. ACM Press.

[93] LINH THI XUAN PHAN, REINHARD SCHNEIDER, SAMARJIT CHAKRABORTY, AND INSUP LEE. **Modeling Buffers with Data Refresh Semantics in Automotive Architectures**. In *Proceedings of the tenth ACM international conference on Embedded software - EMSOFT '10*, page 119, Scottsdale, AZ, USA, October 2010. ACM Press.

[94] BRICE MORIN, OLIVIER BARAIS, JEAN-MARC JEZEQUEL, FRANCK FLEUREY, AND ARNOR SOLBERG. **Models@Run.time to Support Dynamic Adaptation**. *Computer*, **42**(10):44–51, October 2009.

[95] CHRISTIAN BUCKL, MICHAEL GEISINGER, DHIRAJ GULATI, FRAN J. RUIZ-BERTOL, AND ALOIS KNOLL. **CHROMOSOME: A Run-Time Environment for Plug & Play-Capable Embedded Real-Time Systems**. *ACM SIGBED Review*, **11**(3):36–39, 2014.

[96] RICHARD ANTHONY, ACHIM RETTBERG, DEJIU CHEN, ISABELL JAHNICH, GERRIT DE BOER, AND CECILIA EKELIN. **Towards a Dynamically Reconfigurable Automotive Control System Architecture**. *Embedded System Design: Topics, Techniques and Trends*, **231**:71–84, 2007.

[97] COMPAQ; HEWLETT-PACKARD; INTEL; LUCENT; MICROSOFT; NEC; PHILIPS. **Universal Serial Bus Specification Revision 2.0**, 2000.

[98] FRESCOR PARTNERS. **Framework for Real-time Embedded Systems based on COntRACTS (FRESCOR) Deliverable - Final Project Report**. Technical report, Universidad de Cantabria, 2005.

[99] WERNER DAMM, HARDI HUNGAR, BERNHARD JOSKO, THOMAS PEIKENKAMP, AND INGO STIERAND. **Using Contract-based Component Specifications for Virtual Integration Testing and Architecture Design**. In *2011 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, Grenoble, France, March 2011.

[100] ALBERTO SANGIOVANNI-VINCENTELLI, WERNER DAMM, AND ROBERTO PASSERONE. **Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems**. *European Journal of Control*, **18**(3):217–238, January 2012.

[101] AUTOSAR 4.2.1. **Methodology**, 2014.

[102] KLAUS BECKER AND SEBASTIAN VOSS. **Analyzing Graceful Degradation for Mixed Critical Fault-Tolerant Real-Time Systems**. In *2015 IEEE 18th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 110–118, Auckland, New Zealand, April 2015.

[103] FRANCK FLEUREY AND ARNOR SOLBERG. **A Domain Specific Modeling Language Supporting Specification, Simulation and Execution of Dynamic Adaptive Systems**. *Lecture Notes in Computer Science*, **5795 LNCS**:606–621, 2009.

[104] PIERRE BIEBER, FRÉDÉRIC BIONIOL, MARC BOYER, ERIC NOULARD, AND CLAIRE PAGETTI. **New Challenges for Future Avionic Architectures**. *Aerospace Lab Journal*, **11**(4):1–10, 2012.

[105] JAKOB AXELSSON AND AVENIR KOBETSKI. **On the Conceptual Design of a Dynamic Component Model for Reconfigurable AUTOSAR Systems**. In *ACM SIGBED Review - Special Issue on the 5th Workshop on Adaptive and Reconfigurable Embedded Systems*, pages 45–48, Philadelphia, PA, USA, December 2013.

[106] MARTIN EDER AND ALOIS KNOLL. **Design of an Experimental Platform for an X-by-wire Car with Four-wheel Steering**. In *2010 IEEE International Conference on Automation Science and Engineering*, pages 656–661, Toronto, Canada, August 2010. IEEE.

[107] HAUKE STÄHLE, KAI HUANG, AND ALOIS KNOLL. **Drive-by-Wireless with the eCar Demonstrator**. In *4th ACM SIGBED International Workshop on Design, Modeling, and Evaluation of Cyber-Physical Systems - CyPhy'14*, pages 19–22, Berlin, Germany, April 2014. ACM Press.

[108] ROBERT BOSCH GMBH. **CAN Specification Version 2.0**, 1991.

[109] TIMMO-2-USE. **Language syntax, semantics, metamodel V2 Version 1.2**, 2012.

[110] PETER H. FEILER AND JÖRGEN HANSSON. **Flow Latency Analysis with the Architecture Analysis and Design Language (AADL)**. Technical Report CMU/SEI-2007-TN-010, Software Engineering Institute, Carnegie Mellon University, 2007.

[111] ARDA GOKNIL, JAGADISH SURYADEVARA, MARIE AGNÈS PERALDI-FRATI, AND FRÉDÉRIC MALLET. **Analysis Support for TADL2 Timing Constraints on EAST-ADL Models**. *Lecture Notes in Computer Science*, **7957 LNCS**:89–105, 2013.

[112] JEAN BÉZIVIN, SALIM BOUZITOUNA, MARCOS DIDONET DEL FABRO, MARIE-PIERRE GERVAIS, FRÉDÉRIC JOUAULT, DIMITRIOS S. KOLOVOS, IVAN KURTEV, AND RICHARD F. PAIGE. **A Canonical Scheme for Model Composition**. *Model Driven Architecture - Foundations and Applications*, pages 346–360, 2006.

[113] RODOLFO PELLIZZONI, ANDREAS SCHRANZHOFER, JIAN-JIA CHEN, MARCO CACCAMO, AND LOTHAR THIELE. **Worst Case Delay Analysis for Memory Interference in Multicore Systems**. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 741–746, Dresden, Germany, March 2010. IEEE.

[114] ANDREAS SCHRANZHOFER. *Efficiency and predictability in resource sharing multicore systems*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH), 2011.

[115] GANG CHEN, KAI HUANG, AND ALOIS KNOLL. **Adaptive Dynamic Power Management for Hard Real-time Pipelined Multiprocessor Systems**. In *2014 IEEE 20th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pages 1–10, Chongqing, China, August 2014. IEEE.

[116] JAMES F. KUROSE AND KEITH W. ROSS. *Computer Networking - A Top-Down Approach*. Pearson, sixth edition, 2012.

[117] JORK LOESER AND HERMANN HAERTIG. **Low-latency Hard Real-Time Communication over Switched Ethernet**. In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 2004)*, pages 13–22, Catania, Italy, June 2004. IEEE.

[118] JEAN-PHILIPPE GEORGES, THIENY DIVOUX, AND ERIC RONDEAU. **Comparison of Switched Ethernet Architectures Models**. In *Proceedings of the 9th IEEE International Conference on Emerging Technologies and Factory Automation (EFTA)*, **1**, pages 375–382, Lisbon, Portugal, September 2003. IEEE.

[119] KASPER REVSBECH, HENRIK SCHIØLER, TATIANA K. MADSEN, AND JIMMY J. NIELSEN. **Worst-Case Traversal Time Modelling of Ethernet Based In-Car Networks Using Real Time Calculus**. *Lecture Notes in Computer Science*, **6869/2011**:219–230, 2011.

[120] MATTHIAS A. TRAUB. *Durchgängige Timing-Bewertung von Vernetzungsarchitekturen und Gateway-Systemen im Kraftfahrzeug*. PhD thesis, Karlsruhe Institut für Technologie (KIT), 2010.

[121] DEVESH B. CHOKSHI AND PURANDAR BHADURI. **Modeling Fixed Priority Non-Preemptive Scheduling with Real-Time Calculus**. In *2008 14th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 387–392, Kaohsiung, China, August 2008. IEEE.

[122] SHANE TUOHY, MARTIN GLAVIN, EDWARD JONES, MOHAN TRIVEDI, AND LIAM KILMARTIN. **Next Generation Wired Intra-Vehicle Networks, A Review**. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 777–782, Gold Coast City, Australia, June 2013. IEEE.

[123] MICHAEL D. JOHAS TEENER, ANDRE N. FREDETTE, CHRISTIAN BOIGER, PHILIPPE KLEIN, CRAIG GUNTHER, DAVID OLSEN, AND KEVIN STANTON. **Heterogeneous Networks for Audio and Video: Using IEEE 802.1 Audio Video Bridging**. *Proceedings of the IEEE*, **101**(11):2339–2354, November 2013.

[124] RENE QUECK. **Analysis of Ethernet AVB for Automotive Networks using Network Calculus**. In *2012 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 61–67, Istanbul, Turkey, July 2012. Ieee.

[125] FELIX REIMANN, SEBASTIAN GRAF, FABIAN STREIT, MICHAEL GLAS, AND JURGEN TEICH. **Timing Analysis of Ethernet AVB-based Automotive E/E Architectures**. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–8, Cagliari, Italy, September 2013. IEEE.

[126] ANDREI HAGIESCU, UNMESH D. BORDOLOI, SAMARJIT CHAKRABORTY, PRAHLADAVARADAN SAMPATH, P. VIGNESH V. GANESAN, AND S. RAMESH. **Performance Analysis of FlexRay-based ECU Networks**. In *44th ACM/IEEE Design Automation Conference (DAC)*, pages 284–289, San Diego, CA, USA, June 2007. IEEE.

[127] DEVESH B. CHOKSHI AND PURANDAR BHADURI. **Performance analysis of FlexRay-based systems using Real-Time Calculus, Revisited**. In *Proceedings of the 2010 ACM Symposium on Applied Computing - SAC '10*, page 351, Sierre, Switzerland, March 2010. ACM Press.

[128] ROBERT TARJAN. **Depth-first search and linear graph algorithms**. *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, **1**(2):146–160, 1971.

[129] A. B. KAHN. **Topological Sorting of Large Networks**. *Communications of the ACM*, **5**(11):558–562, 1962.

[130] ERNESTO WANDELER AND LOTHAR THIELE. **Interface-Based Design of Real-Time Systems with Hierarchical Scheduling**. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 243–252, San Jose, CA, USA, April 2006. IEEE.

[131] ANNE BOUILLARD AND ÉRIC THIERRY. **An Algorithmic Toolbox for Network Calculus**. *Discrete Event Dynamic Systems*, **18**(1):3–49, October 2007.

[132] ANNE BOUILLARD, BERTRAND COTTENCEAU, BRUNO GAUJAL, LAURENT HARDOUIN, SÉBASTIEN LAGRANGE, AND MEHDI LHOMMEAU. **COINC Library: a toolbox for the Network Calculus**. In *Proceedings of the 4th International ICST Conference on Performance Evaluation Methodologies and Tools*, pages 1–3, Pisa, Italy, October 2009. ICST.

[133] CHENG-SHANG CHANG. **Performance Guarantees in Communication Networks**. *European Transactions on Telecommunications*, **12**(4):357–358, July 2001.

[134] JOHN HERSHBERGER. **Finding the upper envelope of n line segments in O(n log n) time**. *Information Processing Letters*, **33**(4):169–174, December 1989.

[135] WEI CHEN AND KOICHI WADA. **On Computing the Upper Envelope of Segments in Parallel**. *IEEE Transactions on Parallel and Distributed Systems*, **13**(1):5–13, 2002.

[136] LOTHAR THIELE AND NIKOLAY STOIMENOV. **Modular Performance Analysis of Cyclic Dataflow Graphs**. In *Proceedings of the 9th ACM international conference on Embedded software - EMSOFT '09*, page 127, Grenoble, France, October 2009. ACM Press.

[137] NIKOLAY N. STOIMENOV. *Compositional Design and Analysis of Distributed, Cyclic, and Adaptive Embedded Real-Time Systems*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH), 2011.

[138] KAI HUANG, WOLFGANG HAID, IULIANA BACIVAROV, MATTHIAS KELLER, AND LOTHAR THIELE. **Embedding Formal Performance Analysis into the Design Cycle of MPSoCs for Real-time Streaming Applications**. *ACM Transactions on Embedded Computing Systems*, **11**(1):1–23, March 2012.

[139] HENRIK SCHIOLER, JAN J. JESSEN, JENS D. NIELSEN, AND KIM G. LARSEN. **Network Calculus for Real Time Analysis of Embedded Systems with Cyclic Task Dependencies**. In *20th International Conference on Computers and Their Applications, CATA 2005*, New Orleans, Louisiana, USA, 2005.

[140] LICONG ZHANG, REINHARD SCHNEIDER, ALEJANDRO MASRUR, MARTIN BECKER, MARTIN GEIER, AND SAMARJIT CHAKRABORTY. **Timing Challenges in Automotive Software Architectures**. In *Companion Proceedings of the 36th International Conference on Software Engineering - ICSE Companion 2014*, pages 606–607, Hyderabad, India, May 2014. ACM Press.

[141] LINH T. X. PHAN, SAMARJIT CHAKRABORTY, P. S. THIAGARAJAN, AND LOTHAR THIELE. **Composing Functional and State-Based Performance Models for Analyzing Heterogeneous Real-Time Systems**. In *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pages 343–352, Tucson, Arizona, USA, December 2007. IEEE.

[142] KARINE ALTISEN AND MATTHIEU MOY. **ac2lus: Bringing SMT-Solving and Abstract Interpretation Techniques to Real-Time Calculus through the Synchronous Language Lustre**. In *2010 22nd Euromicro Conference on Real-Time Systems*, pages 207–216, Brussels, Belgium, July 2010. IEEE.

[143] NAN GUAN AND WANG YI. **Finitary Real-Time Calculus: Efficient Performance Analysis of Distributed Embedded Systems**. In *2013 IEEE 34th Real-Time Systems Symposium*, pages 330–339, Vancouver, Canada, December 2013. IEEE.

[144] URBAN SUPPIGER, SIMON PERATHONER, KAI LAMPKA, AND LOTHAR THIELE. **A simple approximation method for reducing the complexity of Modular Performance Analysis**. Technical report, Computer Engineering and Networks Laboratory of the Swiss Federal Institute of Technology, Zurich (ETH), 2010.

[145] SIMON KÜNZLI. *Efficient Design Space Exploration for Embedded Systems*. Phd thesis, Swiss Federal Institute of Technology Zurich (ETH), 2006.

[146] NADINE KEDDIS, GERD KAINZ, CHRISTIAN BUCKL, AND ALOIS KNOLL. **Towards Adaptable Manufacturing Systems**. In *2013 IEEE International Conference on Industrial Technology (ICIT)*, pages 1410–1415, Cape Town, Western Cape, South Africa, February 2013. IEEE.

[147] YORAM KOREN. *The Global Manufacturing Revolution*. John Wiley & Sons, Inc., Hoboken, NJ, USA, May 2010.

[148] NADINE KEDDIS, JONATHAN BURDALO, GERD KAINZ, AND ALOIS ZOITL. **Increasing the Adaptability of Manufacturing Systems by using Data-centric Communication**. In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2014)*, Barcelona, Spain, September 2014.

[149] YANJA DAJSUREN, MARK VAN DEN BRAND, ALEXANDER SEREBRENIK, AND RUDOLF HUISMAN. **Automotive ADLs: A Study on Enforcing Consistency Through Multiple Architectural Levels**. In *Proceedings of the 8th international ACM SIGSOFT conference on Quality of Software Architectures - QoSA '12*, page 71, Bertinoro, Italy, June 2012. ACM Press.

[150] HUANG BO, DONG HUI, WANG DAFANG, AND ZHAO GUIFAN. **Basic Concepts on AUTOSAR Development**. In *2010 International Conference on Intelligent Computation Technology and Automation*, pages 871–873, Changsha, China, May 2010. IEEE.

[151] MICHAEL SEIBT. **Architekturmodellierung mit EAST-ADL2 und AUTOSAR**. *AUTOMOTIVE - Spezial Baden-Baden 2009*, pages 38–41, 2009.

[152] PETER H. FEILER, DAVID P. GLUCH, AND JOHN J. HUDAK. **The Architecture Analysis & Design Language (AADL): An Introduction**. Technical Report CMU/SEI-2006-TN-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2006.

[153] ANDREAS JOHNSEN AND KRISTINA LUNDQVIST. **Developing Dependable Software-Intensive Systems: AADL vs. EAST-ADL**. In *16th Ada-Europe International Conference on Reliable Software Technologies*, **6652 LNCS**, pages 103–117, Edinburgh, UK, 2011. Springer Berlin Heidelberg.

[154] FRÉDÉRIC MALLET AND ROBERT DE SIMONE. **MARTE vs. AADL for Discrete-Event and Discrete-Time Domains**. In PROF. DR. MARTIN RADETZKI, editor, *Languages for Embedded Systems and their Applications*, **36**, pages 27–41. Springer Netherlands, 2009.

[155] HUASCAR ESPINOZA, SÉBASTIEN GÉRARD, HENRIK LÖNN, AND RAMIN TAVAKOLI KOLAGARI. **Harmonizing MARTE, EAST-ADL2, and AUTOSAR to Improve the Modelling of Automotive Systems**. In *Standards workshop (Satellite workshop of the 21st Euromicro Conference on Real-Time Systems)*, Dublin, Ireland, June 2009.

[156] ANDREAS WEINMANN. **Automobile Systeme in der Automatisierung - AUTOSAR und Echtzeit**, 2013.

[157] TIMMO-2-USE. **Methodology description V2 Version 1.0**, 2012.

[158] LOÏC BESNARD, THIERRY GAUTIER, PAUL LE GUERNIC, AND JEAN-PIERRE TALPIN. **Whitepaper: Towards a Synchronous Timing Annex for AADL**, 2013.

[159] CHRISTIAN BUCKL, IRINA GAPONOVA, MICHAEL GEISINGER, ALOIS KNOLL, AND EDWARD A. LEE. **Model-Based Specification of Timing Requirements**. In *Proceedings of the Tenth ACM International Conference on Embedded Software - EMSOFT '10*, page 239, Scottsdale, Arizona, October 2010. ACM Press.

[160] KAI HUANG, LUCA SANTINELLI, JIAN-JIA CHEN, LOTHAR THIELE, AND GIORGIO C. BUTTAZZO. **Periodic Power Management Schemes for Real-Time Event Streams**. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 6224–6231, Shanghai, China, December 2009. IEEE.

[161] PRATYUSH KUMAR. *Hard Real-Time Guarantees in Cyber-Physical Systems*. PhD thesis, Swiss Federal Institute of Technology Zurich (ETH), 2014.

[162] ANNE BOUILLARD, LINH T.X. PHAN, AND SAMARJIT CHAKRABORTY. **Lightweight Modeling of Complex State Dependencies in Stream Processing Systems**. In *2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 195–204, San Francisco, CA, United States, April 2009. IEEE.

[163] KAI HUANG, LOTHAR THIELE, TODOR STEFANOV, AND ED DEPRETTERE. **Performance Analysis of Multimedia Applications using Correlated Streams**. In *2007 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6, Nice Acropolis, France, April 2007. IEEE.

[164] SIMON PERATHONER, TOBIAS REIN, LOTHAR THIELE, KAI LAMPKA, AND JONAS ROX. **Modeling Structured Event Streams in System Level Performance Analysis**. *ACM SIGPLAN Notices*, **45**(4):37, April 2010.

[165] LOTHAR THIELE KÜNZLI, SAMARJIT CHAKRABORTY, MATTHIAS GRIES, AND SIMON. **Design Space Exploration of Network Processor Architectures**. *Network Processor Design: Issues and Practices*, **1**:30–51, 2002.

[166] SAMARJIT CHAKRABORTY. *System-Level Timing Analysis and Scheduling for Embedded Packet Processors*. Phd thesis, Swiss Federal Institute of Technology Zurich (ETH), 2003.

[167] KARSTEN ALBERS, FRANK BODMANN, AND FRANK SLOMKA. **Advanced Hierarchical Event-Stream Model**. In *2008 20th Euromicro Conference on Real-Time Systems*, pages 211–220, Prague, Czech Republic, July 2008. IEEE.

[168] KARSTEN ALBERS. *Approximative Real-Time Analysis*. Phd thesis, University of Ulm, 2008.

[169] DAVID HUTCHISON AND JOHN C MITCHELL. **Model-Based Engineering of Embedded Real-Time Systems**. *International Dagstuhl Workshop, Dagstuhl Castle, Germany, November 4-9, 2007. Revised Selected Papers*, **6100**, 2011.

[170] MAGNUS PERSSON. *Adaptive Middleware for Self-Configurable Embedded Real-Time Systems*. PhD thesis, KTH Schhol of Industrial Technology and Management, 2009.

[171] LEI FENG, DEJIU CHEN, AND MARTIN TÖRNGREN. **Self Configuration of Dependent Tasks for Dynamically Reconfigurable Automotive Embedded Systems**. In *47th IEEE Conference on Decision and Control*, pages 3737–3742, Cancun, Mexico, December 2008. Ieee.

[172] GONZÁLEZ H. MICHAEL, DANIEL SANGORRÍN, AND MIGUEL T. DE ESTEBAN. **FRESCOR Deliverable - Schedulability analysis techniques for distributed systems**. Technical report, Universidad de Cantabria, 2005.

[173] DREAMS CONSORTIUM. **Architectural Conceptualization - Deliverable D1.1.1, Distributed Real-time Architecture for Mixed Criticality Systems (DREAMS)**, 2015.

[174] DREAMS CONSORTIUM. **Architectural Style - Deliverable D1.2.1, Distributed Real-time Architecture for Mixed Criticality Systems (DREAMS)**, 2014.

[175] IBM, FRAUNHOFER FOKUS, THALES, AND TATA CONSULTANCY SERVICES. **Common Variability Language (CVL) OMG Revised Submission**, 2012.

[176] SAOUSSEN ANSSI, KARSTEN ALBERS, MATTHIAS DÖRFEL, AND SÉBASTIEN GÉRARD. **chronVAL/chronSIM: A Tool Suite for Timing Verification of Automotive Applications**. In *2012 Embedded Real Time Software and Systems Congress (ERTS)*, Toulouse, France, February 2012.

[177] KARSTEN ALBERS, STEFFEN KOLLMANN, FRANK BODMANN, AND FRANK SLOMKA. **Advanced Hierarchical Event-Stream Model and the Real-Time Calculus**. Technical report, Embedded Systems / Real-Time Systems, University of Ulm, Ulm, 2008.

[178] HENRIK SCHIOLER, HANS P. SCHWEFEL, AND MARTIN B. HANSEN. **CyNC: A MATLAB/SimuLink Toolbox for Network Calculus**. In *Proceedings of the 2nd International ICST Conference on Performance Evaluation Methodologies and Tools*, page 60, Nantes, France, October 2007. ICST.

[179] JENS B. SCHMITT AND FRANK A. ZDARSKY. **The DISCO Network Calculator - A Toolbox forWorst Case Analysis**. In *Proceedings of the 1st international conference on Performance evaluation methodolgies and tools - valuetools '06*, page 8, Pisa, Italy, October 2006. ACM Press.

[180] MARC BOYER. **NC-Maude: A Rewriting Tool to Play with Network Calculus**. *Lecture Notes in Computer Science*, **6415 LNCS**(PART 1):137–151, 2010.

[181] LUCA BISTI, LUCIANO LENZINI, ENZO MINGOZZI, AND GIOVANNI STEA. **DEBORAH: A Tool for Worst-Case Analysis of FIFO Tandems**. In *Lecture Notes in Computer Science*, pages 152–168. Springer Berlin Heidelberg, 2010.

[182] MARC BOYER, NICOLAS NAVET, XAVIER OLIVE, AND ERIC THIERRY. **The PEGASE Project: Precise and Scalable Temporal Analysis for Aerospace Communication Systems with Network Calculus**. In TIZIANA MARGARIA AND BERNHARD STEFFEN, editors, *4th International Symposium on Leveraging Applications, ISoLA 2010*, **6415** of *Lecture Notes in Computer Science*, pages 122–136. Springer Berlin Heidelberg, Berlin, Heidelberg, October 2010.

[183] LINH T. X. PHAN, JAEWOO LEE, ARVIND EASWARAN, VINAY RAMASWAMY, SANJIAN CHEN, INSUP LEE, AND OLEG SOKOLSKY. **CARTS: A Tool for Compositional Analysis of Real-Time Systems**. *ACM SIGBED Review*, **8**(1):62–63, March 2011.

[184] STEFFEN BONDORF AND JENS SCHMITT. **The DiscoDNC v2 - A Comprehensive Tool for Deterministic Network Calculus**. In *Proceedings of the 8th International Conference on Performance Evaluation Methodologies and Tools*, Bratislava, Slovakia, December 2014. ICST.

[185] MAREK JERSÁK. *Compositional Performance Analysis for Complex Embedded Applications*. Phd thesis, Technical University of Braunschweig, 2005.

[186] INSIK SHIN AND INSUP LEE. **Compositional Real-Time Scheduling Framework with Periodic Model**. *ACM Transactions on Embedded Computing Systems*, **7**(3):1–39, April 2008.

[187] LOTHAR THIELE, SAMARJIT CHAKRABORTY, MATTHIAS GRIES, ALEXANDER MAXIAGUINE, AND JONAS GREUTERT. **Embedded Software in Network Processors - Models and Algorithms**. In *First International Workshop on Embedded Software, EMSOFT 2001*, pages 416–434, Tahoe City, CA, USA, October 2001. Springer Berlin Heidelberg.