Technische Universität München
TUM School of Computation, Information and Technology

A Framework to Design and Synthesize Vehicle E/E Architecture

Hadi Askaripoor, M.Sc.

Vollständiger Abdruck der von der TUM School of Computation, Information and Technology der Technischen Universität München zur Erlangung eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing Jörg Ott

Prüfer der Dissertation:

1. Prof. Dr.-Ing. habil. Alois C. Knoll
2. Prof. Dr. Ali Mosleh (University of California, Los Angeles)

Die Dissertation wurde am 14. Dezember 2023 bei der Technischen Universität München eingereicht und durch die TUM School of Computation, Information and Technology am 11. September 2024 angenommen.

Abstract

In recent years, the field of automotive electrical and electronic (E/E) architecture has undergone substantial evolution. The latest generation of road vehicles necessitates a substantial infusion of computational power to support the execution of a multitude of safety-critical applications and advanced driver assistance systems (ADAS) functionalities. Centralized architecture, enhanced by the incorporation of high-performance computing units, emerges as a pivotal approach to reinforce the capability of vehicles in handling these resource-intensive applications. The high number of sensors and actuators demands high-bandwidth communication protocols to facilitate a seamless data flow. In addition, to harmonize the integration of safety-critical and real-time applications, known as mixed-criticality systems, deterministic and redundancy protocols are necessary. However, configuring and integrating essential applications into a vehicle's E/E architecture, all while meeting various requirements, guaranteeing reliable communication, and considering optimization objectives, can be time-consuming, complex, and error-prone tasks.

This thesis presents a novel model-based framework to facilitate the synthesis of car E/E architectures, which supports modeling for automotive embedded systems. The introduced tool automates mapping of software components to hardware elements and computes schedules for application threads. It establishes network message routing and schedules communication tasks within the car's topology while addressing safety requirements, including redundancy, homogeneous redundancy, and reliability. The proposed computer-aided tool also optimizes the system model, covering multiple optimization objectives. It supports multi-objective optimization and utilizes a single-step approach to solve mixed-integer programming (MIP) constraints, reducing solving time and considering the relationships among various constraints. Moreover, the proposed tool offers a web-based frontend that allows users to model their desired E/E systems and select various hardware and software requirements and properties, along with the boundary and optimization goals. The developed frontend also visualizes the solution of the designed system after it has been solved.

There are situations where a designed E/E architecture is not satisfiable, meaning that feasible solutions cannot be found by a MIP solver. Unlike simple models, navigating and correcting the unsatisfiability of complex E/E models is a complex and time-consuming task, leading to increased development costs. To tackle this issue, this thesis also introduces an approach to identify design errors when violations occur in a constraint set included in a system model after the solving step.

The performance of this model-based framework is assessed at three key stages. Design-time, where the solving and generation times of constraint sets in various scenarios are evaluated, including scalability analysis. Run-time, where the solution is deployed on an experimental setup, and finally, quantitative and qualitative evaluations. The results of the design-time experiments indicate that the formulations can scale to systems of reasonable size. During the run-time experiments, it is observed that there are no instances of timing deadline breaches following the deployment of the design-time solutions on an experimental setup.

Zusammenfassung

In den letzten Jahren hat sich der Bereich der elektrischen und elektronischen (E/E-) Architektur von Kraftfahrzeugen erheblich weiterentwickelt. Die neueste Generation von Straßenfahrzeugen erfordert eine erhebliche Steigerung der Rechenleistung, um die Ausführung einer Vielzahl von sicherheitskritischen Anwendungen und fortschrittlichen Fahrerassistenzsystemen (ADAS) zu unterstützen. Eine zentralisierte Architektur, die durch den Einbau von Hochleistungsrecheneinheiten verbessert wird, erweist sich als entscheidender Ansatz, um die Fähigkeit von Fahrzeugen zur Handhabung dieser ressourcenintensiven Anwendungen zu verbessern. Darüber hinaus erfordert die zunehmende Anzahl von Sensoren und Aktoren Kommunikationsprotokolle mit hoher Bandbreite, um einen nahtlosen Datenfluss zu ermöglichen. Zur Harmonisierung der Integration von sicherheitskritischen und Echtzeitanwendungen, die als Systeme mit gemischter Kritikalität bezeichnet werden, sind außerdem deterministische und redundante Protokolle unerlässlich. Die Konfiguration und Integration wichtiger Anwendungen in die E/E-Architektur eines Fahrzeugs unter Berücksichtigung der verschiedenen Anforderungen, der Gewährleistung einer zuverlässigen Kommunikation und der Optimierungsziele kann jedoch zeitaufwändig, komplex und fehleranfällig sein.

In dieser Arbeit wird ein neuartiges modellbasiertes Framework zur Erleichterung der Synthese der E/E-Architektur eines Fahrzeugs vorgestellt, das die Modellierung für eingebettete Systeme im Automobil unterstützt. Das vorgestellte Werkzeug automatisiert die Zuordnung von Softwarekomponenten zu Hardwareelementen und berechnet Zeitpläne für Anwendungsthreads. Es legt das Routing von Netzwerknachrichten fest und plant Kommunikationsaufgaben innerhalb der Fahrzeugtopologie unter Berücksichtigung von Sicherheitsanforderungen wie Redundanz, homogener Redundanz und Zuverlässigkeit. Das vorgeschlagene computergestützte Tool optimiert auch das Systemmodell und deckt mehrere Optimierungsziele ab. Es unterstützt die Mehrzieloptimierung und verwendet einen Ein-Schritt-Ansatz zur Lösung von MIP-Beschränkungen (Mixed-Integer Programming), wodurch die Lösungszeit reduziert und die Beziehungen zwischen den verschiedenen Beschränkungen berücksichtigt werden. Darüber hinaus bietet das vorgeschlagene Tool ein webbasiertes Frontend, mit dem Benutzer ihre gewünschten E/E-Systeme modellieren und verschiedene Hardware- und Softwareanforderungen und -eigenschaften sowie die Randbedingungen und Optimierungsziele auswählen können. Das entwickelte Frontend visualisiert auch die Lösung des entworfenen Systems, nachdem es gelöst wurde. Es gibt Situationen, in denen eine entworfene E/E-Architektur nicht zufriedenstellend ist, was bedeutet, dass der MIP-Löser keine machbaren Lösungen finden kann. Im Gegensatz zu einfachen Modellen ist das Navigieren und Korrigieren der Unerfüllbarkeit komplexer E/E-Modelle eine komplexe und zeitaufwändige Aufgabe, die zu erhöhten Entwicklungskosten führt. Um dieses Problem anzugehen, wird in dieser Arbeit auch ein Ansatz zur Identifizierung von Entwurfsfehlern vorgestellt, wenn nach dem Lösungsschritt Verletzungen in der im Systemmodell enthaltenen Constraintmenge auftreten.

Die Leistung dieses modellbasierten Rahmens wird in drei Schlüsselphasen bewertet: zur Entwurfszeit, in der die Lösungs- und Generierungszeiten von Constraint-Sets in verschiedenen Szenarien bewertet werden, einschließlich einer Skalierbarkeitsanalyse; zur Laufzeit,

in der die Lösung in einem Versuchsaufbau eingesetzt wird, sowie durch quantitative und qualitative Bewertungen. Die Ergebnisse der Experimente zur Entwurfszeit zeigen, dass die Formulierungen auf Systeme angemessener Größe skaliert werden können. Bei den Laufzeitexperimenten wurde festgestellt, dass nach dem Einsatz der Lösungen aus der Entwurfszeit in einem Versuchsaufbau keine Verstöße gegen die Zeitvorgaben auftraten.

Acknowledgement

I extend my deepest gratitude to all those who have contributed to the realization of this Ph.D. thesis. First and foremost, I express my sincere appreciation to my supervisor, Professor Alois Knoll, for his unwavering support, mentorship, and invaluable guidance throughout this research journey and for giving me the chance to pursue my Ph.D. at his research group. His expertise and commitment have been instrumental in shaping the trajectory of my academic pursuits. I would also like to express my heartfelt thanks to Professor Ali Mosleh, my second examiner from University of California, Los Angeles (UCLA), for his valuable feedback and insightful comments on my research. His expertise has been instrumental in refining the quality of my work. Additionally, my sincere appreciation goes to Professor Jörg Ott, the chairman of my defense, for overseeing the defense process and for his time and support in ensuring everything runs smoothly.

A special acknowledgment is reserved for Dr. Morteza Hashemi Farzaneh as my mentor. His guidance and insightful hints provided in the early stages were pivotal in helping me find my research direction. I wish to express my gratitude to my colleagues at the Chair of Robotics, Artificial Intelligence, and Real-time Systems. In particular, I extend my special thanks to Amy Buecherl for her consistent help and kindness. A heartfelt thank you goes to Ute Lomp, the former member of the chair, for her unwavering support and care. Welcoming Janine Delle, a new member of the chair, deserves my appreciation for her kindness and contributions to the dissertation process. I also want to thank Marie-Luise Neitz for her valuable assistance. Dr. Alex Lenz merits special thanks for his unwavering support, help, and kindness throughout this journey. My sincere gratitude extends to Thilo Mueller, my student, for his contributions. I am indebted to my colleagues and friends, including Hossein Malmir, Walter Zimmer, and Soubarna Banik, for their proofreading of this thesis. A profound thank you goes to my best friend, Andreas Wieser, for his help and for being an exceptional companion and friend during this journey.

In conclusion, I express deep gratitude to my family, including my sister and brothers, for their continuous support, motivation, and assistance. To my parents, Fatemeh and Yousef, my heartfelt and special thanks for their unwavering love, motivation, and constant support throughout this challenging journey. Despite the long distance, my parents and family send their love and support. I am immensely grateful and love you all dearly. This work is dedicated to them.

Los Angeles, 14.12.2023 *Hadi Askaripoor*

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

1.1 Vehicle E/E Architecture and its Development

The automotive industry is profoundly transforming, driven by technological advancements and a growing emphasis on safety, connectivity, and automation. Central to these changes is the concept of electrical and/or electronic (E/E) architecture, which refers to the organizational structure and integration of electrical and electronic systems within a vehicle. The development of E/E systems, including the technical approaches, requirements, design structure decisions, and methods, are heavily influenced by the vehicle's E/E architecture [AFK21a; AMK23].

The E/E architecture can be viewed from multiple perspectives. One is the physical aspect that shows the positioning and connections of elements in the car, such as electronic control units (ECUs), sensors, actuators, gateways, power supply, and switches. It also includes the placement of communication networks, wiring harnesses, and power distribution setups. Furthermore, the software used in the vehicle plays a crucial role in the E/E architecture, especially as the level of autonomy in cars has been advancing in recent years. Another perspective is that the E/E architecture can be seen as a logical representation, emphasizing the connection and interaction between the components and elements incorporated into the vehicle. In this context, the E/E architecture can be understood as pertaining to the transfer of data, the movement of signals, and the protocols for communication and interfaces [AHK22].

Recent advancements in driving automation have led to a significant increase in the number of companies investing in the development of autonomous vehicles. This effort has not only resulted in the inclusion of non-essential features, such as gaming using a vehicle's infotainment system, but also the implementation of advanced driver assistance systems (ADAS) to enhance safety and comfort for drivers. As a result, the E/E architecture of automobiles has undergone significant improvements, including the integration of various sensors, actuators, and powerful computing units to process the large amounts of data collected from the sensors for both critical and non-critical functions [AHK22].

Over the past few years, as the number of functionalities and features in vehicles has grown, the E/E architecture of cars has undergone significant changes [AFK20]. Initially, each ECU was responsible for managing a single function. As the number of functions increased, domain-specific ECUs emerged, responsible for controlling a set of functions related to a specific area or domain. These functional domains that require domain ECUs are usually compute-intensive and connect to many input/output (I/O) devices [AFK21b]. Each domain ECU is connected to multiple function-specific control units. A zonal ECU is responsible for distributing data and power throughout the vehicle, for example, by distributing the power

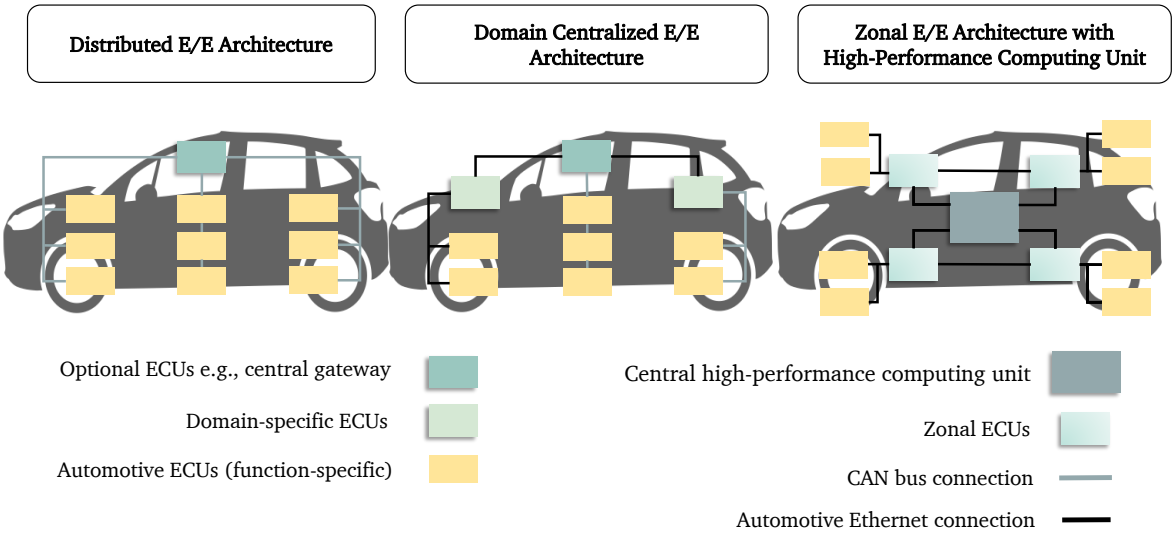


Figure 1.1: The figure presents the evolution of vehicle E/E architecture. Distributed E/E architectures were used until 2019, while domain-centralized architectures are used nowadays as vehicle architectures. The zonal architecture shows the future car E/E architecture [AHK22].

from the vehicle battery to various sensors and actuators using zonal controllers. Similarly, data is spread over the vehicle's network and transferred from sensors to other actuators/sensors and controllers using zonal ECUs. In addition, this type of ECU supports any feature available in the specific vehicle zone and acts as a gateway, switch, and smart junction box. Furthermore, it supports any interface for sensors, actuators, and displays [AMK23; AHK22].

Considering the ADAS and self-driving applications, using domain-specific ECUs increases the number of ECUs with substantial growth in the wiring harness, communication bandwidth, cost, software variants, and software complexity. Therefore, multi-core ECUs can be considered a solution to reduce the number of ECUs, cost, wiring harness, software complication, and variations. In addition, the multi-core technology has promptly been extending in different areas of embedded systems to deliver an appropriate performance for artificial intelligence (AI)-based applications and systems by giving scalable computing power [Bro06; AHK22]. A high-performance computing unit (HPCU) is a multi-core ECU that is composed of multiple system on chips (SoCs) that include several cores, a graphics processing unit (GPU), random access memory (RAM), and deep learning accelerators. These components work together to process high computational power-demand applications such as various object detection applications for ADAS functionalities [AFK21b]. The central computing unit is a fully scalable and upgradeable platform that connects to Edge and Cloud backend and uses cloud computing to process intensive vehicle functionalities. Additionally, it can act as the zonal gateway, meaning it serves as the master and core functionality of the vehicle [AHK22].

As shown in Figure 1.1, the development of E/E architectures has progressed through various stages. The initial stage referred to as distributed E/E architecture, involves using function-specific ECUs and a central gateway connected via a controller area network (CAN) bus. This architecture allows for stronger collaboration among ECUs and the ability to handle more complex functions, such as adaptive cruise control, as well as the potential for cross-functional connections. The next evolution in E/E architecture is known as domain centralized architecture, which utilizes domain-specific ECUs [AHK22]. As shown in Figure 1.1, function-specific control units are connected to domain-specific ECUs using both a CAN bus and an Ethernet connection. Moreover, this type of architecture also utilizes the central gateway ECU. This architecture is capable of handling even more complex functions and can also optimize cost through the consolidation of functions. For example, one domain-specific ECU

can be assigned for the parking assistance system, which includes two function-specific controllers related to vision processing and actuator commands, such as for brake and steering wheel [AHK22].

The domain centralized architecture, which incorporates domain controllers and a central gateway, has become increasingly sophisticated over time, encompassing the car's wiring harness. The implementation of autonomous driving features exacerbates the complexity of the architecture due to the increased number of sensors and actuators, the growth in data processing capabilities, and required bandwidth, as well as the high demand for intelligent power distribution [AMK23; AMK23; AFK21b]. To address the complexity present in previous architectures, the future of E/E architecture is envisioned as a zonal architecture, which will utilize a central HPCU [AMK23]. The zonal architecture is designed to include cutting-edge vehicle functions and technologies while reducing weight and cost. As presented in Figure 1.1, the zonal architecture consists of an HPCU, zonal ECUs, and function-specific ECUs. The central HPCU acts as the master, processing all data from various vehicle zones and controlling the car's operation. Furthermore, the HPCU serves as a central gateway, transmitting data between different zones [Jia19]. The ECUs and HPCU are interconnected using an Ethernet connection, which offers fast and high-bandwidth data transmission [AFK21a; AHK22]. Furthermore, the zonal architecture supports virtual domains, transferring embedded functions to the cloud and providing software updates and downloads via the over-the-air (OTA) service for the HPCU [AHK22].

Further details regarding automotive communication protocols such as CAN bus and automotive Ethernet will be explained in Chapter 2.

1.1.1 The Main Bottlenecks of Current E/E Architecture

Although current E/E architectures can meet a wide range of requirements, they may need to be fully equipped to handle the demands of self-driving or future vehicles. With the growing number of functionalities, applications, sensors, and actuators, as well as the need to adhere to functional safety standards, future ECUs will require a greater level of computational power, communication interfaces, and software integration and architecture compared to their current counterparts [AHK22]. Furthermore, the communication bandwidth required for autonomous vehicles presents a significant challenge in the current E/E architecture. Although current communication networks, such as CAN, have been used for in-vehicle communication in the past decade, their communication distance and rate cannot compare with Ethernet, which offers a wide range of communication protocols and allows for system interoperability, compatibility and efficient resource sharing [AHK22]. In addition, low latency, safe persistency, secure data transmission, and high network security are crucial factors for future vehicles. Furthermore, external communication, such as software updates or vehicle-to-vehicle (V2V) communication, requires higher data traffic and protection. To meet these requirements, communication protocols such as the automotive Ethernet will be necessary [AHK22; AMK23].

Integrating new technology into existing E/E architectures is currently a major challenge in developing autonomous vehicles. To address this issue, the E/E architecture for autonomous cars must be designed to allow for future extensions and the integration of new technologies without compromising the existing structure [AHK22]. To achieve this goal, powerful computing units, high-speed communication protocols that meet automotive safety regulations, and approaches to facilitate the integration of new features into the existing vehicle architecture are crucial. These features will help to overcome the current limitations of the E/E architecture [AFK21b].

1.1.2 The Main Technologies for Future's E/E Architecture

The new functionalities integrated into self-driving cars require advanced technologies to fulfill future automotive feature requirements. Critical technologies for automated driving cars include employing zonal ECUs, which also function as advanced gateways with increased computing power, utilizing central computing units as the vehicle's core, implementing novel in-vehicle communication networks, and establishing new vehicle software architectures [AMK23].

To meet the growing demand for in-vehicle network bandwidth, automotive Ethernet emerges as a viable solution, offering higher bandwidth, enhanced security, and improved compliance with safety requirements outlined in ISO 26262 [ISO18]. Moreover, the achievement of low-latency communication within the network, facilitated by novel message routing mechanisms, plays a pivotal role in advancing in-car networks, particularly in the context of autonomous vehicles [AHK22].

One of the most crucial technologies for transitioning to a centralized E/E architecture is the utilization of multi-core processors, including artificial intelligence (AI) accelerators, as the primary computing units. With the proliferation of AI applications, particularly ADAS applications that employ deep learning and machine learning algorithms demanding significant computational power, especially in the vision domain, there arises a need for incorporating a central HPCU within the vehicle's E/E architecture to efficiently process and compute these applications. Furthermore, the core software architecture for the entire vehicle is established within the HPCU. This architecture accommodates various software domains, encompassing perception, mapping, planning, ADAS applications, and infotainment. Additionally, employing such a centralized architecture, complemented by an advanced software-defined vehicle (SDV) architecture, opens up opportunities for original equipment manufacturers (OEMs) to seamlessly integrate and update advanced software, similar to the approach seen with smartphones [AHK22; AFK21b].

SDV is a type of vehicle that relies heavily on software to control its functions and features. This can include everything from the vehicle's powertrain and propulsion systems to its navigation, entertainment, and safety features. One of the main benefits of using software to define the functions of a vehicle is that it allows for greater flexibility and adaptability. With traditional vehicles, making changes or updates to the hardware can be challenging, as it requires physical modifications to the vehicle itself. With an SDV, however, many of the vehicle's functions can be controlled and updated through software, making adding new features or making changes to existing ones easier. In addition to providing greater flexibility, software-defined vehicles can be more efficient and reliable. Using software to control and optimize various functions can achieve better performance and reduce the risk of mechanical failures. There are also several potential applications for SDVs, including autonomous vehicles, electric vehicles, and connected vehicles that can communicate with each other and with infrastructure. An essential element in advancing software-defined vehicles involves decoupling software and hardware development. A valid comparison can be drawn from the evolution of cell phones. Initially, cellphone software and hardware were closely intertwined. However, the introduction of smartphones transformed phones into software platforms that can host various applications independently of the underlying hardware. Similarly, the automotive industry is undergoing a shift where vehicle software is becoming a platform [Apt23; QNX23].

As mentioned previously, the new software-defined architecture is integral to the future E/E architecture. Hardware virtualization technology should be considered to enable the integration of safety and non-safety-critical software domains into the HPCU in a manner that complies with safety requirements while optimizing hardware resource utilization. This

technology can be implemented using a hypervisor [AHK22; MAK22]. Further details about hypervisors will be provided in Chapter 2.

1.2 Motivation and Research Questions

1.2.1 Motivation

With the increasing level of vehicle automation, such as ADAS, the demand for computational power in vehicle ECUs has grown dramatically in recent years. Currently, cars are equipped with anywhere from 70 to 100 ECUs to manage their software systems [Pel+17]. Furthermore, the complexity and variety of required applications in today's vehicles have substantially increased, especially with the inclusion of ADAS and automated driving features. Meeting both non-safety and safety requirements in compliance with automotive standards, such as ISO 26262 [ISO18] and SOTIF [ISO19], during the design and configuration of automotive architectures has increased complexity. This complexity arises from integrating new applications and features into the vehicles and the limitations of traditional E/E architectures [AHK22; AMK23; AFK21b]. Consequently, the vehicle E/E architecture has been evolving recently concerning the complexity described in Section 1.1.

A significant amount of data must be transmitted over the in-vehicle communication network to support new infotainment and driver assistance features [AMK23]. However, reliable transmission is crucial when vehicles rely on network messages to make safety-critical decisions. Low latency and, in some cases, deterministic message transmission with accurate schedules for each communication frame are also necessary to meet the demands of real-time applications [AFK20; AMK23].

Therefore, developing an E/E architecture with ADAS functionalities and algorithms that meet all safety-related (e.g., timing, freedom from interference (FFI), and redundancy) and non-safety-related requirements is a laborious and time-consuming task that requires domain-specific knowledge [AFK21a; AFK20]. Manually integrating and configuring the software architecture for an automotive HPCU is challenging and prone to errors, given the need to fulfill various hardware, application, operating system (OS), middleware, and hypervisor requirements and properties. The same applies to configuring an automotive communication network, ensuring reliable data transmission for safety-critical ADAS applications. These configuration syntheses can be optimized for multiple goals, comprising power consumption, resource utilization, reliability, bandwidth usage, temperature, cost, response time, end-to-end latency, and more [AHK22; AFK20; AMK23].

Additionally, as the number of hardware and software components continues to grow within the vehicle E/E system, along with their corresponding requirements and properties, the task of finding the ideal configuration synthesis and solutions for specified problems, e.g., mapping problem, becomes increasingly complex for system architects. In addition, the need for a new update in the configuration may lead to unknown risks and becomes costly. For example, Figure 1.2 presents a brief overview of a system integrator's manual process of vehicle E/E architecture synthesis. The system integrator must consider the software model specifications that will be deployed on the car's E/E system, including applications and their requirements, OS and middleware, and virtualization technologies (e.g., hypervisors) [AHK22]. Moreover, as shown in Figure 1.2, the system architect must take into account the E/E system components and their properties, comprising HPCU, communication protocols, ECUs, sensors, and actuators. This holistic approach is necessary to generate a correct configuration synthesis that fulfills all predefined requirements and optimization objectives.

It should be noted that the problem of finding the configuration synthesis can become a

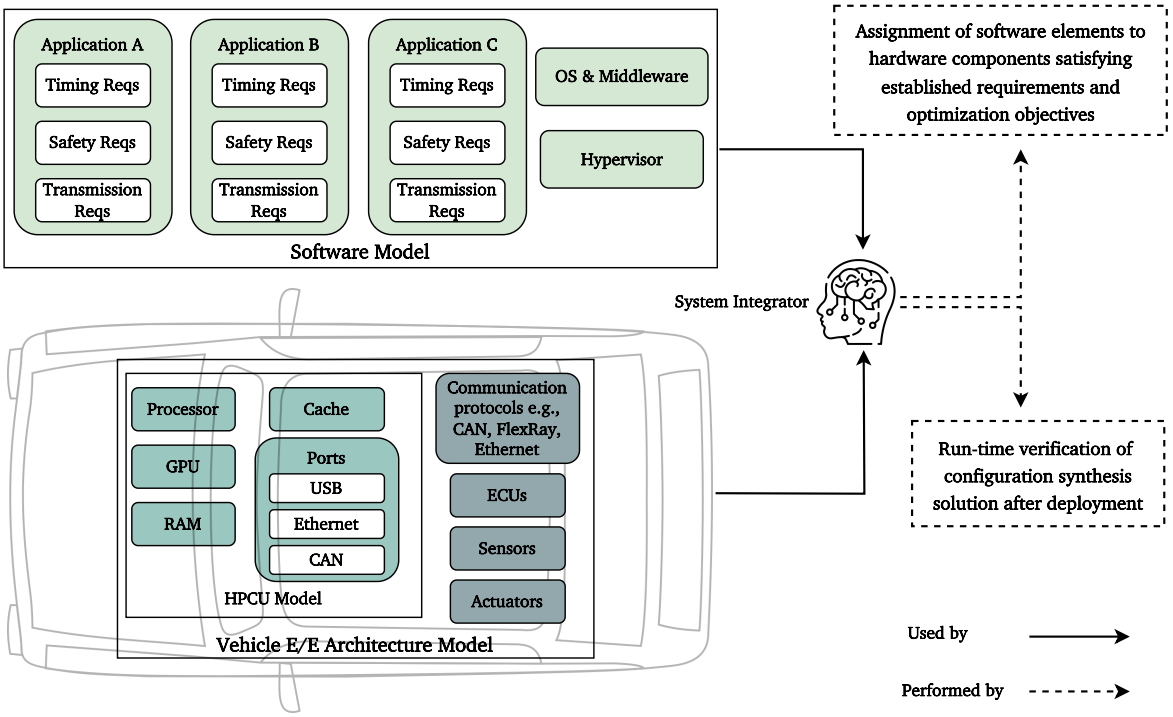


Figure 1.2: The manual procedures must be executed by a system integrator to create configuration syntheses for automotive software components and the vehicle E/E architecture model.

non-deterministic polynomial (NP) problem, depending on the problems [SC13; AHK22]). This is due to the vast number of safety-critical/non-critical application and user requirements, such as reliability [Xie+17b], timing, latency, resource utilization, bandwidth usage, memory usage, power consumption, FFI, ASIL level [AMK23], redundancy, the expansive design space for the specified problems, and the non-deterministic or dynamic workload.

1.2.2 Research Questions

As explained above, current and future E/E architectures encounter various challenges and bottlenecks, including configuration synthesis for software and hardware of the vehicles, as illustrated in Subsection 1.2.1. Making the synthesis of E/E architectures semi-automated or automated facilitates the whole synthesis process and reduces the design process's effort, possible design mistakes, and complexity. Also, it helps to avoid undesired functional safety violations. This can be achieved by using computer-assisted software tools. Following this motivation, it aims to answer two research questions related to the automated configuration synthesis of a car's E/E architecture.

- How to facilitate design and synthesis of E/E architectures?

  As mentioned in Subsection 1.2.1, the design and configuration synthesis of vehicle E/E systems are complex and time-consuming, demanding domain-specific knowledge, particularly when faced with increasing requirements, optimization objectives, and boundary constraints. Specific synthesis problems, such as time-triggered scheduling, message routing, and mapping or resource allocation, are also of critical importance. For example, finding feasible time-triggered schedules is an NP-complete problem.

  Tool-assisted approaches can be utilized to address these challenges in the design phase, simplifying the design and configuration synthesis of E/E systems. In the current state

of the art, a plethora of commercial and non-commercial tools are available for modeling and synthesizing E/E architectures and automotive embedded systems, as elaborated upon in Chapter 3. These tools can accommodate various hardware and software properties, safety and non-safety requirements, synthesis problems, and optimization objectives, thereby facilitating the design and synthesis of car E/E architectures. However, they have limitations regarding the diversity of synthesis problems, optimization goals, and safety and non-safety requirements.

The process involves gathering relevant information for synthesis problems and selected requirements and objectives, realizing them, transforming them into mathematical constraints, and solving them to find optimized solutions using solvers.

- How to simplify analysis of design errors in E/E architectures?

  Ensuring that there are always feasible solutions for the design of E/E systems is not guaranteed. E/E architects can model E/E architectures while selecting various specified requirements and optimization goals. However, after solving the designed E/E models, sometimes the solver concludes that no feasible solutions exist. This occurs due to conflicts within the constraints system. Identifying the source of the violation is necessary to correct the errors and make the system model feasible. This process becomes exceedingly time-consuming and complex when there are many constraints within the system model. Thus, an approach is needed to address this issue, enabling the identification of violated constraints more quickly to make the system model satisfiable.

1.3 Thesis Contributions

Addressing the first research question, automation is pivotal in simplifying the design and synthesis of car E/E architecture. To contribute to this question, we present a model-based software framework. While numerous tools are available for modeling and synthesizing E/E systems, they often come with limitations in areas such as synthesis problems, safety requirements, optimization goals, and more. These limitations are comprehensively analyzed in the literature review in Chapter 3.

Therefore, the introduced modeling tool offers new features including automatic mapping/resource allocation for multi-core computing units and software and hardware components, automatic creation of various types of communication message routings for automotive networks, and the computation of time-triggered schedules for application threads and communication tasks. These synthesis problems are all addressed in a single step to reduce solving time and account for the interrelations between defined constraints. The framework leverages advanced algorithms to achieve this goal. Furthermore, the presented tool encompasses a wide range of safety requirements, including FFI, ASIL considerations, reliability, redundancy, and homogeneous redundancy. It also caters to various optimization goals and boundary conditions, such as end-to-end latency, response time, resource utilization (including memory and processor), link occupation rate (LOR), bandwidth usage, and E/E components cost. Detailed explanations of these terms can be found in Chapters 2 and 4.

To develop this framework, we employ a model-driven development (MDD) approach, wherein an object-oriented metamodel serves as the foundation for all defined synthesis problems, requirements, optimization objectives, and boundary constraints. Linear programming is utilized to implement all the features discussed above in the form of constraints. The complexity of linear programming is hidden from system integrators and E/E system architects by

creating an object-oriented graphical modeling tool as the frontend of the framework. As a result, E/E architectures, comprising hardware and software components, can be easily modeled. The modeler offers various user-friendly features such as drag-and-drop functionality and automatic hardware/software components creation, simplifying the modeling process.

Once the modeling step is complete, all logical requirements and properties are collected to build the E/E system knowledge database. This database is then used to generate constraints related to the previously mentioned problems and goals. A solver subsequently solves and optimizes these constraints, resulting in an optimized configuration synthesis for the designed model. The methodology and approach used in the introduced framework are explained in Chapter 4. The solution is integrated into the frontend to be accessible and observable by the user. The details of the frontend are addressed in Chapter 5.

Various evaluation schemes are employed to evaluate the developed software tool, as described in Chapter 7. These include:

- Design-time evaluation: This assessment focuses on the tool's performance, applicability, and scalability during the design phase. It involves using diverse case studies to test the tool's capabilities.
- Run-time evaluation: In this phase, solutions created by the tool are deployed on a real hardware platform to assess their practical performance.
- Qualitative and quantitative evaluation: This approach examines the tool's performance, usability, and practicality. It involves modeling and synthesizing a series of use cases performed by multiple users, both manually and using the introduced tool.

To address the first research question, Chapters 4, 5, and 7 are introduced.

In response to the second research question, when a solver cannot find feasible solutions, these solutions become unsatisfiable or infeasible, which is common when solving a constraint system. This arises from the conflicts between constraints within the system model. Identifying the source of violation can be time-consuming and complex, especially when dealing with many constraints within the system model. To address this challenge, an approach is presented in Chapter 6 for identifying design errors or pinpointing conflicts between constraints after the solving process. This approach uses two distinct methods to generate a minimal set of unsatisfiable constraints and cores. The proposed approach then utilizes these methods to identify the most critical constraints responsible for model infeasibility. As part of this approach, a list of constraints is created, with each constraint assigned a weight proportional to the number of times it has been identified as the cause of model unsatisfiability. Consequently, using these weighted constraints, the system integrator can pinpoint the origins of conflicts between constraints. Furthermore, it is possible to make the model satisfiable again by addressing and correcting the source of the violation. In addition, in Chapter 6, evaluations of various scenarios for the design error analysis approach are discussed. Each method is assessed individually. However, one particular method is considered the primary approach in this presentation, primarily due to its compatibility with the solver used in the tool.

Most of this thesis's content has been published in various international journals and conferences. In addition, the proposed software framework has become open-source and accessible on GitHub [AMK23], offering the research community an opportunity to explore, utilize, and contribute to its development. It is believed that open sourcing the introduced tool is essential for fostering collaboration, reproducibility, and transparency in research. The most relevant publications are as follows.

- H. Askaripoor, T. Mueller and A. Knoll, "E/E Designer: a Framework to Design and

Synthesize Vehicle E/E Architecture," in IEEE Transactions on Intelligent Vehicles, doi: 10.1109/TIV.2023.3324617.

- Askaripoor, H., Hashemi Farzaneh, M. and Knoll, A., 2022. E/e architecture synthesis: Challenges and technologies. Electronics, 11(4), p.518.

- Askaripoor, H., Farzaneh, M.H. and Knoll, A., 2021, September. A model-based approach to facilitate design of homogeneous redundant e/e architectures. In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC) (pp. 3426-3431). IEEE.

- Askaripoor, H., Farzaneh, M.H. and Knoll, A., 2021, September. A platform to configure and monitor safety-critical applications for automotive central computers. In 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (pp. 1-4). IEEE.

- Askaripoor, H., Farzaneh, M.H. and Knoll, A., 2020, September. Considering safety requirements in design phase of future e/e architectures. In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (Vol. 1, pp. 1165-1168). IEEE.

- Askaripoor, H., Shafaei, S. and Knoll, A., 2021. A flexible scheduling architecture of resource distribution proposal for autonomous driving platforms. In Proceedings of the 7th International Conference on Vehicle Technology and Intelligent Transport Systems.

- T. Müller, H. Askaripoor and A. Knoll, "Advancing E/E Architecture Synthesis: A Perspective on Reliability Optimization and Hypervisor Integration," 2024 IEEE Intelligent Vehicles Symposium (IV), Jeju Island, Korea, Republic of, 2024, pp. 1996-2003, doi: 10.1109/IV55156.2024.10588416.

- Müller, T., Askaripoor, H. and Knoll, A., 2022, October. Performance analysis of KVM hypervisor using a self-driving developer kit. In IECON 2022–48th Annual Conference of the IEEE Industrial Electronics Society (pp. 1-7). IEEE.

1.4 Thesis Structure

The thesis is structured as follows.

- The fundamental concepts and terms are elaborated upon in Chapter 2, where a range of topics, including resource allocation, time-triggered scheduling, message routing in automotive networks, vehicle communication protocols, automotive safety standards, safety requirements relevant to this thesis, and virtualization technologies, are discussed.

- In Chapter 3, a comprehensive analysis of the state of the art is presented. This analysis primarily focuses on studies related to automotive software architecture synthesis, technologies, and frameworks/tools for modeling and synthesizing E/E systems, as well as software integration and configuration in the design process. Additionally, it encompasses research on mapping problem, communication message routing, and time-triggered scheduling. The final section of this chapter includes a discussion and summary.

- The approach of the proposed framework is explained in Chapter 4. This chapter includes various sections, such as framework architecture, framework system model, and mixed-integer programming (MIP) formulation of constraints. Moreover, it discusses boundary constraints, optimization objectives, multi-objective optimization, single-step solving algorithms, and constraint formulation for the Gurobi optimization solver and provides a discussion section [Gur22].
- The frontend of the proposed software tool is detailed in Chapter 5, which comprises sections on modeling, requirements and properties, solving and solutions, model validation, and implementation.
- In Chapter 6, the approach for identifying design errors when violations occur in the constraint set after it has been solved is explained. This chapter involves background, approach, and evaluation sections.
- The performance of the introduced software framework is assessed in Chapter 7, covering design-time, run-time, and both quantitative and qualitative evaluations.
- In Chapter 8, the thesis conclusion, its limitations, and potential avenues for future research are discussed.

2

Basic Concepts and Terms

In this chapter, the foundation for exploring the subject matter is laid by introducing the fundamental concepts and key terminology that will form the basis of this thesis. This section defines relevant terms and provides background information on task mapping, communication message routing, time-triggered scheduling, automotive safety standards, vehicle communication protocols, safety-related requirements in vehicle E/E architecture, design space exploration, and virtualization technology.

2.1 Task Mapping or Resource Allocation

Multi-core architecture can be classified into two categories, homogeneous and heterogeneous, based on the application's and user's specific needs and requirements. In the homogeneous multi-core architecture, all cores are the same, possessing similar computing capacity and instruction set architecture (ISA). On the other hand, heterogeneous architecture comprises a combination of different cores, each specifically designed to meet particular needs, such as high performance or low power consumption [GBI21]. Integrating multiple types of processing units onto a single chip through heterogeneous architecture results in lower energy consumption and increased flexibility [Sin+13]. This approach is beneficial for automotive applications that have varying contexts and requirements. However, it is essential to note that homogeneous processors require less task mapping complexity than heterogeneous processors, as they have identical cores. Furthermore, homogeneous multi-cores do not require an analysis of core properties for task execution, unlike heterogeneous multi-core architecture.

The allocation of tasks in a multi-core system can occur either during design-time or run-time. Design-time task allocation is performed when no application is running, in contrast to run-time mapping, which allows tasks to be assigned to different cores while the system is in operation. In cases where the application requirements are predictable and stable, design-time mapping is favored. However, the run-time assignment should be utilized when the application requirements are subject to change in dynamic scenarios, and there is a requirement for reassignment [AHK22; AMK23].

Design-time mapping is preferably utilized when the application requirements are fairly deterministic. On the other hand, in case of changes in the requirements in dynamic scenarios, the run-time assignment is preferred because reassignment is required. Static mapping uses all system information (e.g., hardware and application properties) to find the optimal solution. In addition, this type of mapping is appropriate when there is a set of predefined

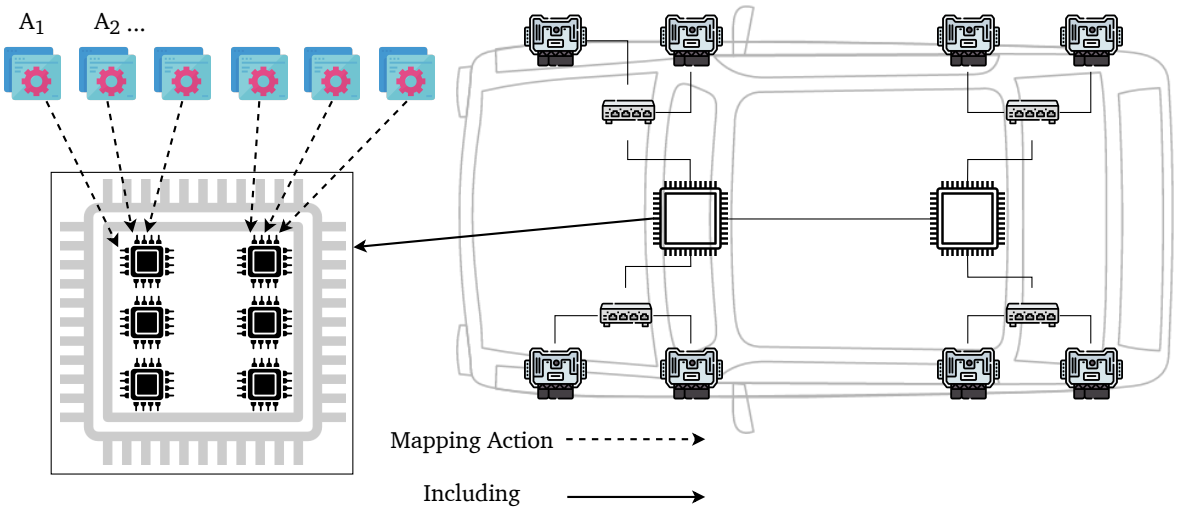


Figure 2.1: Assignment of applications to an HPCU consisting of six cores.

requirements for the applications and hardware. In other words, the design-time methodology cannot solve the mapping problem dynamically. Higher quality solutions can be obtained with the design-time mapping rather than the run-time allocation due to less limitation of the computational power [AHK22; AMK23]. Figure 2.1 shows the static mapping of several applications, each including threads with different periods and execution times, to an HPCU comprising six cores.

2.2 Time-triggered Scheduling

In mixed-critical systems, especially those focused on real-time applications, enabling deterministic message delivery (i.e., all computations must be complete before their respective deadlines), avoiding message overlap, and ensuring low latency are extremely important. Scheduling policies help to meet these requirements using various algorithms. Time-triggered scheduling is one of the most common scheduling schemes used in the automotive industry to schedule communication tasks and activities. In the automotive domain, time-triggered scheduling is used in various systems, including engine control, braking, steering, and infotainment. This scheduling scheme also controls safety systems, such as airbags and seatbelts. These systems must respond quickly and reliably to potential hazards, and this scheduling scheme can ensure that the tasks related to these systems are executed at the appropriate times. It is also used in other real-time and mixed-critical systems, such as aviation and industrial control, where predictability and reliability are critical [Zen+10; Luk+12; AMK23].

In this type of scheduling, activities/tasks that are periodic are initiated at predefined times (see Figure 2.2). A deterministic scheduling approach ensures that tasks in a system are executed at a predictable and repeatable rate. In other words, everything is scheduled before the system is deployed, and it is known in advance which activity will run and when [SW00]. Moreover, all tasks must not overlap during execution [Zha+14]. The running processes in ECUs and HPCUs can be scheduled using the time-triggered method. This policy can be applied to the in-vehicle communication network to schedule communication frames/tasks over a communication protocol, e.g., Ethernet. Time-triggered scheduling is often used with other scheduling algorithms, such as event-triggered scheduling, to provide a balance between predictability and flexibility [Zen+10]. In an event-triggered system, a processing task begins in response to the occurrence of a notable event [Tab07]. While in a time-triggered system, the

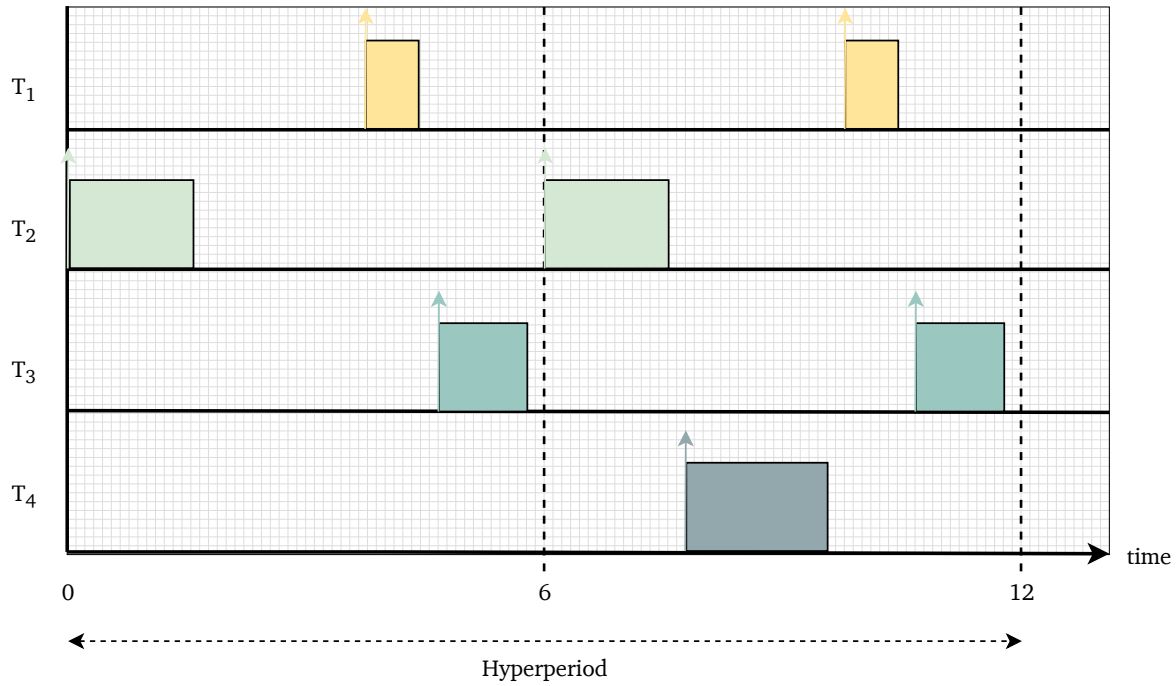


Figure 2.2: The time-triggered slots of four periodic tasks, namely T_1, T_2, T_3, and T_4. An arrow alongside each slot indicates the starting point of the respective task.

activities are initiated periodically at predetermined points in real-time.

To illustrate how time-triggered scheduling works, an example of scheduling four tasks is considered within an automotive real-time operating system (RTOS) running on an HPCU. Each task is specified as $T_i = \{T_i^p, T_i^e\}$, where T_i^p represents the task's period, and T_i^e denotes its execution time. The four periodic tasks are as follows: $T_1 = \{6, 0.5\}$, $T_2 = \{6, 1.5\}$, $T_3 = \{6, 0.8\}$, and $T_4 = \{12, 2\}$. In the context of time-triggered scheduling, the objective is to ensure that each task executes once within its specified period without overlapping with other tasks. As illustrated in Figure 2.2, each task completes within its defined period, e.g., T_4 finished its job once within its period, which is 12. The visualized schedules confirm no overlap between task execution slots; this demonstrates the correctness of the task schedules under time-triggered scheduling.

In Figure 2.2, hyperperiod refers to the least common multiple (LCM) of the periods of all periodic tasks or events in a system. It represents the time interval after which all periodic tasks will simultaneously repeat or hyperperiodically align. It helps to determine the maximum scheduling interval required to ensure that all tasks can be scheduled without deadline violations. When the scheduling interval is set to the hyperperiod, it guarantees that all tasks will complete their cycles within that interval, thus avoiding task deadline misses [AMK23]. In this specific example, the hyperperiod is calculated as 12, which is the LCM of the periods T_1, T_2, T_3, and T_4.

2.3 Communication Message Routing

Message routing plays a crucial role in automotive networks, where many ECUs collaborate to ensure smooth vehicle operations. As modern automobiles become increasingly sophisticated, incorporating various functionalities and advanced systems, efficient and reliable message routing becomes paramount.

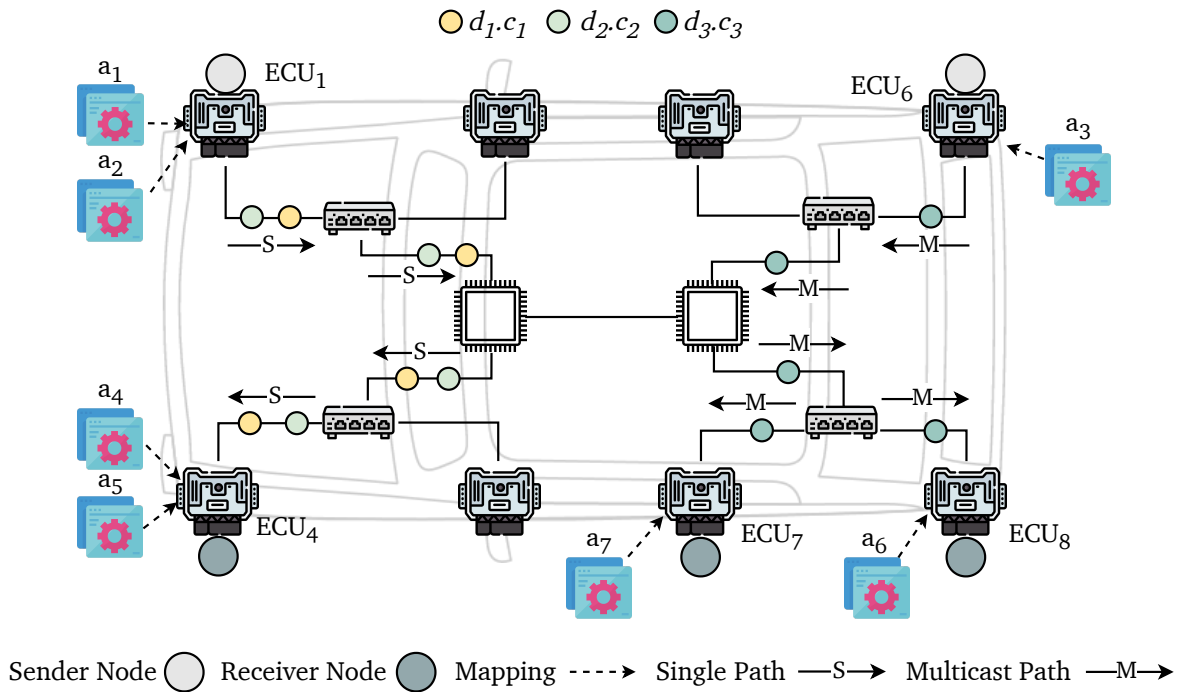


Figure 2.3: Single and multicast routes within a car E/E topology for transferring communication messages from senders to receivers. The communication messages are generated by applications executing on ECUs.

The ADAS capabilities and new infotainment systems introduce an increasing amount of computational power in control units and data, which needs to be transmitted via an in-car communication network. A valid routing is required in an in-vehicle network to transmit messages from a node (e.g., an ECU) as a sender to another node as a receiver. Due to increasing the size of automotive communication networks, which causes a large number of applications, nodes, and transmitted messages, finding a valid path between two nodes becomes extremely time-consuming and complex because of a significant expansion of the design space [AFK21a; Smi+17; AMK23; AFK21b].

One commonly used technique in message routing is the concept of gateways. Gateways act as intermediaries, connecting different network domains or protocols within the vehicle. They receive messages from one network and forward them to another, often performing protocol conversions or message translations. Gateways play a critical role in integrating legacy and newer ones, enabling seamless communication between nodes, e.g., ECUs operating on different protocols. The most essential vehicle communication protocols will be explained in the next section. Another essential aspect of message routing is the consideration of real-time constraints. Many automotive applications demand deterministic and timely message delivery to ensure safety and optimal system performance. Therefore, routing algorithms must prioritize critical messages and allocate network resources accordingly. This involves analyzing the network's traffic patterns, identifying potential bottlenecks, and selecting the most efficient paths to ensure timely message delivery and avoid congestions [AMK23; AFK21a; Smi+18].

The framework introduced in this thesis has the capability to create various types of communication message routings, encompassing single, multi-cast, redundant, and homogeneous redundant paths. Detailed descriptions of each type are provided in Chapter 4. Based on the car topology depicted in Figure 2.3, two types of communication paths, from senders to receivers, are presented. The single route involves the transfer of two communication messages, d_1 and d_2, along with their respective tasks, c_1 and c_2, from ECU_1 as the sender to ECU_4 as the receiver. The multicast route demonstrates the transmission of a communication

message, d_3, from a sender, ECU_6, to two receivers, namely ECU_7 and ECU_8.

2.4 Vehicle Communication Protocols

Vehicle communication protocols play a pivotal role in modern automotive systems, enabling seamless data exchange and coordination among various nodes, sensors, and actuators within a vehicle. The demand for robust and efficient communication solutions has become paramount as the automotive industry rapidly evolves. The following subsections provide concise explanations of the most important communication protocols.

2.4.1 CAN and TTCAN Buses

Controller area network (CAN) is the most common in-vehicle communication network. The CAN protocol, developed by Robert Bosch GmbH in the early 1980s, serves as a multi-master communication interface primarily designed for in-vehicle communication purposes. The CAN bus uses a twisted pair of wires, including CAN low and CAN high, as its physical layer and can deliver data transfer rates of up to one megabit per second (Mbps). One of its notable strengths is its exceptional resistance to electrical interference, simplifying installation due to its ease of wiring. Furthermore, CAN possesses self-diagnostic capabilities, allowing it to identify and rectify errors autonomously. The network's distributed architecture contributes to simplified maintenance procedures and reduces the overall system cost [BSJ18]. CAN bus is a robust, flexible, and low-cost communication system that has become a de facto standard for in-vehicle communication. It uses a message-based protocol, where each message is called a frame and consists of a header and a data field. The header contains information about the message, such as the sender's identifier and the data field's length, while the data field contains the actual data being transmitted [Bus23a].

CAN bus has several advantages over other communication protocols, such as low cost, its ability to operate at high speeds, and its high level of noise immunity. It is also designed to be simple and efficient, making it well-suited for use in the harsh environments of vehicles. However, this protocol has drawbacks, including limited bandwidth and message length (typically eight bytes per frame rate), lack of security [BSJ18], wiring complexity, and limited distance. In addition, CAN has a maximum node limit of 64 devices, and its operation can generate electrical noise due to varying voltage levels [Dis23].

TTCAN

The basic CAN protocol itself does not support time-triggered scheduling. However, there is an extension to the CAN protocol called time-triggered CAN (TTCAN) that does support time-triggered scheduling. TTCAN is a protocol designed specifically for safety-critical automotive applications where reliable and deterministic communication is required [LH02]. It provides a time-triggered communication mechanism, meaning messages are sent and received at specific times rather than in response to events, as illustrated in Section 2.2. In TTCAN, the time-triggered communication is implemented using a time base and a schedule table. The time base is a clock used to synchronize all nodes on the network, and the schedule table specifies the times at which messages are sent and received.

2.4.2 FlexRay

FlexRay is a communication protocol that was developed as a collaborative effort between Bosch, Continental, and Siemens. It is a network protocol specifically designed for use in the automotive industry but has also been used in other industries. FlexRay is a high-speed, fault-tolerant communication system designed to support the real-time requirements of automotive control systems. The dual-channel redundancy ensures high reliability and fault tolerance. It uses a deterministic communication protocol, which guarantees a fixed communication delay for each message, making it well-suited for safety-critical systems [MT06].

FlexRay operates over a physical layer that uses a pair of fiber optic cables or a pair of twisted shielded pairs of wires. It uses a time-division multiple access (TDMA) scheme to share the communication channel among multiple nodes. This protocol has a data rate of up to 10 Mbps and can support up to 64 nodes on the network. It supports time-triggered and event-triggered communication, providing flexibility for various application requirements. FlexRay is used in a variety of automotive applications, including powertrain, chassis, and safety systems. It is also used in other industries, such as aerospace, military, and industrial automation.

Nonetheless, this protocol is more complex than others and can be costly. The bus also has certain disadvantages, such as lower operating voltage levels and asymmetry in signal edges, which can pose challenges when extending the network length [Fle23].

2.4.3 LIN Bus

The local interconnect network (LIN) is a low-speed, low-cost communication protocol that is commonly used in automotive systems for non-critical applications, such as controlling interior lighting and window motors. This protocol is designed to provide a low-cost and low-complexity time-triggered solution for automotive systems without high bandwidth and determinism of protocols like FlexRay or TTCAN. It is, therefore, well-suited for applications that require moderate determinism and reliability, such as some interior lighting and infotainment systems [DeN+01]. Modern automotive networks use a combination of LIN for low-cost applications, primarily in body electronics, CAN for mainstream powertrain and body communications, and the emerging FlexRay bus for high-speed synchronized data communications in advanced systems such as active suspension. The LIN bus uses a master/slave approach that comprises a LIN master and one or more LIN slaves. LIN is a byte-oriented protocol, which means data is sent one byte at a time. A byte field contains a start bit, 8 data bits, and a stop bit. The data bits are sent the least significant bit first [Bus23b].

However, the LIN protocol has certain limitations, including limited bandwidth, unsuitability for complex applications, restricted network size (typically accommodating up to 16 nodes), and reliance on a master-slave configuration where one master node can communicate with up to 15 slave nodes. In the event of a master node failure, the entire network can be affected.

2.4.4 Automotive Ethernet and Ethernet TSN

Communication among ECUs has become more complex, and network throughput has grown in bandwidth with increased software functionality in cars. Ethernet-based communication presents an appealing solution due to its capacity for high bandwidth and greater adaptability, facilitating seamless integration with cloud services and consumer products. Automotive Ethernet is a specialized Ethernet network with a physical layer adopted for automotive

applications. It utilizes advanced PHY transceivers (a transceiver component for transmitting and receiving data or Ethernet frames) to reduce cable costs while meeting automotive electromagnetic compatibility and immunity standards. This technology enables faster communication compared to traditional automotive networks and facilitates the integration of internet protocol (IP) software technologies, including adaptations for automotive use, functional safety, and cybersecurity [SZ18; MK21].

The difference between Ethernet and automotive Ethernet lies within the physical layer. In terms of communication, both utilize IP, similar to other Ethernet variants. However, automotive Ethernet optimizes its physical layer for specific automotive applications. While 100Base-T1 and 1000Base-T1 function as switched networks, similar to standard Ethernet, they employ distinct Phy transceivers and cables. These cables consist of a more cost-effective single twisted pair, enabling full duplex communication instead of the dual twisted pair configuration. The choice between shielded (STP) or unshielded twisted pairs (UTP) depends on the requirements. Moreover, 10Base-T1S employs a single twisted pair but operates as a multi-drop bus, similar to CAN, rather than functioning as a switched network [SZ18; Eth23]. Automotive Ethernet can support data transfer rates of up to 10 Gbps. This is much higher than traditional automotive networking protocols, which can only support rates of up to 1 Mbps. Various Automotive Ethernet standards exist, including 100BASE-T1 (capable of transferring data at speeds of 100 Mbps), 1000BASE-T1 (transferring data at speeds of 1,000 Mbps), and 10GBASE-T1 (transferring data at speeds of 10 Gbps). There is ongoing development of Automotive Ethernet PHY standards to accommodate speeds higher than 10 Gbps, such as 25, 50, and 100 Gbps [Eth23; Tek23].

Ethernet TSN

Time-sensitive networking (TSN) has emerged as a revolutionary technology born out of the necessity to address the evolving demands of real-time communication in networking. Initially developed by the institute of electrical and electronics engineers (IEEE) as part of the IEEE 802.1 working group, TSN introduces a set of standardized protocols and mechanisms to ensure precise and predictable data delivery in Ethernet networks [Fin18; IEE18]. TSN enables several key features for automotive networks comprising redundancy in two areas: data transmission paths and network time masters. IEEE 802.1CB, which refers to frame replication and elimination for reliability, provides the redundancy capability for data transmission. TSN enables deterministic communication, ensuring critical data is delivered with low and bounded latency. This is crucial for real-time applications. This technology provides precise time synchronization across networked devices, allowing for coordinated actions and event triggering. This feature is particularly beneficial in scenarios where multiple devices or systems must act in harmony, such as in distributed automation and autonomous vehicles. It also introduces enhanced quality of service (QoS) mechanisms prioritizing time-sensitive traffic over less critical data streams. It ensures that different types of traffic receive the appropriate level of service. It includes not only prioritization but also traffic shaping, bandwidth reservation, and congestion control. TSN also provides the frame preemption capability, allowing high-priority, time-sensitive Ethernet frames to interrupt and take precedence over lower-priority frames that may be transmitting on the network. This mechanism ensures that critical data gets delivered promptly, even if the network is busy with non-time-sensitive traffic [Fin18; IEE18; AFK21b].

However, automotive Ethernet technology has some disadvantages, including costs related to required switches, overheads for real-time communication (such as TSN), electromagnetic interference, and a more expensive physical layer interface [Eth23].

2.5 Automotive Safety Standards

Automotive safety standards are a set of regulations and guidelines established to ensure the safety of vehicles, passengers, and road users. These standards are developed and enforced by governmental agencies and international organizations to reduce the risk of accidents, injuries, and fatalities in the automotive industry. In the following subsections, two of the most common and pertinent standards relevant to the vehicle E/E architecture, will be explained.

2.5.1 ISO 26262 (Functional Safety for Road Vehicles)

ISO 26262 [ISO18] is an international standard for functional safety in road vehicles. The international organization for standardization (ISO) defined the standard in 2011 and revised it in 2018. It applies to E/E systems in production vehicles, including driver assistance, propulsion, and vehicle dynamics control systems. It provides guidelines for the development and integration of safety-related systems in vehicles, intending to reduce the risk of injuries or fatalities resulting from accidents involving a failure of these systems [AHK22; ISO18; AFK21a].

The standard is organized into eleven parts as follows, each covering a different aspect of functional safety [ISO18].

- Vocabulary (Part 1): This part defines the terms and definitions used in the standard.
- Management of functional safety (Part 2): This part of ISO 26262 determines the requirements for functional safety management for automotive applications.
- Concept phase (Part 3): It describes the early phase of product development. The concept phase includes an impact analysis covered in Part 2.
- Product development at the system level (Part 4): It covers specifications for technical safety, including technical safety concepts, system integration design, item integration, and testing. The purpose of Part 4 is to ensure that the system design and technical safety concept comply with the functional safety requirements.
- Product development at the hardware level (Part 5): Part 5 of ISO 26262 is a standard that specifies the hardware architectural metrics for product development at the hardware level for automotive applications. The standard aims to reduce random hardware failures that impact functional safety.
- Product development at the software level (Part 6): It covers product development at the software level, including design, production, and testing. It also provides requirements for detecting, indicating, and controlling faults in safety-related hardware. This part of ISO 26262 comprises different safety requirements which have been considered in this thesis.
- Production, operation, service land decommissioning (Part 7): This part of the standard includes planning activities for automotive system safety during the remaining phases of the product lifecycle. It involves production, operation, service, and decommissioning.
- Supporting processes (Part 8): It describes a framework for functional safety to help with the development of safety-related E/E systems. The framework is meant to be used to integrate functional safety activities into a company-specific development framework.

- Automotive safety integrity level (ASIL)-oriented and safety-oriented analyses (Part 9): It specifies the requirements for ASIL-oriented and safety-oriented analyses. The standard utilizes a risk classification scheme to define the safety requirements.
- Guidelines on ISO 26262 (Part 10): It provides guidance on the interpretation and use of the standard.
- Guidelines on application of ISO 26262 to semiconductors (Part 11): This part is an adaptation of ISO 26262 to make it easier to apply the standard's requirements to semiconductor devices. It comprises a more detailed definition of transient faults than the original version of ISO 26262.
- Adaption of ISO 26262 for motorcycles (Part 12): It specifies functional safety requirements for motorcycle E/E systems.

2.5.2 SOTIF/ISO 21448

SOTIF [ISO19], which stands for safety of the intended functionality, is a methodology for evaluating and demonstrating the safety of automated systems. It is a risk-based approach that focuses on the potential risks to human safety that may arise from the intended functionality of an automated system. The goal of this standard is to identify and mitigate potential safety hazards before they can cause harm, rather than relying on traditional safety measures that are reactive and only activated after an accident has occurred. To do this, SOTIF includes a series of steps for identifying, analyzing, and mitigating risks related to the intended functionality of an automated system [ISO19].

The SOTIF process typically involves the following steps:

- Identify the intended functionality of the automated system, including the tasks it is designed to perform and the conditions under which it will operate.
- Detect the potential safety hazards that may arise from the automated system's intended functionality, including physical hazards (such as collisions or entrapment) and psychological hazards (such as confusion or discomfort).
- Analyze the likelihood and severity of these hazards, using techniques such as hazard analysis and risk assessment.
- Mitigate the identified hazards by implementing appropriate safety measures, such as design changes, warning systems, or training programs.
- Validate the effectiveness of the implemented safety measures through testing and verification.

2.6 Safety Requirements

Safety requirements are essential throughout the entire car production process, following automotive functional safety standards [ISO18; ISO19]. Consequently, these safety requirements must be taken into account and ultimately met during the car's E/E architecture design. Several safety conditions arise during the architectural design phase, aligning with ISO 26262 [ISO18]. The following subsections discuss the safety conditions most relevant to this thesis.

2.6.1 Redundancy

Redundancy is a key concept in the context of functional safety, as it refers to the use of multiple redundant components or systems in order to provide backup and increase the overall reliability of a system. This is particularly important in safety-critical systems, where a malfunction or failure can have serious consequences. According to ISO 26262 [ISO18], there are two types of redundancy: homogeneous redundancy, which necessitates the duplication of elements, either hardware components or software processes, and heterogeneous redundancy, which can be implemented using diverse elements [AMK23; AFK21a]. Redundancy can be employed in various ways to enhance the reliability of a system. The first method is hardware redundancy, which entails using multiple redundant components within a system, such as multiple sensors or actuators, to provide backup in case one component fails. Another approach is software redundancy, which involves the utilization of multiple redundant software algorithms or programs to perform the same function, thus providing backup in case one algorithm fails. Finally, functional redundancy encompasses using multiple redundant systems or subsystems to perform the same function, serving as a backup if one system fails [ISO18; AFK21a; AFK20].

In this thesis, the need for redundancy is automatically taken into account when designing and synthesizing E/E systems, which may include hardware, software, and functional redundancies. Further details will be provided in Chapter 4.

2.6.2 Freedom from Interference

Freedom from interference (FFI) is a concept denoting the prevention of cascading failures among components within a system. Such failures have the potential to violate safety requirements. FFI refers to the ability of a vehicle's systems to operate correctly and safely without interference from external sources. This is an essential aspect of vehicle safety, as interference can disrupt the functioning of critical systems, leading to accidents or other hazards. This safety requirement holds significant importance in mixed-criticality systems, where it guarantees that components of lower criticality do not impact those of higher criticality. To attain FFI, block partitioning ensures that a fault detected within one block remains isolated and does not propagate into other blocks. This requirement is just one aspect of dependent failure analysis [AMK23; ISO18; AHK22]. FFI requirement is considered in this thesis, and its details will be explained in Chapter 4.

2.6.3 ASIL

ASIL, which stands for automotive safety integrity level, is a risk classification framework applied for the functional safety of road vehicles. The ASIL classification is established by ISO 26262, specifically detailed in its ninth Section [ISO18]. This system is derived from the safety integrity level (SIL) principles initially outlined in IEC 61508 [Com23]. There are typically four ASIL classes, namely ASIL A, ASIL B, ASIL C, and ASIL D. ASIL D represents the most rigorous level of risk management. At the same time, ASIL A is associated with the lowest level of risk. ASIL D imposes the highest integrity and safety requirements, ensuring that components or systems developed for ASIL D adhere to the most stringent safety standards [ISO18; AFK21a]. There is a fifth classification called quality management or QM, which means a risk needs to be higher to require a dedicated safety goal. QM indicates there is no need to implement additional risk reduction measures beyond the industry-acceptable quality system.

ASIL targets critical safety [ISO18; AHK22; AMK23]. Systems like airbags, anti-lock brakes, and power steering require an ASIL D grade, representing the highest level of rigor in safety assurance due to the elevated risks associated with their failure. Moreover, ADAS applications capable of controlling the steering wheel and brakes are also classified as ASIL-D applications. On the other end of the safety spectrum, components like rear lights require only an ASIL A rating. Headlights and brake lights are typically categorized as ASIL B, while cruise control is generally classified as ASIL C. Game applications integrated into the infotainment system are defined as QM applications [Per23].

ASILs are determined through hazard analysis and risk assessment. Engineers evaluate three specific variables for each electronic component in a vehicle as follows:

- Severity: The type of injuries to the driver and passengers
- Exposure: The frequency at which the vehicle encounters the potential hazard
- Controllability: How much control the driver possesses in preventing injuries

Each of the abovementioned variables is further subdivided into subclasses. Severity includes four classes ranging from no injuries (S0) to life-threatening/fatal injuries (S3). Exposure consists of five classes covering the incredibly unlikely (E0) to the highly probable (E4). Controllability comprises four classes ranging from generally controllable (C0) to uncontrollable (C3). All variables and their sub-classifications are analyzed and combined to determine the required ASIL. For instance, a combination of the highest hazards (S3, E4, and C3) results in an ASIL D classification [ISO18]. Details on how the ASIL requirement is applied in this thesis will be described in Chapter 4.

2.6.4 Reliability

Reliability plays a pivotal role in the design of an E/E architecture. It is quantified using the failure rate denoted as λ, which represents the number of failures occurring within a specified time frame. Typically, this metric is expressed in failures in time (FIT), indicating the number of failures per billion hours of operation. In the field of reliability engineering, it is customary to utilize metrics like mean time to failure (MTTF), which measures how long a non-repairable item is expected to last before it fails, and mean time between failure (MTBF), which measures how reliable a product is, instead of the failure rate for assessing reliability [Com+17]. If the failure rate remains constant over time, it can be assumed that

$$MTTF = \frac{1}{\lambda}. \tag{2.1}$$

Note that ISO 26262 does not provide explicit formulas or calculations for reliability, but it does emphasize the importance of understanding the reliability characteristics of safety-critical components. Reliability is considered a factor in assessing the safety of these components, but the standard itself does not prescribe specific methods for calculating reliability metrics like MTTF or failure rate.

The integration of reliability into this thesis follows a component-based approach inspired by the reliability block diagram (RBD) method in the sense that the overall reliability prediction is calculated by looking at the individual components of the system while considering serial and parallel component configurations [Com23; Com+17]. Further details will be expressed in Chapter 4.

2.7 Design Space Exploration (DSE)

Design space exploration (DSE) is a critical aspect of the engineering and design process, serving as a methodical approach to navigating numerous possibilities within the space of potential solutions during system development. It systematically investigates various design alternatives, considering multiple dimensions. This process enables engineers and designers to evaluate trade-offs, identify optimal configurations, and make informed decisions based on a thorough analysis of the available design options. A substantial and complex system may encompass an extensive range, potentially reaching millions or even billions of design alternatives, with some scenarios presenting an infinite design space [KJS11]. DSE is particularly valuable in complex and multidisciplinary domains where diverse factors influence the overall system performance. By embracing DSE methodologies, practitioners gain insights into complicated relationships among design parameters, fostering innovation and efficiency. In the context of synthesizing E/E architectures for vehicles, DSE involves a comprehensive analysis of alternative configurations, considering various factors such as system requirements, performance metrics, and resource constraints [AFK21b; AFK21a].

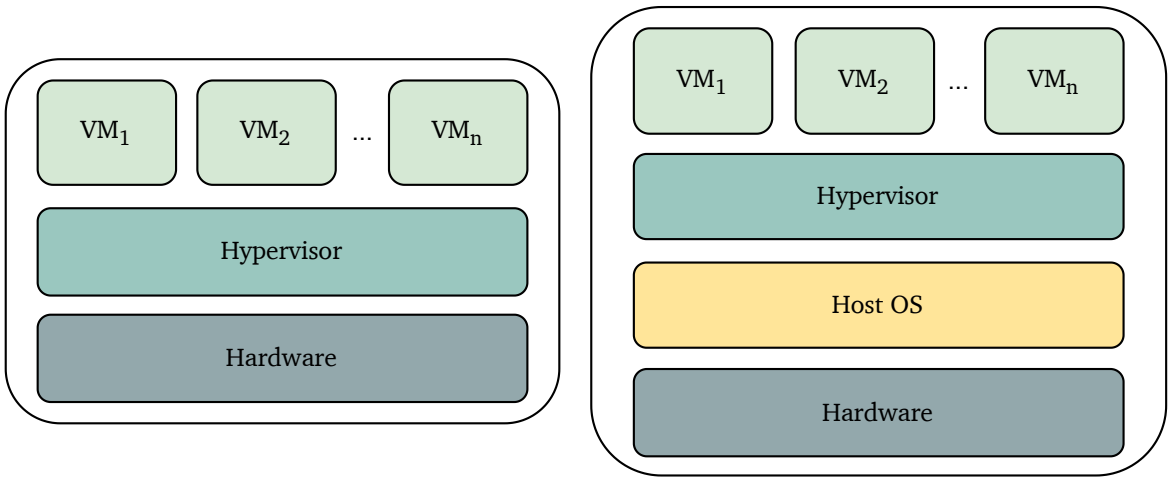


Figure 2.4: Two types of hypervisors including (a) Type-1 or Bare Metal hypervisor and (b) Type-2 or Hosted hypervisor.

2.8 Hypervisor

Hypervisors are commonly used in the automotive software domain to enable the development, testing, and deployment of software applications in a virtualized environment. This can reduce the cost and complexity of hardware infrastructure and provide a stable and isolated environment for testing and deploying applications. Moreover, hypervisors can be used to host multiple operating systems and applications on a single physical host, allowing for greater flexibility and scalability in deploying automotive software [MAK22].

The emerging software-defined architecture plays an integral role in shaping the future E/E architecture [AHK22]. It is essential to consider hardware virtualization technology to seamlessly integrate safety-critical and non-safety-critical software domains into the HPCU while ensuring compliance with safety requirements and optimizing hardware resource utilization. This technology can be effectively implemented through a hypervisor, software

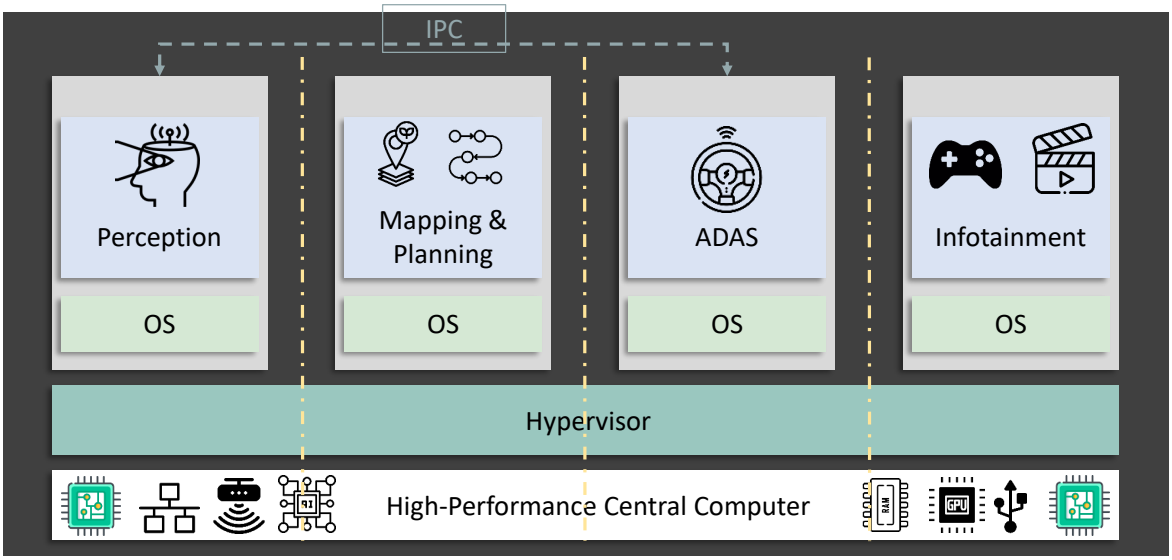


Figure 2.5: The software architecture integrated into the vehicle's HPCU using a type-1 hypervisor consisting of four mixed-critical partitions. The yellow dash lines show the hard separations between partitions starting from the hardware level [AHK22].

designed to create and manage virtual machines (VMs). The hypervisor facilitates sharing virtualized hardware resources among multiple instances running various operating systems [AHK22; MAK22]. There are two types of hypervisors:

- Type-1: It is also known as the Bare Metal or Native hypervisor since it is installed and operates directly on the host's hardware without relying on any host operating system. According to Figure 2.4 (a), this hypervisor possesses direct authority over and access to the hardware resources. For instance, a type-1 hypervisor can allocate a dedicated core to a partition (an execution environment supervised by the hypervisor that utilizes virtualized services) in a manner that prevents other partitions from accessing that core [AHK22; MAK22].

- Type-2: It is also referred to as a Hosted hypervisor. This type of hypervisor operates as an application within the host OS and leverages hardware resources for its virtual machines (VMs) by directing requests through the host's OS based on Figure 2.4 (b). Importantly, the host OS remains unaware of this hypervisor type, treating it like any other regular process [MAK22; AHK22].

The FFI requirement can be satisfied for safety-critical applications by using a hypervisor. For instance, based on the structure of the Bare Metal hypervisor presented in Figure 2.4 (a), safety-critical applications can be run on VM_1. While QM applications can be executed on VM_2, ensuring that safety and non-safety-critical applications do not interfere with each other. Figure 2.5 illustrates a high-level software architecture for a vehicle integrated into a central HPCU. Different partitions are configured to isolate various application domains, including merging mixed-critical applications, to efficiently utilize hardware resources. This is achieved through the use of a type-1 hypervisor [MAK22], allowing each partition direct access to HPCU resources, such as cores, RAM, GPU, cache, network buses, and universal serial bus (USB) interfaces. As shown in Figure 2.5, each partition operates with its OS and application domain, such as perception, layered on top of the OS. This arrangement ensures that each partition can utilize HPCU resources directly, thereby adhering to the principle of FFI as specified in ISO 26262 [ISO18]. However, it is important to note that the extent to

which resources follow the FFI requirement may vary depending on the choice of hypervisor. Various open-source and commercial hypervisors, developed by different companies, may offer different levels of FFI compliance based on the specific hardware configurations to be employed [AHK22].

In the context of the type-1 hypervisor, resources among partitions cannot be shared or accessed by other partitions, as indicated by the yellow dashed lines in Figure 2.5 [AHK22]. Furthermore, the inter-process communication (IPC) feature can activate communication between two partitions using a type-1 hypervisor. For example, suppose one process in a safety-critical partition (e.g., perception) intends to interact with another process in another (e.g., ADAS) to transfer messages. In that case, it can be accomplished by IPC (see Figure 2.5) [AHK22].

3

State of the Art

In this chapter, the related works and studies are presented with respect to our research questions and motivation. Additionally, a wide range of commercial, and non-commercial frameworks/tools have been developed for automotive software integration, configuration, and E/E architecture synthesis. These tools aim to address mapping issues and model software components, including DSE and optimization methods. A comprehensive examination and analysis of these tools are provided in Section 3.4. Furthermore, we discuss our contributions to the current state of the art.

3.1 Communication Message Routing and Synthesis of Time-triggered Schedules in Automotive Networks

In this section, we go through the most relevant studies to create valid paths over the vehicle network in order to transfer a communication message from a sender to a receiver in the automotive architectures. Moreover, studies and approaches related to the synthesis of time-triggered scheduling are illustrated.

3.1.1 Communication Message Routing

In their work, Lukasiewycz *et.al* [LSF14] presented a model based on graphs and a constraint system that enables the creation of message routing for applications running on an ECU within an architecture. They also introduced a binary encoding strategy in a previous work [Luk+09], which uses Satisfiability (SAT)-encoding to allocate resources, bind tasks, and route messages for optimal one-linear objectives. In [Smi+18], an automated method was presented for producing automotive architectures that have low redundancy and single routings. The method takes predefined optimization objectives into account by utilizing SAT-Decoding. It involves assigning resources to the system based on the mapping of applications. Generated routes are then optimized based on the number of allocated links and MTTF. MTTF is defined as the individual failure rate for all the components of the architecture.

This study [Smi+17] focuses on optimizing the communication within automotive networks where multiple systems with varying levels of criticality coexist. The authors propose a message routing and scheduling strategy for time-triggered networks in such environments to ensure efficient and reliable communication. This strategy takes into account the different criticality levels of the messages and assigns appropriate priorities to them to avoid

congestion and minimize latency. Optimization is achieved by using mathematical models to determine the best routing and scheduling decisions in real-time. The results of the proposed strategy are evaluated and compared to other existing methods, demonstrating its effectiveness in improving communication performance in automotive mixed-criticality systems.

The recent introduction of TSN standards has further enhanced the safety of E/E architectures, including mechanisms that ensure deterministic timing and high network reliability. IEEE developed TSN standards [IEE18] to address the hard real-time requirements of Ethernet-based distributed applications. The primary goal of these standards is to facilitate the implementation of distributed applications with varying levels of criticality on the same network infrastructure. In particular, the IEEE 802.1DG TSN standard was developed for the automotive industry and includes profiles for secure, highly reliable networks (e.g., utilizing IEEE 802.1CB for seamless redundancy), deterministic latency, and automotive in-vehicle bridged IEEE 802.3 Ethernet networks. Nayak *et.al* [NDR16] propose a method for ensuring real-time communication of time-sensitive data through transmission scheduling. Their approach employs integer linear programming (ILP) formulations to generate schedules. Their work focuses on the relationship between message routing and the schedule of time-triggered traffic, while they do not address redundant routing. Authors of [Gav+17] present a solution to an optimization problem that involves the synthesis of TSN-based distributed network topologies and stream routing. This approach considers the applications' real-time and redundancy requirements and aims to minimize the cost of the architecture as an optimization goal. Meanwhile, [Smi+19] introduces an automated approach for optimizing routing that takes the regions in a given network into consideration that lack routing variety. The authors also propose an algorithm for identifying proxy areas in the network.

3.1.2 Synthesis of Time-triggered Schedules in Automotive Domain

There are many studies on time-triggered schedule synthesis. In the following, the most relevant ones in relation to this thesis are discussed.

Over the past few years, various research studies have been conducted on software integration and architecture synthesis within the automotive domain. One such example is the study conducted by the author in [Ter18], who proposed an optimized reconfiguration of industrial automation systems. This study utilized the DSE approach to determine optimal architectural configurations for control applications by taking into account constraints and optimization goals. Similarly, a framework was introduced in [Zhe+16] that offers a model for architecture in automotive systems. Authors claimed that the framework they presented enables integration of systems while incorporating optimization techniques and also takes into account validation for various design metrics, such as reliability and timing [AHK22]. In [Zhe+05], the authors discuss a time-triggered scheduling scheme that can help to reduce development and re-verification efforts in the automotive domain. Meanwhile, [Smi+17] presents a methodology for generating valid time-triggered schedules for routing communication frames over network links. This approach leverages a 0-1 ILP formulation. In [Sag+15], the authors describe a technique for synthesizing time-triggered schedules for automotive hardware architectures. This method generates release times for periodic tasks and messages and can also support multi-schedule synthesis for different hardware architecture variants. Authors also conduct several evaluations to measure the effectiveness of their approach, including tests for resource utilization, run-time evaluation for both variant and non-variant use cases, and end-to-end delay analysis.

This study [LC12] proposes a methodology to optimize time-triggered automotive systems with a particular focus on FlexRay bus scheduling. An extended architecture model of this work considers resource utilization and configuration, path delays. Furthermore, the

authors present a strategy to enable an efficient selection of architecture decisions to avoid an infeasible schedule. Authors of [FKK17] propose a graphical modeling tool to reduce configuration effort and overhead by automating gate control list (GCL) schedule synthesis for TSN networks. Moreover, they use a model-based development approach to develop their framework and apply logic programming to transform a designed graphical network to a network knowledge base. Zhang et al. [Zha+14] formulate a method for computing time-triggered schedules for tasks routing over communication links and applications running on network end stations. Authors in [AHM20] aim to address the problem of computing no-wait schedules and multi-path routings for large-scale TSN networks. This study introduces an iterated ILP-based scheduling approach to achieve high scheduling scalability and also provides a degree of conflict (DOC)-aware multi-path routing methodology to achieve fault tolerance. Developing proper scheduling approaches, as well as modeling and verification tools, including experimental setups, is essential in traffic planning and verification of TSN networks. However, this process can be time-consuming and requires advanced expertise, which highlights the need for motivation in this area of research [Cra+16; Ash+17; FK17].

3.2 Software Architecture Synthesis-related Studies

This section presents related papers on software architecture and E/E system synthesis, including those that consider safety requirements.

3.2.1 Software Architecture Synthesis

In [Zim+18], a new methodology is presented for solving optimization problems in engineering design, particularly those with a nested structure of design parameters. The authors introduce the concept of "existence-dependent parameters," which are found to arise in various problems, including the optimization of system architectures. A model-driven approach is proposed in this study to address these issues. A prototype tool is introduced that integrates multiple domain-specific tools. A methodology and tool are demonstrated in the context of a distributed embedded system design example, which includes hardware architecture and software mapping. An analysis of timing in embedded computer systems is the focus of this study [Edm+15], which proposes a new approach. Authors observe that current methods are inadequate as they do not consider the layered, distributed, and heterogeneous nature of these systems. The framework proposed is aimed to be more adaptable and multipurpose, giving developers the liberty to choose and utilize a collection of timing analysis tools that would be appropriate for each individual subsystem. This tool considers the interdependence among the subsystems and delivers a thorough timing analysis from end-to-end. A methodology presented in [TVW18] outlines a process for identifying the best deployment configurations for systems that adhere to the IEC 61499 standard. This approach involves exploring various design options within the system's design space to identify the optimal configuration that satisfies the given requirements and constraints. The objective is to enhance the system's efficiency and performance by deploying it in an appropriate manner. DSE outcomes are utilized to compute deployment configurations that fulfill the desired specifications.

3.2.2 E/E System Synthesis Considering Safety Requirements

Mody [Mod+18] provides an analysis of two automated driving (AD)/ADAS system topologies to explain their characteristics. The study compares the two topologies based on various

parameters, such as bandwidth, functional safety, number of ECUs, and cost. This work also explains the system partitioning specific to the AD/ADAS domain for each topology. The authors of [ZL19] examine the future E/E architectures in the automotive industry and the components that will be necessary for these architectures, such as cyber security, energy management, and appropriate middleware. [DS18] examines the safety aspects of the Ethernet network used in the automotive domain, and provides a comparison of bit flip between Ethernet-based and CAN-based networks. Furthermore, this study presents a short overview of future automotive networks considering some functional safety characteristics such as seamless redundancy. Authors of [Abd+17] suggest a safety method to enhance the safety of the architecture employed in an autonomous vehicle, as well as evaluate the architectural design of the self-driving system during the development stage. In addition, the introduced approach functions as a hazard analysis technique, named system theoretic process analysis (STPA), in compliance with ISO 26262.

Yoneda et al. [Yon+19] propose a technique for safety-critical applications in the automotive industry that is suitable for a centralized ECU. This technique utilizes a network-on-chip (NOC) platform specifically designed to address link and delay faults. In this work, Aniculaesei and colleagues [Ani+16] put forward a method for ensuring the safety of autonomous systems. This method involves combining static verification techniques during the design phase with dynamic verification techniques during operation, allowing for the transfer of results from the design phase to the run time environment. Furthermore, the authors proposed a real-time monitoring approach to assess the accuracy of the system assumptions during operation based on the assumptions made during the design phase. In this study [Chi+17], the focus is on identifying safety requirements for the software of a self-driving car. The study lists four methods that can be used to ensure safety:

- Implementing redundancy at multiple levels of the system.
- Designing the software and hardware architecture based on redundancy requirements.
- Utilizing FFI technique.
- Monitoring the software and hardware at runtime to detect and avoid any potential failures.

In [Xie+17a], the authors investigate the reliability of a heterogeneous automotive system using ASIL decomposition. They propose two heuristic algorithms to increase reliability while minimizing development costs for each ASIL decomposition scheme. The approach calculates reliability by considering tasks mapped on the ECUs and their corresponding failure rates. In both [GPM14] and [TP11], the authors aim to address the real-time requirements of distributed automotive applications while optimizing development costs. Haupt introduced a safety monitoring method for autonomous systems in 2019, as documented in [HL19]. This method uses a collection of safety rules to identify any safety-critical violations that may occur in the system during runtime. Additionally, the method provides the system with the ability to maintain self-awareness and ensure safe operations while in use. In their work, Gosavi et al. [GRC18] put forward three different architectural models for autonomous vehicles that ensure the functional safety of their E/E components in accordance with ISO26262 [ISO18]. Another group of researchers, Tlig et al. [Tli+18], focus on a model-based approach for assessing the safety of automated driving systems. In this approach, a model can be developed that includes the environment, perception, fusion, and control units for an autonomous car. This methodology also involves sequence extraction and analysis to validate particular scenarios. The model is then simulated, which includes visualizing the car and its surroundings.

3.3 Task Mapping in Multi-Core Computing Units

As previously discussed, this section will outline the current mapping techniques and methods, as well as the optimization objectives for task mapping in multi-core platforms.

3.3.1 Mapping Techniques

Static or design-time mapping uses all available system information, including hardware and application properties, to find the optimal solution. This type of mapping is most suitable when there are predefined requirements for both the applications and hardware, as it cannot dynamically address changes in application or hardware requirements. Subsequently, it cannot dynamically be solved using design-time methodology. As a result, solutions of higher quality can be achieved through design-time mapping due to the lack of limitations in computational power compared to run-time mapping [AHK22].

Design-Time Mapping

There are a number of algorithms used for design-time mapping, including graph-theoretic algorithms, mathematical programming algorithms, and heuristic-based algorithms. In graph-theoretic algorithms, the application is divided into separate tasks, which can then be assigned to the cores for execution and take advantage of parallelism [GBI21],[Dev+15]. This approach encompasses various theories such as Levelized Weight Timing [Shi+04], Shortest tree [Bok81], Max-Min [Bra+01], hyper-graph [Dev+15], A* [Shi+04], and Kahn Process networks [Cas+12].

Another design-oriented method for resolving the mapping problem is through mathematical programming, where the requirements are translated into mathematical inequalities. These inequalities are then solved through various mathematical programming solutions, including mixed-integer linear programming (MILP) [NM97], Branch & Bound [HM05], constraint programming (CP) [Bha+12], and ILP [Kai+12]. This approach provides an optimal solution as long as the mapping problem does not fall into the category of NP problems. In such cases, heuristic-based algorithms can be employed.

The final design-time approach is the utilization of heuristic-based algorithms. As previously stated, as the complexity of the problem and proximity to an NP problem increases (for example, with an increase in the number of cores and the complexity of application requirements), finding an optimal solution through mathematical programming algorithms becomes unfeasible within the desired time frame. Hence, heuristic algorithms are introduced to provide a solution that may not necessarily be the best, but can still approximate the exact solution in a faster and more efficient manner.

The alternative options in search algorithms are classified at each branching step based on the available information in order to determine the next branch to follow. These heuristic algorithms can be further divided into two categories: Population-Based and Single Solutions. for example, single solution algorithms refer to iterative search methods, such as greedy randomized adaptive search procedure (GRASP) [PML11], simulated annealing (SA)[Gia+14], and Tabu search[Bra+01]. These algorithms are known for utilizing an iterative search process. In addition to these algorithms, there are several other heuristic-based algorithms that fall under the category of population-based, such as genetic algorithms (GA)[Gan+16], ant colony optimization (ACO)[Fer+10], and particle swarm optimization (PSO) [Xu+16].

Run-Time Mapping

In this type of mapping, the allocation of tasks to various cores is carried out during the execution of applications. The amount of time required to find a feasible and optimal solution is a crucial factor in this mapping approach. Run-time mapping can be categorized into two approaches, On-the-fly mapping and Hybrid mapping, as reported in [Sin+13; GBI21]. On-the-fly mapping refers to a scenario where task allocation is performed online or during application execution without utilizing any offline or design-time information. While in Hybrid mapping, the task planner leverages the results of offline or design-time mapping to perform dynamic task assignments. This approach combines the benefits of both run-time and design-time techniques. For run-time mapping, there are three commonly used algorithms that can be utilized, including Greedy [CCM07], Feedback control theoretic, and Heuristic-based algorithms [GBI21]. These algorithms offer a range of options for run-time mapping tasks.

3.3.2 Optimization Parameters in Mapping

For the purpose of enhancing the task assignment quality in multi-core processors, whether through static or dynamic mapping, it is crucial to take into account certain critical requirements.

Performance

The design of multi-core processors has grown increasingly complex due to the growing number of applications and the increasing requirements placed on them. To optimize performance and improve mapping, various design-time and run-time approaches have been developed, such as reducing task execution time during task allocation [Kai+12] or increasing the utilization of the central processing unit (CPU). When evaluating the performance of a multi-core system, several metrics can be used, including execution time, latency, response time, and throughput [Meh+09]. These metrics allow for an assessment of the system's efficiency and effectiveness in executing tasks.

Power Consumption

Given the increasing popularity of electric vehicles, particularly in the realm of autonomous cars, energy conservation plays a critical role in optimizing the E/E system of the vehicle. Minimizing energy consumption during the application mapping process in multi-core computational units is a critical factor in optimizing the mapping process [Xie+18]. Research has shown that reducing cache miss rates can lead to a substantial decrease in energy consumption, with a 76 percent reduction in energy usage being reported in [Per09].

Reliability

When optimizing multi-core computing units, it is essential to take into account reliability as a key factor. ISO 26262 has established various ASILs to guarantee the reliability of the system [ISO18]. The reliability of the system can be evaluated through the use of MTTF, MTBF, and MTTR as failure metrics [DKV14]. As an illustration of the benefits of the SA approach, improved system reliability is demonstrated in [HYX09]. The study calculates the MTTF by considering the temperature variations of the cores. Based on this calculation, task allocation is performed in a manner that minimizes the MTTF.

A considerable number of studies have been conducted on the topic of the static mapping problem, and many of these have used the ILP method. For example, [Kai+12; NM97; DLM13; Cos+08] focuses on the design-time mapping problem in a homogeneous multi-core architecture and utilized the ILP method to optimize their solutions. These studies aimed to optimize parameters such as performance, energy consumption, and temperature. In [GZ+19], the authors employed a similar approach to address the mapping issue in a heterogeneous architecture, considering various optimization objectives such as execution time, reliability, and temperature. Other studies, including [Gia+14; HTM10; HYX09; Das+14], have explored the mapping problem for homogeneous architecture during the design phase by employing heuristic-based methods, such as SA, ant ACO, and GA, to optimize the final solution based on factors such as response time, reliability, and energy consumption.

A significant amount of research has been conducted to dynamically allocate tasks to multi-core processors. For example, studies by Kinsy et al. [KD14], Lee et al. [LR14], and Liu et al. [LPM13] focused on heterogeneous architectures and employed the ILP method to perform the run-time mapping. These studies aimed to optimize their solutions with respect to performance and power consumption. On the other hand, Bolchini et al. [Bol+16] utilized a heuristic approach, specifically the Age Balancer Heuristics, for dynamic mapping on heterogeneous architectures with the aim of minimizing computation energy and enhancing reliability. As a result of the growing demand for improved efficiency in mapping considering the aforementioned optimization parameters, new challenges have arisen in the area of multi-core processors. These challenges include managing thermal issues in integrated circuits (ICs), implementing machine learning techniques for efficient mapping, and maintaining QoS in multi-core architecture [GBI21], [Sin+13].

3.4 Technologies and Tools for Software Integration and Configuration in Design Process

As previously noted, the manual integration of software for future vehicles is expected to pose significant challenges. This section provides in-depth information on existing technologies for software integration, specifically focusing on model analysis, model checking and validation based on requirements, and mapping problems in embedded systems. Each technology is evaluated based on various attributes, including problem attributes, design metrics, DSE approaches, optimization algorithms, and safety-related and optimization attributes.

It is worth noting that the tools we review in this section play a critical role in the development of complex E/E systems. These systems can range from automotive to aerospace to industrial automation and require a lot of resources, time, and effort to design and develop. However, with the help of the tools that have been reviewed, designers and engineers can significantly reduce the time and effort required for design, implementation, and testing. One of the main reasons for analyzing these tools is to provide a clear understanding of the state-of-the-art in the field of E/E system design and synthesis. It is essential to understand the capabilities and limitations of these tools to make informed decisions about which tool is best suited for specific applications. It is important to note that the synthesis of software architecture in E/E systems is an ongoing and constantly evolving field as new safety requirements and standards continue to emerge. As such, it is essential for researchers and practitioners to stay up-to-date with the latest developments in this field to ensure that their designs are always compliant with the latest safety guidelines.

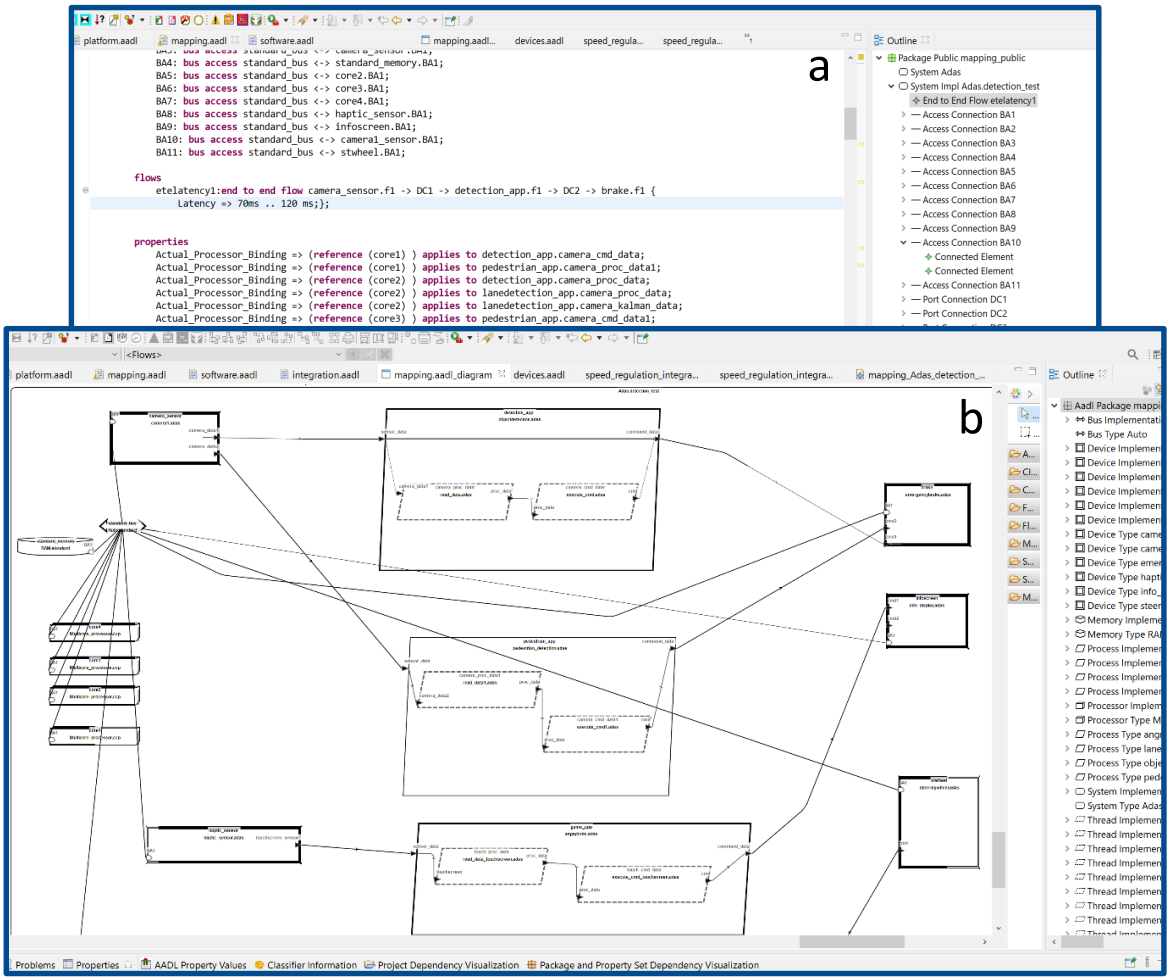


Figure 3.1: The AADL text editor (a) to synchronized graphical editor (b) in OSATE framework.

3.4.1 Non-commercial/Open-source Frameworks

This subsection discusses non-commercial computer-aided tools to design and synthesize embedded and E/E systems. At the end of this subsection, an overview of this analysis is presented as a table, including details regarding each illustrated framework.

OSATE (Open Source AADL Tool Environment)

It is a highly effective open-source tool for creating models using the architecture analysis and design language (AADL) with a syntax-aware text editor and a synchronized graphical editor (as seen in Figure 3.1)[AHK22]. OSATE is an Eclipse-based tool that provides modeling elements specifically designed for aerospace and automotive systems using the AADL language [Fei19],[OSA21].

AADL is a versatile modeling language designed to provide early and recurrent assessments of system architecture with a focus on performance-critical properties. This is achieved through the use of a customizable notation, a comprehensive tool framework, and well-defined semantics. The language is based on formal modeling principles that allow for the characterization and examination of application system architectures through the identification of individual components and their interactions. The scope of this includes abstract representations of software elements such as processes and threads, computational hardware, e.g., processors, buses, devices, and memory, and various system components. This is

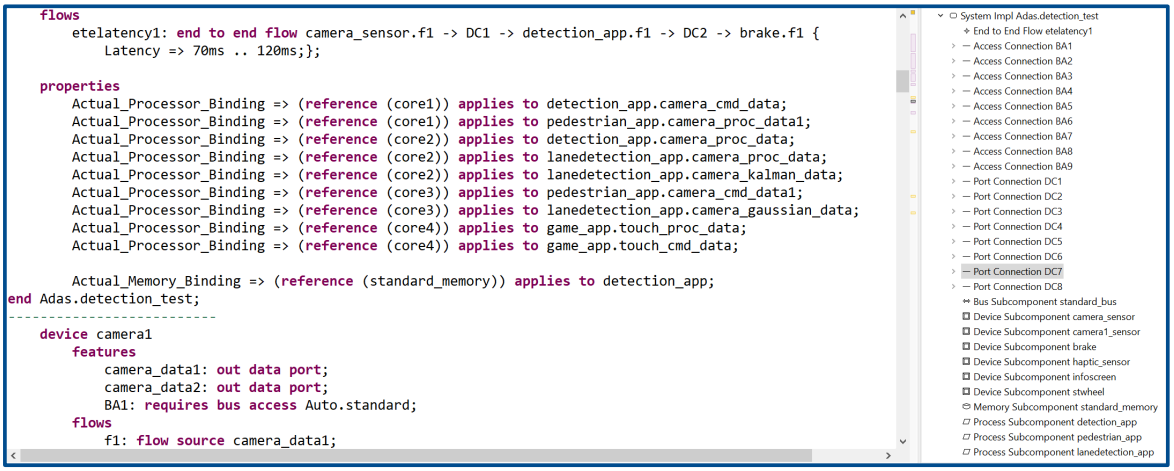


Figure 3.2: The AADL text editor for a lane detection application including flow analysis, processor and memory bindings in OSATE framework.

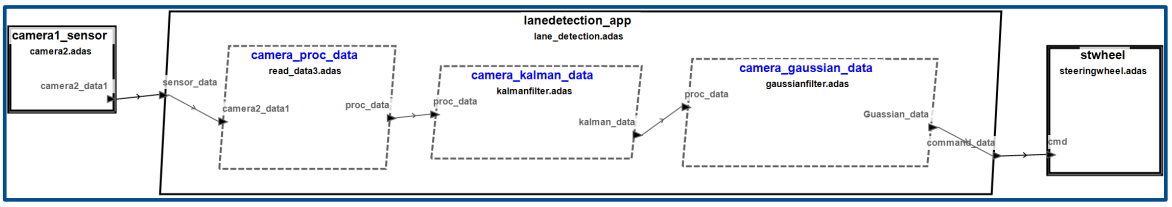
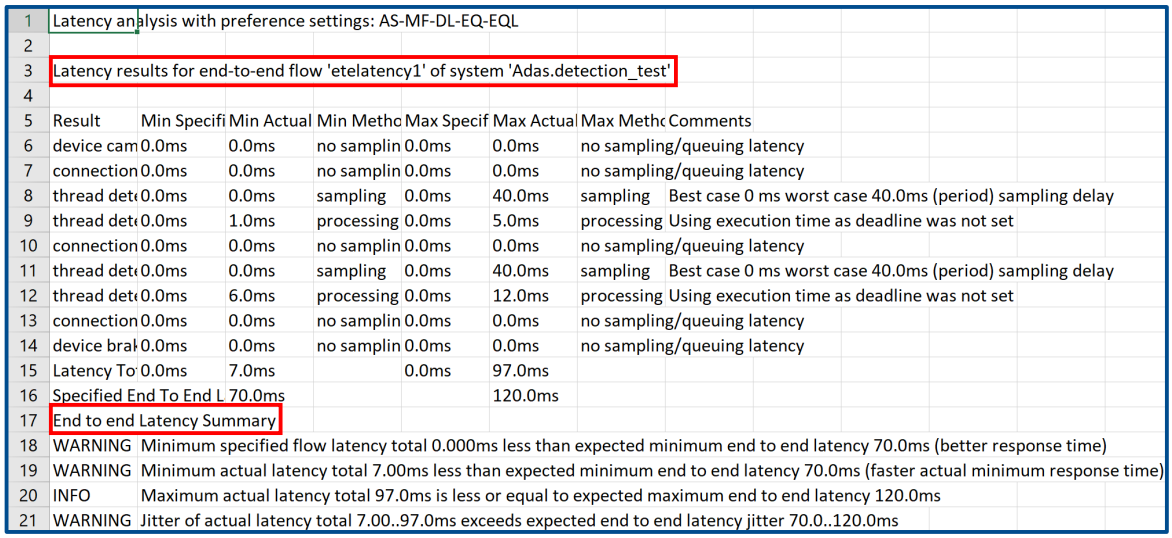


Figure 3.3: The synchronized graphical editor for a lane detection application created based on the AADL text in the OSATE framework.

done with the aim of determining and analyzing automotive, aerospace, and real-time embedded systems, as well as examining the performance capabilities of the designed system through techniques like data-flow analysis. This involves collecting information about the set of values computed at different points within the designed model or system. Additionally, the AADL facilitates the mapping of software components, such as processes, to computational hardware elements, such as processors. AADL has been proven to be particularly useful in the model-based analysis and specification of complex real-time embedded systems [FGH06].

For example, Figure 3.2 shows an AADL text for a lane detection application that takes flow analysis and processor bindings into account. Moreover, Figure 3.3 presents the graphical visualization of the AADL text for the lane detection application.

The OSATE tool allows users to model a system, including both hardware and software components, down to the application level, as demonstrated in Figure 3.1. With this tool, users can model the threads used for each application, specifying properties such as period, computation execution time, million instructions per second (MIPS) budget, and reference processor. The OSATE verifies the model created by the user for syntax errors in the AADL text and property definition violations for each component specified. In addition, this framework allows for a range of model analyses to be performed, including a flow latency check that computes the end-to-end flow latency. For example, Figure 3.4 shows the latency results for the defined flows of a use case in the tool. This tool includes scheduling analysis such as scheduling bound threads, i.e., determining processor utilization as a report, binding and scheduling threads, i.e., thread binding report, and assigning rate monotonic priority. For instance, the binpacking analysis and also thread to processor bindings report for a specific use case generated by the OSATE are displayed in Figure 3.5. The OSATE also covers budget analysis that involves analyzing bus load, power requirements, resource allocations,

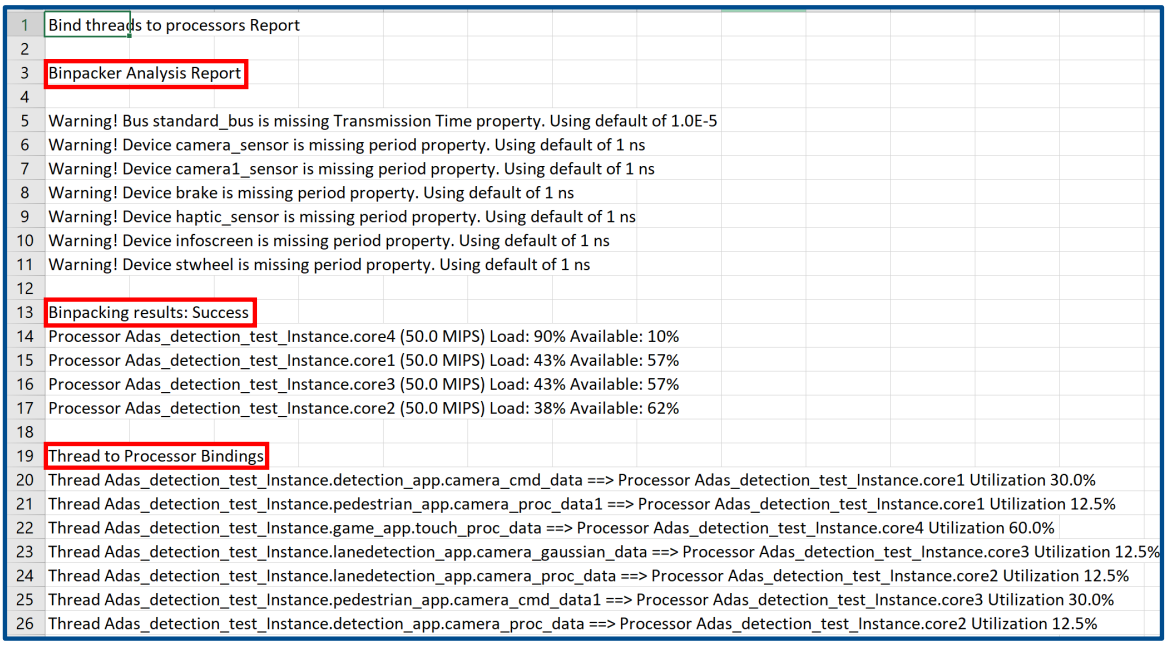


1	Latency analysis with preference settings: AS-MF-DL-EQ-EQL						
2							
3	Latency results for end-to-end flow 'etelatency1' of system 'Adas.detection_test'						
4							
5	Result	Min Specifi	Min Actual	Min Metho	Max Specif	Max Actual	Max Metho	Comments
6	device cam	0.0ms	0.0ms	no samplin	0.0ms	0.0ms	no sampling/queuing latency	
7	connection	0.0ms	0.0ms	no samplin	0.0ms	0.0ms	no sampling/queuing latency	
8	thread det	0.0ms	0.0ms	sampling	0.0ms	40.0ms	sampling	Best case 0 ms worst case 40.0ms (period) sampling delay
9	thread det	0.0ms	1.0ms	processing	0.0ms	5.0ms	processing	Using execution time as deadline was not set
10	connection	0.0ms	0.0ms	no samplin	0.0ms	0.0ms	no sampling/queuing latency	
11	thread det	0.0ms	0.0ms	sampling	0.0ms	40.0ms	sampling	Best case 0 ms worst case 40.0ms (period) sampling delay
12	thread det	0.0ms	6.0ms	processing	0.0ms	12.0ms	processing	Using execution time as deadline was not set
13	connection	0.0ms	0.0ms	no samplin	0.0ms	0.0ms	no sampling/queuing latency	
14	device bra	0.0ms	0.0ms	no samplin	0.0ms	0.0ms	no sampling/queuing latency	
15	Latency To	0.0ms	7.0ms		0.0ms	97.0ms		
16	Specified End To End L		70.0ms			120.0ms		
17	End to end Latency Summary							
18	WARNING	Minimum specified flow latency total 0.000ms less than expected minimum end to end latency 70.0ms (better response time)						
19	WARNING	Minimum actual latency total 7.00ms less than expected minimum end to end latency 70.0ms (faster actual minimum response time)						
20	INFO	Maximum actual latency total 97.0ms is less or equal to expected maximum end to end latency 120.0ms						
21	WARNING	Jitter of actual latency total 7.00..97.0ms exceeds expected end to end latency jitter 70.0..120.0ms						

Figure 3.4: The report of end-to-end latency analysis for the specified flows in the OSATE tool.

computer resource budgets, and calculating the total weight. Moreover, the OSATE tool comprises safety analysis that incorporates fault tree analysis (FTA), functional hazard assessment (FHA), fault impact analysis, failure mode effect analysis (FMEA), and the identification of unhandled faults [OSA21; AHK22]. Furthermore, this framework allows for the performance of a variety of semantic checks and functional integration analyses, including the examination of binding constraints, the consistency of connection bindings, and port connection consistency. This tool also boasts diverse code generation capabilities through the use of various plugins, such as Ocarina, and has the capability of importing models from both MATLAB and Simulink into the OSATE [Oca21; Hug+08; MAT10].

Despite its capabilities, the OSATE does not incorporate any DSE techniques, such as solving mapping problems for multi-core automotive computing units. As a result, it does not



1	Bind threads to processors Report
2	
3	Binpacker Analysis Report
4	
5	Warning! Bus standard_bus is missing Transmission Time property. Using default of 1.0E-5
6	Warning! Device camera_sensor is missing period property. Using default of 1 ns
7	Warning! Device camera1_sensor is missing period property. Using default of 1 ns
8	Warning! Device brake is missing period property. Using default of 1 ns
9	Warning! Device haptic_sensor is missing period property. Using default of 1 ns
10	Warning! Device infoscreen is missing period property. Using default of 1 ns
11	Warning! Device stwheel is missing period property. Using default of 1 ns
12	
13	Binpacking results: Success
14	Processor Adas_detection_test_Instance.core4 (50.0 MIPS) Load: 90% Available: 10%
15	Processor Adas_detection_test_Instance.core1 (50.0 MIPS) Load: 43% Available: 57%
16	Processor Adas_detection_test_Instance.core3 (50.0 MIPS) Load: 43% Available: 57%
17	Processor Adas_detection_test_Instance.core2 (50.0 MIPS) Load: 38% Available: 62%
18	
19	Thread to Processor Bindings
20	Thread Adas_detection_test_Instance.detection_app.camera_cmd_data ==> Processor Adas_detection_test_Instance.core1 Utilization 30.0%
21	Thread Adas_detection_test_Instance.pedestrian_app.camera_proc_data1 ==> Processor Adas_detection_test_Instance.core1 Utilization 12.5%
22	Thread Adas_detection_test_Instance.game_app.touch_proc_data ==> Processor Adas_detection_test_Instance.core4 Utilization 60.0%
23	Thread Adas_detection_test_Instance.lanedetection_app.camera_gaussian_data ==> Processor Adas_detection_test_Instance.core3 Utilization 12.5%
24	Thread Adas_detection_test_Instance.lanedetection_app.camera_proc_data ==> Processor Adas_detection_test_Instance.core2 Utilization 12.5%
25	Thread Adas_detection_test_Instance.pedestrian_app.camera_cmd_data1 ==> Processor Adas_detection_test_Instance.core3 Utilization 30.0%
26	Thread Adas_detection_test_Instance.detection_app.camera_proc_data ==> Processor Adas_detection_test_Instance.core2 Utilization 12.5%

Figure 3.5: Binpacking analysis and thread to processor bindings report in OSATE framework.

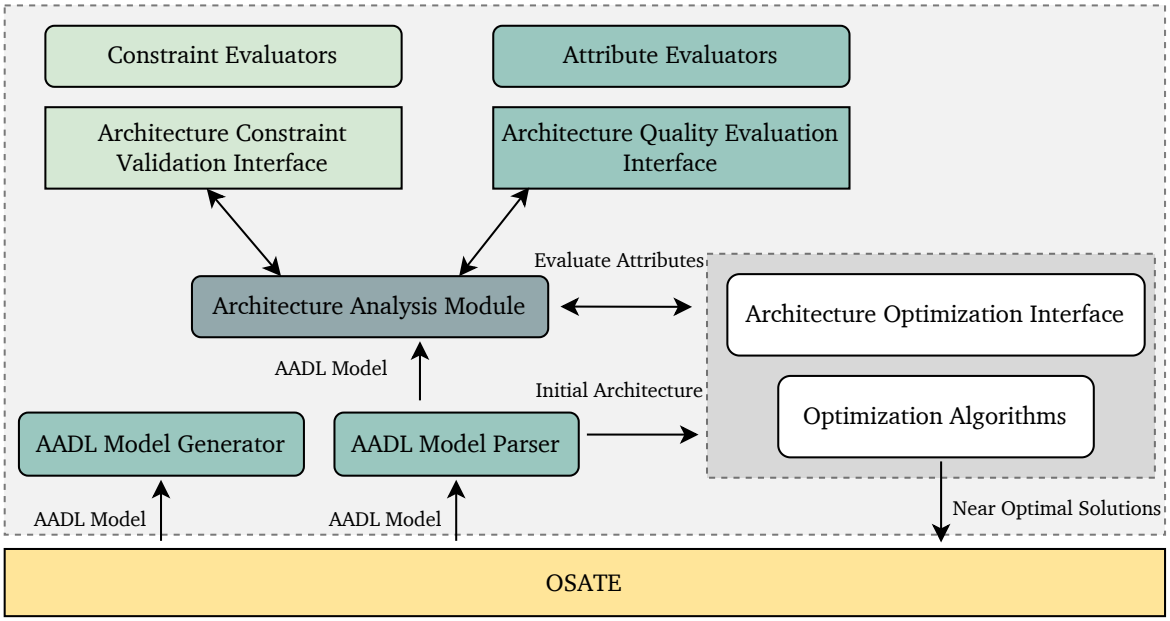


Figure 3.6: The architecture of ArcheOpterix framework [AMK23].

offer any optimization capabilities. Additionally, its coverage of safety attributes is limited as previously discussed [AHK22].

ArcheOpterix

It was previously stated that determining a suitable architecture design is a difficult task for software and system architects, as they must take into account both quality and functional requirements during the design phase. ArcheOpterix is a tool designed to make the task of evaluating, designing, and optimizing AADL specifications easier and more efficient. This open-source tool [Ale+09], which is based on the Eclipse platform, utilizes evaluation techniques, a DSE approach, and optimization heuristics to achieve its goals. The framework in question provides the capability to model software components, as well as the communication between software components, ECUs, buses, and services. Moreover, the tool has the ability to optimize the deployment of software components to ECUs while considering design constraints and optimization objectives such as redundancy allocation and cost-effectiveness [Ale+09].

The ArcheOpterix tool is designed to identify uncertain information related to system parameters and search for the optimal and robust candidate architecture. It also provides a list of the most appropriate optimization algorithms, including multi-objective genetic algorithm (MOGA), non-dominated sorting genetic algorithm (NSGA-II), pareto ant Colony algorithm (P-ACO), simulated annealing (SA), Hill Climbing, Bayesian Heuristic for component deployment optimization (BHCDO), Random Search Algorithm, and Brute-Force Algorithms, allowing the user to select the most suitable one [Mee+11]. As depicted in Figure 3.6, the high-level architecture of the ArcheOpterix framework comprises of multiple modules, the key components of which will be outlined below [Mee+11].

- AADL Model Parser: It is a module that is designed to interpret and extract information from an AADL specification generated by the OSATE tool. This module has the capability to access various AADL elements, such as components, services, and buses. The extracted parameters are then passed on to the Architecture Analysis Module as input, which provides support for two interfaces for analyzing the model, including the

Architecture Constraints Validation Interface and the Architecture Quality Evaluation Interface (as illustrated in Figure 3.6).

- Architecture Constraints Validation Interface: As illustrated in Figure 3.6, offers a connection point for the implementation of Constraint Evaluator modules that assess the compliance of a given architecture with established constraints.

- Architecture Quality Evaluation Interface: In this part, a variety of quality evaluation functions can be considered. The Attribute Evaluator module in ArcheOpterix performs these evaluation functions, which can be expanded to include additional evaluated features. Currently, ArcheOpterix includes the evaluation of Service Reliability, Data Transmission Reliability, and Communication Overhead as integrated features.

- Architecture Optimization Interface: The current framework offers the possibility of incorporating new optimization algorithms. The tool currently includes Exact Algorithms, Genetic Algorithms, and Ant Colony Optimization [Mee+11].

Despite its usefulness, this tool has several limitations when it comes to mapping, architecture synthesis, and software integration for automotive platforms. Initially, the framework is outdated and lacks proper documentation for usage. It does not provide support for mapping analysis or solving mapping problems for multi-core architectures and does not place a focus on multi-core computing platforms for automotive applications. Moreover, the safety attributes specified in ISO 26262, excluding reliability, are not incorporated in this tool. Additionally, the framework itself does not have any model checking or model analysis capabilities. The optimization objectives covered by ArcheOpterix are limited to cost, communication overhead, and data transmission reliability. Furthermore, the tool has a limited set of architectural elements [AHK22].

PerOpteryx

Another open-source framework exists for feature configuration and clustering during the design phase within the software domain, as described by the authors in [KKR11]. They assert that this approach can lead to finding the optimal solutions for software architecture, taking into consideration the predefined requirements and constraints, by utilizing multi-objective evolutionary optimization on software architectures modeled using the Palladio Component Model. With this approach, software architects have the ability to choose the most appropriate architecture for their specific circumstances. The proposed methodology offers software architectural solutions that are optimized based on various quality attributes, including performance, cost, and reliability, utilizing the DSE technique. PerOpteryx is a tool that generates potential software architecture candidates based on various degrees of freedom in component-based software architectures. Afterward, these candidates are evaluated and optimized to meet the specified requirements. The effectiveness of this approach was demonstrated through the application of two different architecture models, including a business reporting system and an industrial control system [BFK19].

However, this framework lacks support for mapping analysis or a DSE approach for task mapping, meaning it cannot find a solution for assigning processes, such as those integrated into automotive high-performance computing units, to cores while meeting all safety and non-safety requirements. Additionally, the framework has limited elements for use in the software architecture and does not include any model checking or analysis. Furthermore, the open-source framework does not define automotive safety parameters, and the PerOpteryx tool is outdated and suffers from inadequate documentation.

MechatronicUML

Due to the complex nature of modern technical systems, particularly reconfigurable mechatronic systems where most control and reconfiguration functionality is embedded in software, it is necessary to meet specific requirements in order to effectively utilize a model-driven development approach for these types of systems. Consequently, a research paper presented an open-source framework, built on the Eclipse framework, for modeling software and hardware components, specifying constraints, and verifying the models through the use of a model checker called UPPAAL for embedded systems. This framework is described in [Beh+06], [BGT04].

This tool is an innovative model-based approach that aims to bring the benefits of model-based design and formal analysis to the field of mechatronics. By utilizing a DSE approach, it helps users to find solutions based on predefined constraints in the model. Furthermore, it offers software reconfiguration capabilities that allow for automatic adaptation of the system at runtime to accommodate changes in the environment, providing the user with greater flexibility in the design process. This framework possesses the ability to generate code in the C programming language, and the models created can be simulated through the use of MATLAB Simulink, Modelica, or the functional mock-up interface (FMI)[FE98],[MAT10],[SW10].

Nonetheless, the limitations of this tool must be acknowledged based on the criteria set for mapping and software integration within the automotive industry. Firstly, the framework does not address mapping analysis or DSE related to mapping problems in computing units, and MechatronicUML does not provide optimization capabilities. Furthermore, the tool does not take into consideration safety-related attributes for modeling, analysis, and problem-solving purposes. Additionally, this tool is outdated and lacks an active community or proper documentation.

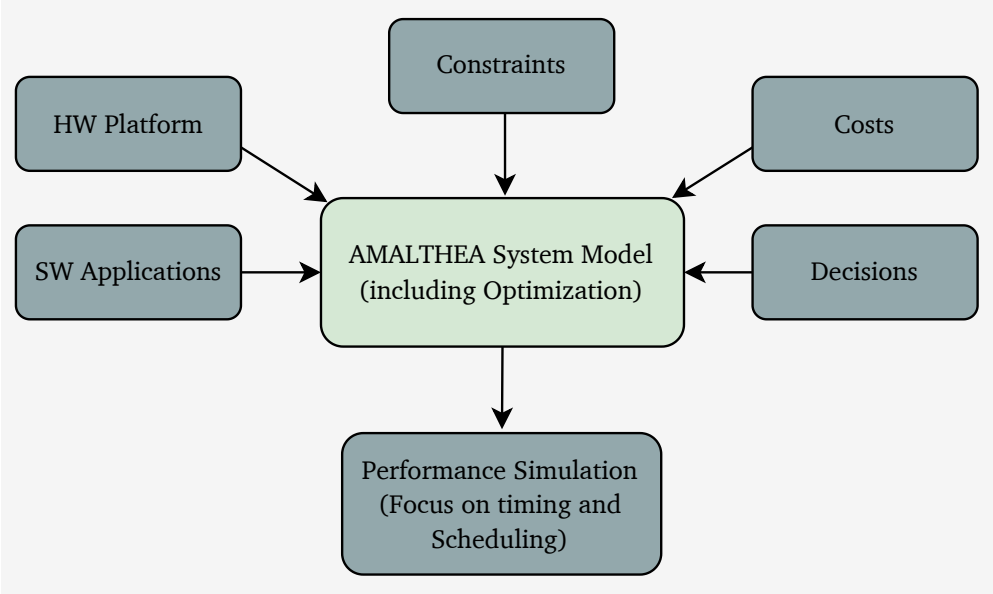


Figure 3.7: The APP4MC architecture [AMK23].

App4MC

As previously mentioned, the automotive industry has increasingly adopted the use of multi-core and many-core systems to manage ADAS and autonomous driving functionalities due to the significant number of applications that require a high level of computational power

for processing. APP4MC is an open-source Eclipse platform that focuses on the performance simulation of scheduling and timing analysis in multi-core platforms. This platform utilizes a model-based development approach, as shown in Figure 3.7. It is dedicated to providing insights into the performance of multi-core systems [HKI15; Höt+17]. The hardware and software components can be modeled, including specific properties like the processor type, connections between various modules, OS schedulers, and task properties such as execution time and deadline. Furthermore, different timing constraints related to the tasks, OS schedulers, and mapping constraints (i.e. the assignment of tasks and schedulers to a specific core) can be defined, visualized, verified, and validated using this tool.

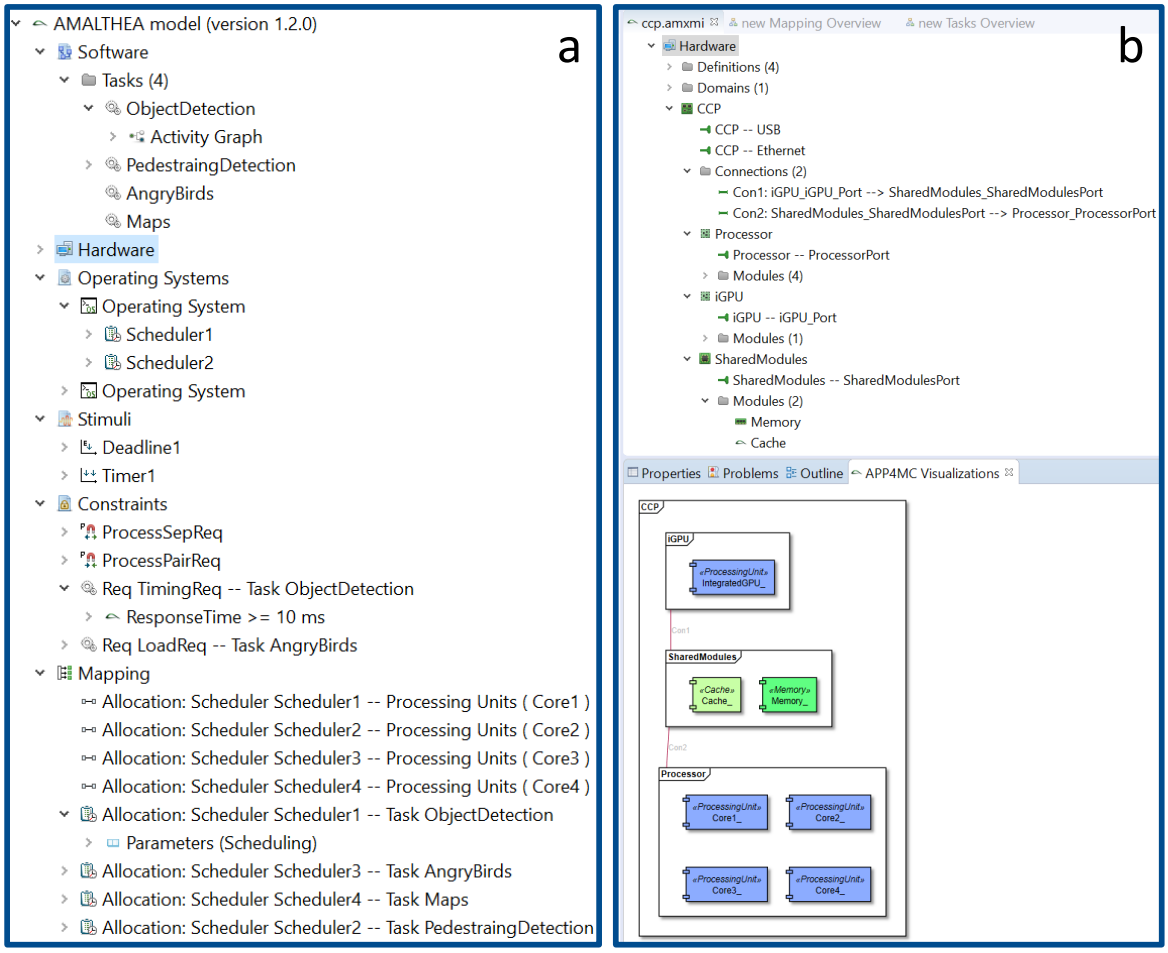


Figure 3.8: A model of the hardware and software system in the automotive industry has been developed using the APP4MC framework. This model encompasses timing and mapping constraints (a) and includes a visual representation of the hardware model (b) [AHK22].

In addition, the hardware and software model, as well as its constraints, can be defined and simulated by utilizing various visual aids such as graphs (e.g., Gantt chart) and tables to present the results. This simulation takes into account different optimization goals, including load balancing, energy consumption, and memory mapping. Figure 3.8 illustrates a modeled automotive system within APP4MC that includes components such as tasks, hardware, operating system, stimuli, constraints, and mapping. For example, the mapping section of the model allows schedulers to be assigned to cores and tasks, as depicted in Figure 3.8 (a). In Figure 3.8 (b), the hardware model, which includes a processor with four cores, an integrated GPU (iGPU), shared modules such as cache and memory, and communication between these components, is visualized using the visualization feature of AAP4MC [AHK22].

However, the APP4MC framework has limitations when it comes to automating the mapping of tasks to hardware elements such as cores, taking into account both safety and non-safety requirements (i.e. the task mapping problem using DSE). While it provides analysis and simulation of task mapping, it does not provide a solution. Moreover, it has limited coverage of safety attributes and optimization goals, and a significant number of E/E architecture elements are not considered in the framework [AHK22].

Autofocus3

Another open-source tool that is based on the Eclipse platform utilizes a model-based development approach to synthesize E/E architectures. This framework supports the modeling of architecture, starting from requirements all the way to the generation of code for embedded systems. Furthermore, the tool is equipped with the capability to simulate the designed model, including the defined constraints, and to perform checks and validate the model. The tool employs a domain-specific modeling language to formalize exploration problems and has the capability to determine end-to-end latency and schedule synthesis through the use of the DSE method. Additionally, optimization algorithms such as binary search have been integrated into the tool to apply defined optimization objectives, such as timing and communication load, to the solution that is explored [Ara+15; VEH14; HF07].

Autofocus3 falls short in addressing the issue of mapping configuration automation on multi-core platforms, taking into consideration both functional and non-functional requirements. There is also a lack of mapping analysis for these platforms. Moreover, it only offers limited support for safety attributes, specifically the ASIL level, and for the optimization goals related to the synthesis of the electronic and electrical architecture [AHK22].

Clafer

A methodology has been developed that combines structural modeling with behavioral formalism to aid in the mapping of feature configurations to component configurations or model templates. This approach, known as Calfer, allows for the integration of feature models (representing variability), component models, and discrete control models (in the form of automata) into a single, unified syntax and semantic framework. The language component is constructed using a combination of first-order logic with quantifiers for modeling structures and linear temporal logic for modeling behavior. This approach engages in DSE with a focus on timing as an attribute, and it allows for model analysis. Additionally, it provides multi-objective optimization for the discovered solution by considering factors such as mass, end-to-end latency, and cost [Juo+18; Ros+19].

Despite its potential, Clafer lacks certain important features for modeling and analysis in embedded systems. Specifically, it does not offer model checking, mapping analysis, or DSE for mapping problems. Additionally, the architectural elements in Clafer are limited and do not include safety-related attributes for model analysis. Furthermore, this approach has not taken into account the modeling and analysis of multi-core computing units. To add, the available information on this approach is not extensive and outdated [AHK22].

AAOL Framework

The model-based approach is a framework for optimizing E/E architecture using a constraint-based methodology and a domain-specific language. The accompanying tool leverages the DSE method to determine the optimal solution for the deployment problem while considering design constraints such as memory capacity, and it implements a multi-objective optimization mechanism with cost and weight as the defined objectives [Kug+15; KP14].

However, this tool has several limitations that hinder its full potential. In terms of architectural elements at the software and hardware level, it lacks a definition of software application parameters and does not address mapping problems in multi-core computing units. Furthermore, it lacks support for model analysis and checking. The tool only considers the ASIL as a safety-related parameter in architecture synthesis and only covers a limited number of optimization objectives. Furthermore, this tool is outdated and lacks comprehensive documentation [AHK22].

Assist

The aviation electronics, or avionics, used aboard airplanes are complex and highly distributed systems. With the constant addition of new functionalities in software, the complexity of these systems continues to increase. The use of multi-core processors enables multiple functions to be performed on a single hardware unit while still meeting all safety requirements, resulting in improved system performance. This represents a major advancement in the aviation industry. A model-based solution, named Assist, is introduced in [HD13] to address mapping problems and optimize deployment of safety-critical applications on avionics hardware in distributed systems within the aviation domain. The approach employs a DSE mechanism to determine optimal mapping solutions while taking into account optimization objectives, such as resource usage, weight, and cost. Furthermore, the Eclipse-based tool has the capability to generate a periodic schedule for real-time tasks and ensure a deterministic timing behavior [HB17].

Despite its benefits, this framework has certain limitations that must be taken into consideration. Firstly, the hardware and software level of the specified architectural elements are quite restricted, and its primary focus is on the aviation domain, rather than the automotive and E/E architecture. Secondly, it only takes into account redundancy as a safety-relevant attribute based on ISO 26262 for mapping purposes, and does not provide support for model checking and analysis. Lastly, the number of defined optimization goals is limited [AHK22].

Deepcompass Framework

The design of embedded systems for multi-processor platforms necessitates the prediction and balance of multiple system quality attributes at an early stage. In [BCK07], a DSE framework is introduced for component-based software systems that provides architects with a means to explore a range of possible design options and evaluate and compare these alternatives. This framework facilitates the creation of multiple alternatives for both software and hardware architectures, and is equipped with the ability to conduct model analysis, mapping analysis, and model validation. Additionally, the tool employs a DSE method to determine the optimal software/hardware architecture options while considering multiple optimization objectives, including cost, throughput, and resource utilization.

However, it lacks any automotive-related elements and multi-core computing unit attributes and does not incorporate a DSE for the mapping problem. It also lacks any considerations for safety-related attributes, and the specified optimization goals are severely limited. Moreover, this open-source tool is outdated and lacks proper documentation [AHK22].

SCALL

The prototype tool described here utilizes an allocation method to provide deployment solutions for system architects during the design phase by leveraging the concept of DSE [ŠC15]. It also supports multi-objective DSE, which encompasses the allocation of heterogeneous components, the optimization of bandwidth, and the minimization of communication cost.

To assist system architects in making complex allocation decisions during the early design stages, the tool incorporates both heuristics and analytic hierarchy process (AHP) approaches.

Despite these advancements, the current approach lacks consideration for model analysis, model checking, or mapping analysis. There is also no coverage of DSE for mapping problems in multi-core computing units and no provisions for safety-related constraints and requirements. Furthermore, the automotive architectural elements have not been clearly defined within this framework, and it lacks any optimization capabilities, rendering it outdated [AHK22].

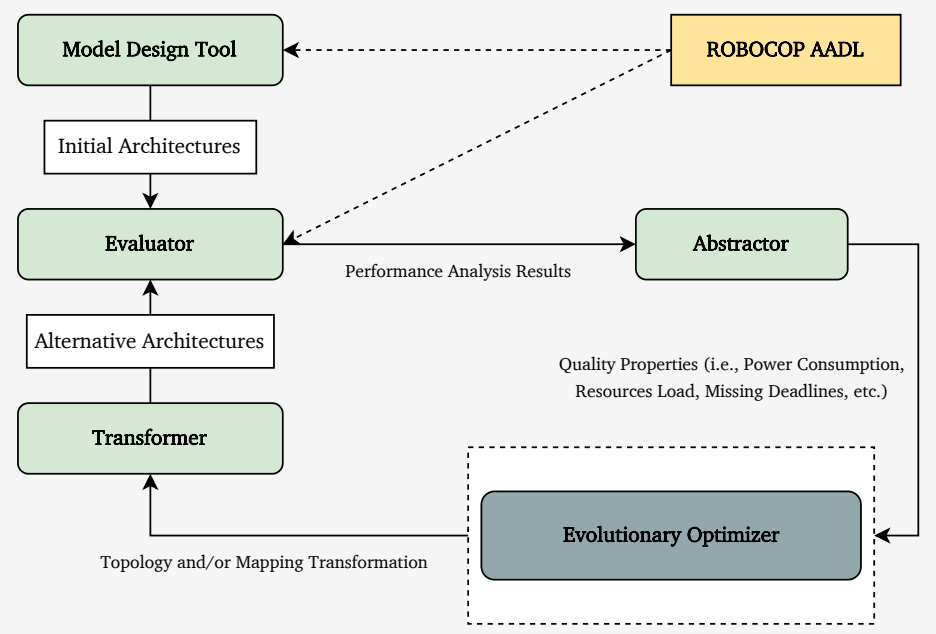


Figure 3.9: The working scheme of the AQOSA toolkit [AHK22].

AQOSA

Authors in [Li+11] present the automated quality-driven optimization of software architecture (AQOSA) toolkit as a solution for improving the design process of software architecture in advanced component-based software development. AQOSA integrates various technologies, including modeling techniques, performance analysis methods, and advanced evolutionary multi-objective optimization algorithms to automate the improvement of non-functional properties in software systems. The toolkit enables the modeling of software components and maps feature configurations to component configurations or model templates, thereby streamlining the design process. Furthermore, the AQOSA framework supports the DSE method, including multi-objective optimization, such as optimization of data flow latency, processor usage, and architecture cost. The architect can easily create the initial architecture using the OSATE tool [OSA21] and the AADL [FGH06], and then import it into the AQOSA framework. With regards to software architecture modeling, AQOSA incorporates the ROBOCOP modeling language [BC+06] (robust open component-based software architecture for configurable devices project) as shown in Figure 3.9.

The AQOSA framework consists of several architectural elements but lacks examination of mapping analysis and DSE for mapping issues. Moreover, it does not consider multi-core platforms and covers no model-checking functionality, with limited optimization targets. The framework also lacks proper documentation and is outdated [AHK22].

SQuAT-Vis

A tool that can be integrated into software architecture optimization methodologies is available to architects, providing them with the ability to analyze the results of their optimization efforts [FH20]. This tool aims to help architects attain an optimal software architecture that meets quality-attribute requirements. The tool employs the DSE approach, incorporating optimization techniques to search for the best design solution for software clustering problems.

On the other hand, the current framework lacks consideration of the mapping problem as a DSE issue, and it does not incorporate model checking or mapping analysis for multi-core processing units. Moreover, the optimization goals are limited to response time and CPU utilization and it does not take into account any safety-related attributes, such as exploration parameters during software clustering.

3.4.2 Overview of Non-commercial Frameworks Analysis

The following tables present the problem being studied, the type of DSE, optimization algorithms and attributes, as well as safety-related attributes for each of the above-mentioned open-source technologies under examination [AHK22].

Table 3.1 showcases the problem type (such as mapping, deployment, model checking, model analysis, etc.) that each technology aimed to address, along with its corresponding problem attributes or DSE items, including resource usage, scheduling, and task response time. The table also highlights the DSE method used in each technology. The table presented below has been adapted from the source [AHK22].

Table 3.1: Problem's type, problem's attributes, and DSE type of the above-mentioned open-source frameworks.

E/E Configurator	Problem	Problem Attributes	Design Space Exploration
ArcheOpterix	Deployment and Mapping	Memory Consumption and Response time	Multi-Objective Optimization and Constraints Satisfaction
PerOpteryx	Software Clustering including Component /Resource Selection, Allocation, and Feature Configuration	Response time	Multi-Objective Optimization
MechatronicUML	Model Checking, Deployment, Formal Analysis of the Requirements, and the Design	Allocation Specification Language	Constraints Satisfaction
APP4MC	Mapping, Resource Management, Performance Simulation, and Validation	Task Response Time, Scheduling, and Partitioning focused on Timing	Multi-Objective Optimization and Constraints Satisfaction
Autofocus3	Model Checking and Deployment	Schedule Synthesis and Latency	Optimization and Constraints Satisfaction
Clafer	Model Analysis and Feature Modelling	Timing	Multi-Objective Optimization and Constraints Satisfaction

Table 3.1. Continued.

OSATE	Model Analysis and Model Checking	Scheduling Analysis, End-to-End Latency, Safety Analysis, Computer Budget Analysis, and Weight Analysis	Constraints Satisfaction excluding DSE method
AAOL	Deployment and Mapping	Memory Usage, CPU Time, Network Bandwidth, and ASIL Level	Multi-Objective Optimization and Constraints Satisfaction
ASSIST	Deployment and Mapping	Redundancy, Scheduling, and Managing Shared Resources	Multi-Objective Optimization and Constraints Satisfaction
Deepcompass Framework	Model Analysis, Model Validation, and Mapping	Task Completion Latency and Missing Deadline in Scheduling	Multi-Objective Optimization and Constraints Satisfaction
SCALL	Software Component Allocation	Heterogeneous Components Allocation, Bandwidth, and Communication Cost	Constraints Satisfaction
AQOSA	Software Clustering and Mapping	Task Latencies, Processor Utilization, and Architecture Cost	Multi-Objective Optimization and Constraints Satisfaction
SQUAT	Software Clustering	Response Time	Multi-Objective Optimization and Constraints Satisfaction

As previously mentioned, there are multiple optimization attributes that must be considered when designing embedded systems for the automotive industry. Table 3.2 focuses on the most crucial optimization attributes. Table 3.2 presents the coverage of optimization parameters achieved by the various approaches discussed before, while also highlighting the optimization algorithms used, such as the genetic algorithm. Furthermore, given the crucial importance of satisfying safety requirements during the configuration of automotive software, Table 3.2 also includes an examination of safety-related attributes taken into consideration by each approach, such as the ASIL level, reliability, FFI, redundancy, and others. Reliability, previously noted, is a crucial optimization parameter and a safety-related aspect of the embedded system, which is calculated based on the failure rate to ensure the reliability of the system [Xie+18].

The cost of a system refers to the expenses incurred in its design, where the goal is to reduce the number of components used. Latency and execution time play a crucial role in improving system performance by reducing the latency during task allocation. Energy consumption, as discussed in Subsection 3.3.2, is also a crucial optimization parameter in embedded systems. The optimal utilization of the system's CPU and memory is achieved by studying CPU utilization and memory usage, which both contribute to enhancing system performance. The following table has been adapted from [AHK22].

Table 3.2: The used optimization algorithms, and the covered optimization and safety-relevant attributes in the above-explained open-source frameworks.

E/E Configurator	Optimization Algorithms	Safety-related Attributes	Optimization Attributes

Table 3.2. Continued.

ArcheOpterix	Genetic Algorithm (GA), ParetoAnt Colony Algorithm (P-ACO), Simulated Annealing (SA), Ayesian Heuristic for Component Deployment optimization (BHCDO), Random Search Algorithm, and Brute-Force Algorithms	Reliability	Cost, data transmission reliability, and communication overhead
PerOpteryx	Genetic Algorithm (GA)	Reliability	Performance, Reliability, and Monetary Cost
MechatronicUML	Not Applicable (N.A.)	N.A.	N.A.
APP4MC	Genetic Algorithm (GA)	Safety parallelization and Traceability	Load Balancing, Energy Consumption, Memory Mapping, and Inter-Core Communication
Autofocus3	Meta Search, e.g. Binary Search	Safety Integrity Level	Timing and Communication Load
Clafer	Guided Improvement Algorithm (GIA) Using Alloy, Z3 SMT, and Choco 3 CSP Solvers	N.A.	Mass, End-to-End Latency, and Cost
OSATE	N.A.	FTA, FMEA, and FHA	N.A.
AAOL	Evolutionary Algorithms	ASIL Level	Cost, Weight
ASSIST	Heuristic approach e.g. Simulated Annealing	Redundancy	Resource Usage, Weight, and Power
Deepcompass Framework	Pareto approach	N.A.	Cost, Throughput, and Resource Utilization
SCALL	Genetic Algorithm (GA)	N.A.	N.A.
AQOSA	Nondominated Sorting Genetic Algorithm, Strength Pareto Evolutionary, and S-metric Selection	N.A.	Data Flow Latency, Architecture Cost, and Processor Usage
SQUAT	Genetic Algorithm (GA)	N.A.	Response Time and CPU Utilization

3.4.3 Commercial Tools for E/E Architecture Configuration

A variety of companies have developed commercial tools that play a crucial role in the design of E/E architectures and the integration and configuration of automotive software [Was+13]. The most noteworthy of these tools will be discussed in detail below [AHK22].

PreeVision

This is a commercially available tool developed for the model-based development of distributed, embedded systems in the automotive industry. This tool offers a wide range of functions for both classic and service-oriented architecture construction and covers all aspects of an E/E system. This includes requirements engineering, AUTOSAR, software and communication design, as well as wiring harness evolution [Für+09]. The integration of a model-based approach simplifies the management of complex tasks, making them both straightforward and manageable. This approach is aligned with established system engineering principles of abstraction, decomposition, and reuse and can serve as a foundational engineering structure. Furthermore, it allows for parallel work to be performed on a shared database from multiple locations, as reported in [AHK22]. The design of E/E architecture platforms for various vehicles is facilitated by this tool [Sch16]. It offers support for the design and evaluation of components, signal routing, checks for model consistency, and functional safety analysis.

MotionWise

This commercial platform offers the ability for users to integrate, test, validate, and schedule a multitude of components and applications, thus simplifying the development, testing, and validation process. Additionally, it helps users to meet the essential safety and mission-critical requirements for both single and multi SoC environments related to automotive platforms. The tool abstracts the hardware and operating system, creating a uniform management environment from heterogeneous elements [TTT21; AHK22].

Volcano Vehicle Systems Architect (VSA)

A commercial platform based on Eclipse technology has been developed to generate a design environment for E/E systems [Ecl23; VSA21]. This platform provides support for both software architecture design, including the definition of software components and compositions, and hardware architecture design, such as defining ECUs, networks, sensors, and actuators. The VSA encompasses the complete AUTOSAR metamodel and its formats, and has the ability to perform automatic code generation [Für+09]. It encompasses a range of capabilities, including mapping analysis, connecting software components to ECUs and system signals, as well as topology and communication design and model validation. Furthermore, VSA supports the exchange of AUTOSAR XML files in both directions for software components and compositions with the use of MATLAB and Simulink [MAT10].

ASCET-DEVELOPER

A commercial software model, specifically designed for the automotive industry, is constructed on the Eclipse platform [ETA21; Wik14]. This software offers support to system architects in developing high-performance, secure, and safe embedded software with minimal overhead. Given its compliance with safety certifications such as ISO26262 ASIL D, it is

well-suited for the development of safety-critical software [ISO18]. This commercial framework provides support for model analysis, which includes graphical and textual specifications as well as model validation. Additionally, it allows for automatic generation of C code from a designed model and includes the capability for unit testing. The framework is designed to be easily integrated into a development process and toolchain through the provision of different interfaces and a standardized file exchange format, which enables toolchain integration.

Autosar Builder

Another commercially available tool for the design, configuration, and simulation of E/E systems that adheres to Autosar standards is the Autosar Builder [Aut21; Aut20]. This tool is built on the Eclipse platform and incorporates the Autosar development environment (Artop). The Autosar Builder framework offers support for the development, verification, and validation of E/E components and the associated embedded software in the automotive industry, as well as for the creation of system descriptions at the application level. Moreover, the system includes graphic visualizations and diagrams to simplify the development of complicated architectures and offers the advantage of effortless integration with third-party tools.

SymTA/S

SymTA/S is a commercial framework designed to evaluate the performance and enhance the efficiency of real-time embedded systems that accommodate diverse architectural designs. This framework serves as a tool for estimating the budget, verifying the scheduling, and improving the performance of processors, ECUs, communication buses, and networks. This tool provides support for the timing and scheduling analysis of distributed embedded architectures and can calculate the worst-case execution time (WCET). It offers a one-of-a-kind capability for end-to-end timing analysis and visualization, and it allows for the design of the system to be optimized by defining multiple optimization objectives, integrating concepts, and determining its reliability and safety through the use of the DSE approach as described in references [Hen+05] and [Ham+04].

ChronSIM/ChronVAL

This commercial framework is a tool for analyzing the timing of automotive systems, leveraging formal verification methods to examine the real-time functionality of safety-critical embedded systems. The DSE approach allows for the end-to-end analysis of distributed functions. The framework also offers graphical validation and simulation of the timing requirements of the system, and supports multi-objective optimization, such as resource utilization and response time objectives, as described in references [Ans+12] and [INC21].

Limitations of Commercial E/E Configurators

While the commercial frameworks discussed above provide various features for E/E architecture synthesis, they also have certain limitations. None of these platforms incorporate mapping analysis, except VSA and ASCET-DEVELOPER, which provide mapping analysis. In addition, none of these frameworks offer a solution for the mapping problem using the DSE method for multi-core computing units or take into account safety-relevant attributes based on ISO 26262 [ISO18]. Furthermore, the above-mentioned frameworks do not have model checking capabilities, except for PreeVision, VSA, and ASCET-DEVELOPER. Only three of these frameworks, ASCET-DEVELOPER, SymTA/S, and ChronSIM/ChronVAL, offer optimization for their synthesis results, but with a limited number of objectives. None of these

platforms take into account a comprehensive range of architectural elements at both the hardware and software levels in the E/E architecture configuration [AHK22].

3.4.4 Overview of Commercial Tools Analysis

As discussed before in the open-source technologies, there are multiple attributes that must be considered when designing embedded systems. Table 3.3 provides a summary of the most essential features related to the design of E/E systems integrated into the commercial tools discussed above. Moreover, it states the DSE coverage and type for the above-discussed tools.

Table 3.3: Features and DSE type of the above-presented commercial tools.

Commercial E/E Configurator	Features	Design Space Exploration
PreeVision	Requirements Engineering, Software Design, Wiring Harness and Communication Design, Functional Safety Analysis, Model Consistency Check, and Signal Routing	N.A.
Motionwise	Abstraction Tool supporting hardware and OS Validation, Test, and Scheduling of Components and Applications	Not Explained
Volcano Vehicle Systems Architect (VSA)	Eclipse-based Platform, software/hardware Architecture Design, Mapping Analysis, Topology and Communication Design, and Model Validation	N.A.
ASCET-DEVELOPER	Eclipse-based Platform, Model Analysis including Graphical and Textual Specifications, and Model Validation	N.A.
Autosar Builder	Eclipse-based Platform, Design, Configuration, and Simulation of E/E Systems, Verification and Validation of E/E Components	N.A.
SymTA/S	Budgeting, Scheduling Verification, WCET calculation and optimization for processors, ECUs, communication buses, and networks and End-to-end Timing Analysis	Multi-Objective Optimization and Constraints Satisfaction

Table 3.3. Continued.

ChronSIM/ ChronVAL	Timing analysis of Automotive Systems, Analysis of Real-time Capability of Safety-critical Embedded Systems using Formal Verification method, Graphical Validation and Simulation, End-to-end Analysis	Multi-Objective Optimization and Constraints Satisfaction

3.5 Summary & Discussion

Various approaches, methods, and software frameworks were explained above while taking our research questions and motivation into account.

In Subsection 3.1.1, the message routing for automotive networks is discussed, and relevant previous works were reviewed. It aims to improve upon existing message routing constraints by extending them to automatically create homogeneous redundant and multi-cast routings, using a model-based development approach. This approach involves using models to represent the behavior and communication of various components in the system, and using those models to automatically generate code or configurations. Additionally, changes are made to the constraints to enable a single-step solving approach. This means that the routing can be determined in a single step, rather than requiring multiple iterations to refine the solution.

Similarly, Subsection 3.1.2 presents a review of related works focused on synthesizing time-triggered schedules for a vehicle's network including applications mapped on different nodes and communications messages routing over network's links. We draw on existing research to support the introduced approach in this thesis, and the time-triggered conditions introduced in one of the reviewed works are extended to align with the proposed single-step solving approach in this dissertation. In addition, path and message dependencies constraints are introduced that are tailored to their automation message routing creation approach. These constraints support our methodology of solving the problem in a single step, which aims to streamline the process of creating time-triggered schedules for processes running on different car's ECUs/HPCUs and communication messages.

Section 3.2 discusses the synthesis of software architecture, particularly in the context of E/E systems. Furthermore, it reviews studies that consider safety requirements specific to the automotive industry, such as redundancy and reliability, during the synthesis process. These studies provide valuable insights into the necessary safety conditions that must be met according to the ISO 26262 standard [ISO18], which sets guidelines for the functional safety of road vehicles. In the automotive industry, ensuring the safety of E/E systems is crucial, as any failure in these systems can potentially lead to catastrophic consequences. Thus, the synthesis process must take into account various safety-related requirements to minimize the likelihood of failures. Some of the key safety requirements that must be addressed during synthesis include identifying potential hazards, assessing their severity, and determining the probability of their occurrence. The studies discussed in this section provide an overview of the various methods and techniques that can be used to meet these safety requirements.

Section 3.4 covers a comprehensive review of both non-commercial and commercial software frameworks/tools that have been developed to support configuration and synthesis for E/E architectures and systems. The focus has been on analyzing the functionalities and capa-

bilities of these tools to determine their suitability for specific problems types and attributes. This section also looked at the DSE methods, optimization attributes and algorithms, as well as safety-relevant features of these tools, to identify their strengths and limitations while considering our research questions and motivation. Overall, it is believed that the review of the tools presented in this section provides a comprehensive overview of the current state of the art in E/E system design and synthesis. This review is useful to the exact contributions of this thesis, as it provides valuable insights into the tools. Based on this analysis, the following limitations are addressed compared to the contributions of this thesis.

None of the aforementioned frameworks/tools provide a comprehensive solution for managing complex distributed systems. In particular, these tools do not provide:

- Automated mapping and time-triggered scheduling for assigned applications, including their threads running on different control nodes. This feature is essential for managing complex systems with multiple nodes and applications that require coordination.
- Automatic message routing creation, comprising single, redundant, multi-cast, and homogeneous redundant paths, and the calculation of time-triggered schedules for communication messages supporting path and message dependencies while applying multi-objective optimization. This is critical for guaranteeing that messages are delivered in a timely and efficient manner, while also ensuring redundancy and fault tolerance in the mixed-critical system.
- Capable of modeling virtualization techniques like hypervisors in the context of vehicle E/E architecture and automating the allocation of hardware and software resources to partitions within each hypervisor.
- A single-step solving approach to determine the last three items in a single step, avoiding multiple iterations to refine the solution and respecting interrelations between defined constraints decisions. This feature is essential for optimizing complex distributed systems and ensuring that all constraints are met efficiently.
- A performance evaluation through deploying solutions created by frameworks/tools on a real automotive-related hardware platform. This is important for ensuring that the solutions are viable in real-world scenarios and can handle the demands of actual hardware.
- Different optimization goals and boundary limits such as response time, end-to-end latency, link occupation rate, bandwidth usage, reliability, cost, and resource usage in one package. This feature is crucial for allowing system designers to optimize their systems according to a variety of different criteria and goals.
- Various safety constraints and optimization objectives, comprising ASIL level, FFI, reliability (using MTTF and failure rate), redundancy, and homogeneous redundancy in one package. This is important for ensuring that the system is safe and reliable and that it meets all required safety standards and regulations.
- An approach to identify design errors in case of having violations in the constraint set included in the system model after the solving step. This feature is important for identifying and addressing errors in the system design within a reasonable amount of time, ensuring that the system is optimized and satisfies all necessary constraints and requirements.

4

Methodology

4.1 Framework Architecture

In order to tackle the difficulties and overcome the obstacles and limitations identified in Section 3.5 and answering the defined research questions in Chapter 1, it is essential to establish a set of requirements that can guide the development of modeling frameworks used for the synthesis of E/E architectures and systems.

- Easing E/E architecture design: various hardware/software components, safety requirements, and other properties need to be taken into consideration when designing E/E systems and vehicle network topology. To accomplish this, graphical and textual modeling elements are necessary. These modeling elements enable E/E architectures/systems to be easily modeled using drag and drop features. This feature allows designers to create and manipulate components with ease and flexibility. Furthermore, the drag and drop feature makes it easy to create and edit models, thus reducing the time and effort required to design an E/E architecture.

- Flexibility and adaptability: The modeling framework should be flexible enough to accommodate various design requirements, and adaptable to changes in the system's specifications or design goals.

- Scalability: The framework should be scalable to handle systems of different sizes and complexities, from simple components to large-scale systems.

- Quick access to components, elements, and properties: In the field of automotive engineering, designing an E/E system or car network topology involves creating a complex architecture that includes a wide range of interconnected components, elements, and properties. Once this architecture has been modeled, it is essential to have a way to quickly and easily access all of these elements and their associated properties. By having quick access to these elements and properties, designers/system integrators can streamline their programming efforts and avoid the time-consuming process of manually coding each component and property. This approach also allows for easier modification and testing of the system, as designers can quickly make changes to specific elements and properties without having to rewrite large portions of code.

- Facilitating E/E architecture synthesis: As vehicles continue to become more complex and require more computational power, there is a growing need for efficient methods

to automate the process of mapping, communication message routing, and scheduling. To achieve this, constraint-based system models can be employed. These models can automate the mapping and scheduling for E/E systems including multi-core architectures, and can also transform safety requirements into constraints that can be easily solved and utilized for E/E configurations. This not only reduces the complexity of the process, but also significantly decreases the time required for synthesis. In addition to these benefits, facilitating E/E architecture synthesis can also lead to improved vehicle safety and reliability. By automating the process, errors and inconsistencies can be minimized, ensuring that the resulting E/E systems fulfill all safety requirements and operate as intended.

- Optimization: It is a critical consideration for any E/E system design, and system integrators should be given the freedom to optimize their designs based on various criteria and objectives. One of the most significant factors in optimization is cost, as the cost of an E/E system can be a determining factor in whether a project is feasible or not. By allowing system integrators to optimize their designs based on cost, they can create efficient and cost-effective solutions that meet the required performance and reliability standards while keeping the project within budget. In addition to cost, redundancy, reliability, FFI, timing, etc.

- Synthesis time: Solving time is a critical parameter for reducing the effort and time required for E/E configuration. Therefore, it is important to consider approaches to speed up the solving process, such as implementing a single-step solving method. This method can help avoid the need for multiple iterations to clarify a solution, while still respecting the interrelations between specified constraints and decisions. In addition, it is important to ensure that any optimization techniques used still produce reliable and high-quality solutions, as compromising on solution quality can result in increased costs or potential safety risks.

- Unsatisfiable system model: Infeasible solutions are common when solving a constraint system, and addressing this issue can be complex and time-consuming, especially when dealing with a large number of constraints or conditions. To address this challenge, a methodology for identifying violated constraints can assist E/E system architects in identifying the source of conflicts much earlier in the design process, thereby enabling them to navigate towards a feasible solution more efficiently.

Considering the first research question specified in Chapter 1, the architecture of the proposed framework, which is called E/E Designer, consists of three main parts. As Figure 4.1 illustrates, the left sidebar of the figure represents the framework's inputs, which can be provided by an E/E system integrator/architect. These inputs include various properties and requirements related to hardware and software components, the mapping process, communication message routing, time-triggered scheduling, and safety. More specifically, they consist of ASIL level, turbo boost technology feature, communication and process/thread execution times and periods, cost considerations, forced mapping, link type, node type, sender and receiver information for communication messages, the desired network topology (e.g., full-mesh or other topologies, including the number of required nodes and links), MTTF, failure rate, etc. Moreover, a graphical E/E system modeler can select different optimization objectives and boundary goals. They can also design a desired E/E architecture or a car network topology using drag-and-drop functionalities.

The middle part of Figure 4.1 primarily represents the back-end of the framework, where all mathematical formulations and calculations are performed. This part remains hidden from

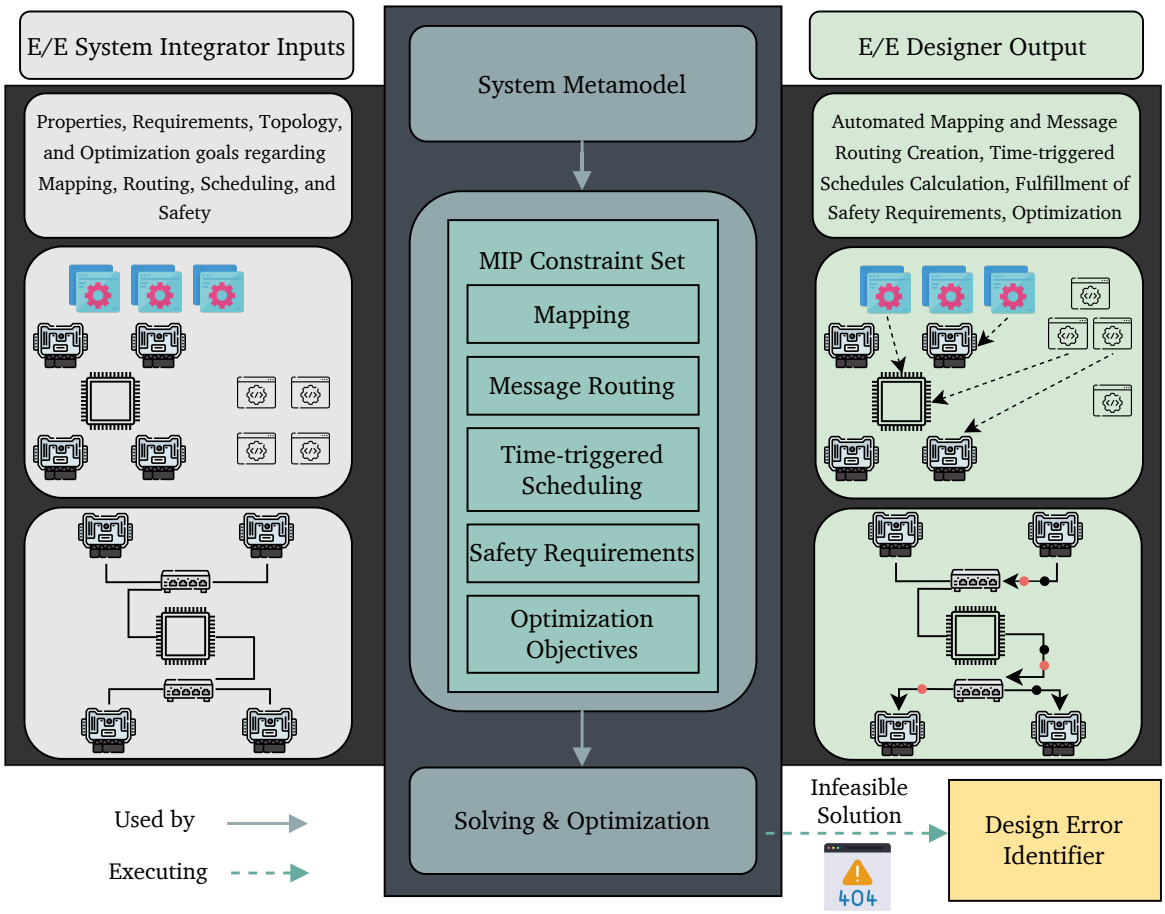


Figure 4.1: The architecture of the proposed model-based framework. The left and right columns, representing E/E System Integrator Inputs and the E/E Designer Output, respectively, constitute the frontend. The middle box denotes the backend of the tool. In the framework's frontend, an E/E architecture is modeled by an E/E system architect. This modeling includes defining requirements, properties, and addressing various problems. The modeled E/E architecture is transformed into MIP formulations using the MDD approach. These formulations are then solved and optimized in the tool's backend. Finally, the optimal solution is visualized in the frontend of the framework.

the user for the sake of simplicity. As observed, the introduced approach utilizes an object-oriented metamodel following the model-driven development (MDD) approach [Sel03] which is explained in the following section. This metamodel serves as the foundation for graphical modeling, which is used by the integrator or modeler to create graphical model instances. Using a formal system metamodel, the graphical model instances, which encompass all selected requirements and properties, are transformed into a set of mixed-integer programming (MIP) constraints within the framework of linear programming (LP) (indicated by the green box in Figure 4.1) [Van+20; FL05].

This constraint set includes conditions for mapping (which can be applied to multi-core architecture and other hardware/software components), message routing for car network topologies, time-triggered scheduling for application threads and communication tasks, safety and non-safety requirements, as well as optimization objectives and boundary goals (refer to Figure 4.1). In the final step, depicted in the middle part of Figure 4.1, a MIP solver is employed to solve and optimize the specified set of constraints while considering the defined optimization goals.

Finally, after the solving step, the E/E Designer framework provides automated mapping

or resource allocation (e.g., automatically assigning various application threads to different cores of a multi-core HPCU) and creates message routes (i.e., finding the correct path from the sender of a communication message to its receiver) for a modeled E/E architecture or network topology that satisfies the predetermined requirements. Furthermore, the E/E Designer computes time-triggered schedules for the assigned running application threads, for example, on a multi-core HPCU, and manages communication task routing over the car's network links. The resulting solution, as the tool's output, fulfills all requirements selected by the E/E system modeler and is optimized based on the chosen optimization goals.

Moreover, in the event of a feasible solution, an error identification approach (depicted by the yellow box in Figure 4.1) is invoked to locate the constraints most likely responsible for conflicts. This approach significantly assists the task of identifying violated constraints, which can be an intricate and time-intensive process, especially when dealing with a substantial number of conditions. This pertains to the second research question outlined in Chapter 1.

4.1.1 Model-Driven Development

Model-driven development (MDD) is a software development methodology that has gained popularity in recent years due to its ability to increase development efficiency and reduce errors. MDD is based on the idea that software development can be greatly improved through the use of visual models and pre-constructed application components. This approach allows developers to focus on the high-level design of an application rather than the low-level details of programming. MDD provides a structured and standardized approach to software development, which makes it easier for developers to create complex applications quickly and accurately. Instead of writing code from scratch, developers use graphical models to describe the behavior and structure of the application. These models can then be automatically transformed into executable code using specialized tools [Sel03; HT06].

Advantages of model-based development include:

- Increased efficiency: By using models to represent the system, developers can analyze and design the system more quickly and accurately than if they had to do so manually. This can lead to shorter development cycles and faster time to market.
- Improved quality: Models can help developers identify and fix problems early in the development process, reducing the risk of defects in the final product.
- Enhanced communication: Models can be used to communicate complex ideas and designs to stakeholders, making it easier for everyone involved in the development process to understand the system.
- Reusability: Models can be reused and repurposed for different projects, which can save time and effort in the development process.

With MDD, software developers can create sophisticated applications through the use of pre-constructed application components and graphical models. MDD's primary objective is to automate challenging programming tasks. In addition, this method can speed up the redeployment, rebuild, and testing procedures, especially when developing multiple applications, compared to the traditional approach [AK03]. Considering all the aforementioned advantages of MDD, this approach has been employed in the presented framework by developing a system metamodel to serve as the foundation for the graphical modeler. There are various types of MDD tools available for creating models for software design purposes. In this work, an open-source graphic modeling tool that employs unified modeling language (UML) [Med+02] from *Eclipse Foundation* was utilized [AFK21a; AMK23; Ecl23].

4.1.2 Object-oriented Metamodel

A metamodel is a model that defines the structure, elements, and relationships of other models. An object-oriented metamodel is a fundamental concept that defines how UML itself is structured and represented. It essentially provides a set of rules and constructs for defining UML elements and their relationships. This typically involves representing and defining modeling elements and relationships using object-oriented concepts such as classes, objects, attributes, methods, and inheritance. Such a metamodel provides a structured approach to define and organize the elements and semantics of a modeling language, simplifying the process of creating, comprehending, and manipulating models in a consistent and modular manner [HT06; Med+02; AFK21a].

A metamodel is developed for the proposed tool, as depicted in Figure A.1, building the foundation for the graphical modeler. The developed metamodel, which employs UML to describe another model as an instance, comprises 39 elements. This metamodel includes 33 classes, with 14 of them serving as the primary classes within the system model. These 14 key elements encompass Node, Application, Link, Data, Data_in, Data_out, Process, Task, Mapping, ECU, Processor, Core, Hypervisor, and Settings. Furthermore, the metamodel encompasses various attribute types, including ASIL level, memory, optimization goal, link, and node types. It also includes a data type used by the system model's elements for the Gurobi solver, which serves as a MIP solving engine [Gur22]. In the following paragraph, the relationships between primary classes, as illustrated in Figure A.1, are explained

According to Figure A.1, the Node class encompasses several attributes and types. Node types include ECU, HPCU, gateway, and network switch. Each Node can have *one-to-many* links, *zero-to-many* applications as both senders and receivers, and *zero-to-many* mapping elements. Mapping class refers to the action of mapping or resource allocation. The Application class can have *zero-to-many* processes and maintains a *one-to-one* relationship with the Mapping class, signifying that each application has a unique mapping attribute. Each process is associated with only one application, and each application process can send/receive only one communication message indicated as Data class in Figure A.1. Each communication message consists of a *one-to-many* relationship with communication tasks (depicted in Figure A.1 under as Task), meaning it can be received by one or multiple application processes but can only be sent by one process. Furthermore, each communication message exhibits a *one-to-many* relationship with outgoing and incoming messages, denoted as Data_Out and Data_In, respectively, in accordance with the metamodel presented in Figure A.1. Furthermore, there exists a bidirectional reference between each Data_In and Data_Out [AFK21a]. A communication link has two *one-to-one* relations with the Node, serving as the starting and finishing points of each link. Additionally, each Link can have *one-to-many* communication tasks and outgoing and incoming messages. It is important to note that each communication task is associated with only one link. The Settings class includes several attributes related to different requirements, such as reliability, message routing, boundary goals, optimization objectives, and visualization-related options. This incorporates the display of desired mapping solutions for applications and paths for communication messages. Each ECU, as a type of node, can own *zero-to-many* processors (where each processor can have *zero-to-many* cores) and memories. In the case of an HPCU, the same relationships apply. Moreover, an HPCU can own *zero-to-many* GPUs, hypervisors, and partitions [AFK21a; AMK23].

4.1.3 Constraint Set

Constraint Satisfaction Problem

Constraint satisfaction problem or CSP is a type of problem in computer science, artificial intelligence, and mathematics that involves finding a solution to a set of constraints or conditions. In a CSP, the problem is typically defined by a set of variables, each with a corresponding set of possible values or domains and a set of constraints that must be satisfied by the variables' values. The goal of the problem is to discover a consistent assignment of values to the variables that fulfills all the constraints. CSPs have numerous applications, including scheduling, resource allocation, planning, and design. They find use in various fields, such as computer science, operations research, artificial intelligence, and engineering. Solving a CSP can be complex due to the potentially vast number of solutions and the challenge of satisfying all constraints. Many algorithms and techniques have been developed to efficiently solve CSPs, including backtracking, forward checking, and constraint propagation [PM10; Rus10].

A constraint satisfaction problem includes three components, X, D, and C where X is a set of variables, $\{X_1, ..., X_n\}$, D is a set of domains, $\{D_1, ..., D_n\}$, one for each variable, and C is a set of constraints that specify allowable combinations of values [Rus10].

ILP and LP

In integer linear programming (ILP) or integer programming (IP), all the decision variables are required to be integers, whereas in linear programming (LP), all the decision variables are continuous. ILP is a type of optimization problem where the objective function and constraints are linear, but some or all of the decision variables are restricted to be integers [Van+20]. While LP can be efficiently solved in the worst case, IP problems can pose significant challenges in many practical situations. This is particularly true for problems with bounded variables, where the number of possible solutions is limited. As a result, integer programming problems are often classified as NP-hard. NP-hard problems are a class of computational problems that are at least as hard as the hardest problems in the complexity class NP. The abbreviation NP stands for nondeterministic polynomial time, which refers to the set of decision problems that can be solved by a nondeterministic Turing machine in polynomial time. An NP-hard problem is a problem that is at least as difficult as any problem in NP. In other words, if an NP-hard problem can be solved in polynomial time, then every problem in NP can also be solved in polynomial time. This is because an NP-hard problem can be reduced to any problem in NP in polynomial time. There are many important computational problems that are known to be NP-hard, including the traveling salesman problem, the knapsack problem, and the Boolean satisfiability problem. These problems have important applications in fields such as operations research, computer science, and artificial intelligence [For09].

One special case of integer programming is 0-1 integer programming, also known as binary integer programming (BIP). In this type of problem, variables are restricted to take on values of either 0 or 1 rather than arbitrary integers. Despite this restriction, 0-1 integer programming is still classified as NP-hard and is known to be one of Karp's 21 NP-complete problems. The decision version of 0-1 integer programming involves determining whether there exists a feasible solution that satisfies a given set of constraints. This decision problem is known to be NP-complete, meaning that it is at least as hard as any other NP problem and cannot be solved in polynomial time. However, despite the theoretical difficulty of solving 0-1 integer programming problems, there exist powerful algorithms and techniques that can be used to find good approximate solutions in practice [Van+20; GJS74].

MIP and MILP

MIP is a more general type of optimization problem, where some of the decision variables can be continuous, and others can be restricted to be integers. It can contain both integer

and continuous variables. MIP is also able to include quadratic constraints; in other words, it combines the features of both LP and ILP. MIP problems can be more challenging to solve than LP or ILP problems, as including integer and continuous variables can make the problem non-convex and non-linear, leading to a more complex solution process.

Mixed-integer linear programming (MILP) is a subset of MIP where all variables, whether continuous or integer, have linear relationships in both the objective function and constraints. In MILP, the objective function and constraints are linear, but some or all of the decision variables can be integers, binary (0 or 1), or a combination of both. MIP and MILP are widely used in various fields, including operations research, engineering, finance, and economics. They are a powerful tool for solving many real-world optimization problems, such as production planning, scheduling, resource allocation, transportation planning, and portfolio optimization. MIP and MILP can help decision-makers to make better decisions by optimizing a wide range of objective functions, such as minimizing cost, maximizing profit, or maximizing efficiency [Jün+09; Van+20]. These problems are generally also NP-hard because they are even more general than ILP programs. Mixed-integer linear programs are problems that can be expressed in canonical form as follows:

$$min_{x,y} \quad c^T x + h^T y$$

subject to:

$$Ax + Gy \geq b$$

$$(x, y) \in \mathbb{R}^n_+ \times \mathbb{Z}^p_+$$

where A and G are $m \times n$ and $m \times p$ matrices, respectively. Moreover, b, c, and h represent m-, n-, and p-dimensional vectors, respectively. In the last constraint, $\mathbb{R}^n_+$ is the n-dimensional space of all non-negative real numbers ($\mathbb{R}^n_+ = \{x \in \mathbb{R}^n : x \geq 0\}$) and $\mathbb{Z}^p_+$ is the p-dimensional space of all non-negative integer numbers ($\mathbb{Z}^n_+ = \{y \in \mathbb{Z}^n : y \geq 0\}$). Accordingly, a set of feasible solutions can be defined below as X:

$$X = \{(x, y) \in \mathbb{R}^n_+ \times \mathbb{Z}^p_+ : Ax + Gy \geq b\}$$

In the following, a simple mathematical example of MILP is presented :

$$max: \quad x + y$$

subject to:

$$C_1 : -2x + 2y \geq 1$$

$$C_2 : 8x - 10y \leq 13$$

$$C_3 : x, y \geq 0$$

$$C_4 : x, y \in \mathbb{R}$$

In this example, x and y are decision variables for which the optimal values are sought. The objective function to be maximized is $x + y$. The first two constraints (C_1 and C_2) are linear inequalities that restrict the possible values of x and y. The third and fourth constraints (C_3 and C_4) specify that x and y must be greater than zero, and they belong to the set of real numbers, respectively. One approach to solving this MILP problem is to use a branch and bound algorithm. This algorithm works by branching on the integer variables, creating subproblems, each of which is a smaller version of the original problem. The algorithm solves each subproblem by applying linear programming techniques and then determines whether the solution is integer-feasible. If the solution is not integer-feasible, the algorithm branches further and repeats the process until an integer-feasible solution is found [LW66]. Therefore,

the optimal solution of its LP relaxation is $(x, y) = (4, 4.5)$ with an objective value of 8.5 using the branch and bound algorithm.

One of the key challenges of MIP and MILP is to find an optimal solution within a reasonable time frame. These problems are generally more complex and computationally intensive than LP or IP problems due to the inclusion of integer variables. Solving a MIP problem involves exploring a large solution space and requires a combination of mathematical algorithms and computational methods. To solve MIP problems, various software packages are available, such as Gurobi, CPLEX, and GLPK [Gur22; Cpl09; GNU00]. These software packages employ advanced algorithms and optimization techniques, including branch-and-bound, cutting planes, and heuristics, to discover an optimal solution or a near-optimal solution within a reasonable amount of time.

Based on the explanation provided above, MIP is employed to formulate the problems and requirements presented in this thesis. These encompass tasks such as mapping or resource allocation, message routing, time-triggered scheduling, addressing message and path dependencies, and ensuring safety-related conditions. These requirements are then transformed into a set of constraints. To solve these constraints, the Gurobi MILP solver [Gur22] is utilized as referenced in [AFK21a; AMK23]. The forthcoming sections will detail the formulation of these constraints and provide additional insights into their specifics.

4.1.4 Optimization

To apply MIP to the synthesis of E/E architecture, the problem must first be formulated as a mathematical model, as mentioned before. This involves identifying the variables, constraints, and optimization goals of the problem. Once the model is formulated, the optimization goals can be specified [AFK21a; AFK21b]. In the automotive domain, a system model can be used to optimize the performance of various components and systems in a vehicle. A system model is a representation of a complex system that can be used to simulate and analyze its behavior under different conditions. Optimization involves finding the best possible set of inputs or parameters to achieve a specific goal or objective. These goals may include minimizing the cost of the architecture, maximizing its reliability, or optimizing its performance. Different optimization goals will result in different optimal solutions, so it is essential to choose the goals carefully based on the specific requirements of the E/E architecture. For instance, cost optimization can be used to reduce the cost of production and operation of a vehicle by optimizing the design of the components and systems to reduce material and labor costs, and improve efficiency.

Once the optimization goals are specified, the MIP solver can be used to solve the optimization problem and generate an optimal solution. The solver searches for the values of the variables that minimize or maximize the optimization goals while satisfying the constraints of the model. In the introduced framework, several optimization objectives are integrated, including cost reduction (CR), end-to-end latency, response time, resource utilization (RU), load balancing in-vehicle communication network, and reliability. In addition, the introduced tool supports multi-objective optimization as well. The details of all these objectives will be described in Section 4.4.

4.1.5 Design Error Identifier

As depicted earlier, based on the yellow box in Figure 4.1, it is common to encounter infeasible solutions while solving a constraint system. Tackling this problem can be difficult and time-consuming, particularly with many constraints or conditions involved. To overcome this

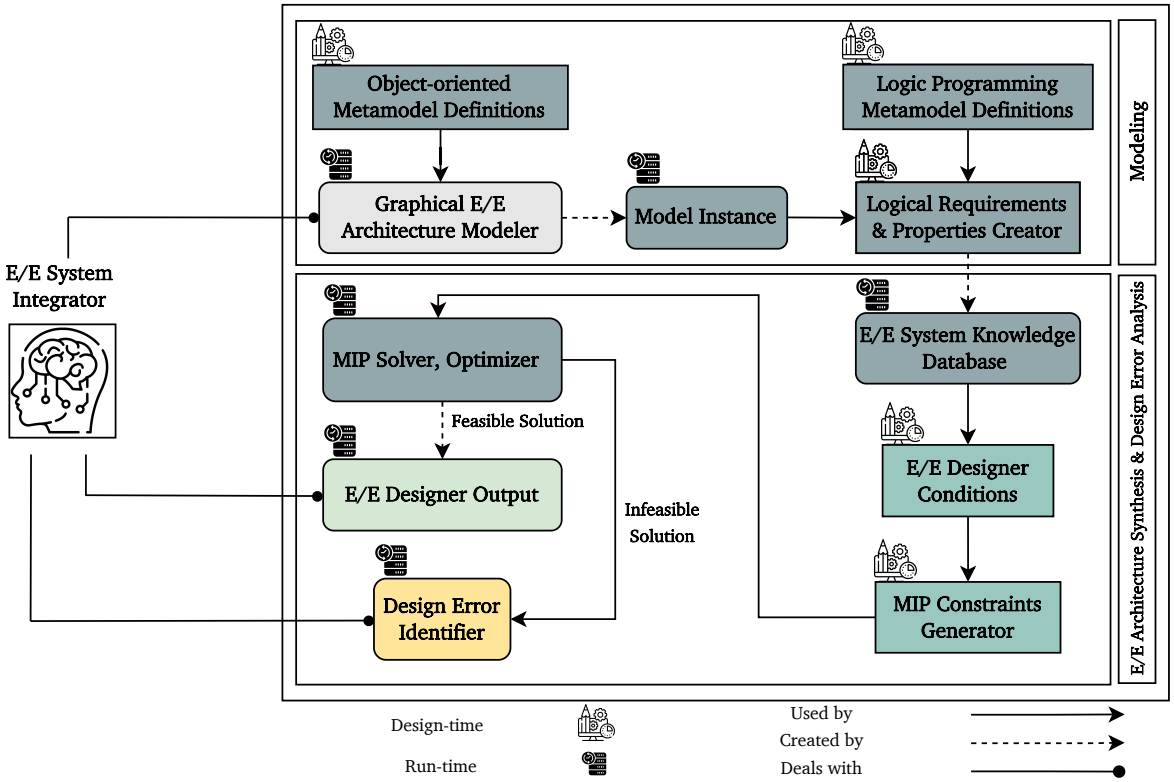


Figure 4.2: The overview of the framework, including modeling, synthesis, and design error analysis parts.

challenge, an approach to identify violated constraints can help E/E system architects identify conflicts' sources early in the design process, allowing them to navigate towards a feasible solution more efficiently. Therefore, an approach is introduced called the design error approach, which uses infeasible inconsistent subsystem (IIS) and minimal unsatisfiable cores (MUC). A MUC is a subset of the negation of the constraint that is itself unsatisfiable, i.e., no assignment of variables can make all of the clauses in the MUC true. The MUC gives information about which parts of the negation of the constraint are causing the violation. IIS is also similar to MUC. This information can pinpoint the violated part of the original constraint used in the approach. Specifically, the negation of the MUC can be taken and intersected with the original constraint to obtain a minimal set of clauses responsible for the violation [LM04; DHN06]. In Chapter 6, this approach will be discussed in detail.

4.1.6 An Overview of Framework Architecture

In Figure 4.2, the modeling approach involves an object-oriented metamodel, which serves as the foundation for a graphical E/E architecture modeler. This modeler enables the creation of visual representations of the model instances. E/E system integrators utilize the graphical modeler, which offers drag-and-drop functionality, to design car E/E systems. These systems consist of both hardware and software components, along with their corresponding physical properties. The user can specify timing requirements and choose from a range of safety conditions, boundary goals, and optimization objectives. For instance, it is possible to define an application that includes a thread with a specific period and execution time, designating it as a safety-critical application. This designation implies that redundancy requirements must be met during mapping and routing for this particular application.

A formal metamodel based on logic programming is used to transform graphical model

instances into a knowledge database for E/E systems. The E/E system database is analyzed to extract conditions and requirements, which are then converted into mathematical formulations called MIP constraints. These constraints cover mapping problems, message routing, time-triggered scheduling, boundary considerations, and optimization objectives. The constraints are solved using an MIP solver and optimizer to determine if a feasible solution exists for the designed model. If a feasible solution is found, it is presented as output for the E/E system integrator. However, if conflicts in the constraint set prevent the tool from solving the system model, the design error identifier approach is activated. This approach identifies the most critical conditions causing the violation and presents the findings to the E/E system integrator. In Figure 4.2, run-time and design-time states for each step is visualized.

4.2 Framework System Model

The system model of the E/E Designer framework is a distributed system comprising multiple nodes and links. It can be depicted as a graph denoted as $G(N, L)$, where the vertices N represent vehicle nodes (such as HPCUs, ECUs, gateways, and network switches), and the edges L represent full-duplex links (like Ethernet, FlexRay, and time-triggered CAN bus). Gateways and switches function as intermediate or networking nodes, facilitating data transfer between different points. On the other hand, HPCUs and ECUs can function as control nodes, responsible for executing applications and sending or receiving data. Control nodes can also act as intermediate nodes [AMK23].

Furthermore, each HPCU may consist of sub-components like processors, cores, GPUs, and memory. We use the notation n^{cz} to differentiate between node types to indicate a control node and n^{nz} to represent a networking node. A single core belonging to a control node, such as an HPCU core, is denoted as n^{cz_c}. A full-duplex link connecting two nodes n_a and n_b is represented by $l_{a,b} \in L$ and $l_{b,a} \in L$, signifying directed links from n_a to n_b and from n_b to n_a, respectively. The E/E Designer tool's system model encompasses several components, which are explained as follows [AMK23; AFK21a].

4.2.1 Application Thread

In the proposed framework's system model, there can be multiple processes/threads for each application that runs on a control node. An application thread refers to a thread running on a control node. An assumption is made that these threads are periodic and represented by a tuple $t_i = \{t_i.p, t_i.st, t_i.e\}$, which consists of three elements: $t_i.p$, $t_i.st$, and $t_i.e$ represent thread's period, starting time, and execution time, respectively. The details of each element are described in Table A.1. In the example provided in Figure 4.3, there are four applications, each containing a single thread, running on n_1^{cz} [AMK23].

4.2.2 Communication Task

A communication task c_i can be specified as $c_i = \{c_i.p, c_i.st, c_i.fl\}$. $c_i.p$ and $c_i.st$ designate the period and starting time of a communication task, respectively. Besides, each communication task is represented as a unique frame with frame length $c_i.fl$ (see Table I). As explained in Chapter 2 and based on Figures 4.4 (a) and 4.5, a route is defined as a path from a sender to a receiver transmitting c_i. For example, in Figure 4.5, there are two communication tasks

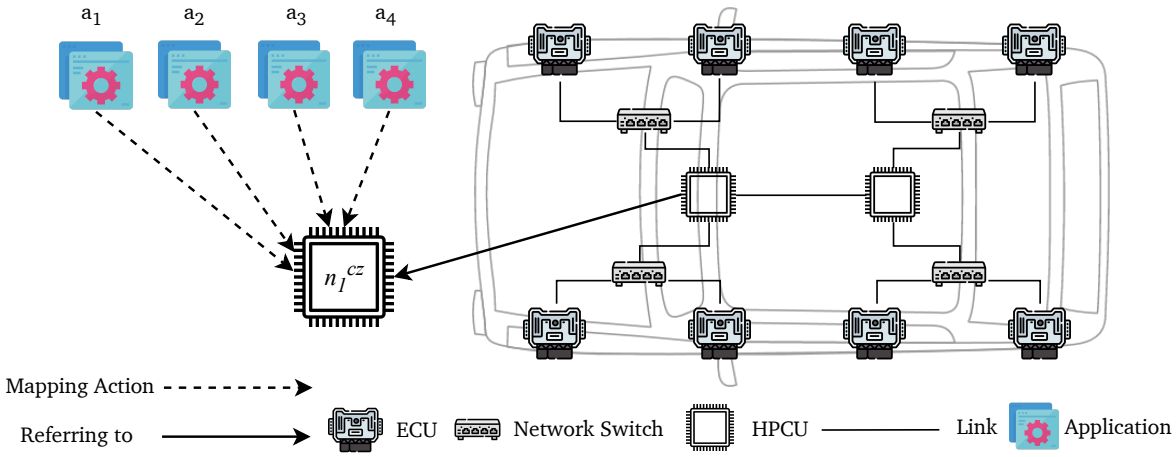


Figure 4.3: A vehicle architecture including assignment of applications to an HPCU (n_1^{cz}).

routed from one control node (ECU_1) to another (ECU_4) over a created path which is indicated with the S arrow. A single schedule on a link $l_{a,b}$ is indicated as $c_i.st^{l_{a,b}}$, which represents the starting time of the frame c_i. It should be noted that the entire process of sending, forwarding, and receiving a frame over the network is represented as a communication task. Note that an application thread and a communication task can have different and arbitrary periods [AMK23].

4.2.3 Mapping Action

Assigning applications, which consist of processes/threads, to control nodes such as ECUs or the cores of an HPCU is defined as a mapping action (see Figure 4.3), as explained as fundamental concepts in Chapter 2. To automate the mapping process, a mapping indication as m_{ij} is considered in the system model comprising different sub-variables. It represents a possible assignment of different applications (j) to various control nodes (i) [AMK23]. According to the example presented in Figure 4.3, assigning a_4 to n_1^{cz} is denoted by $m_{14}.{a_4}^{n_1^{cz}}$ as a mapping variable. Moreover, the mapping variables of a thread belonging to an application that is set as a sender or a receiver are symbolized as ${m_{ij}}^s$ and ${m_{ij}}^r$, respectively. For example, the mapping variables of t_1 from a_1 as the sender running on n_1^{cz}, and t_1 from a_4 as the receiver executing on n_7^{cz} are indicated with $m_{11}^s.{a_1}^{n_1^{cz}}$ and $m_{74}^r.{a_4}^{n_7^{cz}}$, respectively (refer to Figure 4.4 (a)). Similarly, each n_i^{cz} includes all mappings of existing applications, meaning that each a_i, which consists of a t_i or multiple t_i, can be executed on each n_i^{cz} [AMK23].

4.2.4 Communication Message

Multiple variables are defined in the system model to create automatic message routings for a designed vehicle E/E architecture using the introduced computer-aided tool. A communication message is a piece of information sent from a sender to a receiver. In the specified model, a communication message d_i is specified by a tuple $d_i = \{d_i.ch_i, d_i.c_i, d_i^{in}, d_i^{out}, d_i.rt, d_i.el\}$. The message chain is characterized by $d_i.ch_i$, containing the threads, as the sender and receiver, and a communication task that transfers the communication message and constitutes the related application in a correct temporal order. Each d_i includes a communication task, as described before, which carries the message symbolized with $d_i.c_i$. The message d_i received by a node n is represented as d_i^{in} while d_i^{out} designates the d_i sent by the node n.

Following the shown example in Figure 4.4, the message chain $d_1.ch_1$ for d_1 consists of two threads related to applications a_1 and a_4 and a communication task c_1 (yellow dots in Figure 4.4 (a) over the enumerated path) routing over five links in the visualized temporal order (yellow frames in Figure 4.4 (b)). Note that each application thread can only generate one communication message, which is then transmitted via one communication task [AMK23].

In addition, the message d_i sent out from n_a over $l_{a,b}$ is denoted as $d_i^{out}.l_{a,b}^{n_a}$, whereas the message d_i received by n_a over $l_{b,a}$ is characterized as $d_i^{in}.l_{b,a}^{n_a}$ (refer to Table A.1). The response time of a communication message is indicated by $d_i.rt$, which represents the time between the beginning of the period and the end of the last communication task. Meanwhile, the end-to-end latency $d_i.el$ represents the time between the start of the first task and the end of the last task [AMK23]. The significance of these two parameters will be explained in the following sections.

4.2.5 Application

An application a_i is a collection of application threads that can act as senders or receivers of communication messages to perform a specific function. The application is defined by the tuple $a_i = \{a_i.t_{ij}, a_i.t_{ij}^s.d_i, a_i.t_{ij}^r.d_i\}$, where $a_i.t_{ij}$ includes one or multiple application threads $t_{ij} = \{t_{i1}, t_{i2}, ..., t_{ij}\}$, with $j \in \mathbb{N}$, indicating that each a_i can have one or many t. Additionally, $a_i.t_{ij}^s.d_i$ represents the application thread t_j belonging to a_i that sends the communication message d_i, whereas $a_i.t_{ij}^r.d_i$ denotes the t_j of a_i that receives d_i, as stated in Table A.1.

4.2.6 Timing Limitations

The current version of the proposed framework in this thesis offers a non-preemptive time-triggered scheduling scheme for all application threads running on the control nodes and routed communication tasks over the network links. As a result, some additional timing constraints have been incorporated into the system, similar to those presented in [Zha+14], which are explained below.

Serialization Delay

When an application thread completes its task, it takes some time before the data can be packed into frames and sent over the network. This time, often called the serialization delay, is crucial to overall system performance. In the system model, this time is denoted by sd. The serialization delay represents the time required to convert the processed data into a format suitable for transmission. In other words, it is the time it takes to serialize a packet, meaning how long it takes to physically put the packet on the wire. During this stage, the data is typically transformed into a serialized representation, such as binary or JSON, allowing it to be efficiently transmitted across the network. The duration of the serialization delay can vary depending on factors such as the complexity of the data, the chosen serialization method, and the underlying hardware resources. It is worth noting that the network conditions and bandwidth limitations can influence the serialization delay. In scenarios where the available network bandwidth is constrained, the serialization delay may become more noticeable as the system waits for available network resources to transmit the serialized data. Considering these factors during system design and network provisioning is essential to ensure efficient data transfer [AMK23].

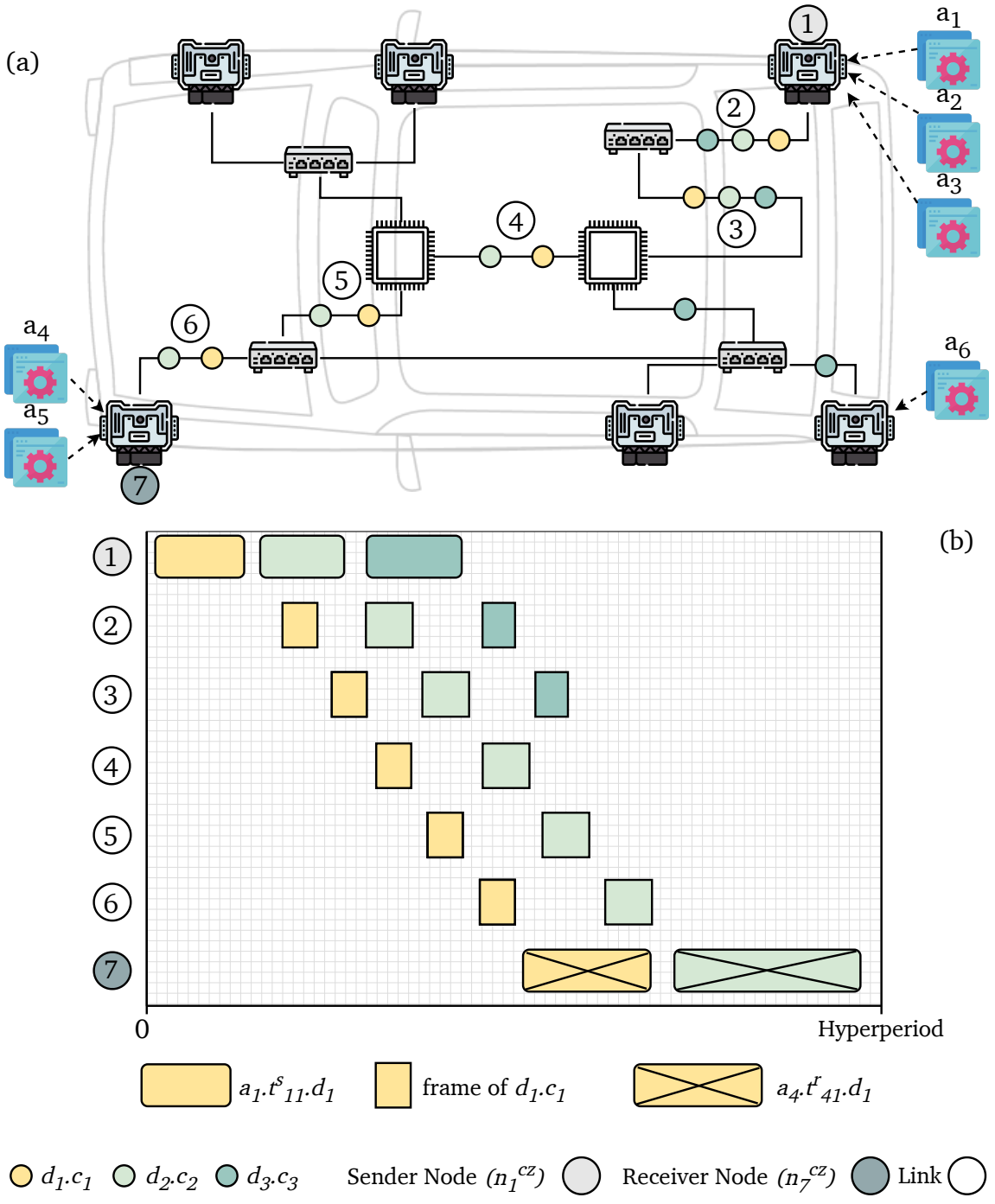


Figure 4.4: (a) A vehicle topology which shows generated paths for communication messages from senders to the receivers and mapped applications to the control nodes. Each colorful dot represents a communication task associated with a communication message. In this example, each application comprises a single application thread. (b) Calculated time-triggered schedules from the sender (n_1^{cz}) to the receiver (n_7^{cz}). It includes schedules for the sender (number one, with thread slots that are not crossed) and the receiver application threads (number seven, with thread slots that are crossed). The schedules of communications tasks (yellow and light green frames of the tasks in numbers two to six) over the generated path (only the enumerated route in (a)) are also included. The path routing two communication messages (d_1 and d_2) is considered. As can be seen, message dependency for the sender and receiver and path dependency for communication tasks are fulfilled. The representations for light green and dark green frames are similar to the yellow ones [AMK23].

Deserialization Delay

When a frame arrives at a control node, it requires a specific amount of time to be unpacked and processed before the relevant node can effectively utilize the contained data. This time, commonly known as the deserialization delay and indicated as *rd* in the framework's system model, plays a vital role in the overall efficiency and responsiveness of the system. This delay stands for the time needed to extract the serialized data from the received frame and convert it back into its original format. This process involves reversing the serialization process, which may comprise operations like decoding binary representations, parsing JSON structures, or reconstructing complex data objects. The duration of the deserialization delay can change due to various factors, such as the complexity of the data, the selected approach, and the computational resources available at the control node [AMK23].

Processing Delay

The maximum processing delay of a communication frame in a networking node is a crucial metric that characterizes the time required for the node to process and forward the received frame. It signifies the delay from when the last bit of the frame is received on the input port until the earliest possible transmission of the first bit on the output port. This delay is represented as *pd*. The processing delay encompasses various stages of frame handling within the networking node. Once the frame is received, it undergoes several essential operations, including header parsing, routing table lookup, potential payload processing, and potential modifications or encapsulations before being transmitted to the next destination. Each of these operations contributes to the overall processing delay. This delay can be influenced by factors such as the frame's complexity, the computational resources available in the node, and the network traffic load. For the complex frames or resource-constrained nodes, the processing delay may be longer, potentially leading to increased latency in the network. Therefore, it is essential to consider these factors during network design and provisioning to ensure that the processing delay remains within acceptable limits [AMK23].

Interpacket Gap

The interpacket gap indicated as *ipg*, is considered, which represents the required time interval between consecutive network packets or frames. The interpacket gap serves as a crucial recovery period that allows network devices to prepare for the reception of the next packet.

The purpose of the interpacket gap is to ensure proper synchronization and coordination between transmitting and receiving devices within a network. It allows devices to handle the processing and forwarding of the received packet, update internal states, and make necessary preparations before the arrival of the subsequent packet. This recovery time is critical in high-speed or heavily loaded networks, where devices require a momentary pause to avoid congestion, buffer overflows, or potential data loss. The duration of the *ipg* is typically defined by network protocols or standards and can be different based on the specific requirements of the network environment. It may be specified in terms of time, e.g., microseconds or milliseconds, or as a multiple of the packet transmission time. The duration of this gap should be carefully chosen to strike a balance between efficient data transmission and allowing sufficient recovery time for network devices. Properly setting and managing the *ipg* can significantly impact network performance and reliability. If the gap is too short, devices may not have enough time to process and prepare for the next packet, leading to increased packet loss, errors, or degraded performance. On the other hand, an excessively long *ipg* can result in reduced network throughput and underutilization of available bandwidth [AMK23].

Network administrators and engineers often fine-tune the *ipg* based on the network infrastructure's specific characteristics and the traffic's nature. They consider factors such as

network speed, latency, device capabilities, and the presence of other network optimization techniques like flow control or congestion avoidance algorithms.

Bandwidth

Finally, the bandwidth, denoted by *bw*, is a fundamental metric that characterizes the maximum potential data transfer rate between two points in a network over a specific link within a given time frame. It plays a crucial role in determining a network connection's overall capacity and performance.

Bandwidth, typically measured in bits per second (bps), represents the amount of data that can be transmitted within a specified time duration. It limits the rate at which information can be exchanged between network devices, such as routers, gateways, switches, or communication links. The higher the bandwidth, the greater the volume of data that can be transferred in a given time, leading to faster and more efficient communication. *bw* can be affected by the underlying infrastructure and technology used in the network. Factors such as cable quality, transmission medium (e.g., copper or fiber optic), and network equipment capabilities influence the achievable bandwidth for wired connections. In wireless networks, variables such as spectrum availability, modulation techniques, signal strength, and interference impact the available bandwidth. Various techniques are employed to make the most efficient use of available bandwidth. Network protocols and algorithms implement strategies such as congestion control, QoS mechanisms, and traffic prioritization to optimize data transfer and ensure fair sharing of network resources [AMK23].

4.3 Constraints MIP Formulation

The E/E Designer framework creates automated mapping and automatic message routings. It considers path and message dependencies and calculates the time-triggered schedules for different application threads and communication tasks c_i for a modeled vehicle E/E system. These threads run on various control nodes n_i^{cz}, and the communication tasks route over the network. The framework takes predefined boundary and optimization objectives into account. The E/E architecture presented in Figure 4.4 is characterized as a distributed system. This is done to proceed with the E/E Designer goals using a set of constraints.

4.3.1 Automated Mapping

To automatically assign applications, including their threads, to control nodes, the following constraint set is formulated:

$\forall i, j, k \in \mathbb{N}$, m_{ij}, m_{ik}, m_{ij}^s, $m_{ij}^r \in \mathcal{M}$, a_j, a_k, a_o, a_p, a^{sc}, a^{asil_D}, $a^{non-asil_D} \in \mathcal{A}$, $n_i^{cz} \in \mathcal{N}$:

if $a_j \notin a^{sc}$ **then**

$$\sum_{i \in \mathbb{N}} m_{ij}.{a_j}^{n_i^{cz}} = 1 \tag{4.1}$$

$$\sum_{j \in \mathbb{N}} m_{ij}.{a_j}^{n_i^{cz}} \geq 1 \tag{4.2}$$

$$m_{ij}^s.{a_j}^{n_i^{cz}} + m_{ik}^r.{a_k}^{n_i^{cz}} \leq 1 \tag{4.3}$$

if $a_k \in a^{sc}$ **then**

$$\sum_{i \in \mathbb{N}} m_{ik}.{a_k}^{n_i^{cz}} = 2 \tag{4.4}$$

$$m_{ij}.{a_j}^{n_i^{cz}} + m_{ik}.{a_k}^{n_i^{cz}} \leq 1 \tag{4.5}$$

if $a_o \in a^{asil_D}$ **and** $a_p \in a^{non-asil_D}$ **then**

$$m_{io}.{a_o}^{n_i^{cz}} + m_{ip}.{a_p}^{n_i^{cz}} \leq 1. \tag{4.6}$$

The condition (4.1) ensures that each application is only mapped once on each n_i^{cz}. Moreover, to distribute the applications, including their threads, to all existing control nodes, each n_i^{cz} must execute at least one application (a_j) which is encoded in Eq. (4.2). Based on the constraint (4.3), the sender and receiver threads from two different applications, transferring a communication message, cannot be executed on the same control node [AMK23].

Redundancy in Mapping

To meet the redundancy requirement following ISO 26262 [ISO18], the safety-critical applications a^{sc} set must be run redundantly. Hence, condition (4.4) forces each a^{sc} to be executed twice on various n_i^{cz} [AMK23; AHK22].

FFI in Mapping

In addition, to fulfill the FFI demand for a^{sc} and a^{asil}, non-safety a_j and safety-critical a_k applications must not be run on the same n_i^{cz} which is formulated in (4.5). Similarly, to provide FFI condition during the automated mapping process for ASIL D applications, constraint (4.6) is stated. Based on (4.6), ASIL D and non-ASIL D applications cannot be executed on the same control node [AHK22; AMK23]. The sets of all mapping variables are defined as $\mathcal{M}$ and all applications, including safety-critical ones, as $\mathcal{A}$. Note that all mapping variables are specified as binary variables (i.e., 0 or 1). The same constraints are applied for each core of an HPCU (n^{cz_c}). That is, the redundancy and FFI conditions are valid for various cores with different ASIL levels.

4.3.2 Automatic Message Routing

Message routing plays a pivotal role in automotive networks, where many nodes, comprising ECUs, HPCUs, switches, and gateways, collaborate to ensure smooth vehicle operations. As modern automobiles become increasingly sophisticated, incorporating various functionalities and advanced systems, efficient and reliable message routing becomes paramount. In automotive networks, messages flow between nodes to exchange vital information about engine control, transmission, braking, safety systems, entertainment, and more. These messages typically follow predetermined protocols and data formats, such as the CAN, time-triggered CAN, FlexRay, LIN, and Ethernet, to ensure compatibility and interoperability across the network. The primary goal of message routing is to deliver messages from their source nodes to their intended destinations, optimizing the data transmission process while adhering to the network's constraints. Several factors influence the routing decisions within automotive networks, including the message's priority and criticality, the available bandwidth, the network topology, and the characteristics of the nodes involved, as illustrated in Chapter 2.

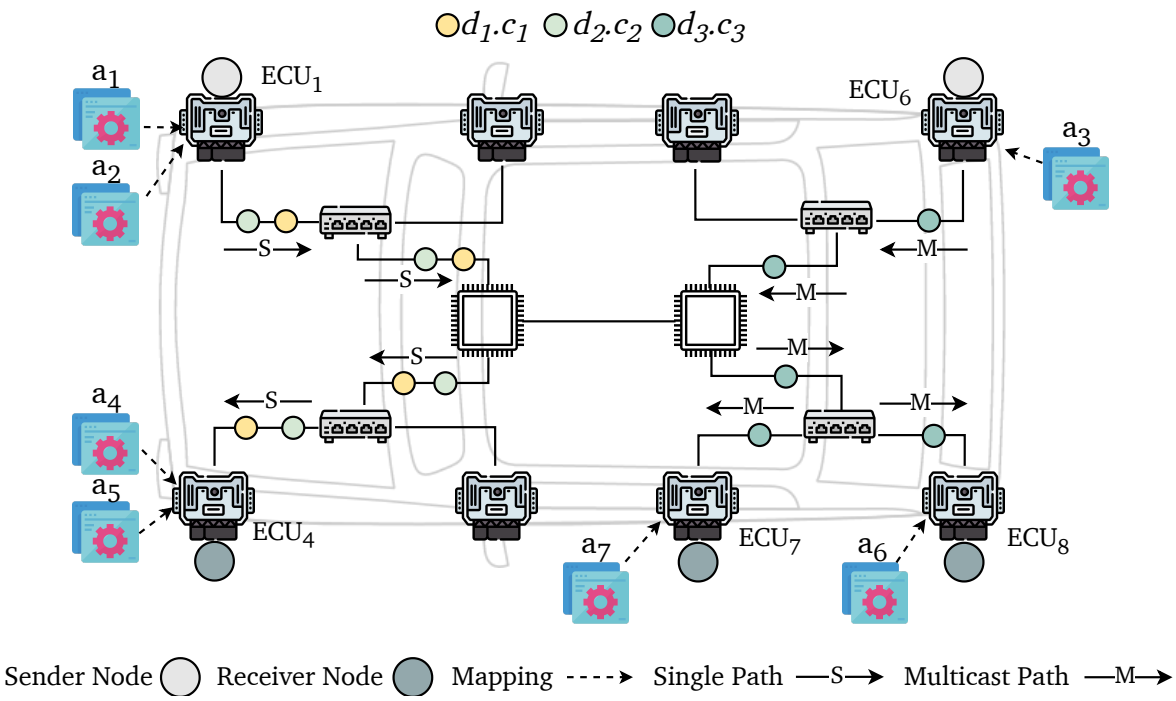


Figure 4.5: A vehicle architecture comprising intermediate and control nodes, links, and assigned applications to ECUs. A single (arrows with S) and a multicast (arrows with M) paths are generated in order to send communication messages (colored dots), created by applications, from sender nodes to receiver nodes.

The presented model-based framework proposes automatic message routing for modeled E/E systems, including single, multicast, redundant, and homogeneous redundant paths. In the following, each of these routes is explained and encoded.

Single Routing

Single routing, also known as unicast routing, involves the transmission of a message from a source control node to a single destination control node (see Figure 4.5). In this type of routing, each message is addressed to a specific destination node, and the routing process focuses on finding the optimal path to deliver the communication message to that particular node. Single routing is commonly used for point-to-point communication, where a message is intended for a specific recipient [AFK21a]. The routing decisions are typically based on various factors such as network topology, available bandwidth, latency requirements, cost, and reliability, which will be illustrated in the following sections.

In Figure 4.5, a single path (indicated as S) is shown to transmit two communication messages (d_1 and d_2) from their senders to their receivers— yellow and light green dots in the Figure 4.5 represent two communication tasks belonging to two communication messages routed over the network's links.

Considering the previous work [AFK21a], the following constraints are utilized to encode the single message routing:

$\forall i, j, k,\ n_j \in \mathcal{N},\ d_k^{in},\ d_k^{out} \in \mathcal{D},\ m_{ij}{}^s,\ m_{ij}^r \in \mathcal{M}$:

$$m_{ij}^s - m_{ij}^r - \sum_{j \in \mathbb{N}} n_j.d_k^{out} \leq 0 \tag{4.7}$$

$$m_{ij}^{s} + \sum_{j \in \mathbb{N}} n_j . d_k^{in} \leq 1 \tag{4.8}$$

$$m_{ij}^{r} - m_{ij}^{s} - \sum_{j \in \mathbb{N}} n_j . d_k^{in} \leq 0 \tag{4.9}$$

$$m_{ij}^{r} + \sum_{j \in \mathbb{N}} n_j . d_k^{out} \leq 1 \tag{4.10}$$

$$m_{ij}^{s} - m_{ij}^{r} + \sum_{j \in \mathbb{N}} n_j . d_k^{in} - \sum_{j \in \mathbb{N}} n_j . d_k^{out} = 0. \tag{4.11}$$

Based on constraint (4.7), there must be at least one outgoing communication message (d_k^{out}) over a link for a sender node. This constraint applies when the automated mapping assigns a mapping variable m_{ij}^{s} equal to 1 for the sender node n_j. However, this condition does not affect the routing process when the node is not a sender. In Eq. (4.8), a sender node n_j is required to block all incoming communication messages (d_k^{in}) related to d_k that are coming from other nodes. This ensures that no activated d_k^{in} messages exist for the sender node n_j, which means that $m_{ij}^{s} = 1$ and $m_{ij}^{r} = 0$. For example, in Figure 4.5, the sender node ECU_1 has only one activated d_k^{out} and no triggered d_k^{in} to route communication messages to the next node.

Similarly to condition (4.7), but this time for a node (n_j) as a receiver ($m_{ij}^{s} = 0$ and $m_{ij}^{r} = 1$) of a communication message, at least one incoming message (d_k^{in}) over a link must be set, according to Eq. (4.9). Also, constraint (4.10) expresses that the receiver node must not have any triggered outgoing data (d_k^{out}). In Figure 4.5, there is only one activated d_k^{in} and no d_k^{out} for ECU_4 as a receiver to create a single route. Considering Eqs. (4.7) and (4.8), it is possible for the sender and receiver to have multiple d_k^{out} and d_k^{in} values, respectively. Therefore, to create a single path, it is necessary to ensure that the sender and receiver nodes have only one triggered outgoing and incoming communication message, respectively, as stated in Eq. (4.11). Furthermore, this condition applies to intermediate nodes that are neither senders nor receivers of a message. These intermediate nodes must either have no triggered message or have exactly one activated incoming message (d_k^{in}) and one triggered outgoing message (d_k^{in}).

Multicast Routing

Multicast routing involves the transmission of a message from a source node to multiple destination nodes simultaneously. In this type of routing, a single message is addressed to a group of destination nodes rather than a specific node. The routing process aims to replicate the message and deliver it to all the nodes in the multicast group efficiently. It is commonly used for applications where the same information needs to be sent to multiple recipients, such as video streaming, software updates, and distributed simulations. A multicast path (indicated by the arrow visualized with M) is shown in Figure 4.5 for transmitting the d_3 message from ECU_6 to two receivers, namely ECU_7 and ECU_8. The E/E Designer uses the following conditions to automatically create multicast routes for the designed automotive networks [AMK23].

$\forall i, j, k,\ n_j \in \mathcal{N},\ d_k^{in},\ d_k^{out} \in \mathcal{D},\ {m_{ij}}^{s},\ m_{ij}^{r} \in \mathcal{M}$:

$$m_{ij}^{s} \times (m_{ij}^{s} - m_{ij}^{r}) - m_{ij}^{s} \times \sum_{j \in \mathbb{N}} n_j . d_k^{out} \leq 0 \tag{4.12}$$

$$m_{ij}^{s} \times \sum_{j \in \mathbb{N}} n_j.d_k^{in} \leq 0 \tag{4.13}$$

$$m_{ij}^{r} \times (m_{ij}^{r} - m_{ij}^{s}) - m_{ij}^{r} \times \sum_{j \in \mathbb{N}} n_j.d_k^{in} = 0 \tag{4.14}$$

$$m_{ij}^{r} \times \sum_{j \in \mathbb{N}} n_j.d_k^{out} \leq 0 \tag{4.15}$$

$$\sum_{j \in \mathbb{N}} n_j.d_k^{out} - \sum_{j \in \mathbb{N}} n_j.d_k^{in} = m_{ij}^{s} - m_{ij}^{r} \tag{4.16}$$

or

$$\sum_{j \in \mathbb{N}} n_j.d_k^{out} \geq m_{ij}^{s} - m_{ij}^{r} + \sum_{j \in \mathbb{N}} n_j.d_k^{in}.$$

Constraint (4.12) reveals that at least one outgoing communication message (d_k^{out}) over a link must be activated for a sender node. This rule is triggered for the sender node, i.e., m_{ij}^{s} equal to 1 for the sender node n_j, as each variable of the Eq. (4.12) is multiplied by m_{ij}^{s}. Moreover, a sender node cannot have any activated communication message coming into the sender from other nodes, according to Eq. (4.13). As a receiver, a node must comprise precisely one triggered inflowing message (d_k^{in}) to fulfill requirements for generating multicast paths (refer to Eq. (4.14)). Considering the multiplication terms, this condition is only valid for the receiver nodes. Furthermore, a receiver must inhibit all outgoing communication messages (d_k^{out}) related to d_k that are coming from other nodes, based on constraint (4.15). This guarantees that no initiated d_k^{out} messages exist for the receiver node n_j.

According to Eq. (4.16), the intermediate nodes must either have an equal number of incoming and outgoing communication messages or the number of d_k^{out} messages must be greater than equal to d_k^{in} messages. This condition enforces the intermediate nodes to include either one input and one output or one input and multiple outputs to meet the multicast routing requirements. In Figure 4.5, considering the multicast path, the network switch has only one entering message or d_k^{in} and two departing messages (d_k^{out}) to receivers ECU_7 and ECU_8.

Redundant Routing

It plays an essential role in ensuring reliable and fault-tolerant communication in automotive communication networks between various nodes, such as control and networking nodes in vehicles. As automotive systems become increasingly complex and interconnected, redundancy becomes essential to maintain system functionality and safety, especially for mixed-critical systems. This type of routing involves the provision of multiple communication paths between nodes, allowing for alternate routes in case of link failures, congestion, or other communication disruptions. These redundant paths serve as backups in case of link failures or congestion. The redundant paths in physical routes, protocols, or transmission media may differ. This redundancy enhances the robustness and resilience of the network, minimizing the risk of communication failures and providing backup options in case of any unforeseen issues. One approach to implementing redundant routing is the use of parallel communication channels. Multiple physical or virtual links between nodes, e.g., ECUs, can be established, allowing for simultaneous data transmission over different paths. If one link experiences a failure or degradation, the system can automatically switch to an alternate path, ensuring uninterrupted communication. This redundancy helps to mitigate the impact of single-point failures and enhances the overall reliability of the network [ISO18; Smi+18; AFK21a; AMK23].

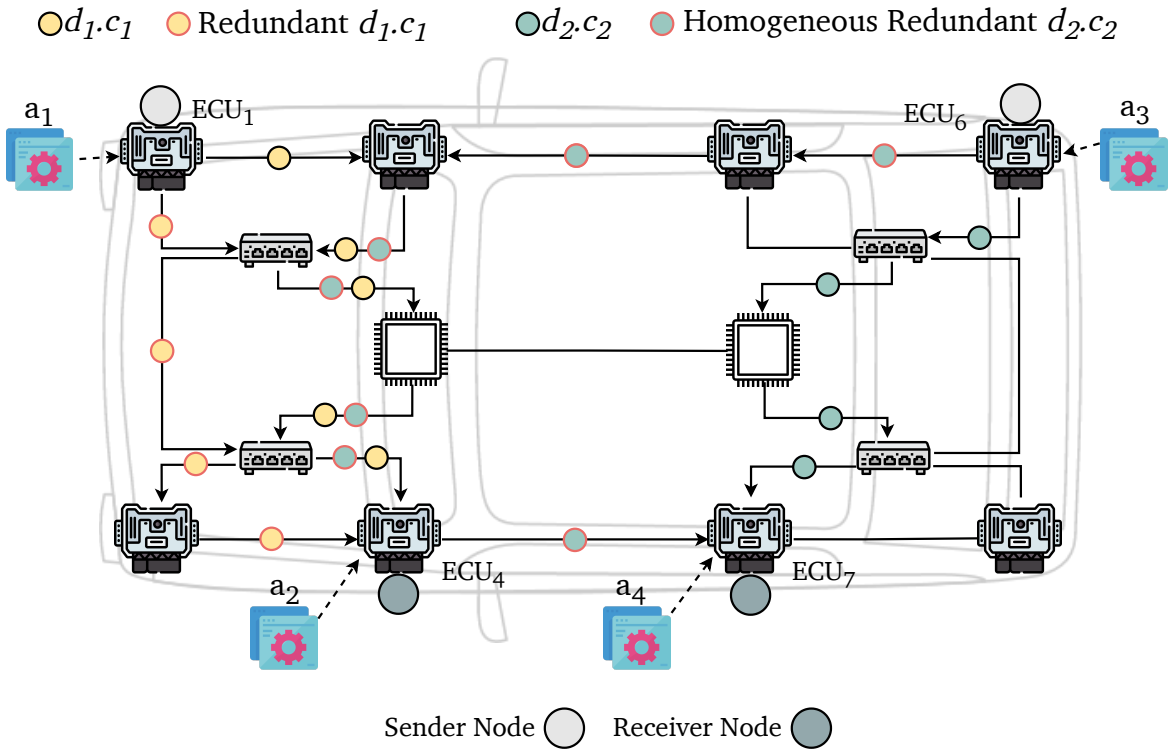


Figure 4.6: A modeled car architecture comprising intermediate and control nodes, links, and assigned applications to ECUs. A redundant (yellow dot with red border line) and a homogeneous redundant (green dot with red border line) routes are created in order to send communication messages (colored dots), created by applications, from sender nodes to receiver nodes.

The E/E Designer framework creates redundant physical paths for safety-critical applications to transmit critical communication messages. Redundant routings are not necessarily disjoint, meaning they may share common nodes or links between senders and receivers. For example, in Figure 4.6, there is one redundant path for the communication message, d_1, from ECU_1 to ECU_4. This route transmits an additional communication task (c_1) belonging to a communication message (d_1) (it is shown with a yellow dot with a red border) from the sender to the receiver. This routing is not disjoint as it shares two network switches with the other related to d_1. The set of constraints below is integrated into the framework's system model to generate automatic redundant routing.

$\forall i, j, k, n_j \in \mathcal{N}, d_k^{in}, d_k^{out} \in \mathcal{D}, m_{ij}{}^s, m_{ij}^r \in \mathcal{M}$:

$$m_{ij}^s - m_{ij}^r - \sum_{j \in \mathbb{N}} n_j.d_k^{out} \leq -1 \times m_{ij}^s \tag{4.17}$$

$$m_{ij}^s \times \sum_{j \in \mathbb{N}} n_j.d_k^{in} \leq 0 \tag{4.18}$$

$$m_{ij}^r - m_{ij}^s - \sum_{j \in \mathbb{N}} n_j.d_k^{in} \leq -1 \times m_{ij}^r \tag{4.19}$$

$$m_{ij}^r \times \sum_{j \in \mathbb{N}} n_j.d_k^{out} \leq 0 \tag{4.20}$$

$$2 \times m_{ij}^{d} - 2 \times m_{ij}^{s} - \sum_{j \in \mathbb{N}} n_j.d_k^{in} + \sum_{j \in \mathbb{N}} n_j.d_k^{out} = 0. \tag{4.21}$$

Constraint (4.17) illustrates that a sender node must activate at least two outgoing communication messages (d_k^{out}) over a link. This requirement is applied to the sender node; in other words, when the mapping variable, m_{ij}^{s}, has a value of 1 for the sender node n_j. In addition, the sender node must not have any activated communication messages coming into it from other nodes, as specified in constraint (4.18). As for a receiver node, it must have at least two incoming triggered messages (d_k^{in}) in order to meet the requirements for creating a redundant path, as indicated in constraint (4.19). This condition is only applicable to receiver nodes. Furthermore, a receiver node must prevent any outgoing communication messages (d_k^{out}) associated with d_k from reaching other nodes, according to Eq. (4.20). This ensures no existing d_k^{out} messages for the receiver node n_j.

When Eq. (4.21) is applied to intermediate nodes, it enforces them to have an equal number of incoming and outgoing messages. Moreover, it obliges sender and receiver nodes to have exactly two outgoing (d_k^{out}) and inflowing (d_k^{in}) messages, respectively. This is crucial because based on Eqs. (4.17) and (4.19), sender and receiver nodes can potentially have more than two d_k^{out} and d_k^{in} messages, respectively. As visualized in Figure 4.6, ECU_1, as a sender, and ECU_4, as a receiver, have two triggered outgoing and incoming paths, respectively. Besides, the network switches, which function as intermediate nodes between ECU_1 and ECU_4, possess two d_k^{out} and d_k^{in} each, applying Eq. (4.21).

Homogeneous Redundant Routing

It refers to a network design approach where there are multiple redundant paths between any two nodes in the network, and each path is capable of carrying the same type of data. This redundancy helps to improve the reliability and fault tolerance of the network, as data can still be transmitted even if one path fails. The redundant paths in homogeneous redundant routing are typically designed to be functionally identical. In other words, each node in the network has redundant communication data flows to other nodes, which are designed to be functionally similar. This redundancy helps to ensure that even if one way fails, the data can still be transmitted using an alternative path. This type of routing is critical in automotive networks, where reliability is crucial for safety-critical applications. Ensuring that multiple redundant paths are available for transmitting critical data reduces the likelihood of a communication failure. In contrast to redundant routing, homogeneous redundant paths are disjoint so that they do not share any common nodes or links [AFK21a; ISO18]. In Figure 4.6, a homogeneous redundant pathway is shown from ECU_6 to ECU_7 to route a communication message (d_2). This path does not share any standard components with the main data flow related to d_2, as displayed in Figure 4.6.

The introduced tool uses the constraints below to provide the homogeneous redundant routing described in our previous paper [AFK21a]. These constraints encode the homogeneous redundant data flows with duplication of the entire routing elements comprising the nodes (except for the source and destination nodes) and the links. Note that the following equations are applicable for the automated mapping approach, which have not been discussed in [AFK21a].

$\forall i, j, k,\ n_j \in \mathcal{N},\ d_k^{in},\ d_k^{out} \in \mathcal{D},\ m_{ij}{}^{s},\ m_{ij}^{r} \in \mathcal{M},\ n_{hmr} \in \mathbb{N}$:

$$m_{ij}^{s} \times (m_{ij}^{s} - m_{ij}^{r}) - m_{ij}^{s} \times \sum_{j \in \mathbb{N}} n_j . d_k^{out} \leq -n_{hmr} \times m_{ij}^{s} \tag{4.22}$$

$$m_{ij}^{s} \times \sum_{j \in \mathbb{N}} n_j . d_k^{in} \leq 0 \tag{4.23}$$

$$m_{ij}^{r} \times (m_{ij}^{r} - m_{ij}^{s}) - m_{ij}^{r} \times \sum_{j \in \mathbb{N}} n_j . d_k^{in} \leq -n_{hmr} \times m_{ij}^{r} \tag{4.24}$$

$$m_{ij}^{r} \times \sum_{j \in \mathbb{N}} n_j . d_k^{out} \leq 0 \tag{4.25}$$

$$m_{ij}^{r} - m_{ij}^{s} + n_{hmr} \times (m_{ij}^{r} - m_{ij}^{s}) + \sum_{j \in \mathbb{N}} n_j . d_k^{out} - \sum_{j \in \mathbb{N}} n_j . d_k^{in} = 0 \tag{4.26}$$

$$(1 - m_{ij}^{r}) \times (1 - m_{ij}^{s}) \times \sum_{j \in \mathbb{N}} n_j . d_k^{out} \leq (1 - m_{ij}^{r}) \times (1 - m_{ij}^{s}) \tag{4.27}$$

$$(1 - m_{ij}^{r}) \times (1 - m_{ij}^{s}) \times \sum_{j \in \mathbb{N}} n_j . d_k^{in} \leq (1 - m_{ij}^{r}) \times (1 - m_{ij}^{s}). \tag{4.28}$$

Based on constraint (4.22), a sender node must at least comprise one plus n_{hmr} outflowing communication messages (d_k^{out}) over a link. Here, n_{hmr} represents the number of required homogeneous redundant routes that can be chosen in the framework fronted by the user. As an example, the number of n_{hmr} is set as one to create one homogeneous redundant pathway in Figure 4.6. Furthermore, all possible inflowing messages d_k^{in} directed into a sender node from other nodes are deactivated, as defined in constraint (4.23). Condition (4.24) indicates that a receiver node must have at least $1+n_{hmr}$ incoming triggered messages (d_k^{in}) to establish n_{hmr} homogeneous redundant routings. In addition, according to Eq. (4.25), messages can not flow out (d_k^{out}) of a destination node. Both sender and receiver nodes must have an exact number of outgoing and incoming messages, respectively, as determined in Eq. (4.26). The same constraint guarantees that intermediate nodes have an equal number of out and inflowing messages (d_k^{out} and d_k^{in}). As mentioned earlier, the homogeneous redundant path is entirely disjoint. Consequently, conditions (4.27) and (4.28) are formulated to ensure all investigated routes are disjoint. When executed on an intermediate node, the number of outgoing and incoming messages must be less than or equal to one, based on Eqs. (4.27) and (4.28), respectively.

All the constraints explained above are simultaneously applied to all nodes, including sender, receiver, and intermediate nodes, and they are valid for all of these nodes.

Outgoing/Incoming Messages Connection and Cycle Breaker

As explained previously, each directed link (e.g., $l_{a,b}$), connecting two nodes (e.g., n_a to n_b), consists of an outgoing (d_k^{out}) and an incoming (d_k^{in}) messages which belong to the same communication message. This is indicated to activate a directed path from, e.g., n_a to n_b. Therefore, in the introduced routing system model, d_k^{out} and d_k^{in} related to a directed link between two nodes must be connected to make the routing process possible. Cycle breaking in communication message routing refers to the process of preventing or resolving cycles in a network topology when routing messages between nodes [AFK21a]. In a network, cycles occur when there is a loop in the routing path, causing messages to circulate indefinitely or causing routing protocols to become stuck. Cycle breaking is essential in communication networks to ensure the efficient and timely delivery of messages and to avoid network congestion or packet loss.

$$d_k^{out}.l_{a,b}^{n_a} - d_k^{in}.l_{a,b}^{n_b} = 0 \tag{4.29}$$

$$d_k^{out}.l_{a,b}^{n_a} - d_k^{out}.l_{b,a}^{n_b} \leq 1 \tag{4.30}$$

or

$$d_k^{in}.l_{a,b}^{n_b} - d_k^{in}.l_{b,a}^{n_a} \leq 1.$$

In order to connect outgoing (d_k^{out}) and incoming (d_k^{in}) messages, which belong to the same communication message and are transmitted on the same directed link between two nodes, Eq. (4.29) is introduced. Eq. (4.29) states that $d_k^{out}.l_{a,b}^{n_a}$, which represents the message d_k being sent out from node n_a over the directed link $l_{a,b}$ between n_a and n_b, must be equal to $d_k^{in}.l_{a,b}^{n_b}$, which indicates the message d_k being received by node n_b over the directed link $l_{a,b}$ between n_a and n_b. Constraint (4.30) prevents routing cycles during path generation. A routing cycle occurs when two d_k^{out} or two d_k^{in}, associated with each directed link between two nodes, are set in both directions.

Algorithm 1 is presented to fulfill the cycle breaker condition for paths and connect the incoming and outflowing messages over the same link. The details of this algorithm are explained as follows. For each application (line 1), all nodes pass through several conditions, and also, for each node (line 2), all incoming messages pass through different rules (line 3). For all d^{in}, based on line (4) each d^{out} goes through two *if conditions*. The first one ensures that the d^{in} and d^{out} belong to the same link; if so, the Eq. (4.29) is applied to connect the incoming and outgoing messages. The constraint (4.30) is executed where a loop/cycle in the path can be created based on the second *if condition* in line (8). In this condition, the possible routing cycle for each communication message between two nodes is identified according to the source and destination nodes for d^{out} and d^{in}. Afterward, to avoid a loop, either a d^{out} or a d^{in} for a related communication message is considered by applying (4.30).

Algorithm 1: Cycle Breaker and Connection of Incoming & Outgoing Messages

Input: $N = \{n_1, n_2, ..n_n\}$, $A = \{a_1, a_2, ..a_a\}$,
$D_{in} = \{ d_1^{in}, d_2^{in}, ..d_q^{in}\}$, $D_{out} = \{d_1^{out}, d_2^{out}, ..d_p^{out}\}$, $n, a, q, p \in \mathbb{N}$

Output: Breaking cycles for created communication message routings and connecting incoming and outgoing messages transmitting only over the same link

```
1  for i ← 0 to a do
2      for j ← 0 to n do
3          for k ← 0 to q do
4              for l ← 0 to p do
5                  if d_k^in.get(link) = d_l^out.get(link) then
6                      Apply constraint (4.29);
7                  end
8                  if d_k^in.get(source_node) = d_l^out.get(source_node) &
                     d_k^in.get(destination_node) = d_l^out.get(destination_node) then
9                      Apply constraint (4.30);
10                 end
11             end
12         end
13     end
14 end
```

4.3.3 Overlapping-Free Application Threads Considering Automated Mapping

To apply the time-triggered scheduling, as illustrated in Section 2.2, only to the assigned application threads on each control node, taking the automated mapping into account, the following set of constraints are introduced. In other words, the constraint set ensures that on every single control node, one mapped application thread is only triggered when the node is idle, i.e., after the last thread is finished. The presented overlapping constraints in (4.31) from [Zha+14] are extended based on the E/E Designer's system model so that automated mapping and time-triggered scheduling for application threads on each control node can be performed in a single-step solving [AMK23].

The set of all application threads is specified as $\mathcal{T}$. In addition, the activated mapping variables, $m_{k\alpha}$ and $m_{k\beta}$, regarding each pair of application threads running on a control node, are specified as the mapping variable denoting $\mathcal{M}$. The pair of threads can belong to the same or different applications. This condition can be encoded as follows:

$\forall i, j (i \neq j), k, \alpha, \beta \in \mathbb{N}; t_i, t_j \in \mathcal{T}; m_{k\alpha}, m_{k\beta} \in \mathcal{M}$:

$$v \times (t_i.p \times w_i + t_i.st + t_i.e) < v \times (t_j.p \times w_j + t_j.st) \tag{4.31}$$

or

$$v \times (t_j.p \times w_j + t_j.st + t_j.e) < v \times (t_i.p \times w_i + t_i.st)$$

$$t_i.st \geq 0 \tag{4.32}$$

$$t_i.st + t_i.e \leq t_i.p \tag{4.33}$$

$$t_j.st \geq 0 \tag{4.34}$$

$$t_j.st + t_j.e \leq t_j.p \tag{4.35}$$

considering

$$v = m_{k\alpha}.a_\alpha{}^{n_k^{cz}}.t_{\alpha i} \times m_{k\beta}.a_\beta{}^{n_k^{cz}}.t_{\beta j}$$

$$\forall w_i \in \left[0, \frac{LCM(t_i.p, t_j.p)}{t_i.p} - 1\right]$$

$$\forall w_j \in \left[0, \frac{LCM(t_j.p, t_i.p)}{t_j.p} - 1\right]$$

, where $LCM(t_j.p, t_i.p)$ symbolizes the least common multiple of periods $t_i.p$ and $t_j.p$. Moreover, $m_{k\alpha}.a_\alpha{}^{n_k^{cz}}.t_{\alpha i}$ and $m_{k\beta}.a_\beta{}^{n_k^{cz}}.t_{\beta j}$ represent the related binary mapping variables of the threads t_i and t_j, respectively. In this set of constraints, all variables are defined as double precision except for binary mapping variables. Based on constraint (4.31), one of the equations is valid at the time; hence, the two conditions must be checked simultaneously in

the constraint system. Constraints (4.32) ensure that the calculated starting time of thread t_i becomes greater than or equal to zero, and condition (4.33) states that the thread job t_i must be finished in its period slot. The same conditions (4.34) and (4.35) are applied for t_j. Following Figure 4.4, there are three sender application threads from three diverse applications $a_1.t^s_{11}$, $a_2.t^s_{21}$, and $a_3.t^s_{31}$ which are assigned to a control node n^{cz}_1 automatically and the correct schedule of each thread is visualized in the number one slot in Figure 4.4 (b) using this constraint set [AMK23].

4.3.4 Overlapping-Free Communication Tasks Considering Automatic Message Routing

Similarly to the previous subsection, the time-triggered scheduling is implemented for communication tasks routing over communication links in the vehicle's network. To ensure that there is no collision of frames being sent over a single directed link, a frame can only start its transmission *ipg* time units after the last frame is finished. The same time-triggered scheduling policy for application threads is applied for the communication tasks routing over a directed link only once the related link is activated, i.e., when the relevant link is part of a created route. The set of all communication tasks is defined as $\mathcal{C}$, and d^{out}_i and d^{out}_j, which are pairs of d^{out} relevant to a directed link, are part of the set of activated routing variables symbolized by $\mathcal{D}$ [AMK23]. This condition can be formulated as follows.

$\forall i, j (i \neq j) \in \mathbb{N}$; c_i, $c_j \in \mathcal{C}$; d^{out}_i, $d^{out}_j \in \mathcal{D}$:

$$r \times (c_i.p \times w_i + c_i.st + c_i.fl/bw + ipg) < r \times (c_j.p \times w_j + c_j.st) \tag{4.36}$$

or

$$r \times (c_j.p \times w_j + c_j.st + c_j.fl/bw + ipg) < r \times (c_i.p \times w_i + c_i.st)$$

$$c_i.st \geq 0 \tag{4.37}$$

$$c_i.st + c_i.fl/bw \leq c_i.p \tag{4.38}$$

considering

$$r = d^{out}_i.c_i \times d^{out}_j.c_j$$

$$\forall w_i \in \left[0, \frac{LCM(c_i.p, c_j.p)}{c_i.p} - 1\right]$$

$$\forall w_j \in \left[0, \frac{LCM(c_j.p, c_i.p)}{c_j.p} - 1\right].$$

Constraints (4.36), (4.37), and (4.38) are encoded for the same reason as constraints (4.31), (4.32), and (4.33) for application threads, with the only difference that here they are applied for pair of communication tasks c_i and c_j instead. Using r ensures that the starting time of communication tasks is only computed for the activated links, which are part of a generated path. Note that $c_i.fl/bw$ represents the transmission time of the packet $c_i.fl$ for a given bandwidth bw. In this constraint set, all variables are specified as double precision float numbers except for binary variables $d_i^{out}.c_i$ and $d_j^{out}.c_j$ which represent d_i^{out} related to communication task c_i and d_j^{out} related to communication task c_j, respectively. In Figure 4.4 (a), for example, in link number two, three communication tasks share the same link to route three messages ($d_1.c_1$, $d_2.c_2$, and $d_3.c_3$) and their schedules are displayed in the number two slot of Figure 4.4 (b) [AMK23].

4.3.5 Path Dependency

Path dependency in computed schedules for a communication network refers to the fact that the schedule of communication events (such as the transmission of data packets) depends on the path the data takes through the network. This means that the order in which the events occur is determined by the route the data takes through the network and not by any other factors [Zha+14]. As mentioned previously, several communication tasks can be sent over each link. In a communication task, a frame must only be forwarded along the routes in the correct temporal order. In other words, the timing order of all tasks belonging to a communication message for a path consisting of a link or multiple links must be correct. In Figure 4.4 (b), the correct temporal order of a communication task c_1 for a route (the numbered path), including five links, is visualized. For example, the finishing times of the frames over link two must be less than the starting times of the same frames over link three, which can be observed in Figure 4.4 (b) [AMK23].

To satisfy path dependency only for the activated routes in communication, the following constraint set is encoded:

$$\forall i \in \mathbb{N};\ c_i \in \mathcal{C};\ d_i,\ d_i^{out} \in \mathcal{D};\ \forall k \in [2, nl] \subseteq \mathbb{N};\ \forall l_k, l_{k-1} \in \mathcal{L}:$$

$$d_i^{out}.l_{k-1} \times (c_i.st_{d_i}^{l_{k-1}} + c_i.fl_{d_i}^{l_{k-1}}/bw + pd + sync) < d_i^{out}.l_k \times c_i.st_{d_i}^{l_k}. \tag{4.39}$$

The constraint (4.39) is encoded to ensure path dependency only for the activated routes in communication. This condition only becomes activated when the corresponding outgoing message over the related link, e.g., $d_i^{out}.l_{k-1}$, is active. In Figure 4.4 (a), the enumerated path consists of five links ($nl = 5$) and on each link, communication tasks are transmitted. The starting time of these tasks is represented with $c_i.st_{d_i}^{l_k}$, must be in the correct temporal order satisfying Eq. (4.39) (see, for example, the yellow schedule slots for links number two to six in Figure 4.4 (b)). Here, k represents the link number that bounds to nl, representing the number of existing links in each possible path, and $\mathcal{L}$ is defined as the set of directed links. In addition, $c_i.fl_{d_i}/bw$ represents the transmission time of the communication packet $c_i.fl$ for a given bandwidth bw. It should be added that *sync* represents the maximum difference between any two clocks in the system, as depicted in Table A.1. It is assumed that all schedules are referenced to the local time of the networking nodes [AMK23].

4.3.6 Message Dependency

In the context of communication networks in automotive systems, message dependency refers to the relationship between different messages or data packets that are exchanged between various components or nodes within the network. It represents the interdependence and order in which messages must be transmitted or processed to ensure the correct functioning of the system. Message dependency is essential because specific messages may depend on the reception or processing of other messages [Zha+14]. For example, consider a scenario where an ECU responsible for brake control needs information from another ECU responsible for wheel speed. The brake control ECU may rely on the wheel speed data to decide when and how to engage the brakes. In this case, the brake control message depends on the availability and timely reception of the wheel speed message. The presented framework supports the temporal dependency for message dependency. Temporal dependency refers to the ordering or sequencing requirements of messages. Some messages must be sent or received before or after specific other messages to maintain the correct functionality of the system [AMK23].

Due to the message dependency, the application threads and communication tasks in the message chain must be executed in the correct temporal order, as illustrated above. Furthermore, constraint (4.40) states that the thread's starting time of the sender, $t_i^s.st_{d_i}$, sending d_i, must be less than the starting time of the related communication task over the first link of the path denoted by l_s. This rule is only applied to the activated link, $d_i^{out}.l_s$, which is part of a created route. While the thread's starting time of the receiver, $t_i^r.st_{d_i}$, must be greater than the related communication task over the last link l_f of the route based on (4.41), considering only the activated edge. For instance, in Figure 4.4 (b), the yellow slot $a_1.t_{11}^s.d_1$, as a sender, is followed by $d_1.c_1$ over link number two, based on (4.40). Also, $d_1.c_1$ over link six is followed by $a_4.t_{41}^r.d_1$, as a receiver, according to the Eq. (4.41) [AMK23].

$\forall i \in \mathbb{N};\ t_i^s,\ t_i^r \in \mathcal{T};\ c_i \in \mathcal{C};\ d_i^{out} \in \mathcal{D};\ l_s,\ l_f \in \mathcal{L}$:

$$d_i^{out}.l_s \times (t_i^s.st_{d_i} + t_i^s.e_{d_i} + sd) < d_i^{out}.l_s \times (c_i.st_{d_i}^{l_s}) \tag{4.40}$$

$$d_i^{out}.l_f \times (c_i.st_{d_i} + c_i.fl_{d_i}/bw + rd) < d_i^{out}.l_f \times t_i^r.st_{d_i}. \tag{4.41}$$

Therefore, the temporal dependencies for a sender thread and its related communication task over the first link of a path, and for the same communication task over the last link of the path and its receiver thread are met in a single step using Eqs.(4.40) and (4.41).

4.4 Boundary Constraints & Optimization Objectives

Boundary conditions refer to the limitations and constraints imposed on the design process. In the automotive domain, these conditions encompass various factors such as technical requirements, safety regulations, industry standards, environmental considerations, and customer expectations. The purpose of defining boundary conditions is to ensure that the resulting design meets the necessary criteria and aligns with the desired objectives and user expectations. The E/E architecture should accommodate future technology advancements,

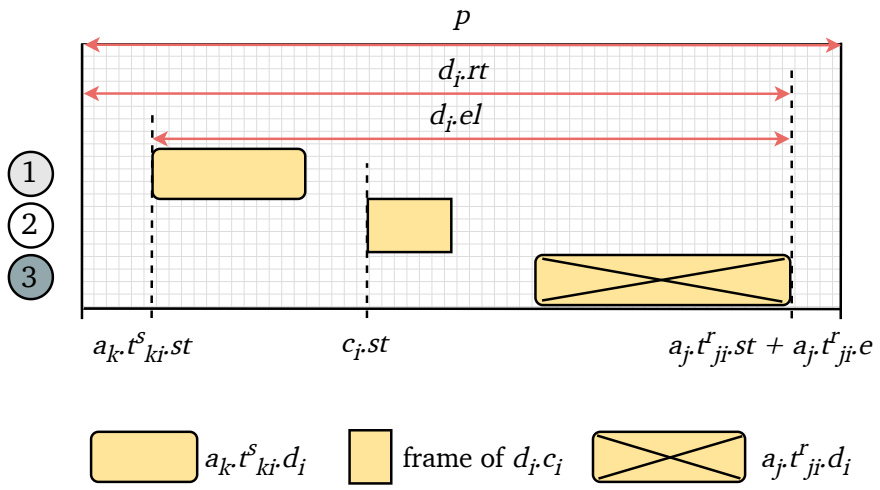


Figure 4.7: End-to-end latency and response time for a communication message ($d_i.el$ and $d_i.rt$, respectively). Number one indicates $a_k.t^s_{ik}$ slot, as a sender, number two represents c_i frame. As a receiver, $a_j.t^r_{ji}$ slot is shown by number three [AMK23].

new features, and vehicle variants. Boundary rules include the ability to integrate additional components, support software updates, and provide a scalable platform for different vehicle models and configurations. Optimization goals in design automation for a vehicle's E/E architecture comprise defining the objectives to be achieved during the design process, as illustrated in Chapter 2. These objectives typically involve enhancing performance, efficiency, safety, and cost-effectiveness. By leveraging computational methods and algorithms, designers aim to optimize various aspects of the automotive design [AHK22; AFK21a; AFK21b; AFK20; AMK23].

As a result, several boundary conditions as requirements and optimization goals are supported by the E/E Designer, which are explained in the following.

4.4.1 End-to-End Latency

In the automotive communication domain, end-to-end latency refers to the time it takes for a message to be transmitted from the sender to the receiver and for a response to be received. This latency can be affected by a number of factors, including the distance between the devices, the type of communication technology being used, the amount of traffic on the communication channel, and the processing time required by the devices to handle the message. This metric considers all the delays that may occur along the way, such as network congestion, processing delays, and transmission delays. Low end-to-end latency means the system can transmit messages quickly and reliably [Zha+14; AHK22; AMK23]. In the context of automotive real-time communication, low end-to-end latency is important for ensuring the smooth operation of the vehicle and the safety of the passengers and in situations where the car relies on real-time data to make decisions or take actions.

As an example, ADAS relies on real-time communication between sensors and other systems; high latency can lead to delays in processing critical information and may negatively impact the performance of the system. Another example can be a car equipped with a sensor that detects obstacles in the road. If the sensor sends a message to the car's HPCU indicating that an obstacle is present, the HPCU needs to process this information and take the appropriate action (such as applying the brakes) as quickly as possible. If the end-to-end

latency is too high, there may be a delay between when the obstacle is detected and when the car takes action, which can result in a dangerous situation. To reduce the end-to-end latency in automotive communication, various measures can be taken, such as optimizing the communication protocol, using high-speed communication technologies, and minimizing the processing time required by the devices during the design phase.

Based on Figure 4.7, the end-to-end latency of a communication message ($d_i.el$) is the time between the start of the application thread acting as the sender ($a_k.t^s_{ki}.st$) and the finishing of the application thread operating as the receiver ($a_j.t^r_{ji}.st + a_j.t^r_{ji}.e$) of its message chain ($d_i.ch_i$) (see Eq. (4.42) and Figure 4.7) [AMK23].

$$d_i.el = a_j.t^r_i.st + a_j.t^r_i.e - a_i.t^s_i.st. \tag{4.42}$$

Figure 4.7 depicts an example of an end-to-end latency computation for a communication message where the message chain comprises two threads belonging to various applications and a communication task. To minimize the end-to-end latency of each communication message, (4.43) is applied as an optimization goal. In addition, a hard constraint, as (4.44), can be used to limit the end-to-end latency to under a specific maximum bearable value, interpreted as a boundary condition.

$\forall i \in \mathbb{N}; d_i \in \mathcal{D}$:

$$el^i_{min} = min(d_i.el) \tag{4.43}$$

$$d_i.el \leq d_i.el_{max}. \tag{4.44}$$

4.4.2 Response Time

Response time refers to the time it takes for a system to respond to a request or input. This metric is important because it reflects the system's ability to process requests quickly and efficiently. A low response time means that the system is able to handle requests quickly and respond promptly [Zha+14]. In the automotive real-time communication domain, response time is critical because it affects the safety and efficiency of the vehicle. For example, if a vehicle's communication system takes too long to respond to a request from the driver or another system, it can cause a delay or malfunction that can lead to an accident. For example, if a car's brake system has a slow response time, it may take longer for the brakes to engage, potentially leading to an accident. Similarly, suppose car's navigation system has a slow response time. In that case, it may take longer to update and provide accurate directions, leading to delays and potential confusion for the driver. Therefore, the communication system needs to have a fast response time in order to ensure the safety and smooth operation of the vehicle.

The presented computer-aided tool supports the response time optimization as well as the response time condition for a communication message. The response time of a communication message ($d_i.rt$) is explained as the time between the beginning of a period, represented as $p.st$, and the time when the job of the last thread in the corresponding message chain is finished [AMK23]. It is given by

$$d_i.rt = a_j.t^r_{ji}.st + a_j.t^r_{ji}.e - p.st. \tag{4.45}$$

Figure 4.7 shows an example of the response time for a communication message (d_i) similar to the end-to-end latency. In practice, because of the unavailability of resources, $a_k.t^s_{ki}.st \neq p.st$. Note that if $a_k.t^s_{ki}.st = p.st$, the response time becomes equal to the end-to-end latency ($d_i.el = d_i.rt$). Like the end-to-end latency optimization objective, to minimize the response time, which is stated as a time limit that the information of time-triggered systems requires to be updated within this time bound, the Eq. (4.46) is presented. Similar to the end-to-end latency, condition (4.47) can be specified as a boundary constraint to enforce the response time to be within a tolerable maximum limit.

$\forall i \in \mathbb{N}; d_i \in \mathcal{D}$:

$$rt^i_{min} = min(d_i.rt) \tag{4.46}$$

$$d_i.rt \leq d_i.rt_{max}. \tag{4.47}$$

Hint: Note that multicast messages are treated as separate messages when calculating the end-to-end latency. Therefore, the end-to-end latency for a multicast message is determined by the maximum of the individually calculated latencies. Furthermore, each route has its latency for redundant and homogeneous redundant routes, and the maximum latency is considered the final latency for a redundant message. The same policy is applied to the response time.

4.4.3 Resource Utilization (RU)

Resource usage optimization is crucial for embedded system developers to avoid facing limited resources. Load balancing can prevent irregularly overloading some control nodes while leaving others idle. Therefore, a boundary condition and an optimization goal are defined to automatically assign applications to available resources, such as ECUs and HPCUs, including cores and processors, while satisfying the resource usage rule.

Based on Eq. (4.48), as an optimization goal, the number of mapped applications on each control node is minimized, i.e., the applications a_j are distributed on control nodes $n^{cz}_i.a_j$ possibly to keep load balancing of each control node the same [AMK23]. The Eq. (4.49) is utilized to average the usage of control nodes as a boundary constraint. The number of mapped applications on each control node must be greater than or equal to a tolerable bound denoted as the apn value, which is the total number of applications divided by the total number of control nodes according to Eq. (4.49). As a result, the applications are as equally as possible distributed on control nodes to maintain the load balancing of each control node. Condition (4.50) shows a boundary constraint similar to Eq. (4.49), which is considered for control node cores such as HPCU cores and apc, indicating a bearable limit [AMK23].

$\forall i, j \in \mathbb{N}; m_{ij} \in \mathcal{M}; a_j \in \mathcal{A}; n^{cz}_i \in \mathcal{N}$:

$$ru^i_{min} = min(\sum_{j \in \mathbb{N}} m_{ij}.n^{cz}_i.a_j) \tag{4.48}$$

$$\sum_{j \in \mathbb{N}} m_{ij}.n^{cz}_i.a_j \geq apn \tag{4.49}$$

$$\sum_{j \in \mathbb{N}} m^c_{ij}.n^{cz_c}_i.a_j \geq apc. \tag{4.50}$$

Maximum Resource Usage

It is necessary to consider maximum resource utilization, such as the maximum usage of memory and processor of an ECU, due to vehicle software updates. In other words, a specific amount of memory and processor usage must be left intact during the design-time configuration process, as a vehicle software update may require extra memory and processor capacities. The updated software may put additional load on the processor; therefore, ensuring that the processor can handle the increased computational requirements without affecting the overall system performance is required. This can be achieved by having enough unused processor usage and sufficient computational power on the processor [AHK22; AMK23].

Memory: The amount of available memory on the ECU should be determined, and the memory requirements of the updated software should be assessed. It must be ensured that the new software fits the memory constraints and leaves enough space for other critical functions. Hence, the proposed tool considers the maximum memory utilization, formulated as Eq. (4.51). Based on this boundary condition, the sum of application memories that is mapped on n_i^{cz} must be within the maximum memory capacity of the node ($n_i^{cz}.m_{max}$) [AMK23].

$\forall i, j \in \mathbb{N}; m_{ij} \in \mathcal{M}; a_j \in \mathcal{A}; n_i^{cz} \in \mathcal{N}$:

$$\sum_{j \in \mathbb{N}} (m_{ij}.n_i^{cz}.a_j \times a_j.mu) \leq n_i^{cz}.m_{max}. \tag{4.51}$$

ECU: Resource usage optimization is important to embedded systems developers to avoid facing limited resources. Moreover, the load balancing scenario, which is the process of distributing a set of tasks over a set of resources, plays a pivotal role in overall processing and makes it more efficient. Using load balancing avoids irregularly overloading some control nodes while others are idle. Therefore, to automatically assign applications to available resources, e.g., ECU, HPCU including core and processor while optimizing the resource usage, an optimization goal is defined and integrated into presented the model-based framework using Eq. (4.52) to minimize the control nodes utilization [AMK23].

$\forall i, j, m_i \in \mathcal{M}, a_j \in \mathcal{A}, n_i^{cz} \in \mathcal{N}$:

$$ru_{min} = min(\sum_{j \in \mathbb{N}} m_j.n_i^{cz}.a_j) \tag{4.52}$$

, based on Eq. (4.52), the number of mapped applications on each control node is minimized, i.e., the applications a_j are equally distributed on control nodes $n_i^{cz}.a_j$ possibly to keep load balancing of each control node the same. Furthermore, similar to the last two optimization goals, a hard constraint can be formulated to restrict the number of assigned applications on each control node to a maximum mapped number as follows.

$\forall i, n_i^{cz} \in \mathcal{N}$:

$$n_i^{cz}.ru \leq n_i^{cz}.ru_{max}. \tag{4.53}$$

4.4.4 Load Balancing in Vehicle Communication Network

Overloading of communication links in automotive networks can occur when the network experiences a higher volume of data traffic than it can handle. As vehicles become more

connected and advanced, the demand for data transmission between various components and systems increases, which can strain the communication infrastructure. This overload can lead to delays in message delivery, increased latency, and even system failures. Load balancing is also a technique used to manage communication messages in a network. It aims to evenly distribute the communication traffic across these links, avoiding overutilizing any specific path and maximizing the network's capacity. This helps prevent bottlenecks, reduces latency, and improves overall network performance. Communication messages or data packets can be distributed across multiple links based on predefined load-balancing algorithms.

Link Occupation Rate (LOR)

The presented tool helps to reduce the overall density of communication tasks and prevents the overloading of links. It also decreases the number of scheduling competitions which mitigates the system's complexity, resulting in shorter synthesis times. To accomplish this, the introduced framework uses an optimization objective as (4.54) that assists in lessening the LOR ratio [AMK23].

$\forall i, j \in \mathbb{N}; d_i^{out} \in \mathcal{D}; l_j \in \mathcal{L}$:

$$lor_{min} = min(\sum_{i \in \mathbb{N}} d_i^{out}.l_j) \tag{4.54}$$

$$\sum_{i \in \mathbb{N}} d_i^{out}.l_j \leq l_j.lor_{max} \tag{4.55}$$

, according to Eq. (4.54), during exploration of message routing for each communication message d_i, the sum of possible outgoing messages d_i^{out} over each link l_j must be minimized, leading to frame balancing for each link. Correspondingly to the last optimization goals, the LOR can also be forced to be within a maximum acceptable bound by specifying a boundary constraint as (4.55) [AMK23].

Maximum Bandwidth Utilization

By defining the maximum bandwidth usage for each communication link, E/E system integrators can efficiently allocate the available network resources. It allows them to ensure that critical systems and applications receive the necessary bandwidth to function reliably. It helps prevent network overloading and ensures bandwidth is distributed appropriately among different components. E/E architects can optimize the overall network's performance by setting maximum bandwidth limits, and they can analyze the bandwidth requirements of different components and applications and design the network topology accordingly. This allows them to avoid bottlenecks and ensure smooth data flow within the network. In some cases, automotive networks may need to provide guarantees on bandwidth availability. For instance, certain data transmissions must occur within specific time constraints in time-critical applications like autonomous driving. Specifying maximum bandwidth utilization ensures that the required bandwidth is reserved and available when required. For future scalability and expansion of the network, considering maximum bandwidth utilization helps. As automotive technologies evolve and new applications are introduced, the bandwidth requirements may change. By defining the maximum bandwidth usage in the design phase, future growth can be accommodated and ensured that the network can handle increased data volumes and communication demands [AMK23].

Therefore, the LOR constraint can be extended by adding the bandwidth of each link to ensure that each network link's maximum bandwidth utilization is not exceeded during finding message paths. This leads to condition (4.56), which states that the sum of bandwidths

used by communication tasks over each link l_j must be less than or equal to the maximum bandwidth specified by the user for the same link ($l_j.bw_{max}$). Here, $transT$ represents the transmission time of each frame [AMK23].

$\forall i, j \in \mathbb{N}; d_i^{out} \in \mathcal{D}; l_j \in \mathcal{L}$:

$$\sum_{i \in \mathbb{N}} (d_i^{out}.l_j \times d_i^{out}.c_i.fl/transT) \leq l_j.bw_{max}. \tag{4.56}$$

4.4.5 Cost Reduction (CR)

Cost optimization in automotive network topologies involves designing and implementing network architectures that balance cost efficiency and performance. Since each communication link in the E/E Designer can have different types, such as Ethernet, FlexRay, and TTCAN-bus, and subsequently varying costs, a cost optimization goal constraint is defined (4.57) to minimize the cost of the created paths. Similarly, this objective can be applied to other software/hardware components [AFK21a; AFK21b; AFK20; AMK23].

$\forall i, j, d_j^{out} \in \mathcal{D}, l_i \in \mathcal{L}$:

$$cr_{min} = min(\sum_{i \in \mathbb{N}} cost.d_j^{out}.l_i) \tag{4.57}$$

$$\sum_{i \in \mathbb{N}} cost.d_j^{out}.l_i \leq path.cost_{max}. \tag{4.58}$$

In Eq. (4.72), the cost of a chosen path is forced to be less than or equal to the boundary cost value of a path specified by the user. This is considered a cost boundary constraint [AMK23; AFK21a].

4.4.6 Reliability

Reliability is an essential concept in the context of ISO 26262 [ISO18] because it is one of the critical factors that must be considered when designing and implementing safety-related systems. Reliability refers to the ability of a system or component to perform its intended function consistently and without failure over a specified time, as expressed in Chapter 2. There are several ways in which reliability is addressed in ISO 26262 [ISO18], including reliability models and reliability testing. Reliability models are mathematical models that are used to predict the probability of failure of a system or component based on various factors, such as operating conditions and environmental factors. Reliability testing involves subjecting a system or component to simulated or actual operating conditions in order to measure its performance and identify any potential failure points [Xie+18].

In addition to reliability models and testing, ISO 26262 also specifies requirements for the design and implementation of safety measures, such as redundant systems and fail-safe mechanisms, to ensure the reliability of safety-related systems. To compute the reliability, there are various metrics which can be used. For instance, the failure rate of E/E system components can be defined by E/E architects, and the reliability, e.g., routing and mapping, can be calculated accordingly. As stated in Chapter 2, our proposed tool follows the component-based approach using the reliability RBD method [Com+17; Men16].

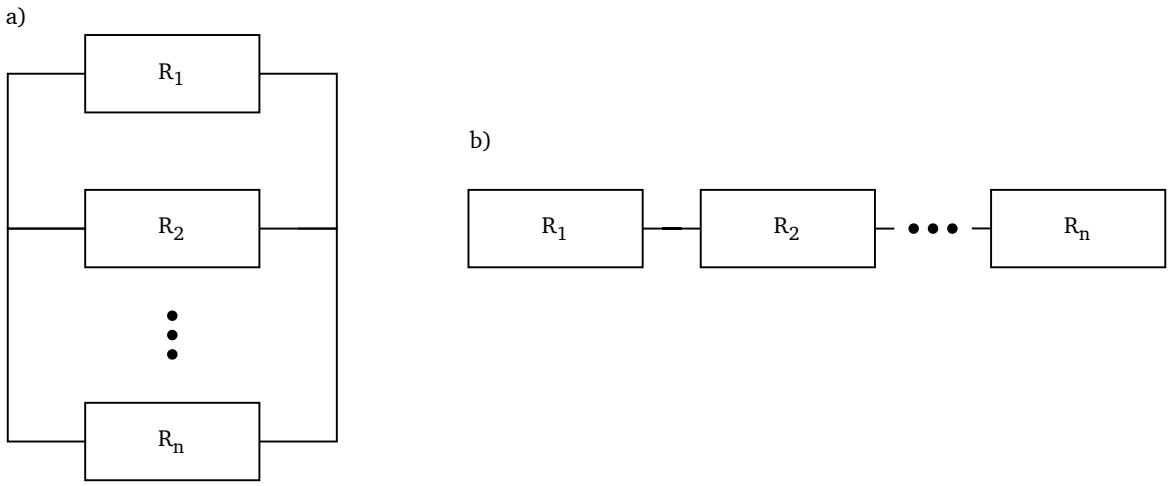


Figure 4.8: a) A parallel system including multiple elements. b) A series system comprising multiple elements.

According to Figure 4.8, the reliability can be computed by considering individual components of the system and taking serial and parallel configurations into account [Men16]. Firstly, the user specifies the reliability/failure rate for each individual (hardware) component. These failure rates can be obtained from norms, data sheets, field experience values, or by exploring thermal, electrical, or mechanical mission profiles. Secondly, the overall reliability score of the system is calculated from the individual reliability scores; thirdly, the computed overall reliability score is constrained by an expected reliability score. Alternatively, constraints can be formulated by requiring reliability scores for sub-systems. These expected reliability scores can be derived from required ASIL levels.

In a series system, as depicted in Figure 4.8 (b), the failure of any component can lead to the failure of the entire system. Therefore, the overall reliability of the series system can be calculated as the product of the individual reliabilities [Men16]:

$$R_s(t) = \prod_{i=1}^{n} R_i(t). \tag{4.59}$$

Given that λ_i represents the failure rate and R_i indicates the reliability of the i-th component:

$$R_i(t) = e^{-\lambda_i t} \tag{4.60}$$

and considering Eq. (4.59), Eq. (4.60) can be expressed as follows:

$$R_s(t) = e^{-(\sum_{i=1}^{n} \lambda_i)t}. \tag{4.61}$$

A parallel system, as shown in Figure 4.8 (a) and which models redundancy, only fails if all its components fail. As a result, the system reliability is calculated as follows [Men16]:

$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)) \tag{4.62}$$

and by using the Eq. (4.60), Eq. (4.62) can be stated as below

$$R_p(t) = 1 - \prod_{i=1}^{n} (1 - e^{\lambda_i t}) = \sum_{i=1}^{n} e^{-\lambda_i t} - e^{-(\sum_{i=1}^{n} \lambda_i)t}. \tag{4.63}$$

MTTF

As mentioned in Chapter 2, MTTF stands for mean time to failure, and it is a commonly used metric in the automotive domain and other engineering fields. MTTF represents the average time elapsed between the start of a system or component's operation and the occurrence of its first failure. It is a measure of reliability and indicates the expected lifespan of a component or system under normal operating conditions. In the automotive domain, MTTF assesses the reliability of various vehicle components such as engines, transmissions, electrical systems, braking systems, and other critical parts. Depending on the specific application, MTTF is typically expressed in time units, such as hours, miles, or kilometers. It is an important factor considered during automotive systems' design and development stages to ensure that they meet the required reliability standards and customer expectations [AMK23; AHK22].

As stated in Chapter 2, if the failure rate remains constant over time, it can be assumed that

$$MTTF = \frac{1}{\lambda} \tag{4.64}$$

, and taking Eq. (4.64) into account, the MTTFs for the series and parallel systems, as visualized in Figure 4.8, are computed using Eqs. (4.65) and (4.66), respectively, as follows [Men16]:

$$MTTF_{seri} = \int_0^\infty e^{-(\sum_{i=1}^n \lambda_i)t} dt = \frac{1}{\sum_{i=1}^n \lambda_i} \tag{4.65}$$

$$MTTF_{par} = \int_0^\infty \sum_{i=1}^n e^{-\lambda_i t} - e^{-(\sum_{i=1}^n \lambda_i)t} dt = \sum_{i=1}^n \frac{1}{\lambda_i} - \frac{1}{\sum_{i=1}^n \lambda_i}. \tag{4.66}$$

To calculate the reliability of a path in an automotive network when the failure rate or MTTF of each link is provided as a constant value, a similar approach to the one used for the series and parallel systems can be applied.

Single Message Routing

The introduced framework calculates the reliability of the entire path by multiplying the reliabilities of all activated links, as they are in series, and this calculation is performed in a single step. Reliability is specified as a boundary constraint and an optimization objective in the proposed tool. During the process of creating paths for communication messages from senders to receivers, the reliability of each possible path is computed, ensuring that the generated route meets the specified reliability boundary value (see Eqs. (4.67) and (4.68)). The same method is employed to find the most optimized route regarding reliability [MAK24].

$$R_{single}(t) = \prod_{i=1}^{n} R_{l_i}(t) \tag{4.67}$$

$$R_{single}(t) = e^{-(\sum_{i=1}^n d^i_{out}.l_i.\lambda_i)t} \tag{4.68}$$

$$MTTF_{single} = \int_0^\infty e^{-(\sum_{i=1}^n d^i_{out}.l_i.\lambda_i)t} dt = \frac{1}{\sum_{i=1}^n d^i_{out}.l_i.\lambda_i}. \tag{4.69}$$

Based on Eq. (4.68), the reliability of each path is calculated only for the activated links. Determining which links are activated depends on the failure rate of each link, as it aligns

with the intended reliability optimization goals. In addition, the MTTF for a single path is computed according to Eq. (4.69), where only the activated links' failure rates are considered.

Redundant Message Routing

The reliability formula for the parallel system, as explained in Eq. (4.62), is employed to calculate the reliability for redundant paths. According to (4.70), the reliability for the redundant message routing, which consists of two single paths (depicted as the upper limit of the product in Eq. (4.70), is computed by treating each single path, as part of the redundant routings, as an element of a parallel system containing two elements. In Eq. (4.70), the single path's reliability is obtained similar to Eq. (4.67) [MAK24].

$$R_{redundant}(t) = 1 - \prod_{j=1}^{2}(1 - R_{single_j}(t)). \tag{4.70}$$

In the proposed system model, the uncertainty of reliability is not taken into account. This means that the failure rates of the E/E components, as provided by the user, are treated as fixed values, not as ranges. In other words, uncertain optimization is not included in the above-mentioned equations.

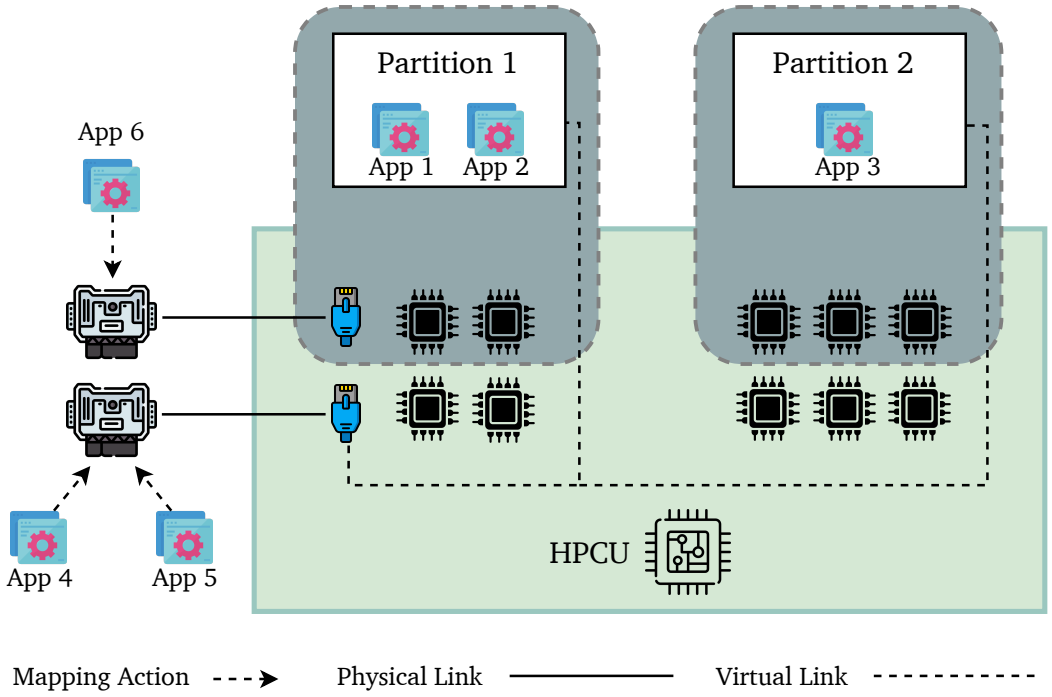


Figure 4.9: Graphical representation of the functioning of the hypervisor constraints implemented in the metamodel presented in Chapter 4, exemplified by the transmission of communication messages. This exemplary scenario shows an HPCU comprising 10 CPU cores and two communication interface devices running two partitions. Partition 1 has exclusive control over two cores and one interface device, while partition 2 controls three cores, as indicated by the dark green boxes. The rest of the CPU cores and the other interface device are shared among both partitions. Assume that communication messages are sent between a_1 and a_6, a_2 and a_4, and a_3 and a_5. The two concepts of exclusive resource allocation and resource sharing are shown in this figure [MAK24].

4.4.7 Hypervisor-related Constraints

Virtualization is increasingly important in the automotive industry as it can facilitate many aspects of the E/E design process by improving flexibility, efficiency, and security. This technology is explained in Chapter 2. This section explains the hypervisor-related constraints integrated into the E/E Designer tool [MAK24].

Exclusive Resource Allocation for Partitions of Type-1 Hypervisor

A fundamental aspect of hypervisor technologies is the exclusive allocation of resources, which refers to dedicating specific hardware resources to individual virtual machines. This ensures that each VM has its isolated portion of CPU, memory, storage, and other hardware resources, preventing interference and contention among VMs [MAK22]. In safety-critical systems, such as those used in automotive applications, sharing critical resources like the CPU is generally not a common practice due to stringent requirements for reliability, predictability, and fault tolerance. Sharing CPU resources among multiple tasks or applications introduces uncertainty and potential contention for processing time, leading to unpredictable behavior and jeopardizing the system's safety. In the proposed framework, exclusive resource allocation is realized by treating each component of the virtualized hardware as an independent entity. Thus, the exclusivity of these resources is captured by the mapping and routing constraints already defined in the framework. For example, Figure 4.9 shows a hardware platform running a hypervisor with two partitions. Through exclusive resource allocation, partition 1 and all its exclusively used resources are treated as stand-alone nodes in the modeling, implicitly ensuring the exclusiveness of the communication interface device and the two CPU cores. The physical link is only available to transfer communication messages between a_1 and a_6 without requiring an explicit constraint [MAK24].

Resource Sharing Among Partitions of Type-2 Hypervisor

In addition to splitting up the available resources as a whole, hypervisor technologies also allow the sharing certain resources among multiple partitions, as explained in Chapter 2. In vehicle E/E architectures, resource sharing is typically used for non-critical I/O devices,

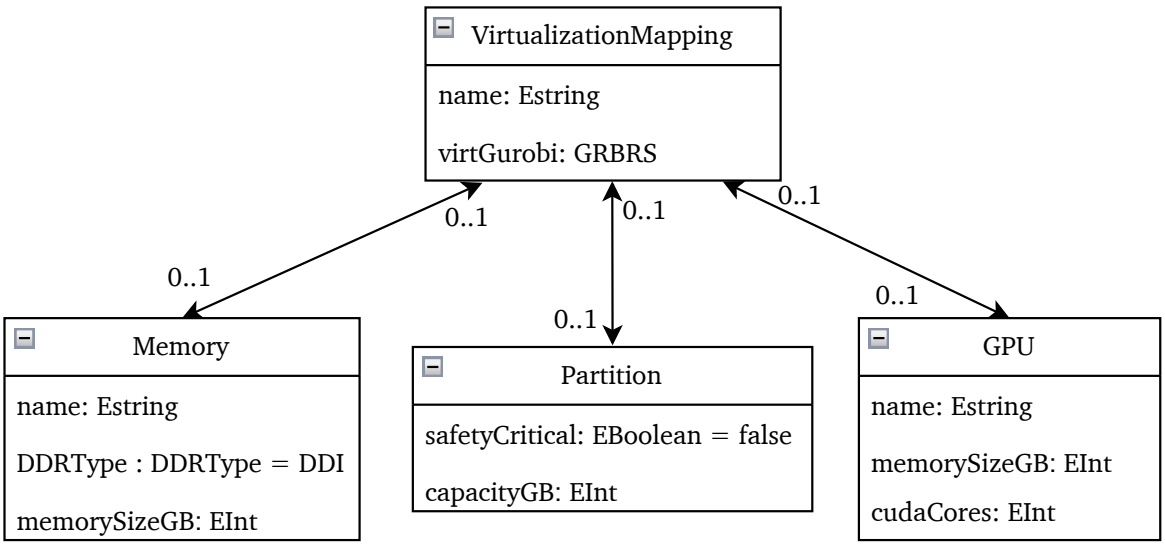


Figure 4.10: The UML classes required for the hypervisor-related constraints.

such as communication interfaces or sensors with less stringent timing requirements, and networking interfaces for diagnostics or non-real-time data transfer.

In the E/E Designer, this concept is realized by introducing virtual links between the shared resource and the respective partitions. These virtual links are not visible in the frontend but have the same properties as regular links. The bottom-most interface device is shared among both partitions in the example presented in Figure 4.9. Interface sharing is achieved by introducing virtual links that connect the partitions with the interface. Thus, messages can be transported between the interface and a_2 or a_3 in parallel; however, only one communication message can be transmitted between the ECU and the interface at a time. Thereby, only the interface device, subject to the hypervisor, controls the flow of messages.

In the UML class diagram shown in Figure 4.10, which is part of the metamodel explained in Chapter 4, the VirtualizationMapping UML class is connected to the hardware resource classes, including GPU and memory, and the Partition class via bidirectional association relations. Other than that, the VirtualizationMapping class has the same relationships to other classes as the Link class in order to replicate its functionality. The basic idea is similar to that presented earlier. Instead of explicitly programming all the constraints necessary to enforce the correct behavior of all components during resource sharing, the virtual links are a way to encode these constraints into the metamodel. Virtual links mimic the independence of access requests. In that way, the shared interface acts as a multiplexer on the partition's messages, imitating the actual physical implementation [MAK24].

Memory Constraint

Using virtualization, the memory of an HPCU can be shared among multiple partitions. From a modeling perspective, this corresponds to the requirement that the sum of the memory utilization of the individual partitions must be smaller or equal to the total available memory of the HPCU:

$\forall h \in \mathcal{H}$:

$$\sum_{i \in \mathbb{N}} p_i.maxMemory \leq h.memory. \tag{4.71}$$

According to constraint (4.71), the sum of allocated memories for hypervisor partitions running on an HPCU must be less than or equal to the memory capacity of an HPCU. $\mathcal{H}$ indicates a set of HPCUs, and p represents a hypervisor partition executing on an HPCU.

4.5 Multi-Objective Optimization

Multi-objective optimization (MOO) is a branch of optimization that deals with problems involving multiple conflicting objectives. In traditional optimization, the goal is to find a single optimal solution that minimizes or maximizes a single objective function. However, in many real-world scenarios, multiple objectives must be simultaneously considered, and these objectives may conflict. The MOO aims to find solutions representing the trade-offs between the different objectives rather than a single optimal solution. These solutions are known as Pareto-optimal or non-dominated solutions. A solution is considered Pareto-optimal if no other solution improves one objective without worsening at least one objective [Gun18; AHK22; AFK21b; AMK23].

4.5.1 Gurobi Multi-Objective Optimization

Gurobi is a widely used commercial optimization solver that provides powerful capabilities for solving optimization problems, including multi-objective optimization. Gurobi supports MOO through its Python interface, allowing users to define and solve problems with multiple goals [Gur22]. Gurobi solver supports two MOO approaches, each explained below and summarized below.

Hierarchical Approach

In a hierarchical or lexicographic approach, a priority is allocated to each objective, and optimization is performed by considering objectives in descending order of priority. At each stage, the best solution for the current objective is determined while ensuring it does not negatively impact the quality of solutions for higher-priority objectives. Priorities for each objective can be specified using the "*setObjectiveN*" function or the "*ObjNPriority*" attribute. Priorities are represented as whole numbers, with higher values indicating higher priorities. The default priority for an objective is 0. For instance, consider a model with two objectives, where the first objective has a priority of 7 and the second has a priority of 3. Assuming the optimal solution for the first objective is 80. In this case, the solver will identify the solution that optimizes the negative of the second objective among all solutions that achieve the first objective value of 100 [Gur22; AMK23].

Blended Approach

To generate a single objective, a blending technique combines linearly various objectives by assigning a weight to each objective using the "*setObjectiveN*" function. Alternatively, the default weight of 1.0 can be adjusted by utilizing the "*ObjNWeight*" attribute in conjunction with "*ObjNumber*" [Gur22].

4.5.2 Multi-Objective Optimization in the E/E Designer Framework

Given the above representation of the presented tool optimization goals, different classes of objectives can be considered in the MOO problem integrated into the framework. For instance, one combination of objectives can simultaneously minimize the end-to-end latency, response time, LOR, and resource usage, with different predefined priorities as shown in Eq. (4.72) to (4.75). The hierarchical approach, as illustrated above, is utilized where it assigns a priority to each objective and optimizes the objectives in reducing priority order [AMK23].

$$Obj_1\ (el_{min}, 4) \tag{4.72}$$

$$Obj_2\ (rt_{min}, 3) \tag{4.73}$$

$$Obj_3\ (LOR_{min}, 2) \tag{4.74}$$

$$Obj_4\ (ru_{min}, 1) \tag{4.75}$$

For example, considering the objective (4.72), the number four represents the priority of the objective; moreover, the higher number is interpreted as a higher priority. As a result, in the shown combination of MOO goals, the end-to-end latency has the highest priority (4.72), while the resource usage has the least (4.75).

Algorithm 2: CMR

Input: $N = \{n_1, n_2, ..n_n\}, D = \{d_1, d_2, ..d_d\}$, m^s, $m^d \in \{0,1\}$, and, $n, d \in \mathbb{N}$
Output: Creation of single, multi-cast, redundant, and homogeneous redundant routes in a single-step solving

```
1  for i ← 0 to d do
2      for j ← 0 to n do
3          for k ← 0 to size.n_j.get(links) do
4              nd_in_list ← addAll(n_j.get(k).get(d_in));
5              nd_out_list ← addAll(n_j.get(k).get(d_out));
6          end
7          for r ← 0 to size.nd_in_list do
8              if nd_in_list.get(r).d = d_i then
9                  d_in_list.add(nd_in_list.get(r));
10             end
11         end
12         for s ← 0 to size.nd_out_list do
13             if nd_out_list.get(s).d = d_i then
14                 d_out_list.add(nd_out_list.get(s));
15             end
16         end
17         mapping_list ← addAll(n_j.get(mapping));
18         for t ← 0 to size.mapping_list do
19             mappingthreads_list.addAll(mapping_list.get(t).get(app).get(thread));
20             for u ← 0 to size.mappingthreads_list do
21                 if mappingthreads_list.get(u).get(receive d) ≠d_i or
                    mappingthreads_list.get(u).get(send d) ≠d_i &
                    mappingthreads_list.get(u).get(receive d) = d_i) then
22                     m^d ← mappingthreads_list.get(u).get(m_binary)
23                 end
24                 if mappingthreads_list.get(u).get(send d) ≠d_i or
                    mappingthreads_list.get(u).get(receive d) ≠d_i &
                    mappingthreads_list.get(u).get(send d) = d_i) then
25                     m^s ← mappingthreads_list.get(u).get(m_binary)
26                 end
27             end
28         end
           // use d_out_list, d_in_list, m^s, and m^d for the following if
              conditions
29         if single route for d_i becomes true then
30             Apply constraints (4.7), (4.8), (4.9), (4.10), and (4.11);
31         end
32         if multi-cast route for d_i becomes true then
33             Apply constraints (4.12), (4.13), (4.14), (4.15), and (4.16);
34         end
35         if redundant route for d_i becomes true then
36             Apply constraints (4.17), (4.18), (4.19), (4.20), and (4.21);
37         end
38         if homogeneous redundant route for d_i becomes true then
39             Apply constraints (4.22), (4.23), (4.24), (4.25), (4.26), (4.27), and (4.28);
40         end
41     end
42 end
```

4.6 Single-Step Solving Algorithms

As depicted earlier, the illustrated framework system model, comprising automated mapping, automatic message routing, and time-triggered scheduling, is solved in a single step. This approach allows us to use the same joint constraint set for mapping, routing, and scheduling throughout the entire optimization run.

When mapping, routing, and scheduling are treated independently, each step introduces its own set of constraints. However, merging these constraints into a single joint set eliminates redundancy and reduces the overall number of constraints. This consolidation simplifies the problem formulation and reduces computational complexity. In addition, these problems are tightly interconnected. Changes in one aspect can affect the feasibility and optimality of the others. The optimization algorithm can explicitly consider these interdependencies, leading to better solutions using the joint set of constraints. For example, when a new application is assigned to a specific ECU/HPCU, it may impact the optimal routing or require adjustments in the schedule. With a joint constraint set, changes made during the optimization process can be propagated efficiently across mapping, routing, and scheduling. When a solution is modified to satisfy a particular constraint, the impacts can be automatically reflected in the other aspects, ensuring consistent and coherent solutions. Furthermore, the search space for the optimization algorithm is reduced. This reduction occurs because incompatible solutions that violate any aspect of the joint constraints can be pruned early on. Consequently, the optimization algorithm can focus on exploring only the feasible and promising solution space, leading to faster convergence [AFK21b; AFK21a; Smi+17; AMK23]. Therefore, several algorithms are applied to satisfy this approach. In this section, three of these algorithms are explained in detail.

4.6.1 CMR Algorithm

Algorithm 2 is used to create various types of routes, such as single, multi-cast, redundant, and homogeneous redundant paths, while considering the automated mapping. According to the CMR algorithm, all nodes pass through multiple conditions for each communication message (d). Initially, each node's incoming and outgoing messages are extracted and stored in a list (lines 3 to 6). Subsequently, the d_{in} and d_{out} carrying the same communication message as the current iteration (d_i in lines 8 and 13) are obtained from the incoming and outgoing lists (lines 7 to 16). To identify the potential mapping variables for both sender and receiver, lines (17) to (28) are introduced.

Firstly, all mapping variables associated with each node are stored in a list (line 17). Then, the mapping variables for the threads are extracted from the mapping list and saved separately (19). The relevant mapping variable for receiving the current iterated communication message (d_i) is investigated and saved using *mappingthread_list* (lines 21 to 23). The same process is followed to identify the related mapping variable for the sender of d_i according to lines (24) to (26). Finally, in the last step, the routing constraints for a communication message d_i are applied based on its routing requirement, including single, multicast, redundant, and homogeneous redundant. This is done by utilizing the d^{out}_list, d^{in}_list, m^s, and m^d (lines 29 to 40) [AMK23].

4.6.2 CSCT Algorithm

Algorithm 3 is utilized to compute the starting time of each communication task, following the time-triggered scheduling and non-conflict frames, only for activated paths. The details of

Algorithm 3: CSCT

Input: $N = \{n_1, n_2, ..n_n\}, L = \{l_1, l_2, ..l_l\}, C = \{c_1, c_2, ..c_c\}$, $D_{out} = \{d_1^{out}, ..d_d^{out}\}$, $v, m_1, m_2 \in \{0,1\}$, and $n, l, c, d, \lambda, \kappa \in \mathbb{N}$

Output: Calculated time-triggered schedules for communication tasks only for activated paths in a single-step

```
1  for i ← 0 to n do
2      for j ← 0 to l do
3          for a ← 0 to c do
4              for b ← 0 to c do
5                  if a ≠ b & a < b then
6                      Calculate w_i and w_j using (4.36);
7                      λ ← maximum of w_i;
8                      κ ← maximum of w_j;
9                      for z ← 0 to d do
10                         if d.c_a is d.d_z^out then
11                             m_1 ← d_z^out.c_a;
12                         end
13                         else if d.c_b is d.d_z^out then
14                             m_2 ← d_z^out.c_b;
15                         end
16                     end
17                     v ← QuadExpr.add(1, m_1, m_2);
18                     for r ← 0 to λ do
19                         for e ← 0 to κ do
20                             Calculate c_a.ST and c_b.ST only for activated paths using
                               condition (4.36);
21                         end
22                     end
23                     for u ← 0 to c do
24                         Apply constraints (4.37) and (4.38);
25                     end
26                 end
27             end
28         end
29     end
30 end
```

this algorithm are illustrated as follows. For each node, all links pass through the conditions specified for each pair of communication tasks (lines 3 and 4). For each non-repetitive pair of tasks (line 5), all conditions from line (6) to (24) are applied. The w_i and w_j presented in condition (4.36) are calculated (line 6). After that, for each out-going data (d^{out}), the related d^{out} of each task is extracted (lines 8 to 15), and their multiplication is assigned to a binary variable (v) (line 17) in order to trigger the schedule computation only for the activated routes. In the next step, for each calculated λ and κ in lines (7) and (8) respectively, the constraint set introduced in (4.36) are applied (line 18 to 26). Finally, for all communication tasks, the Eqs. (4.37) and (4.38) are executed.

Algorithm 4: PD

Input: $D = \{d_1, d_2, ..d_p\}, L = \{l_1, l_2, ..l_p\}, C = \{c_1, c_2, ..c_p\}$, $D_{out} = \{d_1^{out}, ..d_p^{out}\}$, $t \in C$, and $p \in \mathbb{N}$

Output: Path depndency for communication tasks schdules only for activated paths

```
1  for i ← 0 to D do
2      for j ← 0 to L do
3          for a ← 0 to C do
4              if d.c_a = d_i then
5                  t ← c_a;
6              end
7          end
8          link_list ← addAll(l_j.get(dest_node).get(links));
9          for z ← 0 to size.link_list do
10             if link_list.get(z).get(source_node) = l_j. get(dest_node) &
                link_list.get(z).get(dest_node) ≠ l_j. get(source_node) then
11                 task_list←addAll (link_list.get(z).get(task));
12             end
13         end
14         d_out_list ← add (l_j.d_out);
15         for b ← 0 to size.d_out_list do
16             if d_out_list.get(b).d = d_i then
17                 id_out_list ← add (task_list.get(u));
18             end
19         end
20         for u ← 0 to size.task_list do
21             if task_list.get(u).d = d_i then
22                 itask_list ← add (task_list.get(b));
23             end
24         end
25         for r ← 0 to size.id_out_list do
26             for e ← 0 to size.itask_list do
27                 if id_out_list.get(r).get(link) = t.get(link) then
28                     Apply constraint (4.39);
29                 end
30             end
31         end
32     end
33 end
```

4.6.3 PD Algorithm

To meet the path dependency, the PD algorithm is introduced. The path dependency is only triggered for activated paths. In algorithm 4, all links pass through several conditions for each communication message (d). Firstly, the task carrying the same message as d_i for all tasks is obtained and saved in the variable t (lines 3 to 7). The subsequent tasks over subsequent links are investigated and stored in a list (lines 8 to 13). Then, from line (14) to (19), the d^{out}s related to the same d_i are retrieved and saved in a list. In addition, all tasks with the same communication message are acquired from the list of subsequent tasks (lines 20 to

24). In the last step, for each variable of the last created list regarding d^{out} (line 25), all variables of the task list generated in line (22) pass through an if condition, which checks if the extracted task (t) in line 5 has the same related link as each d^{out} listed in id_{out}_list. If the condition is fulfilled, the Eq. (4.39) is applied.

4.7 Constraint Formulation as Mixed Integer Programming for Gurobi Optimization Solver

In this section, the formulations for the constraints, which are considered as either-or conditions, are introduced. All constraints described in Section 4.3 can be converted into a MIP problem supported by the Gurobi solver. Each condition is represented as a single inequality or equation. As an example, constraints (4.32) and (4.33) are already in this format. Nevertheless, as the constraints (4.16), (4.31), and (4.36) are either-or conditions, they need to be converted. Two approaches are used to convert such constraints.

The first approach formulates the either-or condition for constraint (4.16) by multiplying two different binary variables x and y to Eq. (4.76). However, only one of these variables can be one at a time, as specified by Eq. (4.77) [AMK23].

$\forall n^{cz}, n_j \in \mathcal{N}; i, j(i \neq j) \in \mathbb{N};\ d_i^{in},\ d_i^{out} \in \mathcal{D};\ m_{ij}{}^s, m_{ij}^r\ \in \mathcal{M}; x, y \in [0,1]$:

$$x \times \sum_{j \in \mathbb{N}} n_j.d_i^{out} - x \times \sum_{j \in \mathbb{N}} n_j.d_i^{in} = x \times (m_{ij}^s - m_{ij}^r) \tag{4.76}$$

$$y \times \sum_{j \in \mathbb{N}} n_j.d_i^{out} \geq y \times (m_{ij}^s - m_{ij}^r) + y \times \sum_{j \in \mathbb{N}} n_j.d_i^{in}$$

$$x + y = 1 \tag{4.77}$$

4.7.1 Big M Method

The second approach is the *Big M* method [Wil13], which is used for constraints (4.31) and (4.36) to apply the logical "OR". This method is a technique used in linear programming to handle problems with constraints that are not easily expressed in the standard form (i.e., constraints with inequalities and/or logical implications). The method introduces a sizeable positive constant (M) to convert these constraints into equivalent constraints that standard linear programming algorithms can handle. The choice of the value of M is crucial for the success of the *Big M* method. It should be a large enough value to ensure that the auxiliary variables are driven to zero whenever possible but not too large to cause numerical instability or difficulty finding an optimal solution. The approach is an iterative process that aims to find a feasible and optimal solution to the linear programming problem while satisfying all the constraints, including those initially not in the standard form [Wil13; Ars06; AMK23].

The *Big M* technique is utilized by introducing a binary decision variable as an auxiliary variable for conditions (4.31) and (4.36). As a result, these conditions can be represented as (4.78) for application thread scheduling and (4.79) for communication task scheduling. In these equations, q indicates the defined binary variable and M is specified as a sizeable constant number. ω represents the total number of possible collisions between application

threads on each control node for (4.78) and γ denotes the total number of possible overlapping between communication tasks over each link for (4.79) [AMK23].

$\forall i, j(i \neq j) \in \mathbb{N};\ t_i,\ t_j \in \mathcal{T}; c_i,\ c_j\ \in \mathcal{C};\ \lambda \in [1, \omega];\ \kappa \in [1, \gamma]$:

$$v \times (t_i.p \times w_i + t_i.st + t_i.e) < v \times (t_j.p \times w_j + t_j.st + q_\lambda \times M_\lambda) \tag{4.78}$$

$$v \times (t_j.p \times w_j + t_j.st + t_j.e) < v \times (t_i.p \times w_i + t_i.st + (1 - q_\lambda) \times M_\lambda)$$

$$r \times (c_i.p \times w_i + c_i.st + c_i.fl/bw + ipg) < r \times (c_j.p \times w_j + c_j.st + q_\kappa \times M_\kappa) \tag{4.79}$$

$$r \times (c_j.p \times w_j + c_j.st + c_j.fl/bw + ipg) < r \times (c_i.p \times w_i + c_i.st + (1 - q_\kappa) \times M_\kappa)$$

4.7.2 Quadratic Expression

To apply the multiplication of Gurobi variables (e.g., v and r in constraints (4.78) and (4.79), respectively) to each variable of inequality as their coefficients, the Gurobi quadratic expression approach is used (*QuadExpr*). A quadratic expression is formed by adding a linear expression to a collection of coefficient-variable triplets representing the quadratic terms. These quadratic expressions are utilized in constructing quadratic objective functions and quadratic constraints [Gur22]. For example, the following code block shows the multiplication of two binary variables ($k = m_1 \times m_2$) and multiplying the result (k) by a linear expression ($expr10$) using *QuadExpr*.

```
GRBVar m1 = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "m1");
GRBVar m2 = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "m2");
GRBVar k = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "k");
// Model k = m1 * m2
GRBQuadExpr quad_expr_k = new GRBQuadExpr();
quad_expr_k.addTerm(1, m1, m2);
model.addQConstr(quad_expr_k, GRB.EQUAL, k, "c1");

// Multiply k by the linear expression
GRBQuadExpr quad_expr = new GRBQuadExpr();
for (int i=0; i < expr10.size(); ++i) {
  double coeff = expr10.getCoeff(i);
  GRBVar var_1 = expr10.getVar(i);
  quad_expr.addTerm(coeff, var_1, k);
}
```

4.8 Discussion

The presented model-based tool incorporates various architectural modules, as depicted in Figure 4.2, to effectively tackle the challenges mentioned earlier. The future development of E/E architectures necessitates modularity and extensibility as crucial requirements. It is essential to ensure that the implemented modules can be modified, enhanced, or replaced independently, with minimal impact on the other modules.

The E/E Designer tool facilitates easy integration of new features due to its utilization of the MDD approach, as described in Section 4.1.2. For instance, event-triggered scheduling algorithms can be seamlessly integrated into this tool by introducing new constraints to the existing constraint set and new classes and attributes to the current framework metamodel, if required. Furthermore, additional automotive safety features can be accommodated by incorporating new safety requirements into the tool's functionality. This adaptability allows the tool to evolve and meet the changing demands of the automotive industry. Another extensibility aspect lies in enhancing the graphical E/E architecture modeler shown in Figure 4.2. This can be achieved by modifying the frontend, as explained in Chapter 6, and adjusting the presented metamodel. By incorporating new requirements and hardware/software properties into the modeler, the tool can effectively capture and represent the evolving complexities of E/E architectures. Moreover, the choice of solver is not limited to the current Gurobi solver. Alternative solvers, such as SMT solvers, can be utilized by adapting the constraint generation algorithms and implementing the necessary changes. This flexibility empowers the computer-aided tool to accommodate different types of solvers and leverage their strengths in solving the introduced constraints. All the MIP constraints within the presented system model, forming the backend of the introduced framework, were implemented using the Java programming language [AGH05] in the Eclipse integrated development environment (IDE) platform [Ecl23].

In conclusion, the introduced framework is a versatile tool that can be extended and enhanced to address a wide range of requirements and challenges in E/E architecture design. Its modularity, extensibility, and adaptability make it well-suited for future developments in the automotive industry. As mentioned in Chapter 1, the proposed framework is open-source and accessible on Github [AMK23] offering the research community an opportunity to explore, utilize, and contribute to its development

5

The Framework Frontend

In this chapter, the frontend of the presented framework is explained. It presents a graphical modeling tool where the user can model a desired car E/E architecture or vehicle topology comprising various hardware/software components. In the following, the details of the framework's frontend are described.

5.1 Modeling

Modeling is a powerful and versatile technique used across various fields to represent, simulate, or describe real-world systems, processes, or phenomena. It involves creating simplified, abstract representations of complex entities to gain insights, make predictions, or solve problems. Modeling aims to capture the essential features and behaviors of the subject being studied while discarding unnecessary details. In today's fast-paced automotive industry, where innovation drives, creating cutting-edge vehicles requires a seamless integration of complex electrical and electronic systems. This is where sophisticated modeling tools step in, empowering automotive engineers to bring their ideas to life with precision and efficiency.

The proposed framework includes a frontend providing the modeling functionality. The user can graphically design E/E architectures and car topologies using the introduced tool. Figure 5.1 presents a designed example utilizing our tool where, after creating a project, the user is able to select various hardware/software components such as a gateway, network switch, application, ECU, HPCU, communication message, communication task, etc [AMK23].

5.1.1 Web-based Modeling Tool

The E/E Designer framework uses a web-based frontend. In the following, several advantages of web-based tools are illustrated.

One of the key advantages of web-based modeling tools is their accessibility. They can be accessed from any device with an internet connection, eliminating the need for specific software installations [AMK23]. This accessibility allows for collaboration and sharing of models across different teams, locations, and devices. In addition, it eases collaboration among team members. Multiple users can work on the same model simultaneously, making collaborating, discussing, and making real-time updates easier. This enhances team productivity and reduces the time spent on model synchronization and version control. Also, changes made

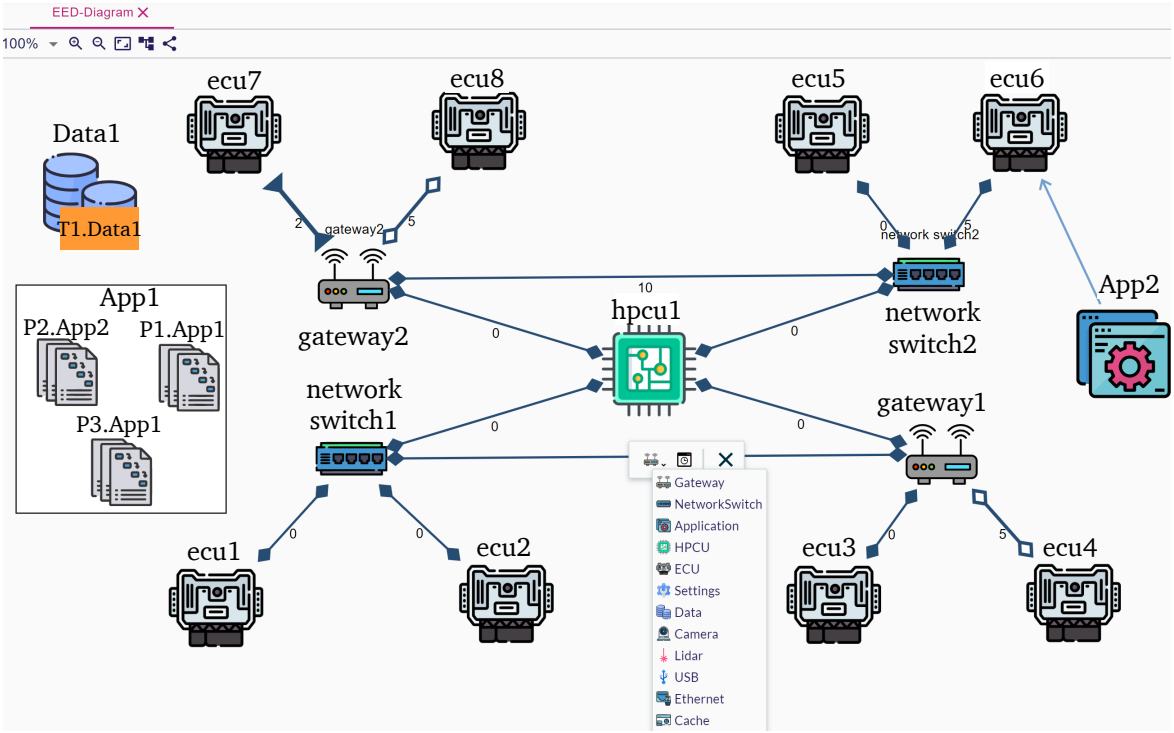


Figure 5.1: An example of zonal E/E architecture model using the presented model-based framework [AMK23].

by one user are instantly visible to all other users. This enables real-time updates and fosters efficient communication and decision-making within the team. It eliminates the need for manual merging of changes and reduces the chances of errors or conflicts in the model. Web-based modeling tools can quickly scale to accommodate projects of varying sizes and complexities. They can handle large models with extensive diagrams, multiple components, and interconnected systems. The cloud-based infrastructure of these tools enables seamless scaling without the need for additional hardware or software upgrades. They often come with built-in version control features. This allows users to track and manage different versions of the model, view revision history, and revert to previous versions if needed. Version control ensures the model remains consistent and provides a safety net for experimentation and exploration.

The web-based frontend stores models and associated data in a centralized location like a cloud server. This centralized storage ensures that the latest version of the model is always accessible and eliminates the risk of data loss due to hardware failures or local storage issues. It also provides a secure backup and facilitates easy sharing and retrieval of models. Another feature of the web-based tools is cross-platform compatibility. This makes them compatible with different operating systems and platforms. Users can access and use these tools on Windows, Mac, Linux, or any other platform that supports a modern web browser. This feature enhances flexibility and convenience for users. Another advantage of web-based tools is integration capabilities with other software tools and systems. They can be easily integrated with version control systems, requirements management tools, issue trackers, and various development environments. Additionally, web-based tools often provide application programming interfaces (APIs) and extensibility options, allowing users to customize and extend the functionality of the tool to suit their specific needs.

The E/E Designer consists of a web-based frontend where all above-explained features are supported.

5.1.2 Drag and Drop Functionality

The E/E Designer tool supports drag-and-drop functionality, a valuable feature in a modeler tool, as it provides an intuitive and user-friendly way to create and manipulate models. Here are some advantages of incorporating drag-and-drop functionality [AMK23]. This feature simplifies the modeling process by allowing users to visually create and arrange elements in the model. Users can drag elements from a palette or toolbox and drop them onto the canvas, eliminating the need for manual drawing or coding. This intuitive approach makes the tool accessible to users with varying technical expertise. Moreover, users can quickly create models by selecting and placing predefined elements onto the canvas. This accelerates the E/E system development process and reduces the time required for the manual creation and positioning of model components. System integrators can focus more on the overall structure and logic of the E/E model rather than spending time on tedious placement and alignment tasks. Model elements can be easily modified and rearranged. Architects can drag existing elements to different locations on the canvas, change their connections, or even drag new elements into existing ones to create hierarchical relationships.

Based on the Figure 5.1, after creating a project, users can select various hardware/software components, such as gateways, network switches, applications (including threads), ECUs (comprising sub-components like processors and memory), HPCUs (including sub-components such as processors, cores, memory, and GPUs), communication messages (in this case, data), communication tasks, and more. Clicking on an empty field brings up a window that displays the available objects that can be selected and generated. Furthermore, users can force applications to be mapped to specific hardware components before the solving step. For example, in Figure 5.1, App_2 is forced to run on ECU_6 (blue arrow) [AMK23].

5.1.3 Full-mesh Topology

In the context of E/E architecture, a full-mesh topology refers to a network configuration where every node (or component) is directly connected to every other node in the system. Each node serves as a point-to-point link with all other nodes, forming an extensive and

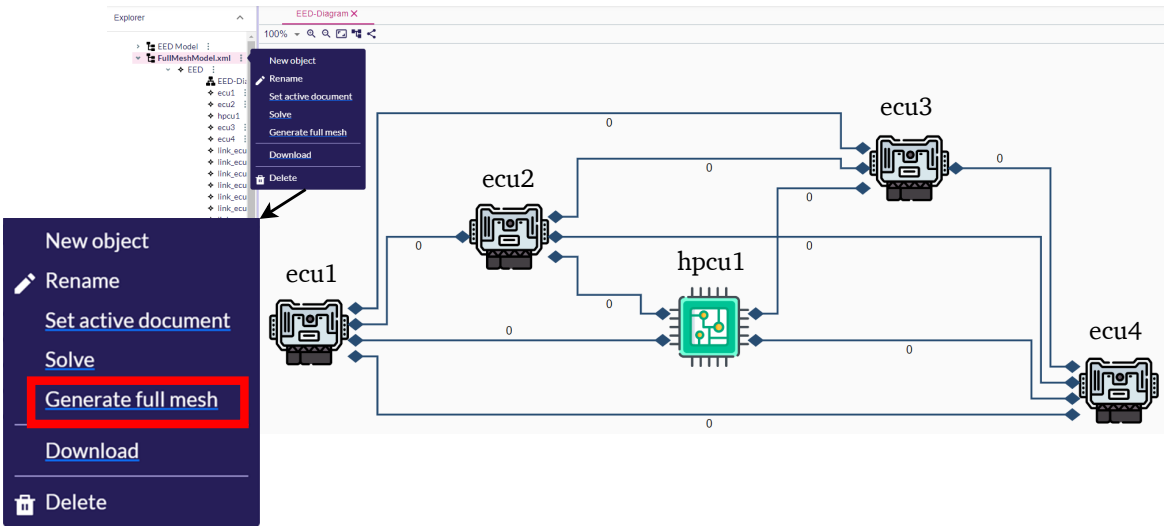


Figure 5.2: A designed full-mesh E/E model including links and ECUs using the E/E Designer tool. By clicking on "Generate full-mesh", each hardware node is connected to other nodes.

redundant interconnection pattern. The full mesh topology is known for its robustness and fault tolerance because if one node fails, the communication pathways remain intact through alternative routes. However, this topology can become complex and costly as the number of nodes increases, as the number of connections grows exponentially [AFK21a; AFK20; AMK23]. To increase the design diversity and facilitate the modeling process, the framework supports a full-mesh generation option that creates a full-mesh network topology for selected hardware nodes where each node is connected directly to all other nodes (see Figure 5.2).

5.1.4 Automatic Creation of Software/Hardware Components

As hardware/software components increase, the modeling and drag-and-drop action become complex and time-consuming. For instance, a designer wants to model an architecture including 100 ECUs and 200 applications. This process takes considerable time. Consequently, an option is integrated into the framework that allows the users to have multiple components at once without dragging and dropping. This saves a significant amount of time for modeling extended topologies/architectures.

5.2 Requirements and Properties

In this section, all safety and non-safety requirements that can be chosen by the user in the frontend are discussed. Moreover, the properties regarding solving and optimization objectives integrated into the frontend are explained.

5.2.1 Hardware/Software Requirements and Properties

Apart from modeling, the requirements and properties relevant to each component can be determined, and they are a must-have for solving a designed model. The details of each component are displayed on the right side of the modeling window. Figure 5.3 illustrates the details regarding several components. As presented in Figure 5.3 (a), for example, as part of an HPCU, a core can have various ASIL levels as a safety-critical property, comprising ASILs A, B, C, D, and QM, a turbo boost feature, a defined clock frequency, a maximum memory utilization, a failure rate, interface to environment choice, and an arbitrary name. For a link, its type (comprising Ethernet, FlexRay, and TT CAN bus), maximum bandwidth capacity, cost, and name can be selected (refer to Figure 5.3 (c)). Moreover, each link's type has a unique visualization symbol, e.g., ECU_7 and ECU_8 are connected with two different links to $gateway_2$ based on Figure 5.1. The application thread properties consist of the execution time, period, and name of each thread (in the tool named process), as shown in Figure 5.3 (g). It should be added that an application can be defined as safety or non-safety-critical in the application properties and have a determined memory usage. Also, the ASIL level of each application can be specified similarly to the core's properties. Additionally, each communication task includes a name, frame length, and period as its properties, as displayed in Figure 5.3 (b). Also, the sender and receiver of a communication message can be specified as Data details depicted in Figure 5.3 (e) [AMK23].

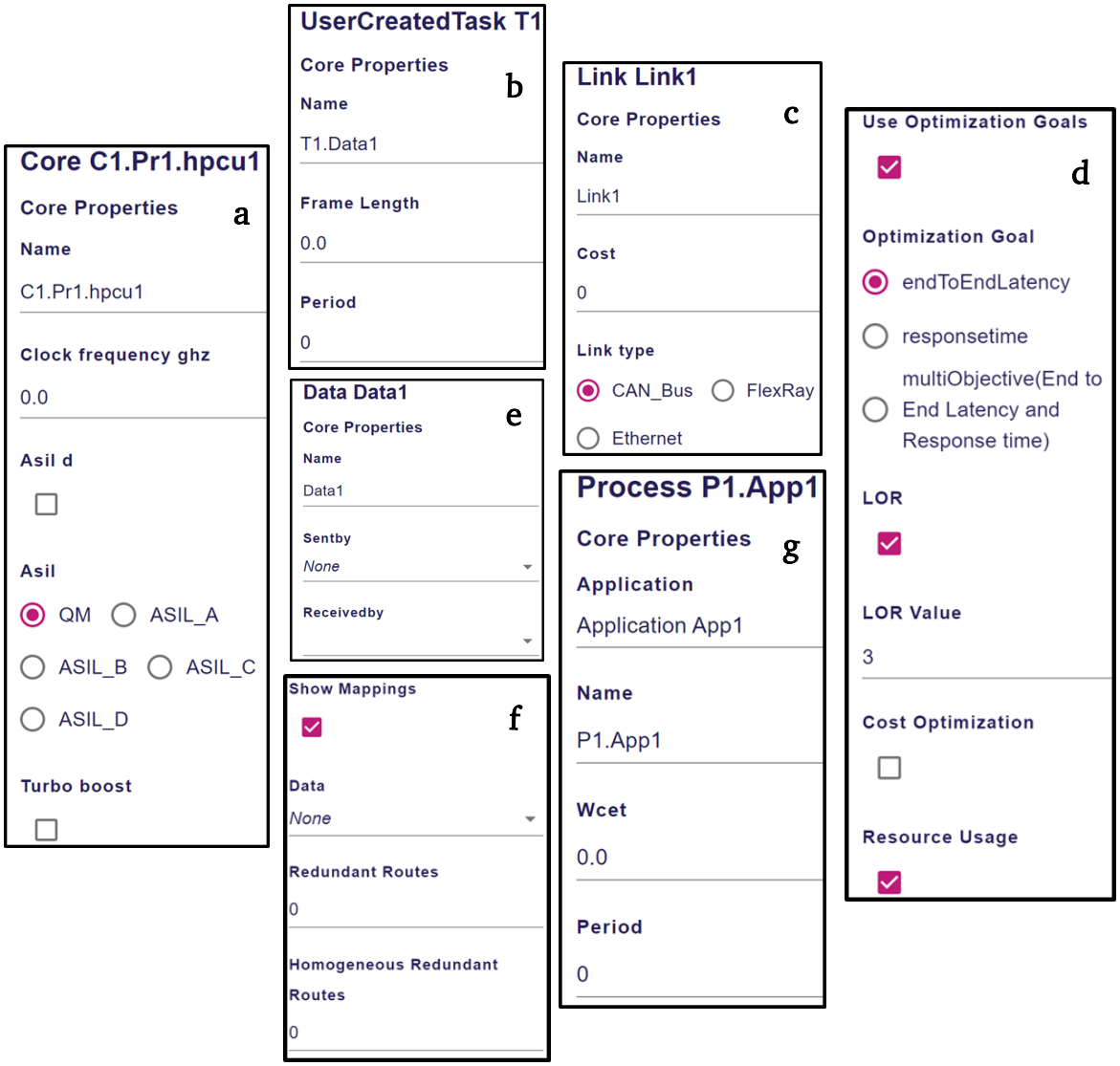


Figure 5.3: Component properties (a), (b), (c), (e), (g), and optimization and solving settings (d) and (f) in the frontend of the presented framework [AMK23].

5.2.2 Optimization and Solving Properties

As mentioned in Chapter 4, the presented model-based framework supports several requirements, optimization goals, and multi-objective optimization [AMK23; AFK21a]. Hence, a setting module has been integrated into the frontend where the user can choose the multi-objective type, activate or deactivate the whole optimization method (*Use Optimization Goals* checkbox in Figure 5.3 (d)), each objective and boundary constraint/requirement, e.g., LOR and LOR maximum tolerable bound, maximum bandwidth utilization, RU, maximum resource utilization, maximum reliability, and cost (see Figure 5.3 (d)). Furthermore, the type of route, such as single, multi-cast, redundant, homogeneous redundant, and the number of required homogeneous redundant paths, can be chosen as indicated in Figure 5.3 (f).

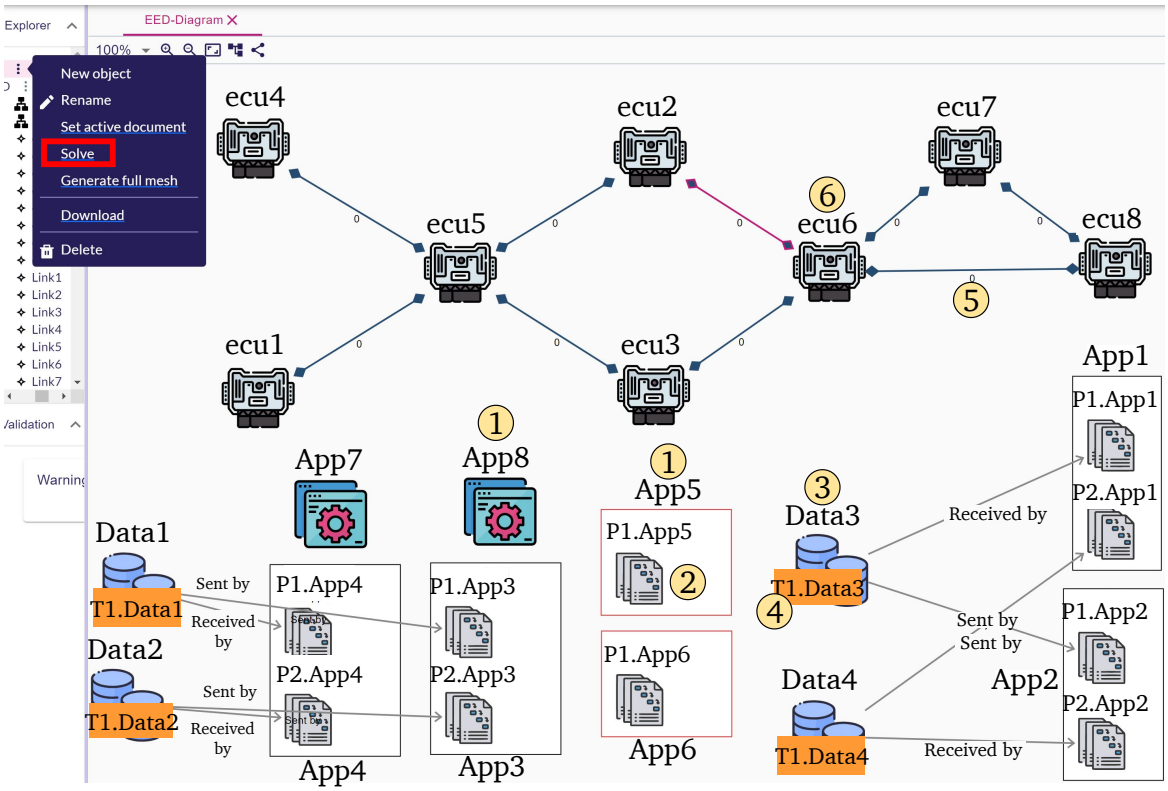


Figure 5.4: A modeled E/E architecture by the presented tool including applications (No. one), application threads (No. two), communication messages (No. three), communication tasks (No. four), links (No. five), and ECUs (No. six). The model is solved and optimized by clicking on the *Solve* option (red rectangle) [AMK23].

5.3 Solving and Solutions

After modeling a vehicle's topology and defining the requirements and properties related to the components, optimizations, and solving, the designed E/E architecture is ready to be solved and optimized.

5.3.1 Solving

To solve and optimize a created model considering the defined properties, requirements, and optimization objectives, e.g., the model shown in Figure 5.4, a *Solve* option is integrated [AMK23]. Clicking on *Solve* option enables the tool to solve the problems associated with the modeled architecture while optimizing the solution and satisfying the specified requirements, as described in Chapter 4. In the model shown in Figure 5.4, there are eight applications, six of which include one or two threads with the same periods and different execution times and two applications without a thread. Two applications here are selected as safety-critical (App_5 and App_6 shown with a red frame). Also, the model consists of four communication messages with the same periods as application threads and various frame lengths. It is aimed to automatically assign the applications to eight ECUs while meeting defined requirements. It is required to ensure that the threads running on each ECU have the correct time-triggered schedules and that all conditions for automated mapping of safety and non-safety-critical applications are fulfilled.

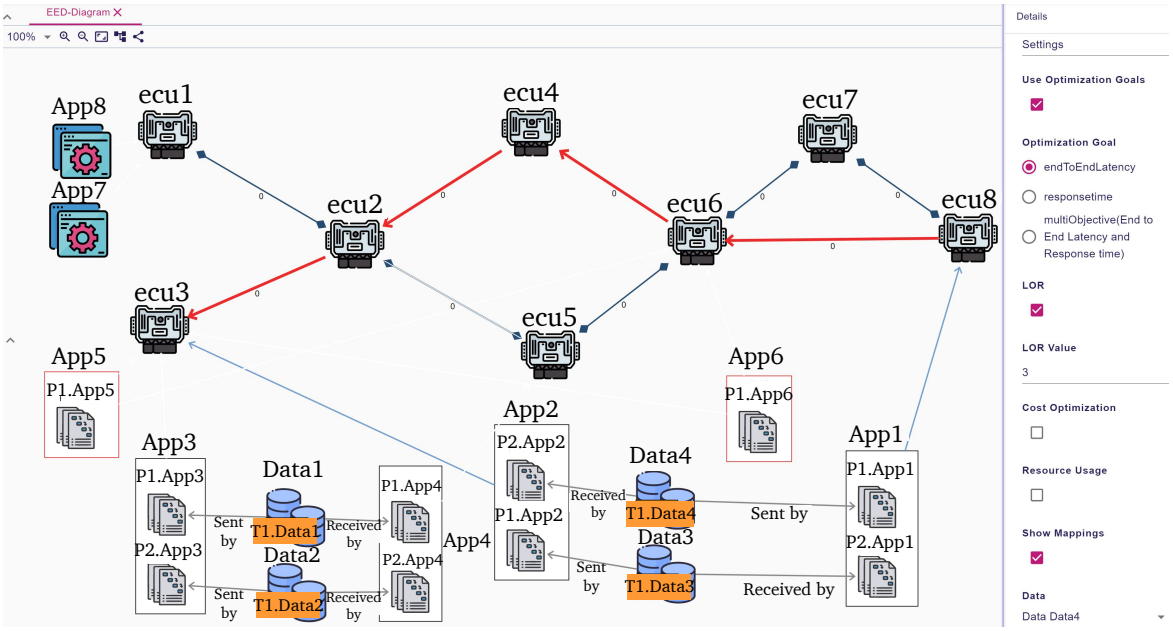


Figure 5.5: A solution of the designed model in Figure 5.4 including mapping, message routing, and scheduling. Here, only the mappings for applications one and two related to the communication message four are displayed [AMK23].

5.3.2 Solutions

After getting the model solved, the user can observe the optimized solutions for mapping, scheduling of application threads and communications tasks, and message routing. Figure 5.5 displays the solution of the design model depicted in Figure 5.4, which includes message routing, mapping, and scheduling for application threads and communication tasks [AMK23].

Mapping

As a mapping solution, the E/E Designer tool determines which applications should be assigned to which ECUs (e.g., blue arrows in Figure 5.5). The current mapping requirements in this example are redundancy conditions for safety-critical applications, and the constraint ensures that the threads running on each ECU are not the sender and receiver of the same communication message. In Figure 5.5, only mappings related to sender and receiver applications of communication message $Data_4$ are visualized.

It is important to note that the visualization in Figure 5.5 exclusively highlights the mapped applications functioning as both senders and receivers for a selected communication message (here, communication message number four). This design choice is motivated by enhancing clarity and comprehensibility within the model, mainly when dealing with extensive complexity. By focusing on this specific aspect, potential confusion is mitigated, resulting in a cleaner representation that facilitates a more intuitive understanding of the mapping scenario.

Time-triggered Scheduling for Application Threads

Figure 5.6 shows the calculated schedules for the application threads executing on ECU_3 in Figure 5.5 within a computed hyperperiod. As can be observed, each color represents an application, and each slot indicates a thread's schedule. Each slot may be executed multiple

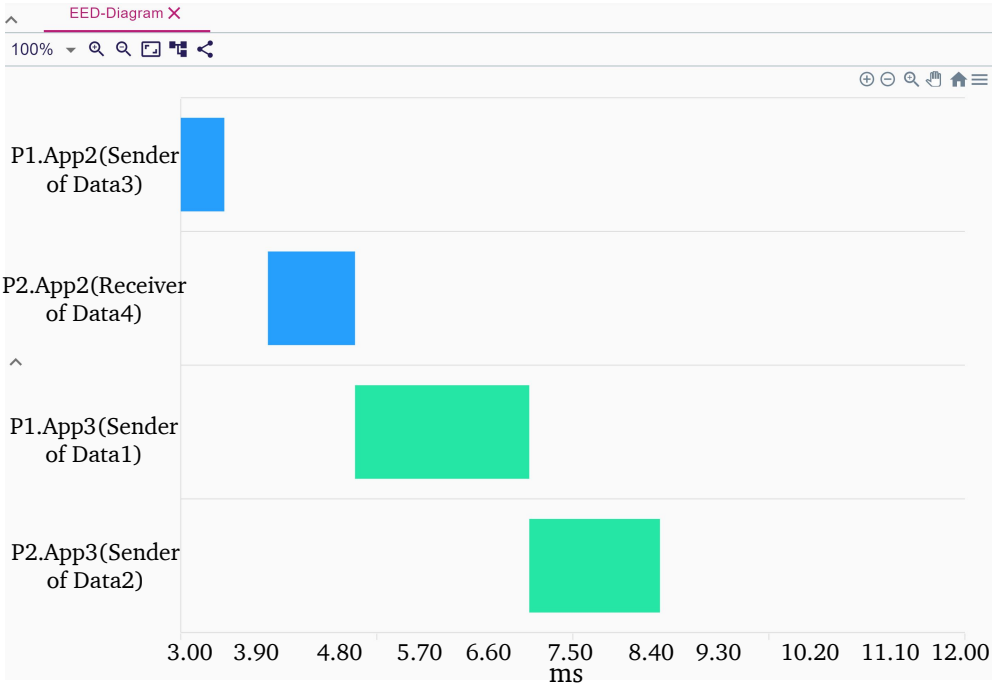


Figure 5.6: Calculated time-triggered schedules by the introduced tool for running application threads on ECU_3 after mapping action as the solution for the model in Figure 5.5 [AMK23].

times within the hyperperiod, depending on the thread's period. Figure 5.7 presents another example of a mapping schedule related to 20 threads executing on an ECU.

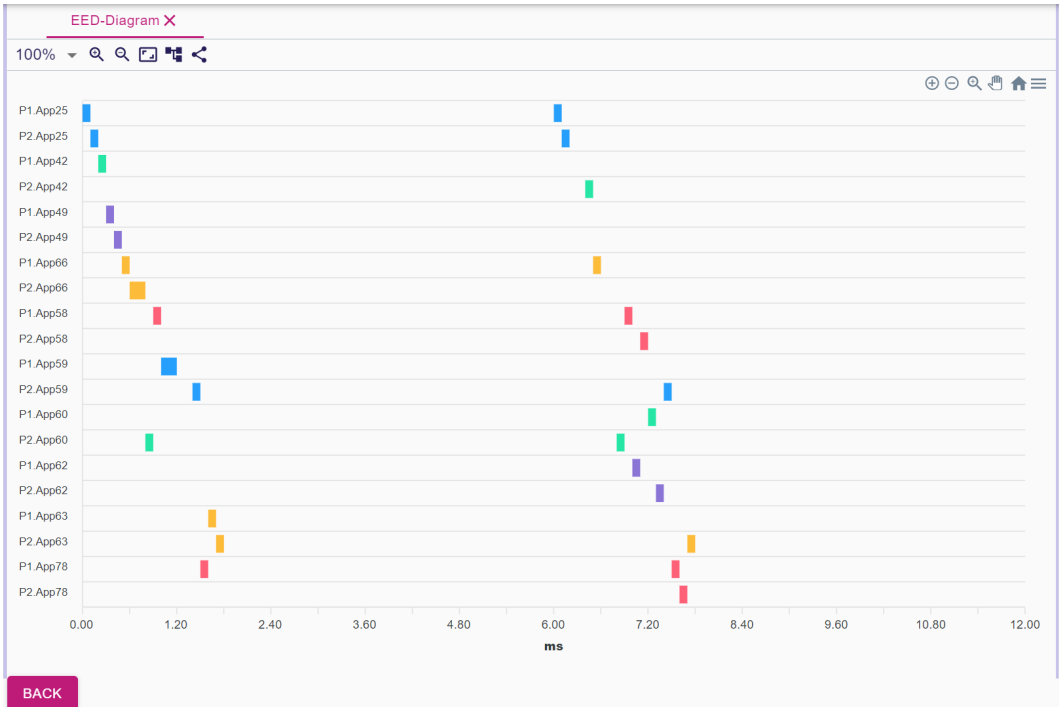


Figure 5.7: Computed time-triggered schedules by the introduced tool for running 20 application threads belonging to 10 applications on an ECU after mapping action.

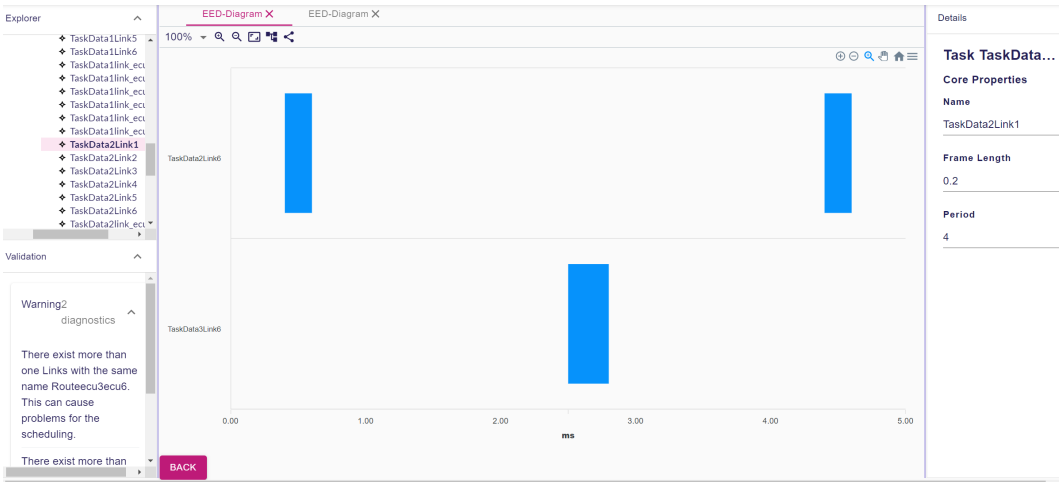


Figure 5.8: Time-triggered schedules of two communication tasks over a link.

Communication Message Routing

A created route for a specific communication message to route a message from a sender to a receiver can be shown in the frontend. In Figure 5.5, four communication messages need to be sent from senders to receivers. As described in Chapter 4, each message comprises a communication task routing over the network links, and the user must specify the threads as the sender and receiver of each message in the frontend before the solving step. Therefore, after solving the model, the user can observe the correct path for each communication message and related mappings for sender and the receiver applications relevant to the same communication message by choosing a desired message. Based on Figure 5.5, the red path shows a generated route from ECU_8 as a sender to a receiver ECU_3 routing the communication message d_4.

Time-triggered Scheduling for Communication Tasks

Similar to the scheduling for application threads, the communication task slots are displayed during the related hyperperiod. It should be added that end-to-end latency optimization goal was applied in the solution presented in Figure 5.5. In addition, the computed time-triggered schedules for four communication tasks routed over links can be visualized similarly to Figures 5.6 and 5.7. For example, Figure 5.8 illustrates correct schedules for two communication tasks over a link. Also, in Figure 5.8, each row relates to each communication task relevant to a specific message.

5.4 Model Validation

Model validation plays a crucial role in ensuring the accuracy, reliability, and quality of an E/E model created using a modeling tool. To effectively utilize the tools for architectural design, it is essential to validate the models against various criteria and requirements. Model validation involves systematically examining and verifying the E/E architecture model to ensure that it conforms to the desired specifications, standards, and constraints. It aims to identify and rectify potential errors, inconsistencies, or design flaws that can impact the final system's performance, safety, or reliability. When modeling an E/E system, validation encompasses multiple aspects. First, it involves verifying the correctness and coherence of

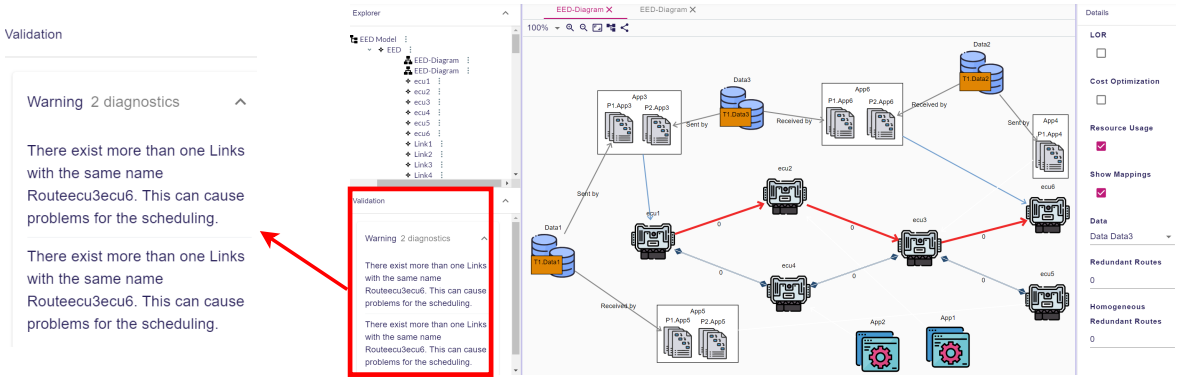


Figure 5.9: A solution of a modeled E/E topology including mapping and message routing. On the left side, several warnings are displayed regarding the validation of the model.

the model's structure and behavior. This includes ensuring that the components, interfaces, signals, and their interactions are accurately represented. Moreover, the validation checks the model's compliance with domain-specific guidelines and defined requirements. Furthermore, model validation focuses on assessing the consistency and completeness of the model. It involves analyzing whether the model adequately captures all the necessary system requirements, functional specifications, and performance constraints. System integrators can detect missing or ambiguous requirements, redundant or conflicting specifications, and incomplete or inaccurate representations by conducting thorough validation. Another critical aspect of model validation in E/E architecture modeling tools is the evaluation of system properties and performance attributes. For example, in the case of safety requirements, if the user assigns an ASIL D application to a core of an HPCU which does not support ASIL D, the validation process gives a warning to the user. Hence, the validation process ensures that the model adheres to these properties and meets the necessary performance targets.

Since several different model requirements must be met to create a valid model, as explained above, errors, warnings, and general tips can assist the user in having a valid model. The validation of the introduced tool is a continuous process, checking the model after each change and displaying validation messages if required. The message validation consists of a problem description and includes the elements, mappings, or attributes that lead to an issue. Moreover, the message indicates the acuteness of the problem and the urgency to change the model [AMK23]. Based on this, three different message types are introduced, including *error*, *warning*, and *message* that each of them are described in the following.

Error

An *error* occurs when there is a violation in the model, which disrupts critical safety measures or breaks the rules of MIP constraints [AMK23]. This message indicates that the user needs to adjust the model to fix the error. Promptly addressing such errors is crucial, as they not only affect how well the system works but also endanger the overall reliability of the system.

Warning

A *warning* serves as an advisory signal intended to highlight potential violations or discrepancies that, while not necessarily leading to critical problems, merit attention and rectification [AMK23]. It points out areas of concern that should be addressed to enhance the overall quality and effectiveness of the subject under consideration. For instance, as illustrated in Figure 5.9, the designed model is accompanied by a series of *warnings*. These cautionary indications shed light on aspects of the model's design that may not align with best practices

or desired standards. While these warnings may not immediately result in severe issues, they serve as essential signposts that prompt further investigation and possible adjustment. Ignoring such warnings can lead to compounded issues or hinder the model's optimal performance.

Message

Finally, the system displays a *message* to provide users with helpful tips, essential notes, and additional information. This feature aims to enhance the user experience by offering valuable insights and guidance [AMK23]. Whether it is a brief pointer to optimize their workflow or a noteworthy caution about potential pitfalls, these messages serve as a valuable resource to users, aiding them in making informed decisions and using the system more effectively.

5.5 Implementation

This section discusses the prototypical implementation of the E/E Designer's frontend. As Chapter 4 mentioned, the Eclipse IDE platform was selected as the development environment. The Eclipse IDE supports various plugins for Java that assist in the development process. One of the main plugins used in this project was the eclipse modeling framework (EMF), which Sirius and Sirius Web utilize for creating class models and aiding in code generation [Ecl23].

5.5.1 Sirius Web

Sirius Web is used as the front-end's foundation of the E/E Designer to provide the web-based functionality. It is an open-source web-based extension of *Sirius Desktop*. Eclipse *Sirius Web* is a framework to easily create and deploy studios to the web [Ecl22]. By using the *Sirius*

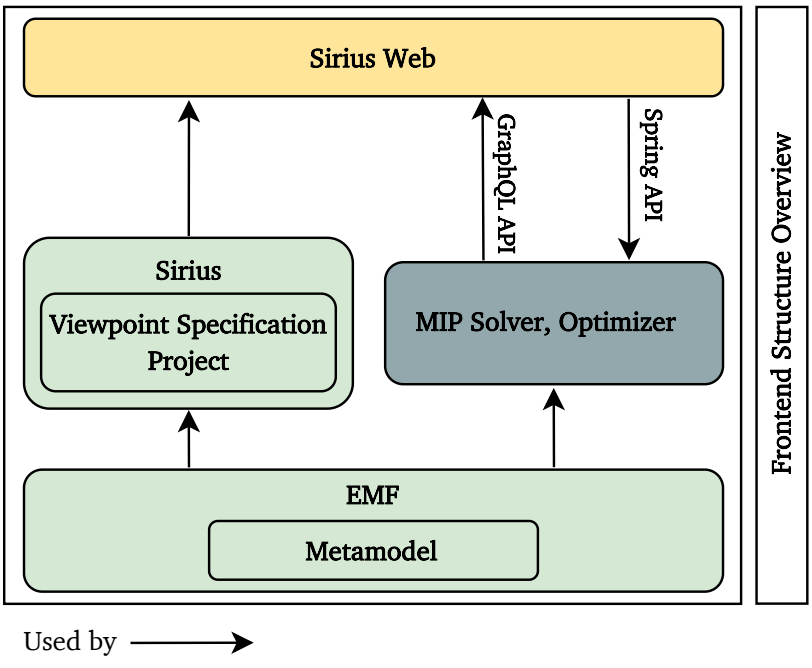


Figure 5.10: An outline of the proposed framework's frontend describing the interaction of modified *Sirius Web* with other modules existing in the backend.

Web, the modeling features of *Sirius* can directly be utilized from a web browser. Once the *Sirius Web* is deployed on a server, users can access and modify editing contexts and download and upload projects without requiring specific installation on their desktops [Ecl22]. To integrate all defined attributes, functional requirements, boundary constraints, and optimization objectives specified in the E/E Designer backend as explained in the Chapter 4, the *Sirius Web* has been modified and developed to provide an appropriate frontend for the introduced tool. Furthermore, all outputs generated by the framework, such as mapping, routing, and scheduling, can be visualized by the developed frontend. The Figures 5.1 to 5.9 shown above are screenshots from the E/E Designer frontend, which is based on the *Sirius Web*.

Figure 5.10 presents an overview of how the frontend (i.e., modified *Sirius Web*) interacts with other modules integrated in the backend. The EMF plugin allows creating a metamodel, which other plugins can then utilize. The metamodel is employed by the MIP solver and optimizer (the presented tool uses the Gurobi Solver as mentioned in Chapter 4) and by *Sirius*. *Sirius* is the local version of *Sirius Web* and is technically a sub-project of *Sirius*. *Sirius* identifies the existing object classes and their relationships through the model. This enables *Sirius* to define the appearance and behavior of each instance object. *Sirius Web* reuses some of the functionality of Sirius. A distinction is also made within the system between the frontend, which uses JavaScript, and the backend, which employs Java. Both systems need to interact with each other. An interface is required since two separate programming languages cannot directly communicate. Therefore, *Sirius Web* comes natively with a GraphQL interface that offers various types of data and methods for data uploading. For the Java component, the Spring Framework is used. A call to a Spring method is executed in the development process when a user wants a model solved or requests other features, such as full-mesh generation. After the solver completes solving the model, the GraphQL API uploads the new model to *Sirius Web*, where it becomes available for the user in the web interface.

6

Design Error Analysis

This chapter focuses on situations where a designed E/E architecture is not satisfiable, which means that the solver cannot find feasible solutions. Unlike simple models, navigating and correcting the unsatisfiability of complex E/E models is an intricate and time-consuming task, leading to increased development costs.

To address this issue, an approach is introduced to identify design errors when violations occur in the constraint set included in the system model after the solving step. This feature is crucial for detecting and rectifying errors in the system design within a reasonable timeframe, ensuring that the system is optimized and meets all necessary constraints and requirements [AMK23; AHK22; AFK21b].

6.1 Background

In numerous domains of technology and science, a prevalent problem-solving strategy involves discovering solutions that meet a set of formal constraints. These constraint systems are utilized across various sectors, including formal verification, automated configuration of hardware or software, planning tasks, and other applications within artificial intelligence.

Propositional logic stands out as an extensively used formal framework for representing constraints. It facilitates modeling logical connections between facts that can be either true or false [BL99]. The boolean satisfiability (SAT) problem concerns determining whether a combination of facts makes a propositional formula valid. Due to its NP-completeness, the SAT problem plays a crucial role in many realms of computer science, and there's no known algorithm capable of efficiently solving every instance [BHM09]. A substantial area of formal methods research focuses on unsatisfiable constraint sets where sets of constraints are proven unsolvable through formal methodologies. In various scenarios, such sets emerge due to design flaws it aims to identify. A productive approach to gaining insights into these errors involves extracting a minimal explanation from a collection of conflicting constraints. This explanation elucidates which constraints or interactions between them are causing the issue.

In the context of the SAT problem, constraints are expressed using formulas in propositional logic. In this scenario, every variable has a binary value of either TRUE or FALSE. When the constraint conditions are met, the outcome of the boolean logic evaluation is considered True. Conversely, if the conditions are unmet, the result is False [BHM09]. Conventionally, SAT problems are often represented using conjunctive normal form (CNF) [Sül+08].

Although most of present research is directed towards extracting minimal unsatisfiable sets from an unsatisfiable constraint system, only a few studies delve into recognizing re-

quirements and rectifying them to transform the constraint system into a satisfiable state. Hence, this chapter aims to bridge this gap by investigating the process of identifying minimal unsatisfiable sets linked to the introduced set of constraints within the framework. In addition, this chapter suggests pragmatic approaches to render the constraints feasible.

6.1.1 Conjunctive Normal Form

CNF is a fundamental concept in propositional logic, a branch of formal logic that deals with manipulating and analyzing logical statements involving propositions and their truth values [Sül+08]. CNF serves as a structured representation for logical formulas, allowing complex statements to be broken down into simpler components that are easier to analyze, manipulate, and process using automated tools and algorithms.

A logical formula is said to be in CNF if it is a conjunction $\wedge$ (AND) of one or more clauses, where each clause is a disjunction $\vee$ (OR) of literals. A literal is either a propositional variable (denoted by a letter or its negation) or the negation of a propositional variable. In CNF, the logical operators are reduced to only conjunction and disjunction. The following is a breakdown of the components of CNF:

- Clause: A clause is a disjunction of literals. It represents a statement that is true if at least one of the literals in the clause is true. For example, (A OR B) is a clause where A and B are literals.
- Literal: A literal is either a propositional variable or the negation of a propositional variable. For example, A and $\neg B$ (not B) are literals.
- Conjunction ($AND/\wedge$): The conjunction operator combines multiple clauses or literals with the AND operator. It indicates that all the statements it connects must be valid for the entire formula to be true.
- Disjunction ($OR/\vee$): The disjunction operator combines literals within a clause. It indicates that at least one of the literals in the clause must be valid to be true.

Below are examples of two formulas in CNF:

$$(A \vee \neg B) \wedge (\neg A \vee C) \wedge (B \vee C \vee D) \tag{6.1}$$

$$(m_1 + m_2 \leq 3) \wedge (m_1 \geq 5) \wedge (m_2 \leq 7) \wedge (m_1 \leq 4). \tag{6.2}$$

A CNF formula is satisfied when an assignment of truth values (true or false) to the variables in the formula makes the entire formula accurate. In other words, for a CNF formula to be satisfied, every clause within the formula must have at least one literal that evaluates to true under the given assignment. For example, the logical formula presented in (6.1) consists of three clauses. To identify each clause, they are separated by AND/conjunctions. To determine if the CNF formula is satisfiable, it is required to find assignments for the variables (A, B, and C) that satisfy at least one literal in each clause. If A, B, C, and D are set as TRUE, FALSE, FALSE, and TRUE, respectively, this assignment satisfies all clauses, so the CNF formula is satisfiable.

Mathematical statement in (6.2) illustrates a CNF formula as a SMT interpretation including four clauses. Because of the conflicting m_1 clauses which define m_1 value, the CNF formula is unsatisfiable.

6.1.2 Minimal Unsatisfiable Subset (MUS) or Unsatisfiable Core

An unsatisfiable core, also known as a minimal unsatisfiable subset (MUS), is a subset of the original set of clauses (constraints) that, when taken together, is itself unsatisfiable. In other words, if any clause gets removed from the unsatisfiable core, the remaining subset becomes satisfiable. The concept of an unsatisfiable core is often utilized in solving constraint satisfaction problems, including SAT problems. It helps identify a smaller subset of conflicting clauses that are causing the unsatisfiability of the entire problem [Sül+08; BHM09].

In the example presented in (6.1), the CNF clause, $B \vee C \vee D$, is considered. If B and C are both FALSE and A is TRUE, the entire clause evaluates to TRUE, which means the original CNF formula is satisfiable. However, when dealing with a more extensive set of clauses and a particular combination of them results in an unsatisfiable formula, the goal is to identify the smallest possible subset of clauses responsible for the unsatisfiability. This subset is known as the unsatisfiable core.

The concept of a MUS gives rise to various algorithmic tasks, showcasing significant differences in complexity and achievable performance and the methods employed to address them. Among these tasks, the simplest one entails discovering a single MUS, a process commonly called MUS extraction. Note that a minimal unsatisfiable subset may not necessarily be the smallest size, as smaller unsatisfiable subsets can exist. This simplifies the task into a linear search space traversal, continuing until a subset that fulfills the definition is found.

Moving to a more challenging endeavor, the aim is to uncover a MUS with the smallest possible cardinality, often known as the smallest MUS (SMUS). In other words, an SMUS is the most minor collection of constraints that cannot be satisfied simultaneously. In the realm of algorithms for this task, the initial reliance was on non-chronological backtracking [LM04], although a more efficient approach has emerged in the form of a branch-and-bound technique [Mne+05]. However, the practical applicability of this problem appears to be somewhat limited. This is due to the fact that discovering a MUS of minimal size is notably more intricate than finding any MUS and frequently yields minimal additional information. The most ambitious task is exhaustively enumerating all MUSes within an unsatisfiable clause set. Current methodologies addressing this challenge incorporate ingenious enumeration techniques to optimize the assessment of subset candidates [BSW03]. Alternatively, these methodologies capitalize on the duality shared by MUSes and maximal satisfiable subsets through an interleaved approach [BS05] or a two-level approach [LS08] to address a hypergraph transversal problem. Considering that a clause set may potentially comprise an exponential multitude of distinct MUSes, a universally efficient technique for identifying all MUSes is improbable. Nonetheless, even in substantial industrial scenarios, the count of unique MUSes often proves surprisingly limited, lending practical significance to extant tools for MUS enumeration. This significance arises from the collective set of MUSes offers an all-encompassing understanding of the underlying error's characteristics.

6.2 Approach

The design error analysis approach aims to bridge the gap in identifying the constraints responsible for model infeasibility in a constraint system. While previous studies have focused on generating sets of unsatisfiable or infeasible constraints, the specific constraints causing the system's infeasibility have yet to be thoroughly explored. This aspect holds significance in the context of this thesis because identifying violated constraints following the solution of elaborate E/E systems models is an intricate and time-consuming task. Consequently, an approach is introduced to identify the core satisfiable constraint by building upon the

algorithms that generate minimal unsatisfiable sets.

To extract the unsatisfiable sets, the proposed approach employs two methods, including the irreducible infeasible subsystem (IIS) and the MUSes using MARCO algorithm [GR90; Lif+16], which will be explained below. Once the MUSes are generated, the design error analysis approach introduces a method to assign weights to each constraint based on its contribution to the system's infeasibility. This facilitates the identification of the precise constraints that result in unsatisfiability within the constraint system. By assigning weights to individual constraints, the approach provides practical solutions to modify the constraint system, making the system model feasible.

6.2.1 Using Irreducible Inconsistent Subsystem (IIS)

As the first approach, the IIS method is used. It focuses on finding a minimal set of constraints that, when removed from the problem, render the system consistent or satisfiable. In other words, an IIS is a subset of the constraints that, when removed, allows the remaining constraints to be satisfied [GR90]. The IIS approach is commonly used in the context of ILP and MIP constraints. As Chapter 4 mentions, the presented model-based tool uses the Gurobi optimizer to solve the MIP constraints. Consequently, the *computeIIS* command, integrated within the Gurobi, creates IISes that encompass unsatisfiable sets [Gur22]. Based on the Gurobi [Gur22] and [GR90], an IIS refers to a subset of constraints and variable limits characterized by the following attributes:

- It remains infeasible, and
- The subsystem becomes feasible if a constraint or boundary is omitted.

The proposed approach using the IIS method consists of several steps, illustrated as a flowchart in Figure 6.1. The process involves providing an infeasible set of MIP constraints and empty lists as inputs. In the subsequent step, the calculation of IIS from the infeasible constraints is executed, and the results are stored in a list. An IIS represents a minimal subset of constraints that renders the system infeasible, as explained earlier. This implies that including any constraints within an IIS leads to an infeasible solution.

Following the computation of IIS, the next step involves examining each constraint within the IIS list. One constraint at a time is removed from the set of infeasible constraints. The IIS calculation is then reapplied to the updated set of infeasible constraints, excluding the removed constraint. The model's feasibility is assessed based on the IIS computation's outcome during this iteration. If the model becomes feasible after excluding the constraint, it indicates that the removed constraint significantly contributes to the unsatisfiability of the entire system model. A weight is assigned to the excluded constraint to capture this influence and saved in a separate list. The assigned weight of a constraint is proportional to the number of times it has been identified as a cause of model infeasibility when excluded from the set of infeasible constraints. According to the flowchart presented in Figure 6.1, the process continues until all constraints in the IIS list are covered. As a result, a list of constraints, along with their respective weights, becomes available.

Finally, by taking into account the weights assigned to each constraint, the most crucial constraints contributing to the unsatisfiability of the system model are identified. With the aid of these identified constraints, users can rectify the model by addressing issues present in the critical constraints. For instance, this may involve validating the provided inputs as requirements for the tool. Subsequently, after making the necessary adjustments, the designed E/E system can be rendered satisfiable upon being re-solved.

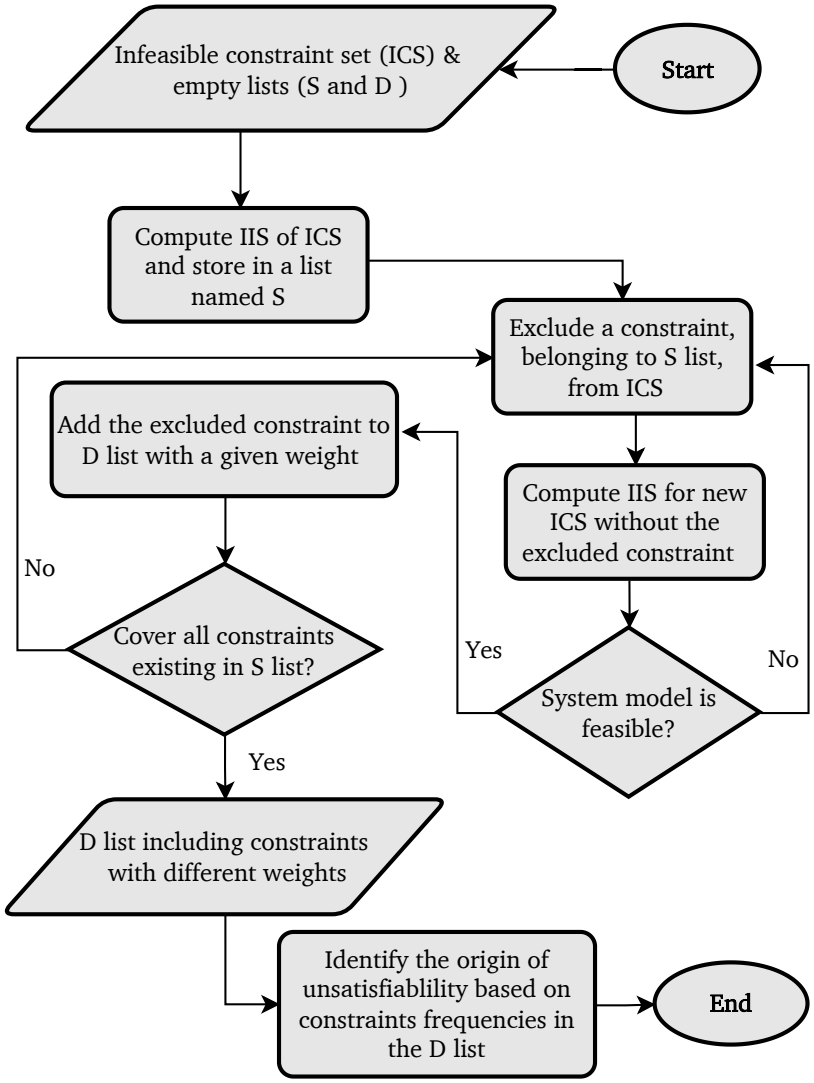


Figure 6.1: The design error analysis flowchart using the IIS method.

Algorithm 5 is also introduced within this approach. Building upon the DVC algorithm, for each constraint present in the *IIS_list*, which is derived from the computation of IIS for the list of infeasible constraints, denoted as *c_list* (line number 1), it undergoes exclusion from the *c_list* upon encountering an identical constraint within it (lines 2 to 5). Subsequently, the modified constraint is saved back into the *c_list*, effectively updating it (line number 5). Following this, the IIS is recalculated for the updated *c_list* and stored within the *result* variable (line number 6). If the result variable yields a null outcome, the constraint that was excluded, attributed either to the *IIS_list* or *s* as per line (8), is assigned a weight and recorded within the *d_list*. However, in the scenario where the constraint *s* already resides within the *d_list* (line 12), the assigned weight is incrementally added to its existing weight (lines 9 to 11).

ConstrIISForce

When determining an IIS for an infeasible model, the *ConstrIISForce* parameter, as an attribute integrated into the Gurobi optimizer, determines the inclusion or exclusion of a general constraint within the IIS. If the value is set to -1 by default, the decision is left to the IIS

Algorithm 5: DVC

Input: A list of infeasible MIP constraints (*c_list*), $s \in Constr$
Output: Weighted constraints S

```
1  IIS_list ← computeIIS(c_list);
2  for i ← 0 to IIS_list.size do
3      for j ← 0 to c_list.size do
4          if IIS_list.get(i) = c_list.get(j) then
5              c_list.remove(j);
6              result← c_list.computeIIS;
7              if result == null then
8                  s←IIS_list.get(i);
9                  if s∈ d_list then
10                     d_list.get(s).setWeight(d_list.get(s).get(Weight) + 1);
11                 else
12                     d_list.add(s.setWeight(1));
13                 end
14             end
15         end
16     end
17 end
```

algorithm itself. Alternatively, if the attribute is set to 0, the constraint is deemed ineligible for the IIS inclusion. Conversely, setting the attribute to 1 guarantees the constraint's inclusion in the IIS, with the algorithm disregarding any consideration of its removal [Gur22].

Note that configuring this attribute as 0 can potentially make the derived subsystem feasible or consistent. Consequently, attempting to construct an IIS may lead to a GRB _ERROR_IIS_NOT_INFEASIBLE error, even if attempted. Similarly, assigning this attribute a value of 1 may yield an IIS that lacks full irreducibility. The system's irreducibility pertains solely to model elements bearing force values of -1 or 0 [Gur22]. Hence, in order to remove a constraint in line (5) of the DVC algorithm, this parameter is utilized.

6.2.2 Using MARCO Algorithm

MARCO is an algorithm designed to analyze infeasible constraint systems with the primary objective of systematically listing all MUSes and maximal satisfiable subsets (MSSes). Generally, MUSes hold a higher degree of significance due to their more comprehensive range of applications. MARCO is an acronym for Mapping Regions of Constraints, succinctly capturing its operational essence. The underlying concept revolves around mapping, which can be effectively visualized on the power set lattice of a given constraint system. The power set of constraints refers to the complete collection of all possible subsets of constraints within that set [Lif+16].

To identify the constraints responsible for unsatisfiability, a brute-force method can be employed, involving the counting and sorting all constraints' frequencies in the MUS sets, as outlined in Kleiman et al.'s work [Kle09]. The constraint with the highest frequency may be the root cause of the unsatisfiability. Altering this criterion increases the likelihood of rendering the constraint satisfiable. However, this method may not always be the most efficient, as it can be computationally demanding, particularly for large systems encompassing numerous constraints.

The MIP constraints introduced in the presented framework tool must be transformed into SMT constraints to use the MARCO algorithm. These type of constraints are used in optimization problems involving continuous and integer variables, while the SMT constraints are employed in logical constraint satisfaction problems with complex logical formulas across various theories. After employing the MARCO algorithm to generate all MUSes from the SMT constraints, the constraints are sorted and grouped based on their corresponding requirements. Next, the frequency of each constraint is analyzed to identify the conditions most likely to lead to system unsatisfiability. This approach shares similarities with the IIS approach, where constraints were prioritized based on their weights to identify the most influential ones. Following the transformation of the MIP to the SMT constraints, the Z3 solver is employed to solve the set of SMT constraints [DB08]. In cases of infeasible solutions after solving, the SMT file generated by the Z3 solver is integrated into the MARCO algorithm. In the final step, the frequency of each constraint is tallied, and the most critical constraints are investigated as the source of the unsatisfiability of the system model.

Since the proposed framework includes the MIP constraints and utilizes the Gurobi optimizer, the IIS approach is preferred over the MARCO approach due to both the implementation effort involved and the effectiveness of the Gurobi solver in solving the MIP problems.

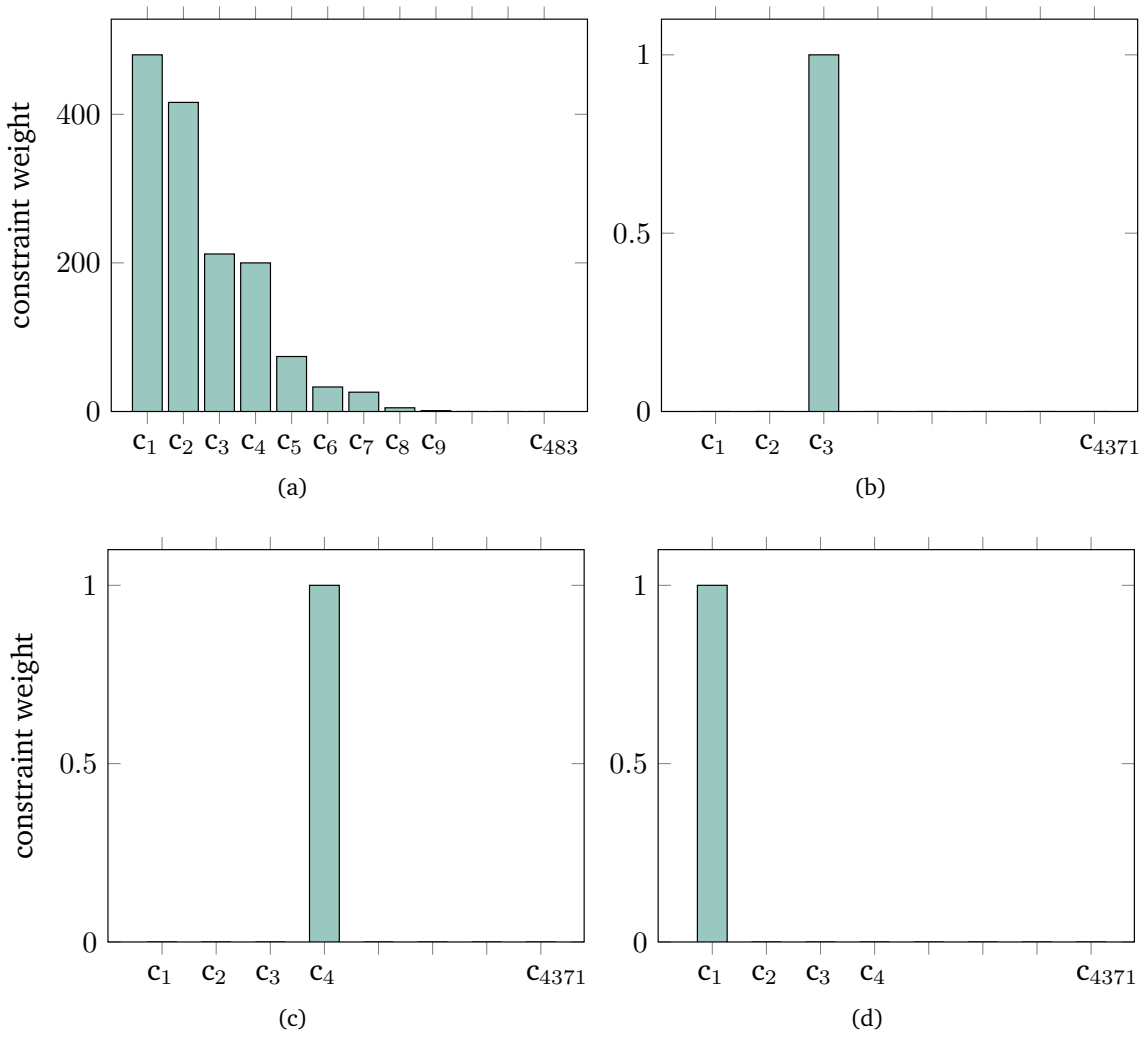


Figure 6.2: The solutions of design error analysis approach created for various case studies using the IIS method.

6.3 Evaluation

This section provides and analyzes the design error solutions generated by the introduced methods. For the IIS approach, the primary design error analysis method used in the E/E Designer tool, solutions for four infeasible case studies are presented as shown in Figure 6.2. Figure 6.2 (a) depicts the weights of each constraint in an infeasible system model after solving, which includes 15 ECUs, six applications, and three communication messages. The generated analysis shows that only eight constraints have a weight higher than zero out of 483. This means that only these nine constraints render the design model unsatisfiable. Furthermore, the higher the weight of a constraint, the more significant its impact on making the system model infeasible.

For instance, in Figure 6.2 (a), the execution times of application threads were defined close to their respective periods. Since this error involves several constraints, different constraints were assigned weights based on their importance in causing an unsatisfiable model. In this example, c_1 has the highest weight related to the message routing constraints, while the other constraints are associated with the mapping and the time-triggered scheduling conditions. To correct the system model, the weighted constraints must be analyzed to identify the source of the violation.

In Figure 6.2 (b), the source of violation was identified for a modeled E/E system comprising 15 ECUs, 60 applications, and 30 communication messages. In this model, the frame length of a communication message was specified to be greater than its period. As observed, only constraint c_3, out of 4371 constraints, has a weight, indicating that the condition stating that the frame length of a message must be less than its period was violated. Since this violation does not involve the other conditions, only c_3 received a weight after the design error calculation using the IIS approach. In the following scenario, the same model as in Figure 6.2 (b) is used. However, this time, the execution time of an application thread is set higher than its period. As can be observed in Figure 6.2 (c), which is the solution provided by the proposed approach using IIS, the related constraint c_4 received a weight from the other 4371 constraints. Therefore, the user can achieve a feasible model by correcting the issue related to this specific constraint during the solving process.

```
1_Activated_Out-Data_&_1_or_more_Acitvated_In_Data_:
 - MMapApplication20Data2Node14 + MMapApplication21Data2Node14
 - Data_INdatain_link_Node14_Node10Data2
 - Data_INdatain_link_Node14_Node11Data2
 - Data_INdatain_link_Node14_Node12Data2
 - Data_INdatain_link_Node14_Node13Data2
 + Data_OUTdataout_link_Node14_Node10Data2
 + Data_OUTdataout_link_Node14_Node11Data2
 + Data_OUTdataout_link_Node14_Node12Data2
 + Data_OUTdataout_link_Node14_Node13Data2 = 0
Process0Application21Data2STofProcess5MustBeLessThanItsPeriod+ExecutionTimeNode0:
 StartingTimeProcess0Application21Data2 <= 9.9
Process0Application21Data2Data2: [
 - 99 Data_OUTdataout_link_Node0_Node1Data2 ^2
 - Data_OUTdataout_link_Node0_Node1Data2 * Data_OUTTaskData2link_Node0_Node1
 + Data_OUTdataout_link_Node0_Node1Data2 * StartingTimeProcess0Application21Data2
 ] >= 0
```

Figure 6.3: The partial output of IIS computation for the case study presented in figure 6.2 (a).

```
One Application related to one Data only be mapped on one Node;480
connect d_out1_n1 to d_in1_n2;378
1 Activated Out-Data;221
1 Activated Out-Data & 1 or more Acitvated In_Data ;185
1 Activated In-Data;74
each App mapped only on one Core;32
Process0Application11Data1STofProcess3MustBeLessThanItsPeriod+ExecutionTimeNode0;22
Process0Application01Data0STofProcess1MustBeLessThanItsPeriod+ExecutionTimeNode0;9
Process0Application21Data2STofProcess5MustBeLessThanItsPeriod+ExecutionTimeNode0;1
TaskData2link_Node9_Node8STofTask2MustBeLessThanItsPeriod-FrameLengthNode9;0
```

Figure 6.4: The result of design error analysis approach using the IIS method for the use case described in figure 6.2 (a).

Finally, in Figure 6.2 (d), a routing constraint in the system model was modified, resulting in an infeasible model after the solving step. This modification was done to verify whether the presented approach can accurately identify the source of the violation. Following the application of the design error approach, the routing constraint (c_1 in Figure 6.2 (d)) was successfully identified.

Note that the computation time required to identify the source of unsatisfiability varies depending on whether the source of conflict impacts a single or multiple constraints. It also varies based on whether the error is self-explanatory, as defined by the constraints, or if the error becomes apparent only after solving half of the system model due to adding values that render the system infeasible. For example, in the use case shown in Figure 6.2 (a), the execution time of a thread was defined to be close to its period. Initially, this does not appear to cause conflicts based on the scheduling constraints. However, after determining the correct schedules for multiple applications, it became evident that the overlapping between threads occurred due to the small gap between their execution time and period, leading to conflicts in other conditions, such as routing. Therefore, the computation time plays a pivotal role in extensive models, mainly when the error involves multiple constraints.

Figure 6.3 shows a partial output comprising 483 constraints after calculating the IIS for the infeasible model described in Figure 6.2 (a) using the *computeIIS* command in the Gurobi optimizer. As can be observed, a list of MIP constraints includes their names. Meanwhile, Figure 6.4 displays the results of the designed error analysis approach for the presented case study in Figure 6.2 (a). A weight is calculated for each constraint name based on the information presented in Figure 6.4.

The solution consists of nine constraints with computed weights, and the c_1 constraint has the highest influence, as indicated by the number in the red box, on making the model infeasible, as also explained in Figure 6.2 (a). With this solution, the user can review the weighted constraints in order to correct the system and obtain a feasible solution after solving it.

An infeasible mapping case study was conducted to assess the design error analysis approach using the MARCO algorithm. This use case involves 16 ECUs and 13 applications. Similar to the previously described approach, the results yielded three constraints with assigned weights. However, it is essential to acknowledge that the MARCO algorithm generates only one minimal unsatisfiable subset (MUS) based on specific constraints within the system. Consequently, the outcomes produced by the MARCO algorithm are significantly influenced by the precise number of constraints associated with particular requirements in the constraint system. To gain a more comprehensive insight into the infeasibility of the constraint system, it becomes imperative to execute the MARCO algorithm multiple times, each employing a

distinct set of constraints.

7

Evaluation

This chapter delves into the comprehensive evaluation and experimentation phase that forms the cornerstone of this research endeavor. The primary objective is to rigorously assess the proposed software framework's viability, effectiveness, and real-world applicability. A systematic series of experiments and assessments, including three types of evaluations, aims to validate the hypotheses formulated in the earlier stages of this study and provide empirical evidence to support the claims. The chapter unfolds as follows:

- Section 7.1 outlines the design-time evaluations of the introduced tool, providing an in-depth analysis of the experimental results for various scenarios.
- Section 7.2 presents an experimental setup using a real hardware platform and detailing methods and parameters utilized to conduct our evaluations. This section also offers insights into the rationale behind selecting these components and their alignment with real-world scenarios.
- In Section 7.3, the focus is shifted towards a qualitative evaluation that offers a nuanced perspective on the introduced tool. Here, the emphasis lies on elucidating how this tool facilitates the design and synthesis of intricate E/E systems. Through this analysis, the tool's potential is uncovered in streamlining the design process and enhancing the synthesis of systems, thus contributing to advancing engineering practices. Moreover, this section undertakes an evaluation of the installation process of the framework, providing a comprehensive assessment of its usability and ease of integration.

7.1 Design-time Evaluation

Through diverse case studies, this section assesses the tool's performance, applicability, and scalability in the design phase. The goal is to highlight its strengths, adaptability, and potential areas for improvement, providing empirical insights into its real-world value.

The proposed framework serves as a valuable asset in streamlining the design process for system architects. However, the practical utility of this framework can be hampered by prolonged computation times involved in both constraint generation and solving processes. To address this challenge, a comprehensive investigation is conducted to discern the impact of various parameters within the presented system model on the time required for efficient constraint set generation and solving [AFK21a; AMK23]. Furthermore, this study includes

a meticulous scalability analysis, underscoring the tool's capability to seamlessly accommodate systems of considerable magnitude. This entails an exploration of the novel single-step solving algorithms and formulations, demonstrating their scalability even when dealing with intricately large systems [AMK23].

Notably, the *Gurobi 9.5* solver [Gur22], as mentioned in Chapter 4, is used for effectively solving the system models. It is worth noting that all experiments conducted during the design phase are performed on a laptop equipped with a robust 2.80GHz Core i7 processor and a 16 GB memory capacity. This setup ensures that the experimental conditions align with real-world scenarios while upholding consistent standards throughout the evaluation process.

7.1.1 Evaluation of Communication Message Routing Generation

This experimental study solely evaluates the synthesis time associated with generating communication message routing constraints concentrating on homogeneous redundant routes. The synthesis time encompasses two fundamental components: the duration required to formulate the system model's variables and constraints and the time to effectively solve the resultant constraint set. This investigation is centered around comprehending the direct influence of the message routing constraint set on the synthesis time, particularly in the context of homogeneous redundant paths. The underlying motivation is to dissect how these constraints shape the temporal aspects of the synthesis process. The analysis is performed under four distinctive scenarios, each carefully curated to encapsulate specific conditions and variables [AFK21a]. It is essential to underline that the chosen experimental scenarios are characterized by deploying fully interconnected topologies or architectures—commonly referred to as full-mesh arrangements as illustrated in Chapter 4. This choice allows us to isolate and interrogate the impact of message routing constraints within a controlled environment.

First Case Study

In the first scenario, the count of homogeneous redundant (HR) routes is increased while keeping the number of applications and nodes constant. Moving to the second scenario, the number of applications remains constant but increases compared to the previous scenario, and the synthesis time is subsequently measured. In the third scenario, the number of applications increases while maintaining a constant number of nodes and HR routes. Lastly, the number of nodes is incremented in the fourth scenario, while the count of HR routes and applications remains unchanged. Each measurement comprises two components: the time to generate MIP constraints and the time to solve. It should be added that the cost optimization goal for the communication links was applied during these experiments [AFK21a; AFK20].

Second Case Study

To assess the impact of increasing the number of HR routes in the synthesis time, refer to Figure 7.1 (a); topologies were generated with the same number of nodes and applications, fixed to 100 and 2 respectively, while the number of routes was increased from one to six. As Figure 7.1 (a) depicts, the number of HR routes does not significantly influence the constraint generation time in contrast to the solving time, which exhibits a noticeable linear rise. A similar experiment to the last scenario was conducted to observe the effect of the number of applications on HR routes, with the only difference being that the number of applications was altered to 20 instead of 2. As shown in Figure 7.1 (b), both the generation and solving

times exhibit similar linear trends to the previous scenario; however, the solving process takes considerably longer in this particular experiment [AFK21a].

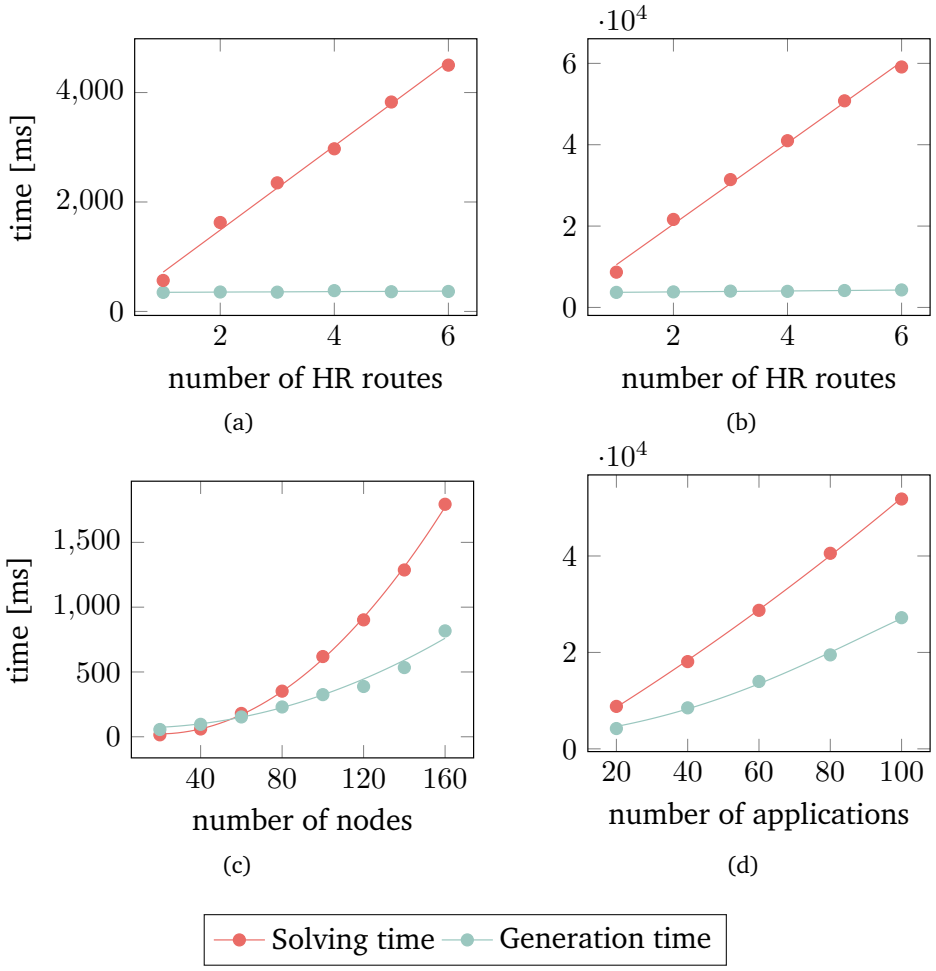


Figure 7.1: The architectural synthesis times for the defined experimental scenarios involving communication message routing integrated into the E/E Designer tool. The constant variables for each scenario are established as follows: (a) Two applications and one hundred nodes. (b) Twenty applications and one hundred nodes. (c) Six HR paths and two applications. (d) One hundred nodes and six HR paths.

Third Case Study

In the third scenario, the number of applications and HR routes remains constant at 2 and 6, respectively, while the number of nodes increases to 160 in order to observe the influence of nodes on the synthesis time. As depicted in Figure 7.1 (c), the generation time expands as the number of nodes increases. Moreover, it is evident that the run-time displays non-linear growth. Furthermore, Figure 7.1 (c) illustrates exponential escalation in the solving time as the number of nodes increases. For instance, the solving time for 80 nodes is approximately 400 milliseconds, while it escalates to 1700 milliseconds for 160 nodes [AFK21a].

Forth Case Study

In the final scenario, the number of applications is increased from 20 to 100 while keeping the number of nodes and HR routes fixed at 100 and 6, respectively. The objective is to investigate the impact of the increased number of applications on constraint generation and

solving times. As shown in Figure 7.1 (d), the run-time for constraint generation non-linearly increases (from approximately 4500 to 27000 milliseconds) in correlation with the growth in the number of applications when transitioning from 20 to 100 applications. Similarly, Figure 7.1 (d) illustrates an increase in solving time as the number of applications increases, while the nodes and number of HR routes remain constant. For instance, solving the constraints for an architecture with 100 nodes and 20 applications supporting six HR routes takes about 10 seconds. In contrast, the architecture with 100 applications and the same number of nodes and HR routes requires approximately 50 seconds. In contrast to the trend seen in Figure 7.1 (c), the solving time displays a less exponential behavior, as depicted in Figure 7.1 (d) [AFK21a]. It is worth noting that, based on Figures 7.1 (c) and (d), the number of nodes has a comparatively more minor impact on constraint generation and solving times than the number of applications does.

7.1.2 Automated Mapping Approach and Application Threads' Scheduling Evaluation

This experiment explores the synthesis time related to the integrated automated mapping approach and the time-triggered scheduling for mapped application threads within the proposed framework. This investigation focuses on understanding the direct impact imposed by the mapping and scheduling constraint sets on the synthesis time. The primary motive is to dissect how these constraints intricately shape the temporal dimensions of the synthesis process. Notably, this experiment unfolds within the context of four different scenarios. These scenarios were carefully designed to help understand how mapping and scheduling constraints influence the system. It is vital to highlight that the selected experimental scenarios involve using zonal topologies or architectures. This specific choice enables isolating and carefully studying the constraints' subtle impacts in a controlled environment [AMK23].

In the first three case studies, Figures 7.2 (a), (b), and (c), the mapping problem is solved by applying the resource utilization (RU) as a hard constraint and the scheduling for threads running on each control node using the E/E Designer framework. Note that in this experiment, the single-step solving algorithms for mapping and scheduling were applied as explained in Chapter 4.

First Case Study

In the initial case study, a distributed E/E system model is constructed comprising ten applications, each encompassing two threads with randomly assigned execution times represented as $t_i.e$, along with even periods denoted as $t_i.p$. Additionally, ten ECUs were employed as control nodes within this setup. For the subsequent case study, the system dimensions is extended to encompass fifty applications and ECUs. This expansion allowed to evaluate both the time taken for solving the constraint set and the time needed for generating MIP constraints (as depicted in Figure 7.2 (a)). The trend observed in the solving time presents a clear exponential pattern, particularly when the count of applications exceeds thirty [AMK23].

Second Case Study

In the second case study, the design of a distributed system featuring 8 ECUs and 10 applications, each application comprising two threads, is undertaken. The same constraints and solving options as those used in the first use case are applied. Subsequently, the scope is extended by increasing the number of applications from 10 to 70 while maintaining a consistent number of ECUs. This expansion allows us to keenly perceive the timing behavior during constraint set solving and generation. As described in Figure 7.2 (b), the solving time

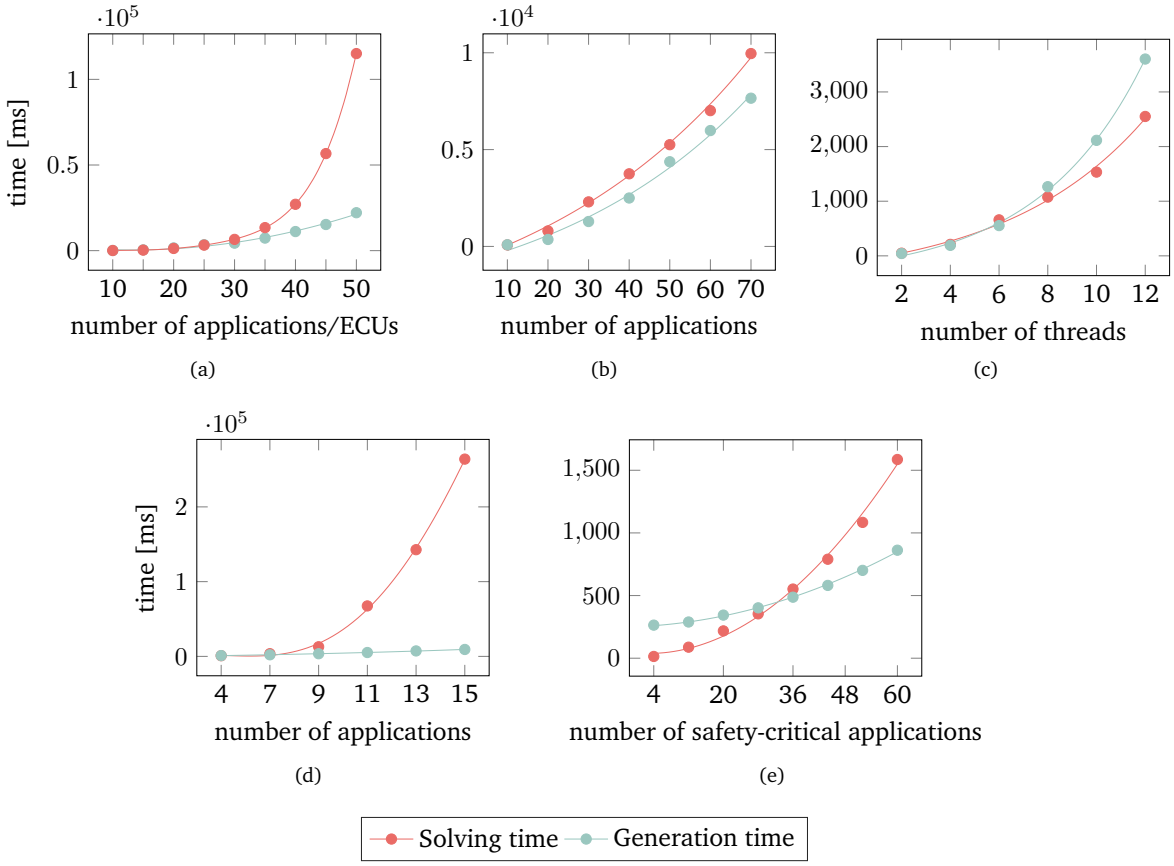


Figure 7.2: Design-time performance evaluation of the introduced model-based framework. The resource utilization constraint is applied to all use cases. (a) Each application consists of two threads. (b) It is applied on a topology including 8 ECUs, and each application has two threads. (c) The topology consists of 8 applications and 8 ECUs. (d) The topology comprises 8 ECUs, and each application consists of two threads with odd periods. (e) A zonal architecture including 15 ECUs is applied in this use case, and each safety-critical application has one thread. The boundary goals for resource utilization, maximum memory usage, and safety-critical mapping constraints are applied.

reveals exponential growth due to the augmented number of thread schedules that must be computed for each ECU. For instance, when we contemplate allocating 70 applications across the 8 ECUs, it implies that each ECU handles a minimum of eight applications or 16 threads (factoring in the RU objective). Each of these threads necessitates an accurate schedule, thus contributing to the observed trend in the solving time [AMK23].

Third Case Study

In the context of the use case depicted in Figure 7.2 (c), a precise measurement is conducted using a model comprising eight applications and eight ECUs. However, in this specific scenario, the thread count for each application is only increased from 2 to 12. As illustrated in Figure 7.2 (c), discernible exponential increments become evident in both solving and generation times [AMK23].

Forth Case Study

Since deriving schedules for multiple threads with odd periods is notably more intricate than even periods due to the complexities of least common multiple and hyperperiod calculations, a practical case is formulated to witness this phenomenon firsthand. As a result, Fig-

ure 7.2 (d) illustrates the observed durations within a system model comprising 5 ECUs and four applications, each housing four threads. The number of applications remains constant, ranging from 4 to 15, with the only variation in the threads' periods, which are odd numbers such as 5, 7, 11, and 13. The graphical representation visually demonstrates the exponential increase in the solving time, substantiating the intricacies involved in computing schedules for threads with odd periods. Conversely, the generation time depicts a relatively consistent trend when juxtaposed with the solving time.

Fifth Case Study

Figure 7.2 (e) illustrates the measurements committed to address a mapping problem and ascertain accurate schedules for application threads. For this purpose, a zonal architecture is employed, similar to the one presented in Figure 5.1 of Chapter 5, comprising 15 ECUs. The range of applications is expanded from 4 to 60, each flagged as safety-critical, encompassing a single application thread. During the solving phase, boundary constraints were established for ECU utilization and maximum memory usage on each ECU. Furthermore, supplementary mapping constraints were incorporated, as delineated within the automated mapping conditions, including redundancy provisions in safety-critical applications. It is crucial to emphasize that owing to the requisite redundancy when dealing with 60 safety-critical applications, a total of 120 applications are concurrently executed within the topology [AMK23].

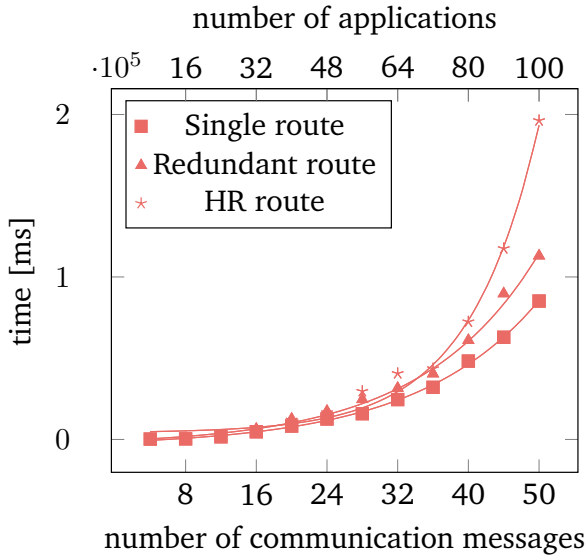


Figure 7.3: The measurement of solving time in a case study includes the full capabilities of the proposed tool while considering different types of paths for transferring communication messages. The same architecture as in Figure 7.2 (e) is used in this experiment. This study incorporates the multi-objective optimization, which encompasses end-to-end latency, response time, and LOR, as well as the goals for resource utilization and maximum memory usage [AMK23].

7.1.3 Evaluation of Full Capabilities of the E/E Designer Framework in a Single-Step Solving

In this experiment, the complete capabilities of the tool are assessed, incorporating single-step solving algorithms like CMR, CSCT, and PD, as detailed in Chapter 4. In this particular use case, the experiment encompasses automatic message routing and mapping, time-triggered scheduling of threads and communication tasks, and resolving path and message

dependencies, all within a single step. Various optimization objectives and boundary constraints are applied during this process. In addition, the impact of creating different types of paths on solving time while employing the features mentioned above is investigated. This investigation aims to comprehensively understand how the interplay between route types, communication volume, and application count manifests in the solving time dynamics. This analysis serves as a crucial step in assessing the efficiency and scalability of the introduced approach within a representative zonal E/E architecture [AMK23].

Figure 7.3 explores the impact of generating various route types on the solving time. This study involves the application of automatic mapping, alongside scheduling application threads and communication tasks within a single step, all within the context of the same zonal E/E architecture depicted in Figure 7.2 (e). The focal point of our investigation is to gauge how the creation of distinct routes influences the time required for solving. The solving time is quantified across three distinct route types, including single, redundant, and HR paths, each responsible for generating three independent routes. These routes are automatically determined based on mapping constraints. Concurrently, the number of communication messages is systematically varied from 4 to 50, and the count of applications is increased from 8 to 100. Each of these applications encompasses a single thread [AMK23].

Moreover, the multi-objective optimization is applied, as illustrated in Chapter 4, including end-to-end latency, response time, and LOR. This optimization uses a hierarchical methodology, with RU and maximum memory usage acting as single boundary objectives. To illustrate, consider Figure 7.3 as an example. In scenarios involving 50 communication messages and 100 applications, the solution consists of a variety of 50 paths. These paths include single, redundant, and HR routes, all carefully constructed from senders to receivers. These solutions also comprise task schedules over links solely for activated routes and application-to-ECU mappings and thread schedules on each individual ECU [AMK23].

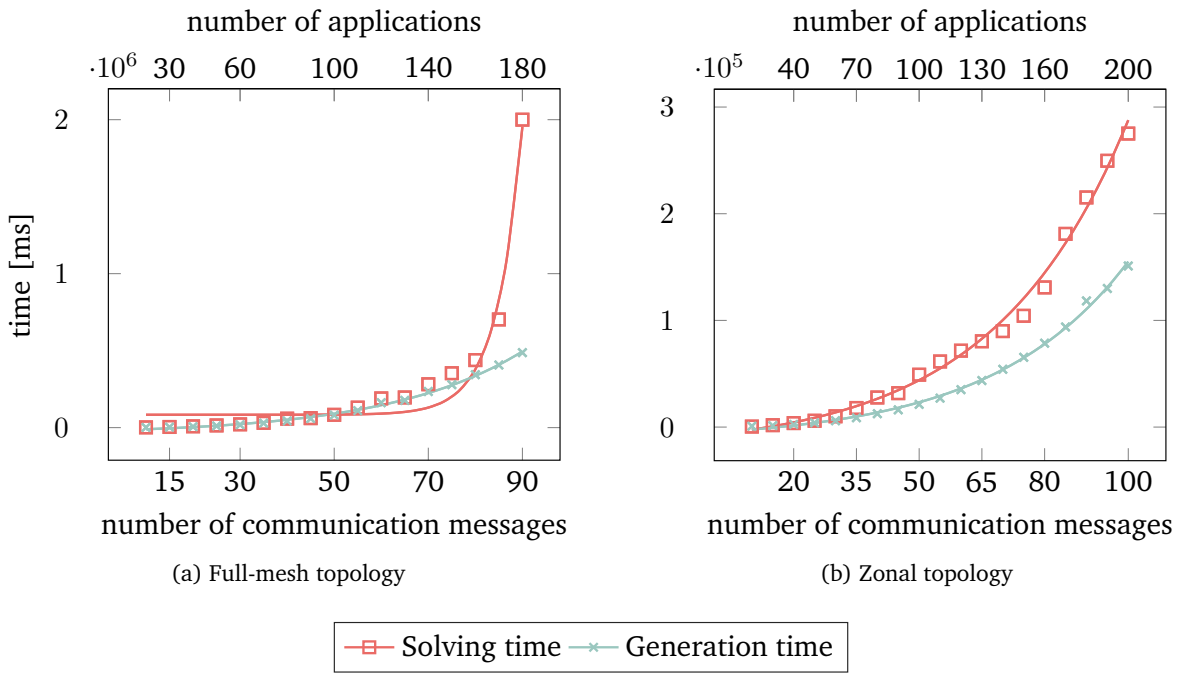
As anticipated, the number of generated routes can impact the solution time. Therefore, referring to Figure 7.3, the HR path necessitates more time due to the requirement of identifying three distinct routes in each scenario. Similarly, a redundant route exhibits a longer solving time than a single route (see Figure 7.3).

7.1.4 Scalability Analysis

In order to assess the scalability of the proposed framework, an evaluation comprising diverse aspects is conducted. The main objective is to evaluate the time required to generate system model variables and the subsequent resolution of the associated constraint set. This evaluation is performed on two distinct architectural paradigms: a comprehensive full-mesh topology and a more streamlined zonal topology. Each of these architectures comprises 15 ECUs. The introduced tool showcases its prowess within this evaluation by seamlessly handling various intricate tasks. The automatic routing of messages, leading to the generation of single paths, mapping, and scheduling application threads and communication tasks, are all adeptly solved in a single step for both the full-mesh and zonal architectures.

To further enrich the analysis, a multi-objective optimization approach comes into play. This approach considers goals, including the end-to-end latency, the response time, and the LOR. Moreover, the RU and the maximal memory usage serve as boundary objectives, effectively guiding the optimization process. Every application thread shares the exact execution times and periods. Likewise, the frame lengths of communication tasks are equal, with their periods aligned to match the time values of their corresponding senders' and receivers' periods.

Referring to Figure 7.4 (a), in the case of the full-mesh topology, the message count is expanded from 10 to 90, along with the number of applications growing from 20 to 180, each



(a) Full-mesh topology

(b) Zonal topology

Figure 7.4: Scalability analysis of the presented computer-aided tool. Generation and solving times for (a) a full-mesh architecture and (b) a zonal topology, each including 15 ECUs [AMK23].

equipped with its dedicated application thread. Meanwhile, as depicted in Figure 7.4 (b) for the zonal topology, the number of messages escalates to 100 while the applications increase to 200. This implies that the illustrated framework generates 100 routes along with their schedules, spanning from senders to receivers. As illustrated in Figure 7.4, the solving time for both scenarios experiences exponential growth. Furthermore, as expected, the full-mesh topology's solving time is notably longer than the zonal topology's. This is attributed to the larger space of exploration involved. Looking at the zonal topology shown in Figure 7.4 (b) and its expansion to accommodate 100 communication messages and 200 applications, the time taken for solving problems like mapping, scheduling, and routing is reasonable. These tasks are recognized as NP-hard problems [AHM19], but the introduced single-step solving approach handles them effectively. Notably, the values showcased here are well-suited for real-world applications in the automotive domain [AMK23].

7.1.5 Discussion

The outcomes of the experiments in the design stage vividly illustrate the efficacy of the aforementioned formulation and approach. It allows users to efficiently create their intended system models, including mapping, routing, and scheduling, all within a commendable time frame and following predetermined criteria. Moreover, the integration of single and multi-objective optimizations empowers the resolution of intricate challenges and diverse scenarios. It is significant to emphasize that the model-based tool, the E/E Designer, is suitable for synthesizing any type of vehicle E/E architecture and network topology, including various configurations of applications, threads, and communication tasks.

7.2 Run-time Evaluation

As mentioned in the previous section, the framework's performance and applicability are evaluated. In this section, the solutions computed by the tool are deployed on a real hardware platform to observe the applicability of the design-time solutions in a real-world implementation. Within the scope of this section, the solutions meticulously calculated by the tool find tangible expression through deployment on an actual hardware platform. This strategic deployment serves as a bridge, connecting the realm of design-time decisions with the dynamic environment of run-time execution. By doing so, it aims to explore how the design-time solutions seamlessly integrate with and translate into practical outcomes during active operational scenarios. In essence, this hands-on deployment provides an opportunity to assess the framework's efficacy in real-world contexts. As it is witnessed the design-time solutions unfolding in real-time scenarios, it gains a more comprehensive understanding of their adaptability, performance, and practical utility. This experiential approach contributes to the holistic assessment of the framework, bridging the gap between concept and implementation [AMK23; AFK21b].

7.2.1 Hardware Platform Analysis

As explained previously, the automotive E/E architecture is shifting towards a centralized architecture, necessitating high-performance computing units capable of processing vast amounts of data. In order to make an informed choice for selecting a vehicle centralized computer or a HPCU, a hardware analysis is conducted [AFK21b; AHK22].

Figures 7.5 (1) and (2) showcase the Drive AGX Xavier and Pegasus Developer Kits, respectively. These kits are designed to provide a comprehensive suite of standard software, hardware, and sample applications tailored to develop self-driving vehicles. Notably, they support diverse input/output (I/O) interfaces, including cameras, LiDAR, radar, and vehicle I/O. Both of these kits are equipped with two Xavier SoCs, each capable of accommodating six distinct processor types. These encompass a CPU boasting eight cores, a GPU, a deep learning accelerator (DLA), a programmable vision accelerator (PVA), an image signal processor (ISP), and a stereo/optical flow accelerator. The Pegasus Developer Kit shown in Figure 7.5 (2) leverages the computational power of additional Turing GPUs, enabling it to achieve an elevated tera operations per second (TOPS) rate [NVI21; AFK21b]. The DRIVE AGX Xavier software stack comprises sample applications, a software development kit (SDK), an embedded RTOS, and a hypervisor. Although Nvidia offers a preconfigured firmware package, it limits industrial partners' access to platform development kit (PDK) details. As a result, Nvidia's proprietary hypervisor is not applicable for academic use.

The MPPA-DEV4 development platform, as depicted in Figure 7.5 (3), serves as another central computer within the vehicle. This platform provides a readily available environment for evaluating, developing, and optimizing applications across domains such as automotive, data-centric, robotics, and communication [KAL20]. Similarly, Figure 7.5 (4) introduces the R-Car H3 and M3 Starter Kits, designed to bolster automotive software development. These products have played a pivotal role in facilitating the establishment of open-source automotive Linux environments [REN21; AFK21b]. AVA3501 is a computing platform designed for autonomous vehicles, featuring components like the Intel Xenon 9th Gen CPU and RTX8000 GPU (refer to Figure 7.5 (5)) [ADL21]. The last associated HPCU, named Nuvo7208VTC, is equipped with an 8-core processor (see Figure 7.5 (6)) [Neo21].

Drawing upon the analysis provided above, several critical factors informed the decision-making process. These factors encompass computational power, the number of cores, the

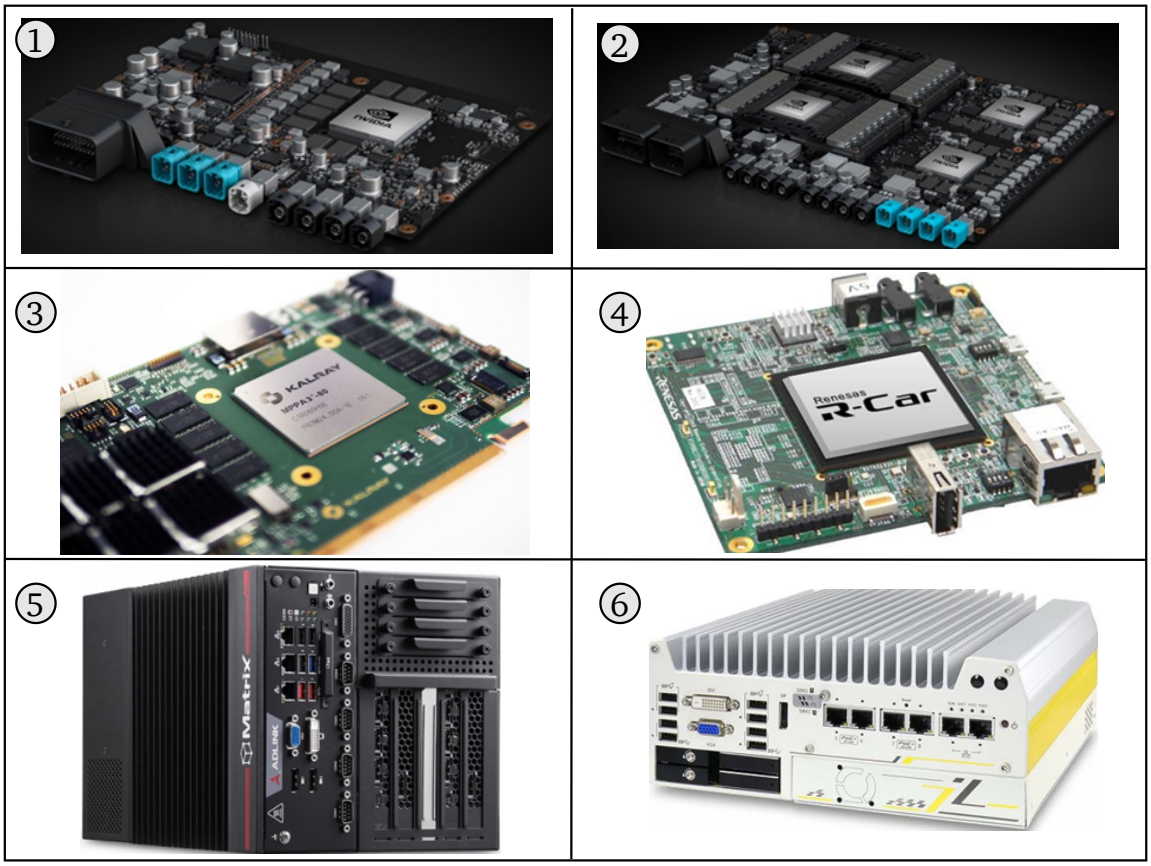


Figure 7.5: The six relevant development kits serve as an HPCU including (1) Nvidia Drive AGX Xavier, (2) Nvidia Pegasus, (3) MPPA-DEV4 development platform, (4) R-Car H3 and M3 Starter Kits, (5) AVA 3501, (6) Nuvo 7208VTC.

spectrum of automotive applications, the diversity of interfaces, the quality of customer support, and the comprehensiveness of documentation. Within these considerations, Nvidia AGX Drive emerged as the chosen HPCU for the envisaged hardware platform, primed for in-depth run-time evaluation. The capabilities offered by Nvidia AGX Drive align seamlessly with the demanding requirements of the introduced evaluation. Its computational may, augmented by a robust core count, promises a solid foundation for the real-time processing of intricate automotive tasks. The array of interfaces it offers caters to the multifaceted connectivity needs of modern automotive systems. Furthermore, the availability of substantial customer support and well-documented resources adds a layer of reliability and ease to the operational endeavors. With Nvidia AGX Drive at the helm, the proposed hardware platform appeared poised for an insightful run-time assessment [AFK21b; AMK23].

Nonetheless, it is important to note that due to technical considerations linked to Nvidia, as elaborated upon in the subsequent subsections, an alternative development kit was also subjected to testing and subsequent comparison with the Nvidia solution.

7.2.2 Experimental Setup

This subsection outlines the experimental evaluation of the design-time solutions computed by the proposed tool as they were deployed on real hardware platforms. The experiments encompassed utilizing three distinct hardware platforms commonly featured in automotive systems.

Figure 7.6: The mapping experimental setup using the Nvidia Drive AGX. On the left is the host computer, and on the right is the Nvidia Drive AGX Xavier with the power adapter. Both are connected to a monitor and other peripherals.

The primary hardware platform chosen for these experiments was the Nvidia Drive AGX Xavier Developer kit, which stands as a prominent selection for HPCUs in autonomous driving (see Figure 7.6). This kit revolves around two Xavier SoCs, each encompassing an 8-core ARM CPU, RAM, and hardware accelerators tailored for deep-learning inference, as illustrated in the previous subsection. This platform orchestrates a modified version of Ubuntu 18.04 integrated with the Preempt-RT-patch as its operating system. In addition to the Nvidia platform, a series of experiments were conducted on an evaluation board grounded on the Intel i210, effectively simulating the functionality of a communication network gateway [Int23]. Lastly, a Discovery kit featuring an STM32L476VG microcontroller was harnessed to exemplify an energy-efficient ECU [ST23]. This triad of hardware platforms collectively forms the foundation for empirically assessing the framework's design-time solutions [AMK23].

Reference computer: As a reference based on Figure 7.6, a standard computer with an Intel Core i7-4770 @ 3.90 GHz processor with eight cores, 16384 MB of DDR3 memory, a 500 GB disk, and an NVIDIA GeForce GTX 645 graphics card with 1024 MB was used. In addition, all performance tests ran on the Nvidia Drive natively to determine the actual overhead produced by the hypervisor configurations.

Software Setup

In the following bullet points, the details of the software setup used in the experimental setup are presented.

- **Tasks**: Each task is modeled as a Linux process running a customized benchmark application. This benchmark employs the Gauss–Legendre algorithm to calculate ten digits of pi in an infinite loop, making it CPU-bound. Additionally, the setup includes sender and receiver applications that facilitate TCP-based message transfers [AMK23].

- **Scheduling and Dispatching**:

  The scheduling process is simulated by employing a separate standard Linux process, which is endowed with the highest real-time priority of 99. This is achieved using a C

timer function that triggers a callback every 1000 nanoseconds. This callback ascertains whether a task should be scheduled or stopped at that specific timestamp. Prior to the start of the simulation, it is imperative to specify the number of hyperperiods. This enables the anticipation and pre-calculation of each task's start and stop times, which are then meticulously stored in a sorted array. Consequently, during the timer callback, the comparison is confined to the current counter value and the leading entry in the aforementioned array. This optimization drastically curtails the callback's invocation time to the bare minimum. When task initiation or cessation is warranted, the scheduler executes the dispatch by transmitting a POSIX signal to the corresponding task. The SIGKILL signal orchestrates the killing of the task, while the SIGCONT signal indicates its restart [AMK23].

- **Task mapping**: The tasks are assigned to specific cores before the simulation commences. Their CPU affinity is established using the taskset command, which instructs the Linux scheduler to associate the process with a designated CPU core to achieve this. To ensure the uninterrupted execution of these tasks, each one is endowed with the second-highest real-time priority of 98. This strategic prioritization by the Linux scheduler places these tasks above all other processes, though they remain susceptible to interruption by the simulation scheduler. Moreover, CPU core 0 is reserved for all remaining system operations and external processes, excluding bounded kernel threads. This deliberate allocation of tasks serves the purpose of isolating the simulated cores, thus maintaining a focused and controlled environment [AMK23].

- **Task synchronization**: The precision time protocol (PTP) was employed to synchronize the system clocks of multiple nodes. Leveraging the inherent PTP hardware support across all utilized devices, a master clock offset value of approximately 100 ns was attained. Through this synchronization of system time across nodes, the starting time of the simulation can be uniformly communicated to all devices, thereby ensuring a synchronized simulation start. Other performance metrics encompassed CPU and RAM utilization, as well as the thermal behavior of the CPU [CBB05; AMK23].

- **Monitoring Mechanism**: As for the run-time behavior of the operating system and middleware within the hardware platform, uncertainty arises due to event-based activities, such as application service discovery and other dynamic, interacting processes

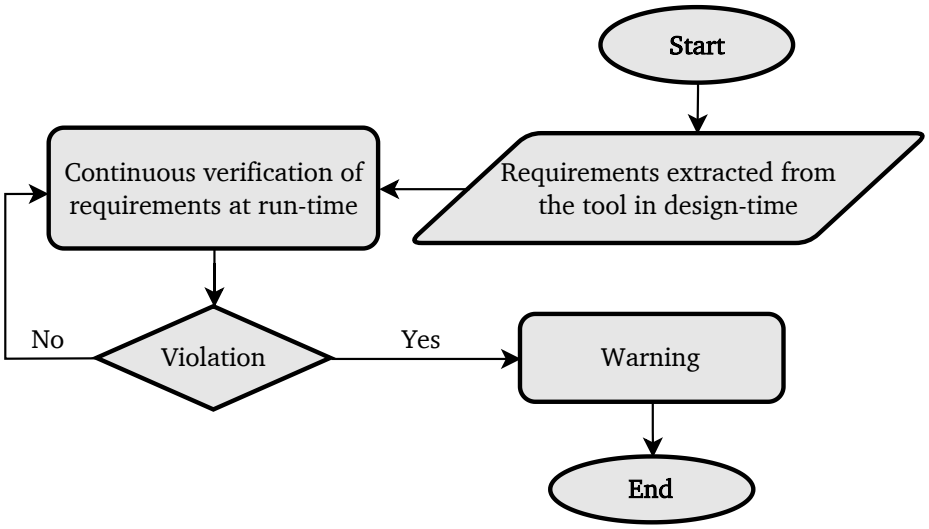


Figure 7.7: The Monitoring mechanism flow chart.

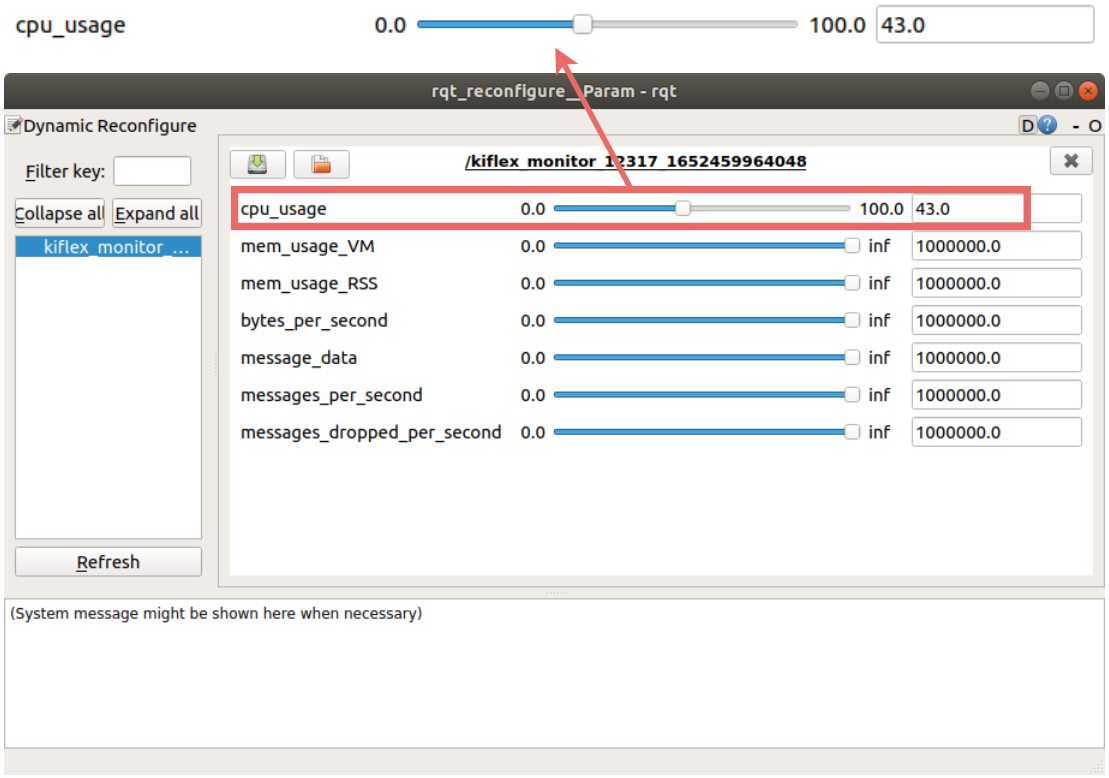


Figure 7.8: The Monitoring GUI. The threshold values for each specified requirement can be chosen. For example, the threshold value of the CPU usage can be defined inside the red rectangle.

that lead to non-deterministic utilization of system resources. Given this complexity, creating and considering pertinent constraints at the design stage by the presented tool becomes impractical. In order to establish the validation of requirements, e.g., timing

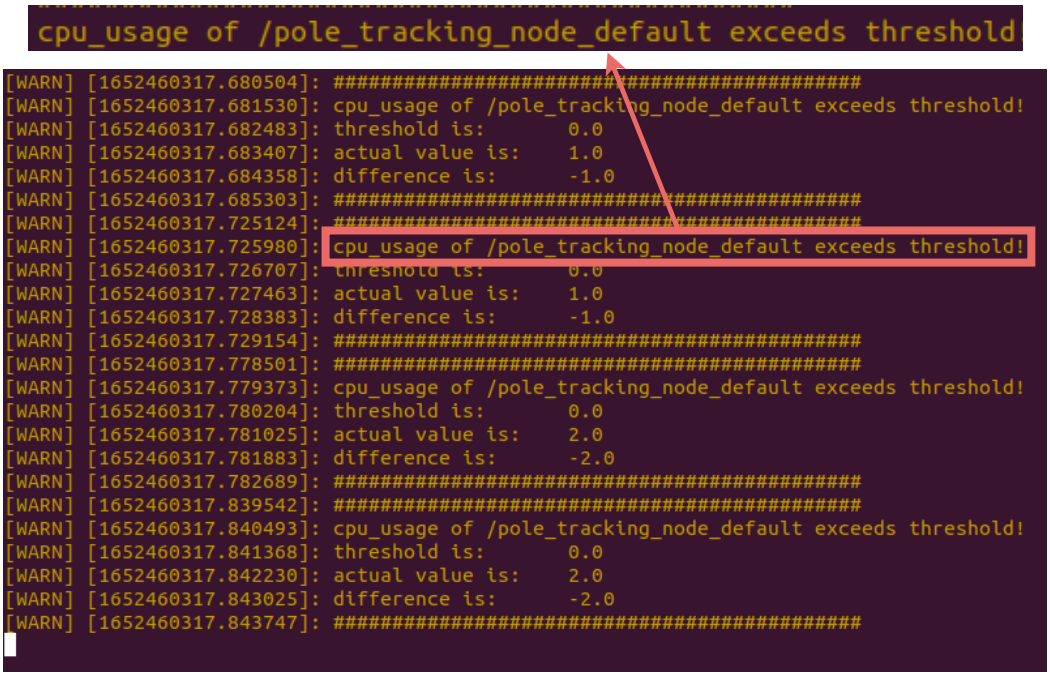


Figure 7.9: The Monitoring GUI. A warning message can be observed in case of violation (red rectangle).

requirements, during run-time after the deployment of solutions computed by the introduced framework onto the hardware platform, a monitoring mechanism has been developed. The methodology outlined in [ASK21] is followed to execute this mechanism. This approach introduces a monitoring mechanism for identifying timing violations in autonomous driving platforms.

The flow chart, depicted in Figure 7.7, illustrates the steps involved. To mitigate risks arising from requirement violations, the monitoring module receives the predefined requirements initially given to the tool. Concurrently, it collects real-time values from the hardware platform. In the subsequent phase, a continuous comparison and verification process is initiated between the design-time requirements and the real-time status concerning these requirements. Furthermore, if any violation occurs, an alert is issued to announce the breached requirement, as illustrated in Figure 7.7 [AFK21b]. Figure 7.8 shows the graphical user interface (GUI) for the integrated monitoring mechanism as part of the main GUI. Within this interface, the option to establish a threshold value for each requirement is presented, exemplified by CPU usage, as depicted in Figure 7.8. Furthermore, in instances where the value of any requirement surpasses the determined threshold, an immediate warning message is generated, as demonstrated based on Figure 7.9 [ASK21; AFK21b; AMK23].

The monitoring mechanism is a substantial part of the performance evaluation as the

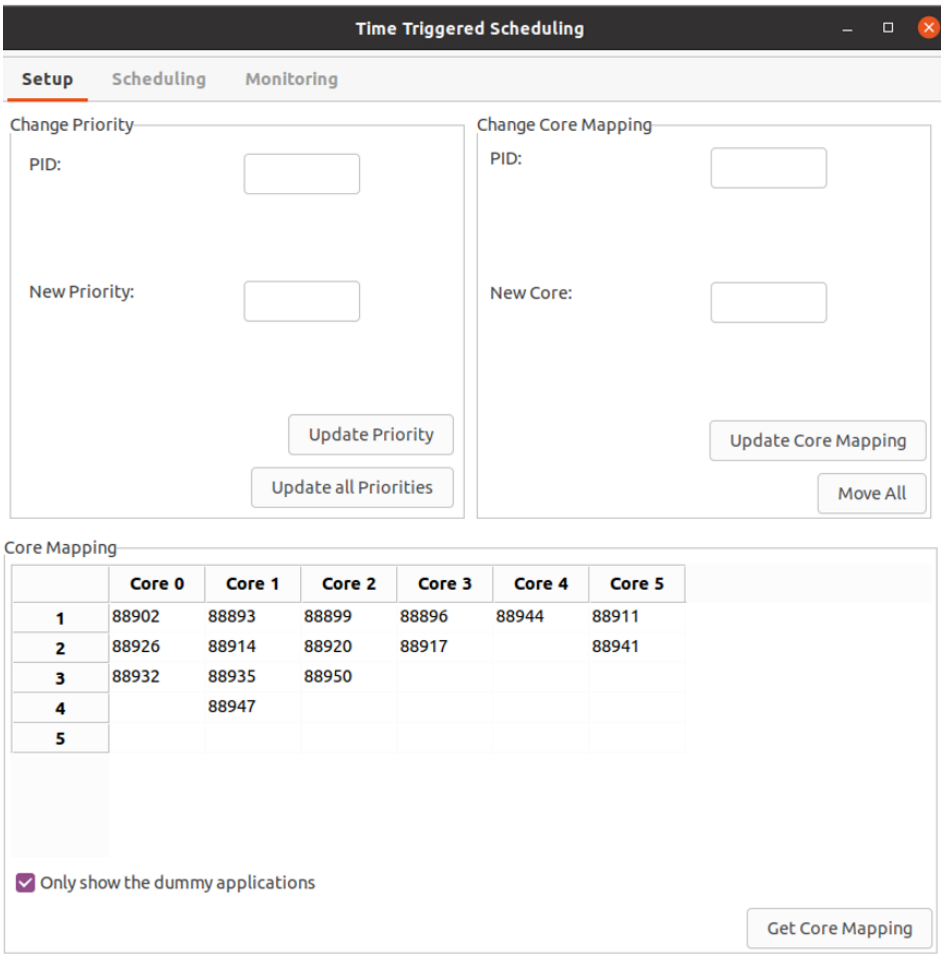


Figure 7.10: The primary GUI including all features. Users can select the priority and assignment for each task on the window.

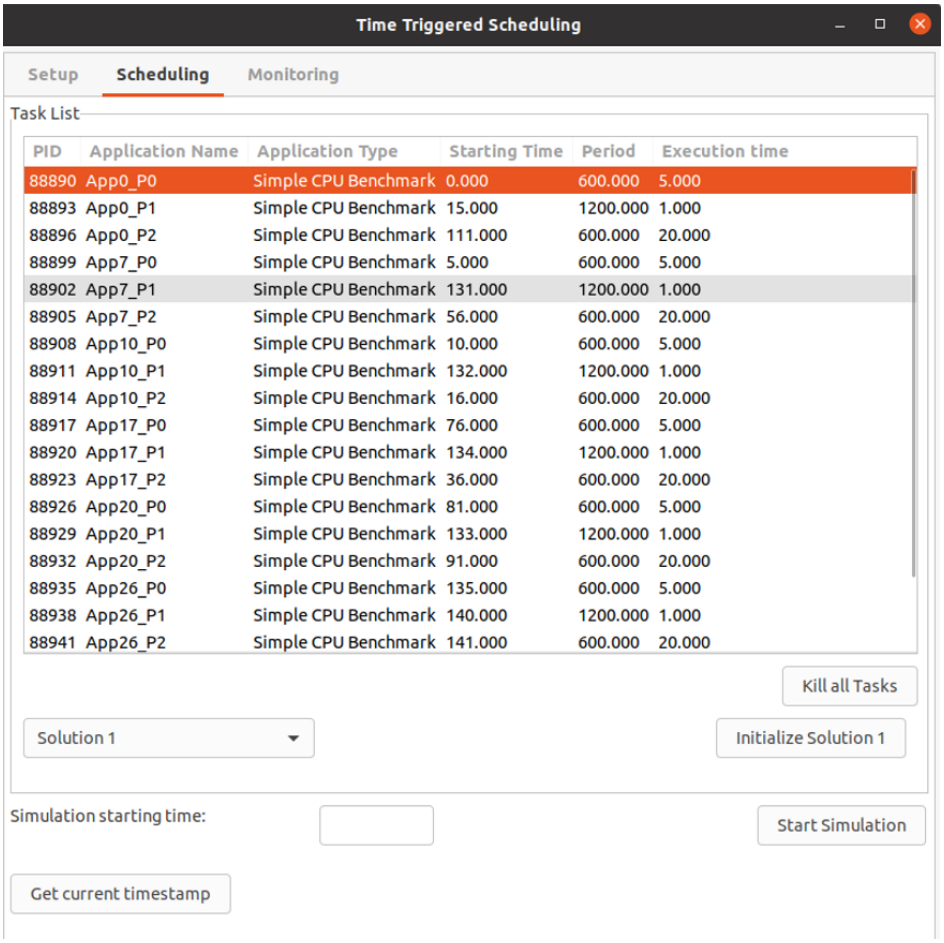


Figure 7.11: The primary GUI including all features. The window visually represents the priority, period, and execution time associated with each task.

measuring must be very accurate but also very light-weight to not influence the measured metrics [ASK21; AFK21b; AMK23]. The primary performance metric during the ongoing run-time evaluation pertained to each task's initiation and cessation times. This specific metric allowed for the computation of both start and stop jitter, representing the variance between the actual and expected start and stop times. The determination of start and stop times was facilitated through a meticulous tracing of system calls that signified transitions in process states, a process efficiently carried out using*strace*. Other performance metrics included CPU and RAM utilization, as well as the thermal development of the CPU, which are discussed in the following.

- **GUI**: A user-friendly graphical interface has been developed to facilitate the deployment of the calculated solutions. This interface seamlessly integrates all the functionalities described above into a single program. This integration enables the automated deployment of mapping and communication solutions onto the hardware, simplifying the process of replicating experiments. Moreover, the developed GUI assumes the role of initiating the monitoring procedures. It adeptly manages the collection and processing of the measured data, a process elucidated in the finer details of the monitoring mechanism. This cohesive integration ensures a seamless and efficient flow of tasks, enhancing the overall user experience and research methodology [AMK23; ASK21]. Figure 7.10 illustrates the GUI configuration for deploying the mapping and time-triggered scheduling solutions on the hardware platform. For example, based on Figure 7.10, the

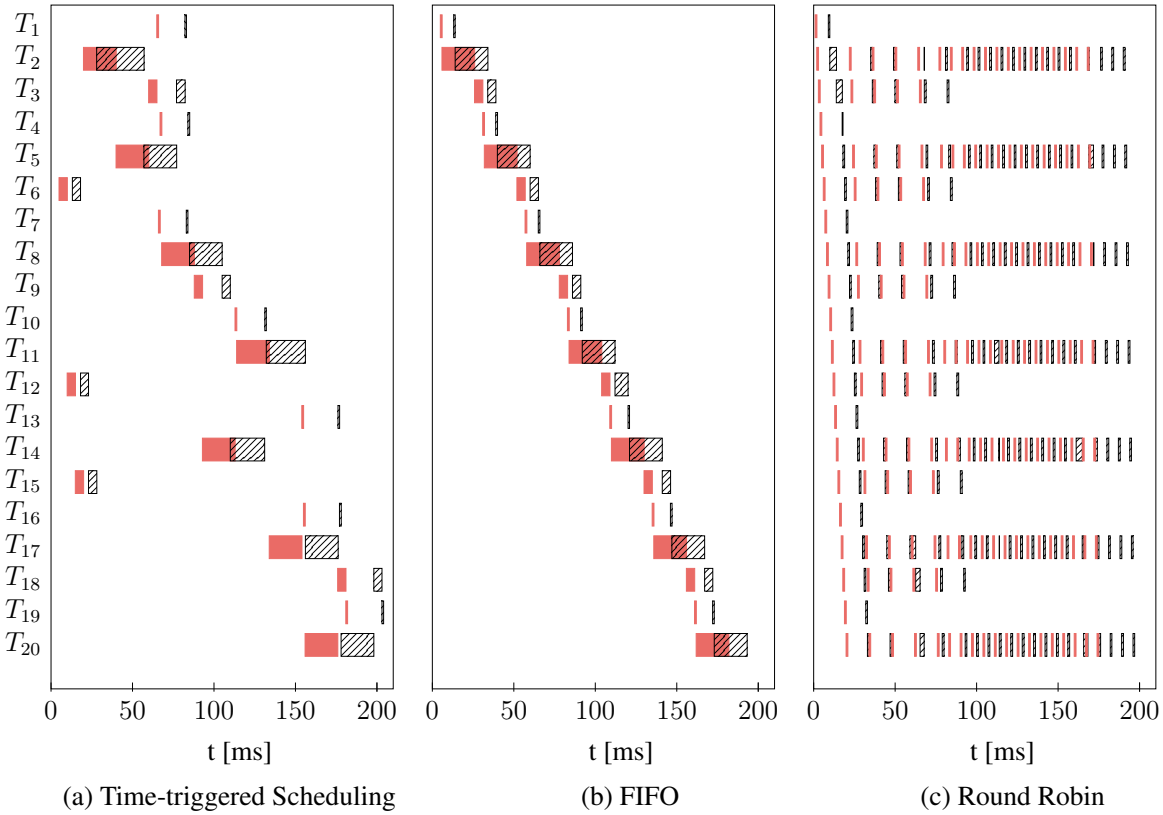
Table 7.1: Periods and execution times of threads for each sample application.

Name	$t.p$ (Period) [ms]	$t.e$ (Execution time) [ms]
T_0	600	5
T_1	1200	1
T_2	600	20

priority of each task and its allocation to a specific core can be managed. In Figure 7.11, the designated period and execution time of each task, along with its name and priority, are presented.

Mapping evaluation

The mapping evaluation was based on 40 applications, each consisting of three threads. The periods and execution times of these threads can be found in Table 7.1. A total of 120 application threads are mapped onto different cores. The time-triggered scheduling was evaluated against first in, first out (FIFO) and Round Robin with a time quantum of 1 ms, which was selected based on the shortest execution time [AMK23; LA07; RT08]. FIFO scheduling is a simple and intuitive scheduling algorithm that operates on a first-come, first-served basis. In this approach, the process that arrives first is executed first, and subsequent processes are executed in the order of their arrival [LA07]. In comparison, Round Robin scheduling is a preemptive scheduling algorithm that allocates a fixed time quantum to each



(a) Time-triggered Scheduling (b) FIFO (c) Round Robin

Figure 7.12: Gantt charts of the different scheduling solutions for CPU 5. The red bars represent the planned execution, while the hatched bars stand for the actual execution [AMK23].

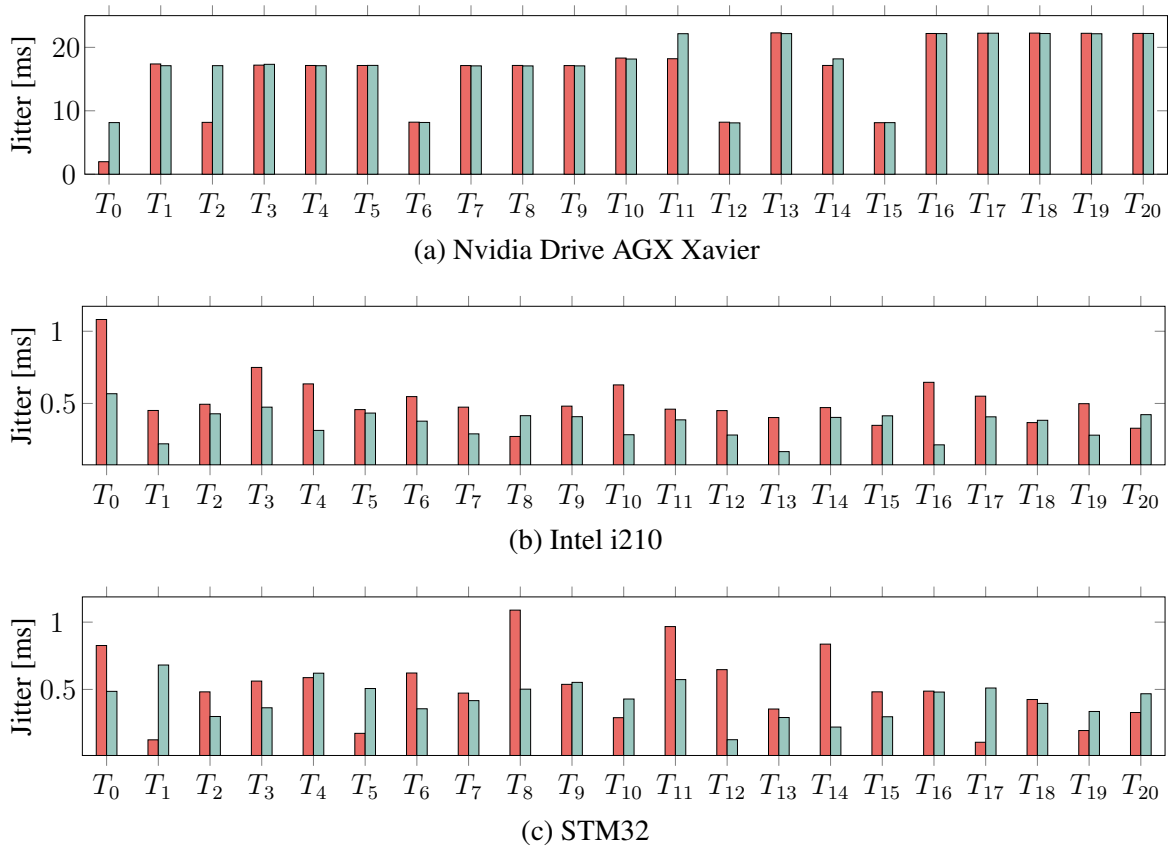


Figure 7.13: Start and stop jitters of each thread measured for the time-triggered scheduling over one hyperperiod. These are the measurements on CPU core 5. Red bars represent the start jitter, and the green bars the stop jitter [AMK23].

process in a cyclic manner. It ensures fairness by allowing each process to execute for a predefined time slice or quantum before moving to the next process. If a process does not complete its execution within the time quantum, it is temporarily suspended, and the next process in the queue is allowed to run. The suspended process is then placed back in the ready queue, and execution resumes from where it left off during the next scheduling cycle [RT08].

Given that the execution of all cores operates independently from each other and the workload is distributed relatively evenly across all cores in the assessed solution, for the sake of illustration, only the results from one core are presented. This is due to the similarity in behavior demonstrated by the other cores.

Figure 7.12 depicts the expected and actual start and stop times for the three evaluated scheduling policies. It is noticeable that across all setups, there are cases where threads commence even after the expected stop time. This tendency is more pronounced with shorter execution times, making it especially notable in the Round Robin configuration.

Figure 7.13 shows the jitter measurements for time-triggered scheduling across various hardware platforms, namely Nvidia Drive AGX Xavier, Intel i210 development kit, and STM development kit [Int23; ST23; NVI21]. The observation reveals a noteworthy disparity in jitter generation. Particularly, the Nvidia Drive exhibits a notable level of jitter. The average jitter on the Nvidia Drive surpasses that on the Intel i210 by a factor of approximately 36.6, despite both setups being the same [AMK23].

The results of the CPU, RAM, and temperature measurements are illustrated in Figure 7.14. While there are observable distinctions among all the setups, they are not deemed significant. Moreover, none of the scheduling schemes exhibits superior performance across

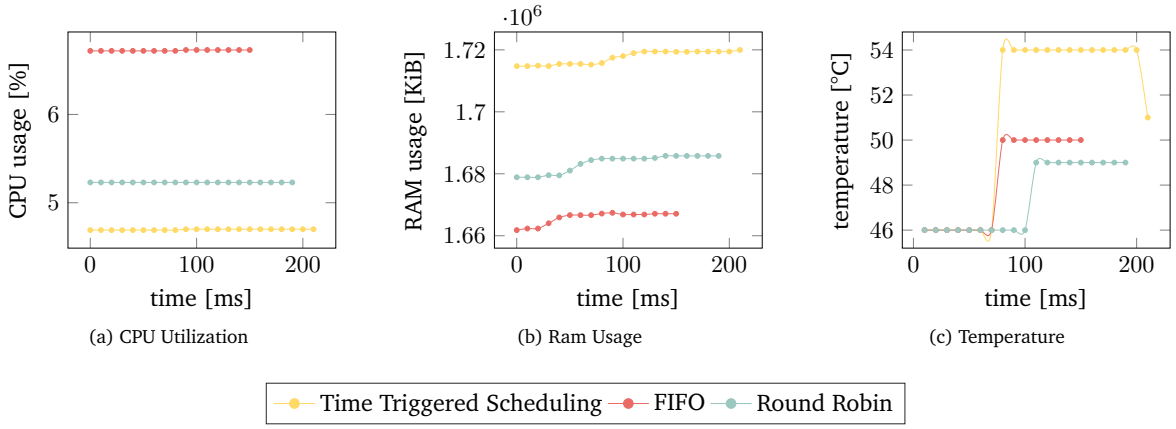


Figure 7.14: Comparison of different metrics. Yellow is the time-triggered scheduling, red is FIFO scheduling, and green is Round Robin scheduling [AMK23].

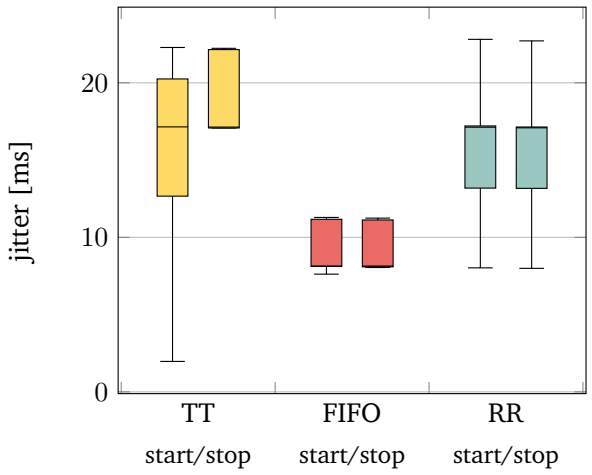


Figure 7.15: Spread of start and stop jitter for the tested scheduling policies. The box plot shows the minimum and maximum values, the lower and upper quantiles, and the mean. The yellow, red, and green boxes represent results for time-triggered, FIFO, and Round Robin scheduling schemes, respectively.

all metrics.

Furthermore, the offset between the actual and expected execution times remains comparatively constant with FIFO scheduling. This can also be seen by looking at Figure 7.15, which shows that the jitter is much more predictable for FIFO than for the time-triggered scheduling solution. This indicates that the jitter produced by the dispatcher is centered around a constant value with a low spread. Nevertheless, all threads finished within their specified period, and no deadline violation occurred in any of the tested setups [AMK23].

Communication evaluation

Apart from the mapping evaluation, a communication assessment was performed where an E/E architecture is modeled using the introduced computer-aided tool, and its provided solution is deployed on an actual hardware setup with the same topology designed by the tool. The solution consists of schedules for application threads executing on each ECU, as a sender and receiver, and for communication tasks routing over each link. The multi-objective optimization comprising end-to-end latency and response time was applied to the solution. This experiment measured the end-to-end latency and response time for communication messages [AMK23].

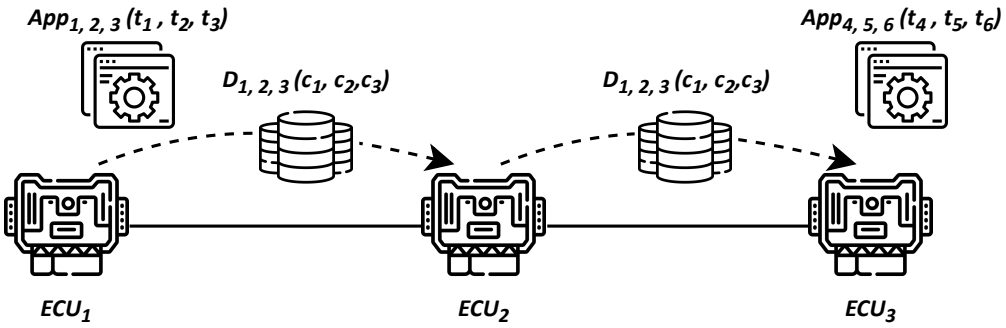


Figure 7.16: Topology of the tested communication setup. Threads 1-3 on ECU_1 each send a communication packet to threads 4-6 on ECU_3. All three packets (c_1, c_2, c_3) are routed over ECU_2 [AMK23].

Communication was tested using a scenario consisting of three connected nodes: an Nvidia TX2 Developer kit (ECU_1), the Nvidia Drive AGX Xavier (ECU_2), and an Intel i210 Developer kit (ECU_3) based on Figure 7.16. On ECU_1 ran three applications, each comprising a thread, each of those sending a communication packet to a respective receiver task on ECU_3 via ECU_2. Figure 7.16 visualizes the setup. As mentioned earlier, the end-to-end latencies and response times of the three communication messages, each including a communication task defined above, are measured [AMK23].

Table 7.2: Communication solution calculated by the presented framework.

Name	Start time [μs]	Stop time [μs]
$t_1.st$	0.00	90.00
$t_2.st$	90.00	290.00
$t_3.st$	290.00	440.00
$c_1.st^{l_1}$	100.0	112.60
$c_2.st^{l_1}$	300.00	312.60
$c_3.st^{l_1}$	450.00	462.00
$c_1.st^{l_2}$	127.60	140.20
$c_2.st^{l_2}$	327.60	340.20
$c_3.st^{l_2}$	477.60	490.20
$t_4.st$	150.20	250.20
$t_5.st$	650.20	850.20
$t_6.st$	500.20	650.20

The communication links had a theoretical bandwidth of 1000 Mbit/s; however, 940 Mbit/s were measured when also running PTP synchronization over the same links. The size of the communication messages was chosen to fit into one single Ethernet frame of 1500 B size. Therefore, the frame length of each communication task was set to 12.6 μs. The solution calculated by the proposed tool is presented in Table 7.2. It shows the start and stop times for the sender and receiver applications and the communication tasks over one hyperperiod [AMK23].

Figure 7.17 shows the results of the experiments. It is noticeable that the measured end-to-end latencies do not vary much but are in the order of 10 ms. Even though the actual frame length is considerably smaller than these values, the influence of start and stop jitters

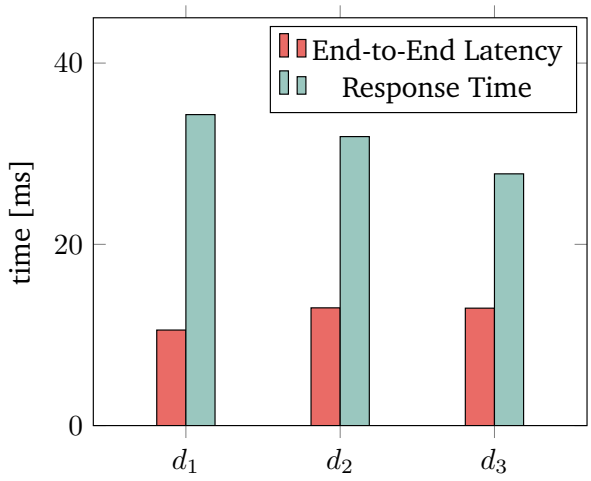


Figure 7.17: Measured end-to-end latency and response times of the three communication messages using Nvidia Drive AGX, Nvidia TX2, and Intel i210 developer kits as the setup.

in the application threads profoundly impacts the overall latency metrics. This, in turn, led to instances of deadline violations during the communication testing phase [AMK23]. This phenomenon can be attributed to the significant jitter introduced by the Nvidia AGX Drive, vividly illustrated in Figure 7.13.

Due to this effect, a decision was made to execute the communication experiment using STM32-based development boards [ST23]. These boards exhibit a notably lower scheduling

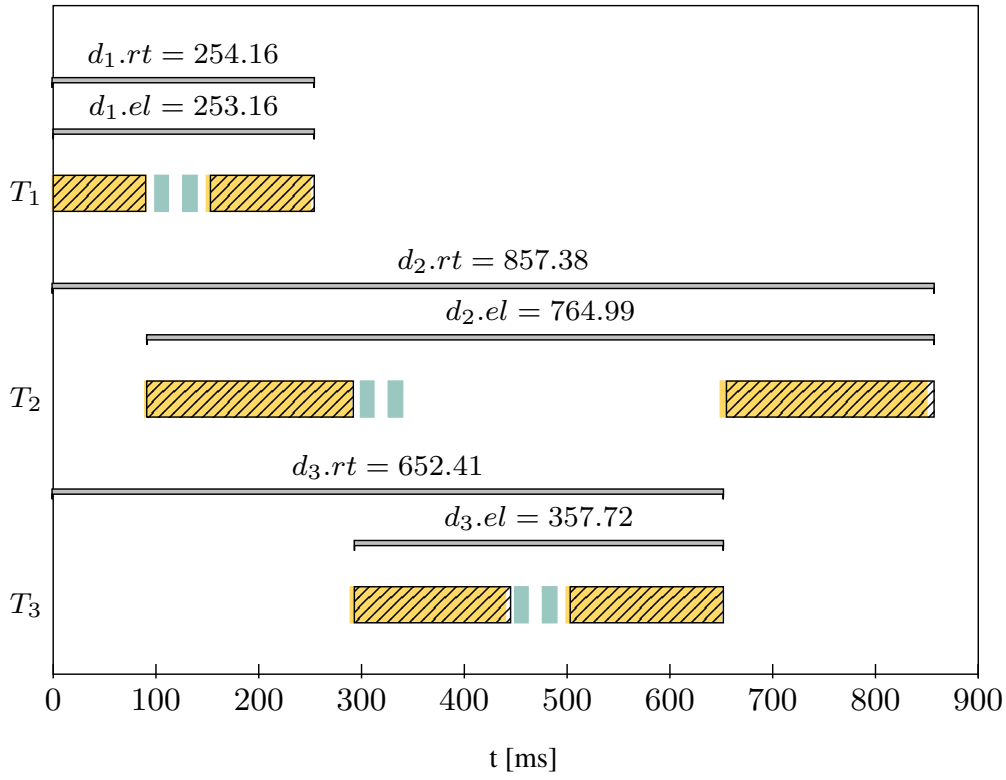


Figure 7.18: Results of the communication evaluation experiment using STM development kits. The yellow bars represent the expected execution, while the hatched bars represent the actual execution times of the sender and receiver applications. The green bars indicate the planned schedule of the communication messages. The response time and end-to-end latency of each communication message are also shown [AMK23].

jitter, ensuring minimal disruption to the execution of the synthesized communication solution. It is important to note that this particular experiment adhered to the same topology and design-time outcomes, shown in Table 7.2, as the initial communication experiment, both of which were generated by the tool.

The CAN configuration was chosen as the communication protocol between ECUs in this setup. This choice stems from its widespread usage within the automotive domain and its comparative simplicity when contrasted with other protocols regarding the essential hardware setup and software implementation, as described in Chapter 2. Moreover, it has relatively little overhead and thus simplifies the calculations. The communication links had a theoretical bandwidth of 11,520 bytes/s, which is a common baud rate. The size of the communication messages was chosen to be 145 bytes, deliberately kept small to fit into a single frame and avoid fragmentation and long verification times that can potentially influence the results. Each communication task's frame length was set to 12.6 ms, calculated by dividing the size of the communication message by the bandwidth. The interpacket gap was set to 10 ms. The user can specify these communication properties for each network protocol in the framework, similar to how timing parameters can be determined in the framework's frontend.

Figure 7.18 shows the results of the communication experiments using the STM boards. It can be seen that the measured response times and end-to-end latencies for all three communication messages, d_1, d_2, and d_3, closely align with the anticipated values, indicating the absence of any deadline violations.

7.3 Quantitative and Qualitative Evaluation

To evaluate the performance, usability, and practicality of the illustrated model-based tool, a series of use cases were addressed through both manual and automated approaches. The following subsections provide a detailed analysis of the findings, encompassing both quantitative and qualitative aspects.

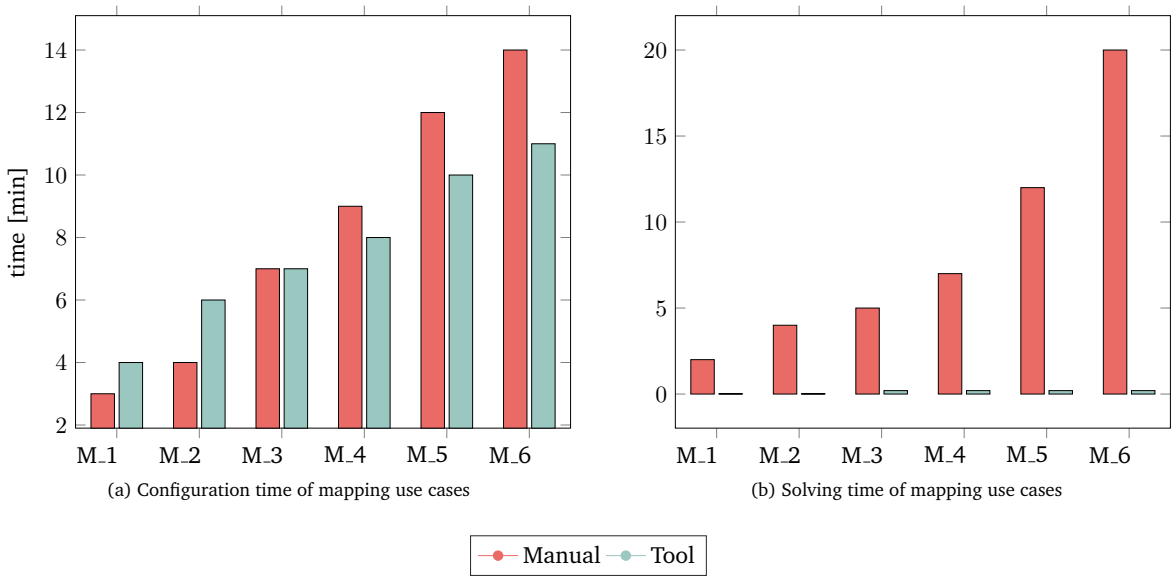
7.3.1 Quantitative Analysis of Various Case Studies

For the quantitative evaluation of the proposed framework, a series of mapping and mapping & routing use cases are undertaken. These cases are addressed through two distinct approaches: manual solution and utilization of the E/E Designer tool.

Mapping Case Studies

The mapping use cases center solely on accurately assigning applications to diverse ECUs and determining schedules for the application threads. However, these use cases do not encompass aspects such as message routing and scheduling communication tasks. To facilitate the evaluation process, the ensuing use cases have been established. The following case studies are given to a group of students to execute the quantitative and qualitative analysis utilizing manual and tool-assisted approaches.

- M_1 :4 ECUs, 4 applications
- M_2: 6 ECUs, 8 applications
- M_3: 8 ECUs, 12 applications



(a) Configuration time of mapping use cases

(b) Solving time of mapping use cases

Figure 7.19: Results of the quantitative evaluation for the mapping case studies using a manual approach and the E/E Designer tool. (a) The required setup time for each use case. (b) The required solving time for each use case.

- M_4: 8 ECUs, 18 applications
- M_5: 8 ECUs, 24 applications
- M_6: 8 ECUs, 32 applications.

Note that each application consists of two threads within the above-mentioned use cases. To address these use cases, the following constraints are taken into consideration.

- Time-triggered scheduling constraints for application threads. This ensures the feasibility of application execution
- Each ECU must execute at least one application

Figure 7.19 (a) illustrates the time required for modeling and designing each of the use cases, as discussed earlier, employing both the manual and proposed framework approaches. Notably, for smaller topology sizes, the proposed model-based framework introduces a slight overhead that leads to a marginally extended configuration time. However, as the topology size scales up, particularly when the case study encompasses multiple objects sharing similar properties, a discernible acceleration in the process becomes evident. This acceleration is closely tied to the intrinsic capability outlined in Chapter 5, which pertains to the automatic creation of software/hardware components. This feature empowers the simultaneous generation of multiple elements, thereby contributing significantly to the observed speedup.

In Figure 7.19 (b), the mapping use cases are illustrated using two distinct approaches: manual and tool-assisted. It is noteworthy that the solving time exhibits exponential growth as the number of components in the topologies increases when these case studies are solved manually. While it may be relatively straightforward to intuitively derive solutions for use cases with a limited number of constraints, the complexity rises exponentially as the constraints multiply. In contrast, referring to Figure 7.19 (b), the time required for solving these same use cases via the tool remains remarkably low and consistent in comparison to the manual method. This holds true even as the scale of topologies expands, with the time needed to solve the mapping constraint set remaining relatively unaffected. Furthermore, the addition of further constraints does not significantly impact the solution time. This serves as

evidence of how the proposed framework adeptly streamlines the design process for vehicle E/E architectures.

Mapping & Routing Case Studies

In addition to the mapping use cases, routing and mapping use cases have also been taken into account. These specific use cases serve the purpose of comprehensively evaluating the overall functionality of the E/E Designer framework. The following case studies are considered and given to a group of students to perform the quantitative and qualitative analysis similar to the mapping case studies.

- M&R_1: 5 ECUs, 2 applications with up to one thread, 1 communication message
- M&R_2: 5 ECUs, 8 applications with up to two threads, 4 communication messages
- M&R_3: 5 ECUs, 12 applications with up to two threads, 6 communication messages
- M&R_4: 5 ECUs, 16 applications with up to two threads, 8 communication messages
- M&R_5: 5 ECUs, 20 applications with up to two threads, 10 communication messages
- M&R_6: 5 ECUs, 28 applications with up to two threads, 14 communication messages
- M&R_7: 5 ECUs, 74 applications with up to two threads, 37 communication messages

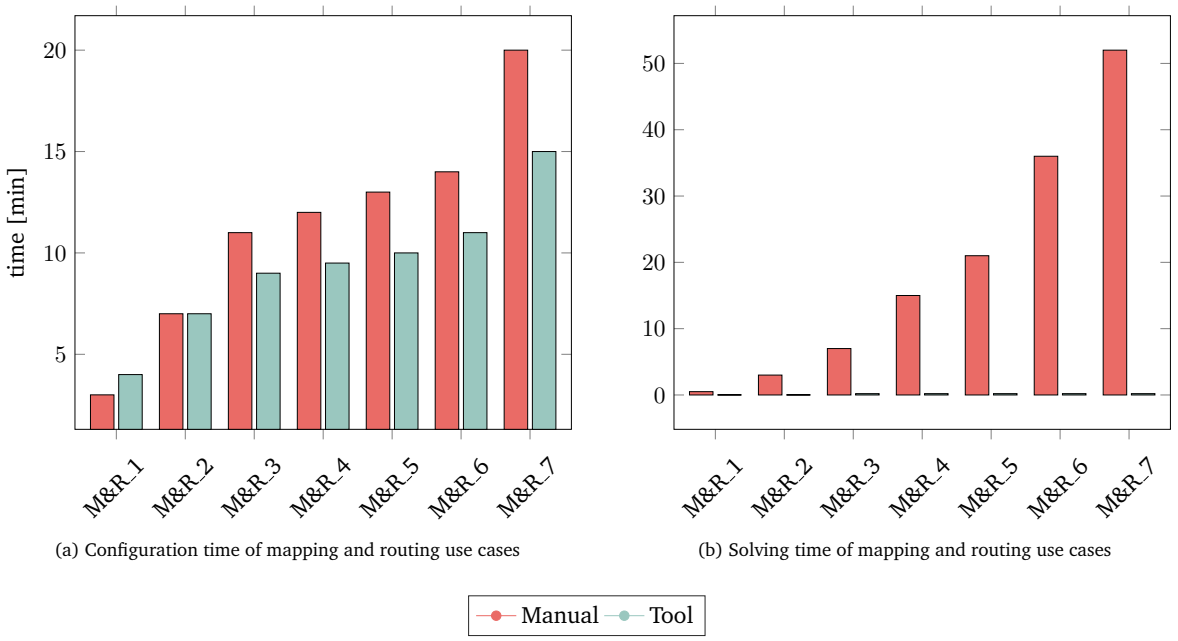
To handle the routing and mapping use cases, the following constraints have been taken into account.

- Scheduling constraints for applications threads and communication tasks
- Message routing conditions to find a reliable path from a sender to a receiver
- Each ECU must execute at least one application
- Each LOR's link must be maximum three

Moreover, within the solving process, the optimization objectives of end-to-end latency and response time are duly taken into account.

Figure 7.20 (a) expresses the configuration times associated with the design of the mapping and routing case studies illustrated above using both manual and tool-assisted approaches. In parallel to the mapping experiment, the framework's modeling process consumes more time at a smaller scale than the manual approach. However, as the model size increases, the utilization of the tool becomes advantageous, resulting in an expedited design process, as evident in Figure 7.20 (a).

The solving process of the mapping and routing experiment demonstrates significantly longer durations than the mapping experiment within the manual solving, as illustrated in Figure 7.20 (b). The extended durations can be attributed to the complexity introduced by optimization goals and the number of problems necessitating resolution. Moreover, as can be seen, there is a significant difference in solving time between using the manual and tool-based approaches. It should be added that the manual solving time does not account for solution optimality or visualization. The comparisons drawn here distinctly emphasize the pronounced enhancements offered by the proposed framework, not only in the setup but particularly in the resolution of use cases, from a quantitative standpoint. These insights underscore the tool's efficacy and efficiency in navigating intricate design scenarios.



(a) Configuration time of mapping and routing use cases

(b) Solving time of mapping and routing use cases

Figure 7.20: Results of the quantitative evaluation for the mapping and routing case studies using manual and tool-assisted approaches. (a) The required setup time for each use case. (b) The required solving time for each use case. The manual solving time does not include visualization and optimality of the solutions.

7.3.2 Qualitative Analysis

In addition to the quantifiable benefits, qualitative distinctions exist between the outcomes of manual approaches and the solutions offered by the introduced tool.

- The manual solving process necessitates a complete restart whenever a single object property changes. While the need to re-solve remains applicable to the introduced framework's approach, it is evident that the framework enables rapid exploration of various property permutations. This accelerated process is an important advantage.
- The tool swiftly offers an overview of the feasibility of a desired configuration. Conversely, the manual method mandates a comprehensive exploration of all feasible combinations, a time-consuming task. This process can potentially be expedited when partial mappings breach the constraints. Nevertheless, validating this condition requires substantial additional time.
- The visualization of results quickly becomes disorganized when manual mapping is carried out. In contrast, the E/E Designer automatically presents the outcomes in a visually coherent manner, adeptly avoiding the issue of overlapping visuals.
- The E/E Designer offers a significant advantage. The tool ensures the finding of an optimal solution. In contrast, when attempting manual solutions, especially for more extensive use cases, it is possible to find solutions; however, discovering the optimized solution becomes exceedingly challenging or unattainable.

8

Conclusion and Future Work

The demand for applications in modern vehicles has grown significantly, primarily due to the integration of ADAS and automated driving technologies. To ensure that automotive designs meet both safety and non-safety criteria in accordance with established standards like ISO 26262 and SOTIF [ISO18; ISO19], designers face increasingly intricate challenges when configuring automotive architectures. This complexity stems from the need to seamlessly integrate new applications and features into vehicles while working within the confines of conventional E/E architectures [AHK22; AMK23; AFK21b].

Developing an E/E architecture with ADAS functionalities and algorithms that not only fulfill safety-related aspects like timing, FFI, and redundancy but also meet various non-safety-related requirements is a demanding and time-consuming endeavor. This task requires a deep understanding of the specific domain [AFK21a; AFK20]. The manual integration and configuration of a software architecture for an automotive HPCU pose considerable challenges and are susceptible to errors. This complexity arises from the need to align with many hardware, application, OS, middleware, and hypervisor prerequisites and attributes. The same level of intricacy applies to an automotive communication network setup, which must guarantee secure data transmission for safety-critical ADAS applications. In addition, synthesizing these configurations can be optimized for various objectives. These goals encompass minimizing power consumption, efficient resource utilization, enhanced reliability, bandwidth optimization, temperature control, cost efficiency, response time, end-to-end latency, and more [AHK22; AFK20; AMK23]. The following sections summarize the key contributions of this thesis and provide perspectives on the limitations of this thesis and future work.

8.1 Summary

This thesis addresses two research questions, and the results for each question are summarized below. Before delving into these questions, a comprehensive analysis of current approaches and frameworks/tools for synthesizing vehicle E/E systems and embedded systems is performed in Chapter 3.

- How to facilitate design and synthesis of E/E architectures?

To address this question, a novel model-based framework called E/E Designer is presented for modeling and synthesizing automotive E/E architectures. The framework enables users

to model mixed-critical networks, providing automated mapping of applications, calculating schedules for mapped application threads on different ECUs, processors, and cores, creating valid paths for communication messages between senders and receivers (including single, multicast, redundant, and homogeneous redundant routings), and computing schedules for communication tasks routing over network links while considering message and path dependencies. The proposed tool also supports and facilitates modeling hypervisors, covering both types by considering different requirements and constraints, as explained in Chapter 4. It also supports various safety requirements, such as ASIL, redundancy, FFI, and reliability. These requirements are considered during the configuration process, which includes resource allocation and message routing. For example, considering the failure rates of components, the reliability of each communication route can be calculated; hence, the most reliable path can be selected to route messages from senders to receivers.

The presented framework considers a range of boundary goals and optimization objectives for the designed model, including end-to-end latency, response time, resource utilization (including maximum resource usage, memory, and ECU), load balancing in the vehicle communication network (comprising LOR and maximum bandwidth utilization), reliability for single and redundant paths, and CR. Additionally, the introduced tool supports multi-objective optimization using a hierarchical approach. This approach assigns priority to each objective, and optimization is performed by considering objectives in descending order of priority.

As illustrated in Chapter 4, the proposed framework utilizes an object-oriented metamodel following MDD methodology. This metamodel is the foundation for graphical modeling, which the E/E system integrator or modeler uses to create graphical model instances. Using a formal system metamodel, the graphical model instances, which include the requirements, boundary goals, and optimization objectives, are transformed into MIP constraints. Furthermore, an approach is employed to solve all constraint sets for mapping, application thread scheduling, message routing, and communication task scheduling in a single optimization run. This single-step approach reduces the solving time and maintains the interrelations between the specified constraint decisions. Moreover, the proposed tool offers a web-based frontend that allows users to model their desired E/E systems and select various hardware and software requirements and properties, along with the aforementioned boundary and optimization goals. The developed frontend also visualizes the solution of the designed system after it has been solved, as described in Chapter 5.

The performance of the model-based tool is assessed using three methods: a design-time evaluation, where the solving and generation times of constraint sets in different scenarios are evaluated, including scalability analysis; a run-time evaluation, where the solution is deployed on an experimental setup; and quantitative and qualitative evaluation, where the performance, usability, and practicality of the E/E Designer are assessed by addressing a series of use cases through both manual and automated approaches. The design-time experiments show that our formulations scale to systems with reasonably large sizes. During the run-time experiments, it was noted that there were no instances of timing deadline breaches following the deployment of the design-time solutions on an experimental setup. The evaluation is explained in Chapter 7.

- How to simplify analysis of design errors in E/E architecture?

To address this question, an approach, called design error analysis, is introduced, as depicted in Chapter 6. This approach focuses on situations where a designed E/E architecture is not satisfiable, meaning that the solver cannot find feasible solutions. Unlike simple models, navigating and correcting the unsatisfiability of complex E/E models is a complex and time-consuming task, which leads to increased development costs. To tackle this issue, the design

error analysis approach is introduced to identify design errors when violations occur in the constraint set included in the system model after the solving step. This feature is crucial for detecting and rectifying errors in the system design within a reasonable timeframe, ensuring that the system is optimized and meets all necessary constraints and requirements.

Two methods are used and evaluated, which include the IIS and Marco algorithms, to apply the introduced approach. Since the proposed framework incorporates MIP constraints and utilizes the Gurobi optimizer, the IIS method is preferred over the MARCO algorithm. This preference is based on the implementation effort required and the effectiveness of the Gurobi solver in handling MIP problems.

8.2 Limitations

8.2.1 Constraints Formulation

In this thesis, logical requirements and properties are automatically generated from the graphical model and metamodel definitions. Also, the specified problems, converted into MIP constraints, are acquired from analyzing the defined E/E system database. However, the existing problems, boundary and optimization objectives do not cover all possible problems, scenarios, and optimization goals related to vehicle E/E architecture, including real-time and mixed-critical systems. Formulation of problems and goals for synthesizing the modeled E/E systems (which can comprise various scheduling schemes and other hardware/software requirements) requires a background in logic programming; in other words, the E/E system integrator must have knowledge about MIP in order to develop and add new problems and optimizations goals to the current system model. Therefore, this framework does not use specific modeling-based languages such as object constraint language (OCL) [WK03] and AADL or any other modeling-based languages to define the problems. However, these languages have their limitations in defining various problems.

8.2.2 Verification

As explained in Chapter 4, the developed tool, as a modular framework, uses Eclipse modeling framework, Sirius Web, and Gurobi optimizer as software artifacts. These software modules are utilized for the synthesis of E/E systems, which include safety requirements as depicted in Chapter 4. The quality and correctness of the synthesis results depend highly on the correct implementation of these software modules. Verification of the software components directly impacts the certification of the safety-relevant parts of the model configured using the framework. Ada programming language [Bar84] can be considered in the context of formal verification. Ada is known for its strong support of formal methods and formal verification. Ada's type system and design principles make it more amenable to formal verification techniques. However, using Ada depends on whether that can fit into the system model and if it is necessary to be used in the context of E/E architecture synthesis.

8.2.3 Placement of E/E Components

Modeling the positioning of E/E components in a car's body is important in terms of cost, safety-criticality, wiring harness design, thermal management, and overall performance. However, the introduced computer-aided tool does not support the exact placement of E/E com-

ponents in the car, considering the size of the car's body. This can assist E/E system architects in calculating and predicting various parameters such as wiring harness, material usage, overheating of components, electromagnetic interference, damage to ECUs in case of crash, etc.

8.2.4 Design Error Analysis

The presented design error analysis approach for finding the source of system model unsatisfiability contains the information that aids in navigating the origin of violation in the set of constraints and reduces time and complexity. In other words, the most critical constraints, which have caused the model infeasibility, are weighted based on their criticality level.

However, this approach does not create any explanations or correcting recommendations or proposals. There is also no guarantee of how many iterations are required to achieve the satisfiability of the E/E model. The number of iterations depends on the correction actions and modifications of the model. Incorrect modifications can also cause more conflicting constraints. Hence, knowledge of E/E systems and understanding of defined problems are required to derive adequate modifications in the designed model.

8.3 Future Works

8.3.1 New Requirements and Features

As part of future work, additional challenges, safety requirements, and optimization objectives can be incorporated into the existing system model of the framework. For example, new scheduling schemes, such as priority-based scheduling, event-triggered scheduling, earliest deadline first (EDF), rate monotonic (RM) scheduling, and others, can be seamlessly integrated into the tool. This expansion will provide a more comprehensive coverage of scheduling possibilities for application threads and communication tasks. Furthermore, there is potential for an increased level of granularity in the modeling and synthesis of E/E architectures. This means that more intricate details, encompassing additional requirements and properties, can be included in the current hardware and software components model. This extension may encompass various elements such as applications, threads, hypervisors, ECUs, HPCUs, switches, gateways, etc.

The ability to position and configure E/E components such as ECUs, switches, wiring harnesses, connectors, and HPCUs within a vehicle's body, taking into account standard vehicle dimensions, assists engineers in determining the optimal component locations and configurations, as discussed in Subsection 8.2.3. This process can also generate estimates for various aspects of the vehicle's wiring harness, including its length, weight, type, required space, cost, connector types, voltage and current ratings, and more. Proper placement of E/E components helps minimize the length of wiring harnesses. Shorter harnesses reduce material usage, which can lead to cost savings. Reduced harness complexity and simplified assembly lead to cost-efficient manufacturing processes. Accurate placement of safety-critical E/E components is vital to ensure their proper operation, and proper positioning helps prevent electromagnetic interference, overheating, and damage, which can impact the vehicle's safety. Optimal placement ensures that wiring harnesses are routed efficiently, reducing complexity and the likelihood of signal interference. It also minimizes signal propagation delays and the risk of signal degradation or electromagnetic interference. Effective modeling considers heat dissipation from E/E components and can help avoid overheating issues, ensuring long-term

reliability. Furthermore, the positioning of HPCUs must be ensured to minimize signal latency and facilitate efficient data flow for advanced features. The same condition is applied to ECUs related to safety-critical systems (e.g., airbag deployment), which require careful positioning to ensure that they operate reliably and are protected from damage in case of a collision.

8.3.2 Run-time E/E Configurator

The developed framework solely focused on the design phase for modeling and synthesizing car E/E systems. However, synthesizing E/E systems in real-time is both innovative and valuable. To achieve this objective, various perspectives and requirements must be considered. This concept becomes relevant during OTA updates for vehicle software. In this scenario, when a new application is installed on the current software platform, it is vital to check the application's requirements and perform a new synthesis to ensure it does not disrupt the existing E/E configuration before the software update. This process can occur while the car is charging overnight or parked in a designated slot. The synthesis may encompass the above-mentioned issues and potentially introduce new challenges. For cases where new requirements have not been integrated into the existing system model, a new approach must be devised to incorporate these new requirements and conditions into the previous system model before re-synthesizing the car's E/E system.

8.3.3 Uncertain Optimization

As described in Chapter 4, to compute reliability, constant failure rates of components provided by the user are considered. However, this thesis does not address the uncertainty in these failure rates. As a future endeavor, exploring the introduction of uncertainty into reliability calculations can be investigated. This would involve defining a range for each component's failure rate, including its mean and standard deviation. Based on this information, reliability for each sample at a specific time can be computed. In this scenario, multiple configurations with varying reliabilities can be generated while considering other optimization goals such as cost, response time, end-to-end latency, etc., using different uncertain optimization approaches like robust and scenario-based optimizations. The failure rate can also be modeled by considering a specific failure rate for each failure mode.

8.3.4 Run-time Simulation

Another aspect of future work involves testing and evaluating synthesized configurations within a simulated environment. In this thesis, the solutions created by the E/E Designer framework were deployed on a real hardware platform for assessment. However, incorporating a simulation environment that is interconnected with the introduced tool enhances the visualization and analysis of design-time solutions, thereby improving the usability of this framework. Furthermore, the simulation can illustrate the differences between optimized and non-optimized solutions, presenting how an optimized solution impacts the system's performance.

A

Appendix 1

The following table illustrates the notation references for the variables used in the equations presented in Chapter 4.

Table A.1: Notation Reference.

	MIP Input
a^{asil}	ASIL level of an application
apn	a tolerable limit for number of assigned applications
bw	maximum amount of data transfer over a network link
$c_i.p$, $c_i.fl$	period and frame length of communication task c_i
$G(N, L)$	set of vehicle topology nodes and full-duplex links
ipg	required time between network packets
pd	maximum processing delay of a communication frame
rd	required time for preparation of receiving a packet by a n^{cz}
sd	required time for preparation of sending a packet by a n^{cz}
$sync$	maximum difference between two any clocks in the system
$t_i.p$, $t_i.e$	period and execution time of thread t_i, $i \in \mathbb{N}$
t_{ij}^{s}	sender thread of application a_j for communication message d_i, $j \in \mathbb{N}$
t_{ij}^{r}	receiver thread of application a_j for communication message d_i
	MIP Decision Variables
$c_i.st$	continuous: starting time of communication task c_i
${d_i}^{in}$	binary: 1 if communication message d_i enters to a node via link
${d_i}^{out}$	binary: 1 if communication message d_i goes out from a node via link
m_{ij}	binary: 1 if application a_j is executing on node n_i^{cz}
${m_{ij}}^{s}$	binary: 1 if sender thread t from a_j is executing on node n_i^{cz}
${m_{ij}}^{r}$	binary: 1 if receiver thread t from a_j is executing on node n_i^{cz}
$t_i.st$	continuous: starting time of thread t_i
v, r, q	binary: decided based on expression's solution

Table A.1. Continued.

	Assistive Terms
$\mathcal{A}$	a set of applications
A^{sc}	a set of safety-critical applications
$a_i.t_{ij}$	one/many threads j belong to a_i
$a_i.t^s_{ij}.d_i$	sender thread t_j, which belongs to a_i, sending d_i out
$a_i.t^r_{ij}.d_i$	receiver thread t_j, which belongs to a_i, receiving d_i
$a_j.mu$	memory usage of a_j
$\mathcal{C}$	a set of communication tasks
$c_i.st^{l_j}_{d_i}$	starting time of c_i related to d_i over link l_j
$\mathcal{D}$	a set of communication messages
$d_i.ch_i$	message chain of d_i containing sender and receiver threads for d_i
$d_i.c_i$	communication task of d_i
$d_i.rt, d_i.el$	response time and end-to-end latency of d_i
$d^{out}_i.l^{n_a}_{a,b}$	message d_i is sent out from n_a over $l_{a,b}$
$d^{in}_i.l^{n_a}_{b,a}$	message d_i is received by n_a over $l_{b,a}$
$\mathcal{L}$	a set of links
$l_{a,b}$, $l_{b,a}$	directed link from n_a to n_b and directed link from n_b to n_a
$\mathcal{M}$	a set of mapping variables
$m_{ij}.a_j{}^{n^{cz}_i}$	mapping variable of a_j running on n^{cz}_i
$m^s_{ij}.a_j{}^{n^{cz}_i}$	mapping variable of sender thread from a_j running on n^{cz}_i
$m^r_{ij}.a_j{}^{n^{cz}_i}$	mapping variable of receiver thread from a_j running on n^{cz}_i
$\mathcal{N}$	a set of nodes
$n_j.d^{in}_p$	message d_p enters to n_j over a link
$n_j.d^{out}_p$	message d_p goes out from n_j over a link
n^{cz}, n^{nz}	control node and networking node
n^{cz_c}	single processor core of a control node
$n^{cz}_i.m_{max}$	maximum memory capacity of n^{cz}_i
nl	number of links related to each possible path
$\mathcal{T}$	a set of application threads
$t^s_i.st_{d_i}$	starting time of sender t_i sending d_i
$t^r_i.st_{d_i}$	starting time of receiver t_i receiving d_i

In the following, a visual representation of the object-oriented metamodel created for the computer-assisted tool introduced in this thesis is provided.

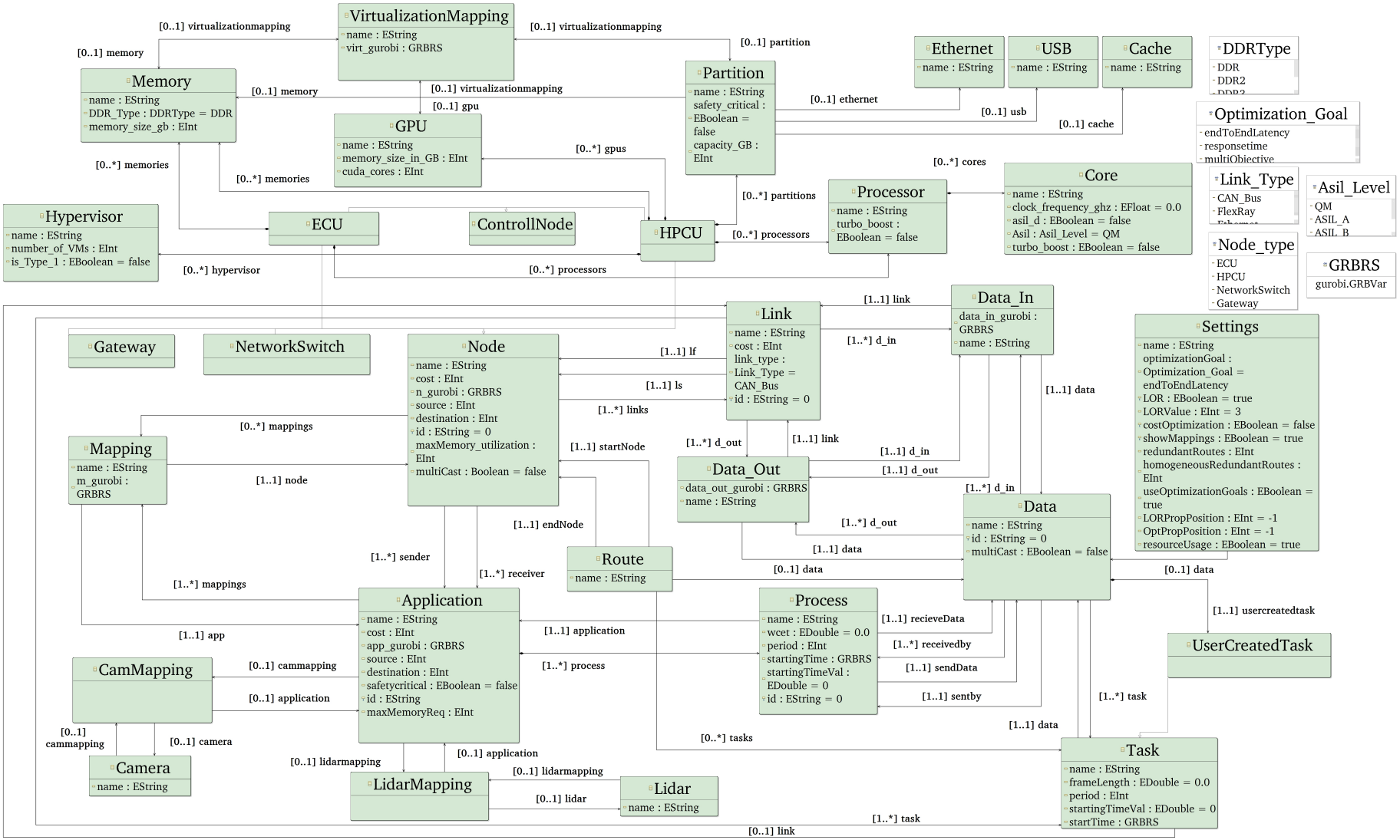


Figure A.1: Object-oriented metamodel for the E/E Designer framework. The green boxes indicate the classes, and the white boxes represent the types of data and elements used within the classes.

Bibliography

[Abd+17] Abdulkhaleq, A., Wagner, S., Lammering, D., Boehmert, H., and Blueher, P. "Using STPA in compliance with ISO 26262 for developing a safe architecture for fully automated vehicles". In: *arXiv preprint arXiv:1703.03657* (2017).

[ADL21] ADLINK. *ADLINK AVA-3501*. 2021. URL: https://www.adlinktech.com/en/Connected-Autonomous-Vehicle-Solutions.

[Ale+09] Aleti, A., Bjornander, S., Grunske, L., and Meedeniya, I. "ArcheOpterix: An extendable tool for architecture optimization of AADL models". In: *2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software*. IEEE. 2009, pp. 61–71.

[Ani+16] Aniculaesei, A., Arnsberger, D., Howar, F., and Rausch, A. "Towards the verification of safety-critical autonomous systems in dynamic environments". In: *arXiv preprint arXiv:1612.04977* (2016).

[Ans+12] Anssi, S., Albers, K., Dörfel, M., and Gérard, S. "chronval/chronsim: A tool suite for timing verification of auto-motive applications". In: *Embedded Real Time Software and Systems (ERTS2012)*. 2012.

[Apt23] Aptiv. *SDV*. 2023. URL: https://www.aptiv.com/en/insights/article/what-is-a-software-defined-vehicle.

[Ara+15] Aravantinos, V., Voss, S., Teufl, S., Hölzl, F., and Schätz, B. "AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems." In: *ACES-MB&WUCOR@ MoDELS* 1508 (2015), pp. 19–26.

[AGH05] Arnold, K., Gosling, J., and Holmes, D. *The Java programming language*. Addison Wesley Professional, 2005.

[Ars06] Arsham, H. "A big-M free solution algorithm for general linear programs". In: *International Journal of Pure and Applied Mathematics* 32.4 (2006), p. 549.

[Ash+17] Ashjaei, M., Patti, G., Behnam, M., Nolte, T., Alderisi, G., and Lo Bello, L. "Schedulability analysis of Ethernet Audio Video Bridging networks with scheduled traffic support". In: *Real-Time Systems* 53 (2017), pp. 526–577.

[AFK20] Askaripoor, H., Farzaneh, M. H., and Knoll, A. "Considering Safety Requirements in Design Phase of Future E/E Architectures". In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2020, pp. 1165–1168. DOI: 10.1109/ETFA46521.2020.9212001.

[AFK21a] Askaripoor, H., Farzaneh, M. H., and Knoll, A. "A Model-Based Approach to Facilitate Design of Homogeneous Redundant E/E Architectures". In: *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. 2021, p. 3426 3431. DOI: 10.1109/ITSC48978.2021.9565115.

[AFK21b] Askaripoor, H., Farzaneh, M. H., and Knoll, A. “A Platform to Configure and Monitor Safety-Critical Applications for Automotive Central Computers”. In: *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. 2021, pp. 1–4. DOI: 10.1109/ETFA45728.2021.9613692.

[AHK22] Askaripoor, H., Hashemi Farzaneh, M., and Knoll, A. “E/E Architecture Synthesis: Challenges and Technologies”. In: *Electronics* 11.4 (2022), p. 518.

[AMK23] Askaripoor, H., Mueller, T., and Knoll, A. “E/E Designer: a Framework to Design and Synthesize Vehicle E/E Architecture”. In: *IEEE Transactions on Intelligent Vehicles* (2023), pp. 1–18. DOI: 10.1109/TIV.2023.3324617.

[ASK21] Askaripoor, H., Shafaei, S., and Knoll, A. C. “A Flexible Scheduling Architecture of Resource Distribution Proposal for Autonomous Driving Platforms.” In: *VEHITS*. 2021, pp. 594–599.

[AHM19] Atallah, A. A., Hamad, G. B., and Mohamed, O. A. “Routing and scheduling of time-triggered traffic in time-sensitive networks”. In: *IEEE Transactions on Industrial Informatics* 16.7 (2019), pp. 4525–4534.

[AHM20] Atallah, A. A., Hamad, G. B., and Mohamed, O. A. “Routing and Scheduling of Time-Triggered Traffic in Time-Sensitive Networks”. In: *IEEE Transactions on Industrial Informatics* 16.7 (2020), pp. 4525–4534. DOI: 10.1109/TII.2019.2950887.

[AK03] Atkinson, C. and Kuhne, T. “Model-driven development: a metamodeling foundation”. In: *IEEE software* 20.5 (2003), pp. 36–41.

[Aut20] Autosarbilder. *Simulation Toolset: Autosarbuilder*. Last accessed 10 October 2021. 2020. URL: https://www.3ds.com/fileadmin/Welcome_to_AUTOSAR_Builder_2020x.pdf.

[Aut21] Autosarbilder. *Simulation Toolset: Autosarbuilder - Dassault Systemes*. https://www.3ds.com/products-services/catia/products/autosar-builder/. Last accessed 9 October 2021. 2021. URL: https://www.3ds.com/products-services/catia/products/autosar-builder/.

[BS05] Bailey, J. and Stuckey, P. J. “Discovery of minimal unsatisfiable subsets of constraints using hitting set dualization”. In: *Practical Aspects of Declarative Languages: 7th International Symposium, PADL 2005, Long Beach, CA, USA, January 10-11, 2005. Proceedings 7*. Springer. 2005, pp. 174–186.

[BSW03] Banda, M. G. de la, Stuckey, P. J., and Wazny, J. “Finding all minimal unsatisfiable subsets”. In: *Proceedings of the 5th ACM SIGPLAN international conference on Principles and practice of declaritive programming*. 2003, pp. 32–43.

[Bar84] Barnes, J. G. *Programming in ADA*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[Beh+06] Behrmann, G., David, A., Larsen, K. G., Håkansson, J., Pettersson, P., Yi, W., and Hendriks, M. “Uppaal 4.0”. In: (2006).

[Bha+12] Bhatti, Z. W., Miniskar, N. R., Preuveneers, D., Wuyts, R., Berbers, Y., and Catthoor, F. “Memory and communication driven spatio-temporal scheduling on MPSoCs”. In: *2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI)*. IEEE. 2012, pp. 1–6.

[BHM09] Biere, A., Heule, M., and Maaren, H. van. *Handbook of satisfiability*. Vol. 185. IOS press, 2009.

[Bok81] Bokhari, S. H. “A shortest tree algorithm for optimal assignments across space and time in a distributed processor system”. In: *IEEE transactions on Software Engineering* 6 (1981), pp. 583–589.

[Bol+16] Bolchini, C., Carminati, M., Mitra, T., and Muthukaruppan, T. S. “Combined on-line lifetime-energy optimization for asymmetric multicores”. In: *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE. 2016, pp. 35–40.

[BC+06] Bondarev, E., Chaudron, M., et al. “A process for resolving performance trade-offs in component-based architectures”. In: *International Symposium on Component Based Software Engineering*. Springer. 2006, pp. 254–269.

[BCK07] Bondarev, E., Chaudron, M. R., and Kock, E. A. de. “Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework”. In: *Proceedings of the 6th International Workshop on Software and Performance*. 2007, pp. 153–163.

[BSJ18] Bozdal, M., Samie, M., and Jennions, I. “A survey on can bus protocol: Attacks, challenges, and potential solutions”. In: *2018 International Conference on Computing, Electronics & Communications Engineering (iCCECE)*. IEEE. 2018, pp. 201–205.

[Bra+01] Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., et al. “A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems”. In: *Journal of Parallel and Distributed computing* 61.6 (2001), pp. 810–837.

[Bro06] Broy, M. “Challenges in automotive software engineering”. In: *Proceedings of the 28th international conference on Software engineering*. 2006, pp. 33–42.

[BL99] Büning, H. K. and Lettmann, T. *Propositional logic: deduction and algorithms*. Vol. 48. Cambridge University Press, 1999.

[BGT04] Burmester, S., Giese, H., and Tichy, M. “Model-driven development of reconfigurable mechatronic systems with mechatronic UML”. In: *Model Driven Architecture*. Springer, 2004, pp. 47–61.

[Bus23a] Bus, C. *CAN Bus*. 2023. URL: https://www.csselectronics.com/pages/can-bus-simple-intro-tutorial.

[Bus23b] Bus, L. *LIN Bus*. 2023. URL: https://www.ni.com/en/shop/seamlessly-connect-to-third-party-devices-and-supervisory-system/introduction-to-the-local-interconnect-network-lin-bus.html.

[BFK19] Busch, A., Fuchß, D., and Koziolek, A. “Peropteryx: Automated improvement of software architectures”. In: *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE. 2019, pp. 162–165.

[CCM07] Carvalho, E., Calazans, N., and Moraes, F. “Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs”. In: *18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP’07)*. IEEE. 2007, pp. 34–40.

[Cas+12] Castrillon, J., Tretter, A., Leupers, R., and Ascheid, G. “Communication-aware mapping of KPN applications onto heterogeneous MPSoCs”. In: *DAC Design Automation Conference 2012*. IEEE. 2012, pp. 1262–1267.

[Chi+17] Chitnis, K., Mody, M., Swami, P., Sivaraj, R., Ghone, C., Biju, M., Narayanan, B., Dutt, Y., and Dubey, A. “Enabling functional safety ASIL compliance for autonomous driving software systems”. In: *Electronic Imaging* 29 (2017), pp. 35–40.

[Com+17] Commission, I. E. et al. *Electric components- Reliability- Reference conditions for failure rates and stress models for conversion: IEC 61709*. 2017.

[Com23] Commission, I. E. *IEC 61508*. 2023. URL: https://webstore.iec.ch/publication/5515.

[CBB05] Correll, K., Barendt, N., and Branicky, M. “Design considerations for software only implementations of the IEEE 1588 precision time protocol”. In: *Conference on IEEE*. Vol. 1588. 11. 2005.

[Cos+08] Coskun, A. K., Rosing, T. S., Whisnant, K. A., and Gross, K. C. “Temperature-aware MPSoC scheduling for reducing hot spots and gradients”. In: *2008 Asia and South Pacific Design Automation Conference*. IEEE. 2008, pp. 49–54.

[Cpl09] Cplex, I. I. “V12. 1: User’s Manual for CPLEX”. In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[Cra+16] Craciunas, S. S., Oliver, R. S., Chmelık, M., and Steiner, W. “Scheduling real-time communication in IEEE 802.1 Qbv time sensitive networks”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 2016, pp. 183–192.

[DKV14] Das, A., Kumar, A., and Veeravalli, B. “Communication and migration energy aware task mapping for reliable multiprocessor systems”. In: *Future Generation Computer Systems* 30 (2014), pp. 216–228.

[Das+14] Das, A., Kumar, A., Veeravalli, B., Bolchini, C., and Miele, A. “Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–6.

[DB08] De Moura, L. and Bjørner, N. “Z3: An efficient SMT solver”. In: *International conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2008, pp. 337–340.

[DeN+01] DeNuto, J. V., Ewbank, S., Kleja, F., Lupini, C. A., and Perisho Jr, R. A. “LIN Bus and its Potential for use in Distributed Multiplex Applications”. In: *SAE transactions* (2001), pp. 135–142.

[DHN06] Dershowitz, N., Hanna, Z., and Nadel, A. “A scalable algorithm for minimal unsatisfiable core extraction”. In: *Theory and Applications of Satisfiability Testing-SAT 2006: 9th International Conference, Seattle, WA, USA, August 12-15, 2006. Proceedings 9*. Springer. 2006, pp. 36–41.

[Dev+15] Deveci, M., Kaya, K., Uçar, B., and Çatalyürek, Ü. V. “Hypergraph partitioning for multiple communication cost metrics: Model and methods”. In: *Journal of Parallel and Distributed Computing* 77 (2015), pp. 69–83.

[DS18] Dijk, L. van and Sporer, G. “Functional safety for automotive ethernet networks”. In: *Journal of Traffic and Transportation Engineering* 6.4 (2018), pp. 176–182.

[DLM13] Ding, H., Liang, Y., and Mitra, T. “Shared cache aware task mapping for WCRT minimization”. In: *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2013, pp. 735–740.

[Dis23] Dis, C. B. *CAN Bus*. 2023. URL: https://www.itrelease.com/2021/12/advantages-and-disadvantages-of-controller-area-network-can/.

[Ecl22] Eclipse. *Sirius Web*. https://www.eclipse.org/sirius/sirius-web.html. Last accessed 16 September 2022. 2022. URL: https://www.eclipse.org/sirius/sirius-web.html.

[Ecl23] Eclipse Foundation, I. *The Community for Open Innovation and Collaboration: The Eclipse Foundation*. http://www.eclipse.org/. Last accessed 29 September 2023. 2023. URL: http://www.eclipse.org/.

[Edm+15] Edman, R., Shackleton, H., Shackleton, J., Smith, T., and Vestal, S. "A Framework for Compositional Timing Analysis of Embedded Computer Systems". In: *2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems*. 2015, pp. 1001–1004. DOI: 10.1109/HPCC-CSS-ICESS.2015.239.

[ETA21] ETAS. *ASCET-DEVELOPER*. [Online;Last accessed 25 September 2021]. 2021. URL: https://www.etas.com/en/products/ascet-developer.php.

[Eth23] Ethernet, A. *Automotive Ethernet*. 2023. URL: https://www.plm.automation.siemens.com/global/en/our-story/glossary/what-is-automotive-ethernet/109722.

[FK17] Farzaneh, M. H. and Knoll, A. "Time-sensitive networking (TSN): An experimental setup". In: *2017 IEEE Vehicular Networking Conference (VNC)*. IEEE. 2017, pp. 23–26.

[FKK17] Farzaneh, M. H., Kugele, S., and Knoll, A. "A graphical modeling tool supporting automated schedule synthesis for time-sensitive networking". In: *2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2017, pp. 1–8.

[Fei19] Feiler, P. *The Open Source AADL Tool Environment (OSATE)*. Tech. rep. Carnegie Mellon University Software Engineering Institute, 2019.

[FGH06] Feiler, P. H., Gluch, D. P., and Hudak, J. J. *The architecture analysis & design language (AADL): An introduction*. Tech. rep. Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2006.

[Fer+10] Ferrandi, F., Lanzi, P. L., Pilato, C., Sciuto, D., and Tumeo, A. "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29.6 (2010), pp. 911–924.

[Fin18] Finn, N. "Introduction to time-sensitive networking". In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 22–28.

[Fle23] FlexRay. *FlexRay Bus*. 2023. URL: https://automotivetechis.wordpress.com/flexray/#:~:text=Disadvantages%20of%20Flexray%3A,%2C%20non%2Dsafety%20critical%20applications.

[FL05] Floudas, C. A. and Lin, X. "Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications". In: *Annals of Operations Research* 139 (2005), pp. 131–162.

[For09] Fortnow, L. "The status of the P versus NP problem". In: *Communications of the ACM* 52.9 (2009), pp. 78–86.

[FH20] Frank, S. and Hoorn, A. van. “SQuAT-Vis: Visualization and Interaction in Software Architecture Optimization”. In: *European Conference on Software Architecture*. Springer. 2020, pp. 107–119.

[FE98] Fritzson, P. and Engelson, V. “Modelica—A unified object-oriented language for system modeling and simulation”. In: *European Conference on Object-Oriented Programming*. Springer. 1998, pp. 67–90.

[Für+09] Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., and Lange, K. “AUTOSAR–A Worldwide Standard is on the Road”. In: *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*. Vol. 62. 2009, p. 5.

[GPM14] Gan, J., Pop, P., and Madsen, J. *Tradeoff analysis for dependable real-time embedded systems during the early design phases*. DTU Compute, 2014.

[Gan+16] Gan, Z., Zhang, M., Gu, Z., and Zhang, J. “Minimizing energy consumption for embedded multicore systems using cache configuration and task mapping”. In: *2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC)*. IEEE. 2016, pp. 328–334.

[GJS74] Garey, M. R., Johnson, D. S., and Stockmeyer, L. “Some simplified NP-complete problems”. In: *Proceedings of the sixth annual ACM symposium on Theory of computing*. 1974, pp. 47–63.

[Gav+17] Gavrilut, V., Zarrin, B., Pop, P., and Samii, S. “Fault-tolerant topology and routing synthesis for IEEE time-sensitive networking”. In: *Proceedings of the 25th International Conference on Real-Time Networks and Systems*. 2017, pp. 267–276.

[Gia+14] Giannopoulou, G., Stoimenov, N., Huang, P., and Thiele, L. “Mapping mixed-criticality applications on multi-core architectures”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–6.

[GZ+19] Girault, A., Zarandi, H. R., et al. “Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores”. In: *IEEE Transactions on Parallel and Distributed Systems* 30.10 (2019), pp. 2193–2210.

[GR90] Gleeson, J. and Ryan, J. “Identifying minimally infeasible subsystems of inequalities”. In: *ORSA Journal on Computing* 2.1 (1990), pp. 61–63.

[GNU00] GNU. *The GNU Linear Programming Kit (GLPK)*. http://www.gnu.org/software/glpk/glpk.html. Last accessed 28 February 2023. Oct. 2000. URL: http://www.gnu.org/software/glpk/glpk.html.

[GRC18] Gosavi, M. A., Rhoades, B. B., and Conrad, J. M. “Application of functional safety in autonomous vehicles using ISO 26262 standard: A survey”. In: *SoutheastCon 2018*. IEEE. 2018, pp. 1–6.

[Gun18] Gunantara, N. “A review of multi-objective optimization: Methods and its applications”. In: *Cogent Engineering* 5.1 (2018), p. 1502242.

[GBI21] Gupta, M., Bhargava, L., and Indu, S. “Mapping techniques in multicore processors: current and future trends”. In: *The Journal of Supercomputing* (2021), pp. 1–56.

[Gur22] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2022. URL: https://www.gurobi.com.

[HT06] Hailpern, B. and Tarr, P. “Model-driven development: The good, the bad, and the ugly”. In: *IBM systems journal* 45.3 (2006), pp. 451–461.

[Ham+04] Hamann, A., Henia, R., Racu, R., Jersak, M., Richter, K., and Ernst, R. “Symta/s-symbolic timing analysis for systems”. In: *WIP Proc. Euromicro Conference on Real-Time Systems 2004 (ECRTS04)*. Citeseer. 2004, pp. 17–20.

[HTM10] Hartman, A. S., Thomas, D. E., and Meyer, B. H. “A case for lifetime-aware task mapping in embedded chip multiprocessors”. In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2010, pp. 145–154.

[HL19] Haupt, N. B. and Liggesmeyer, P. “A runtime safety monitoring approach for adaptable autonomous systems”. In: *Computer Safety, Reliability, and Security: SAFECOMP 2019 Workshops, ASSURE, DECSoS, SASSUR, STRIVE, and WAISE, Turku, Finland, September 10, 2019, Proceedings 38*. Springer. 2019, pp. 166–177.

[Hen+05] Henia, R., Hamann, A., Jersak, M., Racu, R., Richter, K., and Ernst, R. “System level performance analysis–the SymTA/S approach”. In: *IEE Proceedings-Computers and Digital Techniques* 152.2 (2005), pp. 148–166.

[HB17] Hilbrich, R. and Behrisch, M. “Experiences gained from modeling and solving large mapping problems during system design”. In: *2017 Annual IEEE International Systems Conference (SysCon)*. IEEE. 2017, pp. 1–8.

[HD13] Hilbrich, R. and Dieudonné, L. “Deploying safety-critical applications on complex avionics hardware architectures”. In: (2013).

[HF07] Hölzl, F. and Feilkas, M. “13 autofocus 3-a scientific tool prototype for model-based development of component-based, reactive, distributed systems”. In: *Workshop on Model-Based Engineering of Embedded Real-Time Systems*. 2007, pp. 317–322.

[HKI15] Höttger, R., Krawczyk, L., and Igel, B. “Model-based automotive partitioning and mapping for embedded multicore systems”. In: *International Conference on Parallel, Distributed Systems and Software Engineering*. Vol. 2. 1. Citeseer. 2015, p. 888.

[Höt+17] Höttger, R., Mackamul, H., Sailer, A., Steghöfer, J.-P., and Tessmer, J. “APP4MC: Application platform project for multi-and many-core systems”. In: *it Information Technology* 59.5 (2017), pp. 243–251.

[HM05] Hu, J. and Marculescu, R. “Energy-and performance-aware mapping for regular NoC architectures”. In: *IEEE Transactions on computer-aided design of integrated circuits and systems* 24.4 (2005), pp. 551–562.

[HYX09] Huang, L., Yuan, F., and Xu, Q. “Lifetime reliability-aware task allocation and scheduling for MPSoC platforms”. In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2009, pp. 51–56.

[Hug+08] Hugues, J., Zalila, B., Pautet, L., and Kordon, F. “From the prototype to the final embedded system using the Ocarina AADL tool suite”. In: *ACM Transactions on Embedded Computing Systems (TECS)* 7.4 (2008), pp. 1–25.

[IEE18] IEEE. *Institute of Electrical and Electronics Engineers, Inc, Time-Sensitive Networking (TSN) Task Group*. https://1.ieee802.org/tsn/. Last accessed 21 May 2020. 2018. URL: https://1.ieee802.org/tsn/.

[INC21] INCHRON. *ChronVALWorst-Case Timing Analysis*. https://www.inchron.com/chronval/. Last accessed October 2021. 2021. URL: https://www.inchron.com/chronval/.

[Int23] Intel. *Inteli210*. 2023. URL: https://www.intel.com/content/www/us/en/products/details/ethernet/gigabit-controllers/i210-controllers.html.

[ISO18] ISO. *ISO 26262-1:2018*. https://www.iso.org/standard/68383.html. Last accessed 30 August 2021. 2018. URL: https://www.iso.org/standard/68383.html.

[ISO19] ISO. *Safety Of The Intended Functionality (SOTIF)*. Last accessed 16 September 2021. 2019. URL: https://www.iso.org/standard/70939.html.

[Jia19] Jiang, S. *Vehicle e/e architecture and its adaptation to new technical trends*. Tech. rep. SAE Technical Paper, 2019.

[Jün+09] Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G., and Wolsey, L. A. *50 Years of integer programming 1958-2008: From the early years to the state-of-the-art*. Springer Science & Business Media, 2009.

[Juo+18] Juodisius, P., Sarkar, A., Mukkamala, R. R., Antkiewicz, M., Czarnecki, K., and Wasowski, A. "Clafer: Lightweight modeling of structure, behaviour, and variability". In: *arXiv preprint arXiv:1807.08576* (2018).

[Kai+12] Kaida, J., Hieda, T., Taniguchi, I., Tomiyama, H., Hara-Azumi, Y., and Inoue, K. "Task mapping techniques for embedded many-core socs". In: *2012 International SoC Design Conference (ISOCC)*. IEEE. 2012, pp. 204–207.

[KAL20] KALRAY. *Safe compute acceleration for automotive*. 2020. URL: https://www.kalrayinc.com/automotive/.

[KJS11] Kang, E., Jackson, E., and Schulte, W. "An approach for effective design space exploration". In: *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems: 16th Monterey Workshop 2010, Redmond, WA, USA, March 31-April 2, 2010, Revised Selected Papers 16*. Springer. 2011, pp. 33–54.

[KD14] Kinsy, M. A. and Devadas, S. "Algorithms for scheduling task-based applications onto heterogeneous many-core architectures". In: *2014 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE. 2014, pp. 1–6.

[Kle09] Kleiman, M. A. *When brute force fails: How to have less crime and less punishment*. Princeton University Press, 2009.

[KKR11] Koziolek, A., Koziolek, H., and Reussner, R. "PerOpteryx: automated application of tactics in multi-objective software architecture optimization". In: *Proceedings of the joint ACM SIGSOFT conference–QoSA and ACM SIGSOFT symposium–ISARCS on Quality of software architectures–QoSA and architecting critical systems*. 2011, pp. 33–42.

[KP14] Kugele, S. and Pucea, G. "Model-based optimization of automotive e/e architectures". In: *Proceedings of the 6th International Workshop on Constraints in Software Testing, Verification, and Analysis*. 2014, pp. 18–29.

[Kug+15] Kugele, S., Pucea, G., Popa, R., Dieudonné, L., and Eckardt, H. "On the deployment problem of embedded systems". In: *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE)*. IEEE. 2015, pp. 158–167.

[LW66] Lawler, E. L. and Wood, D. E. “Branch-and-bound methods: A survey”. In: *Operations research* 14.4 (1966), pp. 699–719.

[LR14] Lee, S. and Ro, W. W. “Workload and variation aware thread scheduling for heterogeneous multi-processor”. In: *The 18th IEEE International Symposium on Consumer Electronics (ISCE 2014)*. IEEE. 2014, pp. 1–2.

[LH02] Leen, G. and Heffernan, D. “TTCAN: a new time-triggered controller area network”. In: *Microprocessors and Microsystems* 26.2 (2002), pp. 77–94.

[LA07] Leontyev, H. and Anderson, J. H. “Tardiness bounds for FIFO scheduling on multiprocessors”. In: *19th Euromicro Conference on Real-Time Systems (ECRTS’07)*. IEEE. 2007, pp. 71–71.

[Li+11] Li, R., Etemaadi, R., Emmerich, M. T., and Chaudron, M. R. “An evolutionary multiobjective optimization approach to component-based software architecture design”. In: *2011 IEEE Congress of Evolutionary Computation (CEC)*. IEEE. 2011, pp. 432–439.

[Lif+16] Liffiton, M. H., Previti, A., Malik, A., and Marques-Silva, J. “Fast, flexible MUS enumeration”. In: *Constraints* 21 (2016), pp. 223–250.

[LS08] Liffiton, M. H. and Sakallah, K. A. “Algorithms for computing minimal unsatisfiable subsets of constraints”. In: *Journal of Automated Reasoning* 40 (2008), pp. 1–33.

[LPM13] Liu, G., Park, J., and Marculescu, D. “Dynamic thread mapping for high performance, power-efficient heterogeneous many-core systems”. In: *2013 IEEE 31st international conference on computer design (ICCD)*. IEEE. 2013, pp. 54–61.

[LC12] Lukasiewycz, M. and Chakraborty, S. “Concurrent architecture and schedule optimization of time-triggered automotive systems”. In: *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. 2012, pp. 383–392.

[Luk+12] Lukasiewycz, M., Schneider, R., Goswami, D., and Chakraborty, S. “Modular scheduling of distributed heterogeneous time-triggered automotive systems”. In: *17th Asia and South Pacific design automation conference*. IEEE. 2012, p. 665 670.

[LSF14] Lukasiewycz, M., Shreejith, S., and Fahmy, S. A. “System simulation and optimization using reconfigurable hardware”. In: *2014 International Symposium on Integrated Circuits (ISIC)*. IEEE. 2014, pp. 468–471.

[Luk+09] Lukasiewycz, M., Streubuhr, M., Glaß, M., Haubelt, C., and Teich, J. “Combined system synthesis and communication architecture exploration for MPSoCs”. In: *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE. 2009, pp. 472–477.

[LM04] Lynce, I. and Marques-Silva, J. P. “On computing minimum unsatisfiable cores”. In: (2004).

[MT06] Makowitz, R. and Temple, C. “Flexray-a communication network for automotive control systems”. In: *2006 IEEE International Workshop on Factory Communication Systems*. IEEE. 2006, pp. 207–212.

[MK21] Matheus, K. and Königseder, T. *Automotive ethernet*. Cambridge University Press, 2021.

[MAT10] MATLAB. *version 7.10.0 (R2010a)*. Natick, Massachusetts: The MathWorks Inc., 2010.

[Med+02] Medvidovic, N., Rosenblum, D. S., Redmiles, D. F., and Robbins, J. E. “Modeling software architectures in the unified modeling language”. In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11.1 (2002), pp. 2–57.

[Mee+11] Meedeniya, I., Buhnova, B., Aleti, A., and Grunske, L. “Reliability-driven deployment optimization for embedded systems”. In: *Journal of Systems and Software* 84.5 (2011), pp. 835–846.

[Meh+09] Mehrara, M., Jablin, T., Upton, D., August, D., Hazelwood, K., and Mahlke, S. “Multicore compilation strategies and challenges”. In: *IEEE Signal Processing Magazine* 26.6 (2009), pp. 55–63.

[Men16] Menčík, J. “Reliability of Systems”. In: *Concise Reliability for Engineers*. Ed. by Mencik, J. Rijeka: IntechOpen, 2016. Chap. 5. DOI: 10.5772/62358. URL: https://doi.org/10.5772/62358.

[Mne+05] Mneimneh, M., Lynce, I., Andraus, Z., Marques-Silva, J., and Sakallah, K. “A branch-and-bound algorithm for extracting smallest minimal unsatisfiable formulas”. In: *Theory and Applications of Satisfiability Testing: 8th International Conference, SAT 2005, St Andrews, UK, June 19-23, 2005. Proceedings 8*. Springer. 2005, pp. 467–474.

[Mod+18] Mody, M., Jones, J., Chitnis, K., Sagar, R., Shurtz, G., Dutt, Y., Koul, M., Biju, M., and Dubey, A. “Understanding vehicle E/E architecture topologies for automated driving: System partitioning and tradeoff parameters”. In: *Electronic Imaging* 2018.17 (2018), pp. 358–1.

[MAK22] Müller, T., Askaripoor, H., and Knoll, A. “Performance Analysis of KVM Hypervisor Using a Self-Driving Developer Kit”. In: *IECON 2022 – 48th Annual Conference of the IEEE Industrial Electronics Society*. 2022, pp. 1–7. DOI: 10.1109/IECON49645.2022.9968908.

[MAK24] Müller, T., Askaripoor, H., and Knoll, A. “Advancing E/E Architecture Synthesis: A Perspective on Reliability Optimization and Hypervisor Integration”. In: *2024 IEEE Intelligent Vehicles Symposium (IV)*. 2024, pp. 1996–2003. DOI: 10.1109/IV55156.2024.10588416.

[NDR16] Nayak, N. G., Dürr, F., and Rothermel, K. “Time-sensitive software-defined network (TSSDN) for real-time applications”. In: *Proceedings of the 24th International Conference on Real-Time Networks and Systems*. 2016, pp. 193–202.

[Neo21] Neosys. *Nuvo-7208VTC*. 2021. URL: https://omtec.de/industrie-pc/rugged-embedded/nuvo-7000-serie/nuvo-7208vtc.

[NM97] Niemann, R. and Marwedel, P. “An algorithm for hardware/software partitioning using mixed integer linear programming”. In: *Design Automation for Embedded Systems* 2.2 (1997), pp. 165–193.

[NVI21] NVIDIA. *NVIDIA DRIVE HARDWARE*. 2021. URL: https://www.nvidia.com/en-us/self-driving-cars/drive-platform/hardware/.

[Oca21] Ocarina. *Introduction to Ocarina Plugin*. https://ocarina.readthedocs.io/en/latest/introduction.html. Last accessed 10 November 2021. 2021. URL: https://ocarina.readthedocs.io/en/latest/introduction.html.

[OSA21] OSATE. *Welcome to OSATE*. https://osate.org/. Last accessed 11 November 2021. Oct. 2021. URL: https://osate.org/.

[PML11] Pascual, J. A., Miguel-Alonso, J., and Lozano, J. A. “Optimization-based mapping framework for parallel applications”. In: *Journal of Parallel and Distributed Computing* 71.10 (2011), pp. 1377–1387.

[Pel+17] Pelliccione, P., Knauss, E., Heldal, R., Ågren, S. M., Mallozzi, P., Alminger, A., and Borgentun, D. “Automotive architecture framework: The experience of volvo cars”. In: *Journal of systems architecture* 77 (2017), pp. 83–100.

[Per09] Peress, Y. “Multi-core Design and Memory Feature Selection Survey”. In: 2009. URL: https://api.semanticscholar.org/CorpusID:42876232.

[Per23] Perforce. *ASIL Level*. 2023. URL: https://www.perforce.com/blog/qac/what-is-iso-26262#:~:text=Automotive.

[PM10] Poole, D. L. and Mackworth, A. K. *Artificial Intelligence: foundations of computational agents*. Cambridge University Press, 2010.

[QNX23] QNX. *SDV*. 2023. URL: https://blackberry.qnx.com/en/ultimate-guides/software-defined-vehicle.

[RT08] Rasmussen, R. V. and Trick, M. A. “Round robin scheduling–a survey”. In: *European Journal of Operational Research* 188.3 (2008), pp. 617–636.

[REN21] RENESAS. *R-Car-H3-M3-Starter-Kit*. 2021. URL: https://www.renesas.com/jp/en/products/automotive-products/automotive-system-chips-socs/r-car-h3-m3-starter-kit.

[Ros+19] Ross, J. A., Murashkin, A., Liang, J. H., Antkiewicz, M., and Czarnecki, K. “Synthesis and exploration of multi-level, multi-perspective architectures of automotive embedded systems”. In: *Software & Systems Modeling* 18.1 (2019), pp. 739–767.

[Rus10] Russell, S. J. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[Sag+15] Sagstetter, F., Waszecki, P., Steinhorst, S., Lukasiewycz, M., and Chakraborty, S. “Multischedule synthesis for variant management in automotive time-triggered systems”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35.4 (2015), pp. 637–650.

[SC13] Sahu, P. K. and Chattopadhyay, S. “A survey on application mapping strategies for network-on-chip design”. In: *Journal of systems architecture* 59.1 (2013), pp. 60–76.

[SZ18] Samii, S. and Zinner, H. “Level 5 by layer 2: Time-sensitive networking for autonomous vehicles”. In: *IEEE Communications Standards Magazine* 2.2 (2018), pp. 62–68.

[SW10] Schäfer, W. and Wehrheim, H. “Model-driven development with mechatronic uml”. In: *Graph transformations and model-driven engineering*. Springer, 2010, pp. 533–554.

[Sch16] Schäuffele, J. *E/e architectural design and optimization using preevision*. Tech. rep. SAE Technical Paper, 2016.

[SW00] Schild, K. and Würtz, J. “Scheduling of time-triggered real-time systems”. In: *Constraints* 5 (2000), pp. 335–357.

[Sel03] Selic, B. “The pragmatics of model-driven development”. In: *IEEE software* 20.5 (2003), pp. 19–25.

[Shi+04] Shivle, S., Castain, R., Siegel, H. J., Maciejewski, A. A., Banka, T., Chindam, K., Dussinger, S., Pichumani, P., Satyasekaran, P., Saylor, W., et al. “Static mapping of subtasks in a heterogeneous ad hoc grid environment”. In: *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.* IEEE. 2004, p. 110.

[Sin+13] Singh, A. K., Shafique, M., Kumar, A., and Henkel, J. “Mapping on multi/many-core systems: survey of current and emerging trends”. In: *2013 50th IEEE Design Automation Conference (DAC)*. IEEE. 2013, pp. 1–10.

[Smi+17] Smirnov, F., Glaß, M., Reimann, F., and Teich, J. “Optimizing message routing and scheduling in automotive mixed-criticality time-triggered networks”. In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2017, pp. 1–6.

[Smi+19] Smirnov, F., Pourmohseni, B., Glaß, M., and Teich, J. “Variety-aware Routing Encoding for Efficient Design Space Exploration of Automotive Communication Networks.” In: *VEHITS*. 2019, pp. 242–253.

[Smi+18] Smirnov, F., Reimann, F., Teich, J., Han, Z., and Glaß, M. “Automatic optimization of redundant message routings in automotive networks”. In: *Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems*. 2018, pp. 90–99.

[ST23] ST. *STM32L476VG*. 2023. URL: https://www.st.com/en/evaluation-tools/stm32-nucleo-boards.html.

[Sül+08] Sülflow, A., Fey, G., Bloem, R., and Drechsler, R. “Using unsatisfiable cores to debug multiple design errors”. In: *Proceedings of the 18th ACM Great Lakes symposium on VLSI*. 2008, pp. 77–82.

[ŠC15] Švogor, I. and Carlson, J. “SCALL: Software component allocator for heterogeneous embedded systems”. In: *Proceedings of the 2015 European Conference on Software Architecture Workshops*. 2015, pp. 1–5.

[Tab07] Tabuada, P. “Event-triggered real-time scheduling of stabilizing control tasks”. In: *IEEE Transactions on Automatic control* 52.9 (2007), pp. 1680–1685.

[TP11] Tamaş-Selicean, D. and Pop, P. “Optimization of time-partitions for mixed criticality real-time distributed embedded systems”. In: *2011 14th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*. IEEE. 2011, pp. 1–10.

[Tek23] TekEthernet, A. *Automotive Ethernet*. 2023. URL: https://www.tek.com/en/solutions/industry/automotive-test-solutions/in-vehicle-networks/automotive-ethernet.

[TVW18] Terzimehic, T., Voss, S., and Wenger, M. “Using Design Space Exploration to Calculate Deployment Configurations of IEC 61499-based Systems”. In: *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*. 2018, pp. 881–886. DOI: 10.1109/COASE.2018.8560591.

[Ter18] Terzimehić, T. “Optimization and reconfiguration of iec 61499-based software architectures”. In: *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 2018, pp. 180–185.

[Tli+18] Tlig, M., Machin, M., Kerneis, R., Arbaretier, E., Zhao, L., Meurville, F., and Van Frank, J. “Autonomous driving system: Model based safety analysis”. In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE. 2018, pp. 2–5.

[TTT21] TTTechAuto. *MotionWise*. https://www.tttech-auto.com/products/safety-software-platform/motionwise. Last accessed 20 October 2021. 2021. URL: https://www.tttech-auto.com/products/safety-software-platform/motionwise.

[Van+20] Vanderbei, R. J. et al. *Linear programming*. Springer, 2020.

[VEH14] Voss, S., Eder, J., and Hölzl, F. "Design Space Exploration and its Visualization in AUTOFOCUS3." In: *Software Engineering (Workshops)*. 2014, pp. 57–66.

[VSA21] VSA. *Volcano Vehicle Systems Architect (VSA)*. https://www.mathworks.com/products/connections/product_detail/volcano-vehicle-systems-architect.html. Last accessed 1 October 2021. 2021. URL: https://www.mathworks.com/products/connections/product_detail/volcano-vehicle-systems-architect.html.

[WK03] Warmer, J. B. and Kleppe, A. G. *The object constraint language: getting your models ready for MDA*. Addison-Wesley Professional, 2003.

[Was+13] Waszecki, P., Lukasiewycz, M., Masrur, A., and Chakraborty, S. "How to engineer tool-chains for automotive e/e architectures?" In: *ACM SIGBED Review* 10.4 (2013), pp. 6–15.

[Wik14] Wikipedia. *Matrix exponential*. [Online; accessed 12-May-2014]. 2014. URL: http://en.wikipedia.org/wiki/Matrix_exponential.

[Wil13] Williams, H. P. *Model building in mathematical programming*. John Wiley & Sons, 2013.

[Xie+17a] Xie, G., Chen, Y., Liu, Y., Li, R., and Li, K. "Minimizing development cost with reliability goal for automotive functional safety during design phase". In: *IEEE Transactions on Reliability* 67.1 (2017), pp. 196–211.

[Xie+18] Xie, G., Peng, H., Li, Z., Song, J., Xie, Y., Li, R., and Li, K. "Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems". In: *IEEE Transactions on Industrial Informatics* 14.12 (2018), pp. 5447–5462.

[Xie+17b] Xie, G., Zeng, G., Liu, Y., Zhou, J., Li, R., and Li, K. "Fast functional safety verification for distributed automotive applications during early design phase". In: *IEEE Transactions on Industrial Electronics* 65.5 (2017), pp. 4378–4391.

[Xu+16] Xu, X.-X., Hu, X.-M., Chen, W.-N., and Li, Y. "Set-based particle swarm optimization for mapping and scheduling tasks on heterogeneous embedded systems". In: *2016 Eighth International Conference on Advanced Computational Intelligence (ICACI)*. IEEE. 2016, pp. 318–325.

[Yon+19] Yoneda, T., Imai, M., Saito, H., Mochizuki, A., Hanyu, T., Kise, K., and Nakamura, Y. "Network-on-Chip based multiple-core centralized ECUs for safety-critical automotive applications". In: *VLSI Design and Test for Systems Dependability* (2019), pp. 607–633.

[Zen+10] Zeng, H., Di Natale, M., Ghosal, A., and Sangiovanni-Vincentelli, A. "Schedule optimization of time-triggered systems communicating over the FlexRay static segment". In: *IEEE Transactions on Industrial Informatics* 7.1 (2010), pp. 1–17.

[ZL19] Zerfowski, D. and Lock, A. "Functional architecture and E/E-Architecture–A challenge for the automotive industry". In: *19. Internationales Stuttgarter Symposium*. Springer. 2019, pp. 909–920.

[Zha+14] Zhang, L., Goswami, D., Schneider, R., and Chakraborty, S. "Task-and network-level schedule co-synthesis of Ethernet-based time-triggered systems". In: *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2014, pp. 119–124.

[Zhe+16] Zheng, B., Liang, H., Zhu, Q., Yu, H., and Lin, C.-W. “Next generation automotive architecture modeling and exploration for autonomous driving”. In: *2016 IEEE computer society annual symposium on VLSI (ISVLSI)*. IEEE. 2016, pp. 53–58.

[Zhe+05] Zheng, W., Chong, J., Pinello, C., Kanajan, S., and Sangiovanni-Vincentelli, A. “Extensible and scalable time triggered scheduling”. In: *Fifth International Conference on Application of Concurrency to System Design (ACSD’05)*. IEEE. 2005, pp. 132–141.

[Zim+18] Zimmermann, A., Maschotta, R., Wichmann, A., and Hilbrich, R. “Optimization of systems with nested design space”. In: *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE. 2018, pp. 1–8.