electronics

Article

E/E Architecture Synthesis: Challenges and Technologies

Hadi Askaripoor [1,*], Morteza Hashemi Farzaneh [2] and Alois Knoll [1]

[1] Chair of Robotics, Artificial Intelligence and Real-Time Systems, Technical University of Munich, Botlzmannstrasse 3, 85768 Garching bei Muenchen, Germany; knoll@in.tum.de
[2] RealtimeIT MHF GmbH, Landwehrstr. 61, 80336 Muenchen, Germany; hashemi@realtime-it.de
* Correspondence: hadi.askari@tum.de

Abstract: In recent years, the electrical and/or electronic architecture of vehicles has been significantly evolving. The new generation of cars demands a considerable amount of computational power due to a large number of safety-critical applications and driver-assisted functionalities. Consequently, a high-performance computing unit is required to provide the demanded power and process these applications while, in this case, vehicle architecture moves toward a centralized architecture. Simultaneously, appropriate software architecture has to be defined to fulfill the needs of the main computing unit and functional safety requirements. However, the process of configuring and integrating critical applications into a vehicle central computer while meeting safety requirements and optimization objectives is a time-consuming, complicated, and error-prone process. In this paper, we firstly present the evolution of the vehicle architecture, past, present, and future, and its current bottlenecks and future key technologies. Then, challenges of software configuration and mapping for automotive systems are discussed. Accordingly, mapping techniques and optimization objectives for mapping tasks to multi-core processors using design space exploration method are studied. Moreover, the current technologies and frameworks regarding the vehicle architecture synthesis, model analysis with regard to software integration and configuration, and solving the mapping problem for automotive embedded systems are expressed. Finally, we propose four research questions as future works for this field of study.

Keywords: E/E architecture; automotive software configuration; task mapping; multi-core processors; functional safety; architecture synthesis; design space exploration; optimization

check for updates

Citation: Askaripoor, H.; Hashemi Farzaneh, M.; Knoll, A. E/E Architecture Synthesis: Challenges and Technologies. *Electronics* **2022**, *11*, 518. https://doi.org/10.3390/electronics11040518

Academic Editors: Javier Alonso Ruiz, Angel Llamazares and Martin Lauer

Received: 29 December 2021
Accepted: 2 February 2022
Published: 10 February 2022

1. Introduction

The complexity and types of required applications in today's cars have been growing substantially, particularly as a result of advanced driver-assistance systems (ADAS) and automated driving features. Furthermore, meeting all non-safety and safety requirements in compliance with automotive standards (i.e., ISO 26262, and safety of the intended functionality (SOTIF)) during the design and configuration of automotive software architecture, increases complexity [1,2]. Accordingly, vehicle electrical and/or electronic (E/E) architecture has been evolving recently concerning the aforementioned complexity, which stems from the new applications and features integrated into the car, and the limitations of the traditional E/E architectures. The car E/E architecture started with distributed/decentralized architectures, where a considerable number of electronic control units (ECUs) are interconnected, and each of them has specific vehicular functionality. Then, it moves to domain-centralized, where centralized domain controllers are used, and centralized/zonal, where vehicle architecture makes use of zone controller architectures [3].

Considering ADAS and self-driving applications, using domain-specific ECUs results in an increase in the number of ECUs, substantial growth in the wiring harness, communication bandwidth, cost, software variants, and software complexity. Therefore, multi-core ECUs can be considered as a solution in order to reduce the number of ECUs, the cost, the wiring harness, and the complication and variations of the software. In addition, multi-core technology has rapidly 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 [4].

Furthermore, the design of automotive E/E architecture using ADAS functionalities and algorithms, comprising all their safety and non-safety-critical requirements, is an elaborate and time-consuming task that requires domain-specific knowledge [5]. Moreover, manual integration and configuration of the software architecture for an automotive high-performance central computer, while satisfying all safety essentials, is a challenging and error-prone task due to the high number of requirements and properties related to hardware, applications, operating system (OS), middleware, hypervisor, etc. Such configurations can be optimized as well by determining various optimization goals such as power consumption, resource utilization, reliability, temperature, and others. Therefore, approaches/tools, in order to automate the software configuration process during the design-time while taking all requirements and properties into account, can cope with growing complexity and facilitate and improve the design process focused on modeling and mapping of software components on the automotive high-performance computing unit (HPCU).

This paper summarizes the evolution of the car E/E architectures describing the main three E/E architecture types, the current main issues, and the key technologies for the future; in addition, the high-level software architecture of the automotive HPCU is illustrated. It goes through the current challenges for E/E architecture concentrated on software integration, configuration, the multi-core ECUs. Moreover, it provides an overview regarding task mapping approaches for multi-core computing units. In this study, furthermore, the available approaches and tools for automotive software integration, configuration, and E/E architecture synthesis, focused on mapping problems and modeling of the software components comprising design space exploration(DSE) approaches and optimization goals, are presented. As a result of this technology analysis, four research questions are depicted as future works in this research area. The analysis result can be useful for other researchers and E/E system architects in the industry in order to assist in automating and facilitating the configuration process of the software components, particularly applications for the automotive HPCUs.

This study is outlined as follows: Section 2 discusses the basic concepts about E/E architecture, its main key challenges, and technologies. Section 3 describes the challenges and technologies related to mapping in multi-core architecture and software integration and configuration. Section 4 describes further research areas, and finally, Section 5 expresses the conclusion of this work.

2. Basic Concepts

Almost all aspects of E/E system development, including technical approaches, specified requirements, decision making about design structure, and developments methods, are affected by vehicle E/E architecture. Vehicle architecture can be seen from various perspectives. The physical vision illustrates the positioning and the connection of the used elements in the car such as ECUs, sensors, actuators, gateways, power supply, and switches. Moreover, it includes communication networks, wiring harness placement, and power distribution setup. In addition, the automotive software plays a pivotal role in the vehicle E/E architecture as the autonomous level of cars has been improving in the last decade.

Another perspective of E/E architecture is logical vision so that the focus is the interaction and interconnection among various components and elements integrated into the car. Based on this point of view, the E/E architecture can be interpreted to be about data exchange, signal flows, and communication and interface protocols.

The level of driving automation has been improving in recent years, and a lot of companies have spent a huge amount of effort as well as cost to move towards an autonomous vehicle. This has enabled vehicles to provide not only non-critical functionalities and features, e.g., gaming using vehicle infotainment system, but also giving ADAS functionalities to drivers to have safe and more comfortable driving experiences. Accordingly, the automotive E/E architecture has advanced significantly, from various sensors and actuators to more powerful computing units to process a huge amount of data coming from the sensors for critical and non-critical functionalities [6].

With increasing number of functionalities and features, the vehicle E/E architecture has been altering during the last years. The car ECU starts as function-specific, where each ECU handles each function. Domain-specific ECU represents the next level of controllers where a computer controls a set of functions related to a specific area or domain. Functional domains that require a domain ECU are typically compute-intensive and connect to a large number of input/output (I/O) devices. Each domain ECU connects to multiple function-specific control units. A zonal ECU provides and distributes data and power; in other words, the vehicle battery power is diffused over the car architecture, e.g., for running various sensors and actuators, using zonal controllers. Similarly, the data are spread over 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 it acts as a gateway, switch, and as a smart junction box. Moreover, it supports any kind of interface for sensors, actuators, and displays.

The central HPCU is a multi-core ECU which comprises a multi-system on chips (SoCs) (each includes several cores), graphics processing unit (GPU), random access memory (RAM), and deep learning accelerators to process high computational power-demand applications such as various object detection applications used for ADAS functionalities [7]. The central computing unit is a fully scalable and upgradeable platform that connects to Edge and Cloud back-end and uses cloud computing for processing intensive vehicle functionalities. Furthermore, it can function as the zonal gateway. In other words, this ECU functions as the master and the core functionality of the vehicle.

Figure 1 shows the progress of E/E architectures. It starts from distributed E/E architecture, which consists of function-specific ECUs and a central gateway which are connected via controller area network (CAN) bus. Utilizing the central gateway provides stronger collaboration among ECUs, the ability to handle more complex functions, e.g., adaptive cruise control, and the potential of cross-functional connection. The next evolution represents domain centralized E/E architectures which utilize domain-specific ECUs [8]. As Figure 1 shows, function-specific control units bind to domain-specific ECUs using a CAN bus and Ethernet connection. Moreover, the central gateway ECU is used in this type of architecture [9]. This architecture is capable of handling more complex functions; furthermore, the architecture cost can be optimized using the consolidation of the functions. For instance, one domain-specific ECU is assigned for the parking assistance system which includes two function-specific controllers related to vision processing and actuator commands, e.g., for the brake and steering wheel.

Domain centralized architecture, using domain controllers and central gateway, has grown over time and become extremely elaborate, including the car wiring harness. Furthermore, the autonomous driving feature significantly increases the complexity of the architecture due to the increase in the number of sensors and actuators, growth of data processing capabilities and required bandwidth, and high demand for intelligent power distribution [10]. The future E/E architecture, which is a zonal architecture, can deal with the existed complexity in the last two architectures using the central HPCU [3]. The zonal architecture blends future vehicle functions and technologies with savings in weight and cost. As Figure 1 presents, the zonal architecture comprises a HPCU, zonal ECUs, and function-specific ECUs. The central HPCU acts as the master to process all data coming from different vehicle zones and consequently operate the car. In addition, the HPCU functions as a central gateway to pass the data from one zone to another [11]. The ECUs and HPCU are interconnected via the Ethernet connection for transmitting the data over the vehicle's network due to its speed and high bandwidth for data transmission [8]. More importantly, the zonal architecture supports the virtual domain in such a way that the embedded functions can be transferred into the cloud as well as providing software download/update via update over the air (OTA) service for HPCU [12].

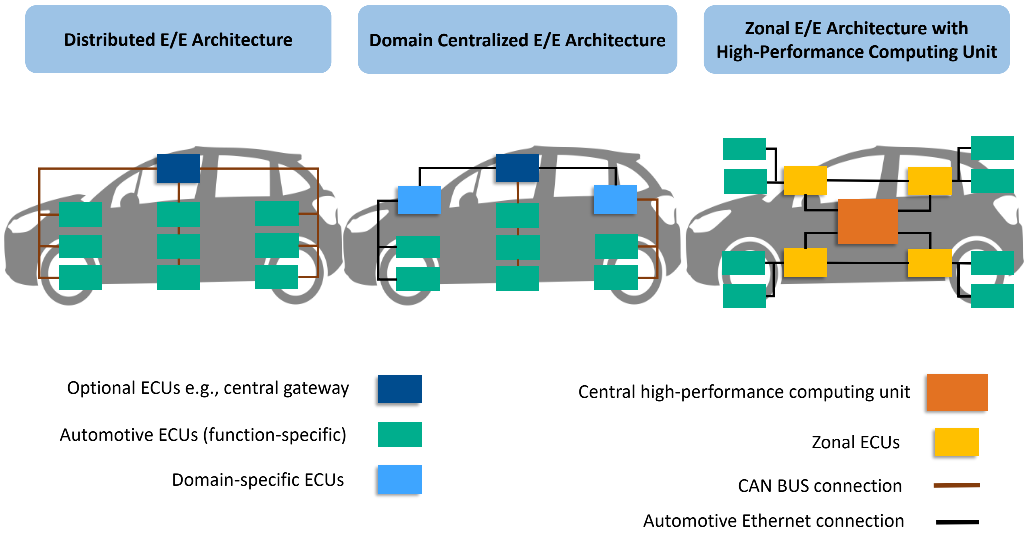

Distributed E/E Architecture
Domain Centralized E/E Architecture
Zonal E/E Architecture with High-Performance Computing Unit
Optional ECUs e.g., central gateway
Automotive ECUs (function-specific)
Domain-specific ECUs
Central high-performance computing unit
Zonal ECUs
CAN BUS connection
Automotive Ethernet connection

Figure 1. The figure presents the evolution of vehicle E/E architecture. Distributed E/E architecture was used until 2019, while domain-centralized architecture is today's vehicle architecture. The zonal architecture shows the future car E/E architecture.

2.1. The Main Bottlenecks of Current E/E Architecture

Current E/E architectures are capable of dealing with many requirements; however, their capability cannot be sufficient to handle the requirements for self-driving/future vehicles. With the increasing number of functionalities, applications, sensors, and actuators, as well as the necessity of meeting safety demands based on the functional safety standards, the future ECUs require more capability than current ones in terms of computational power, communication interfaces, and software integration and architecture. Moreover, the required communication bandwidth for autonomous cars is another bottleneck for today's E/E architecture. Although current communication networks (e.g., CAN) have been handling in-vehicle communication for the last decade, neither its communication distance nor its communication rate can be compared with Ethernet, which supports a variety of communication protocols and includes system interoperability, compatibility, and strong ability to share resources [13]. In addition, low latency, safe persistency, safe data transmission, and high security in network play a pivotal role in future vehicles. Moreover, external communication, for instance, software download/update or vehicle-to-vehicle (V2V) communication causes the need for higher data traffic and higher data protection [14]. Therefore, communication protocols such as automotive Ethernet will be required to tackle the above-mentioned requirements [15].

Another bottleneck for today's architecture is the implementation of new technology into existing E/E architectures. Consequently, the autonomous car's E/E architecture must be designed and developed in such a way that it supports extensibility and feasibility for launching new technologies without changing the architecture backbone [16,17]. Therefore, having powerful computing units, using communication protocols capable of transferring a high volume of data at a fast speed while satisfying automotive safety regulations, and developing the approaches to facilitate the new feature integration into existing vehicle architecture are the most prominent features to overcome the current issues of the E/E architecture.

2.2. The Main Technologies for Future's E/E Architecture

The new functionalities used in self-driving cars require advanced technologies to meet the requirements of the features in future cars. Having zonal ECUs (functioning as advanced gateways as well) with higher computing power, using central computing units as the core of the vehicle's functionality, applying new in-vehicle communication networks, and defining new software architecture for the vehicle are the major key technologies that can be considered for automated driving cars.

To fulfill the demand for in-vehicle network bandwidth, automotive Ethernet can be used to provide higher bandwidth, higher security, and better fulfillment of the safety requirements based on ISO 26262. In addition, providing low latency in the communication network by using new message routing mechanisms plays a significant role in accelerating the progress of the in-car network for autonomous vehicles. For instance, time-sensitive networking (TSN) standards can be utilized in the automotive network to ensure certain safety-related communication requirements and tasks including guaranteed packet transport with bounded latency, low packet delay variation, low packet loss, etc. [15,18].

One of the most important key technologies for moving towards centralized E/E architecture is utilizing multi-core processors, comprising AI accelerator, as the master of computing units. With the increasing number of AI applications, specifically ADAS applications using deep learning and machine learning algorithms that require high computational power including vision part, there will be a need to utilize central HPCU in the vehicle's E/E architecture to process and compute these applications. Furthermore, the main software architecture of the whole vehicle is defined in the HPCU, where various software domains can be integrated such as perception, mapping, planning, ADAS applications, and infotainment; moreover, utilizing such a centralized architecture with advanced software-defined vehicle (SDV) architecture provides the opportunity for original equipment manufactures (OEMs) to integrate and update advanced software as they are developed by analogy with a smartphone [19,20].

As mentioned before, the new software-defined architecture is an inseparable part of the future E/E architecture [19]. Hardware virtualization technology should be taken into consideration to provide integration of safety and non-safety-critical software domains into the HPCU in such a way that does not violate the safety requirements and utilize the hardware resources as optimized as possible. This technology can be approached by using a hypervisor. A hypervisor is software that creates and executes virtual machines (VMs) so that the virtualized hardware resources will be shared among several instances using various types of OS [21]. There are two types of hypervisors:

- Type 1: also called Bare Metal or Native hypervisor, as it is installed and runs directly on top of the host's hardware without using any host OS. This type of hypervisor has direct control over and access to hardware resources. For example, a type-1 hypervisor can assign a specific core to a partition (an execution environment managed by the hypervisor which uses the virtualized services) in such a way that other partitions cannot access that core.
- Type 2: also known as Hosted hypervisor. It runs as an application in the host OS and uses the hardware resources for its VMs by coordinating calls through the host's OS. The host OS does not have any knowledge about this type of hypervisor and it treats it as any other normal process [22].

Figure 2 shows a high-level defined software architecture for the car, which is integrated into a central HPCU. Different partitions are configured to isolate various application domains, i.e., merging mixed-critical applications, and make use of the hardware resources as efficiently as possible using a type-1 hypervisor so that each partition can approach the HPCU resources directly, including cores, RAM, GPU, cache, network buses, universal serial bus (USB), etc. As Figure 2 describes, each partition has its OS and application domain, e.g., perception, which is placed on top of the OS. As mentioned before, each partition can use HPCU resources directly which achieves freedom from interference (FFI) based on ISO 26262 [2]. In other words, in the type-1 hypervisor, the resources among the partitions cannot be either shared or accessed by another partition (see yellow dash lines in Figure 2). However, some resources may not follow the FFI requirement depending on which hypervisor, from a variety of open-source and commercial hypervisors developed by different companies, and hardware are going to be utilized.

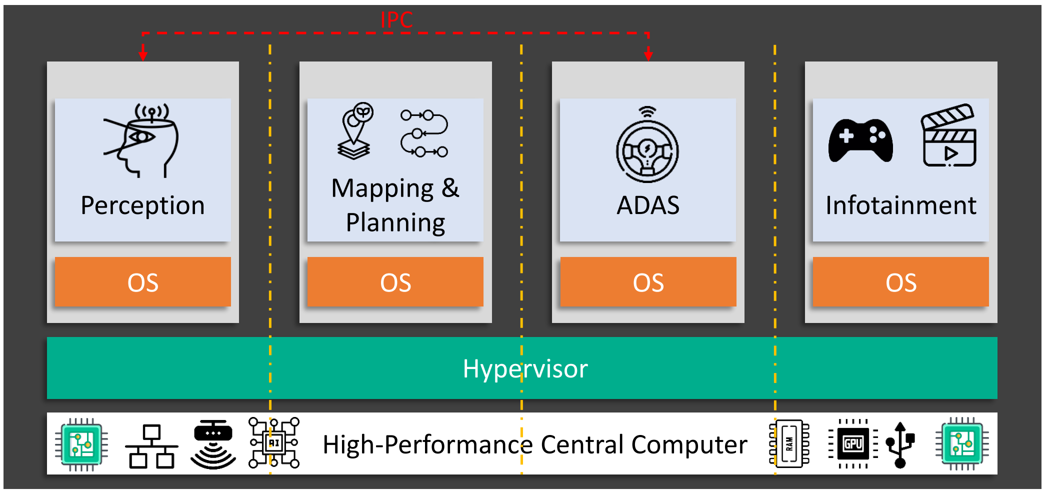


Figure 2. The software architecture integrated into the vehicle's HPCU using a type-1 hypervisor which consists of four mixed-critical partitions. The yellow dash lines show the hard separations between partitions starting from the hardware level.

Furthermore, the inter-process communication (IPC) feature can be used to activate communication between two partitions using type-1 hypervisor [23]. For example, if one process in a safety-critical partition (e.g., perception) intends to interact with another process in another safety-critical partition (e.g., ADAS) to transfer messages, it can be accomplished by IPC (see Figure 2).

3. Technologies and Challenges

In the last years, with the increasing levels of vehicle automation (e.g., ADAS) the need for computational power in vehicle ECUs has grown significantly. Current cars are equipped with 70 to 100 ECUs to manage the software system [24]. The number of ECUs could be reduced considerably by merging various mixed-critical applications into one multi-core ECU. Figure 3 depicts the configuration procedures in design and run-time which are required to be performed by a system architect to find a mapping solution for deploying mixed-critical software components to the HPCU considering their properties and configuration parameters, safety-critical/non-critical requirements of the applications, and optimization objectives. In addition, the system architect must verify the fulfillment of the requirements after solution deployment in run-time due to the unpredictable behavior of OS, middleware, and applications (see Figure 3) [25].

However, designing such configurations and mappings turns out extremely complicated and challenging [26,27]. Moreover, depending on the mapping problem, finding the configuration may become a non-deterministic polynomial-time (NP)-hard problem ([28–30]) due to the huge number of safety-critical/non-critical application and user requirements (e.g., reliability [31], timing, latency, worst-case execution time, central processing unit (CPU) utilization, bandwidth, memory usage, power consumption, FFI, automotive safety integrity (ASIL) level [32], redundancy, etc.), the extensive design space of possible task mappings, and non-deterministic/dynamic workload. Furthermore, with the increase in numbers of applications and components, including their requirements and properties, discovering the configuration and mapping solution becomes more and more complex for the system architect, and the need to have a new update in the configuration might lead to unknown risks and become costly (see Figure 3) [7].

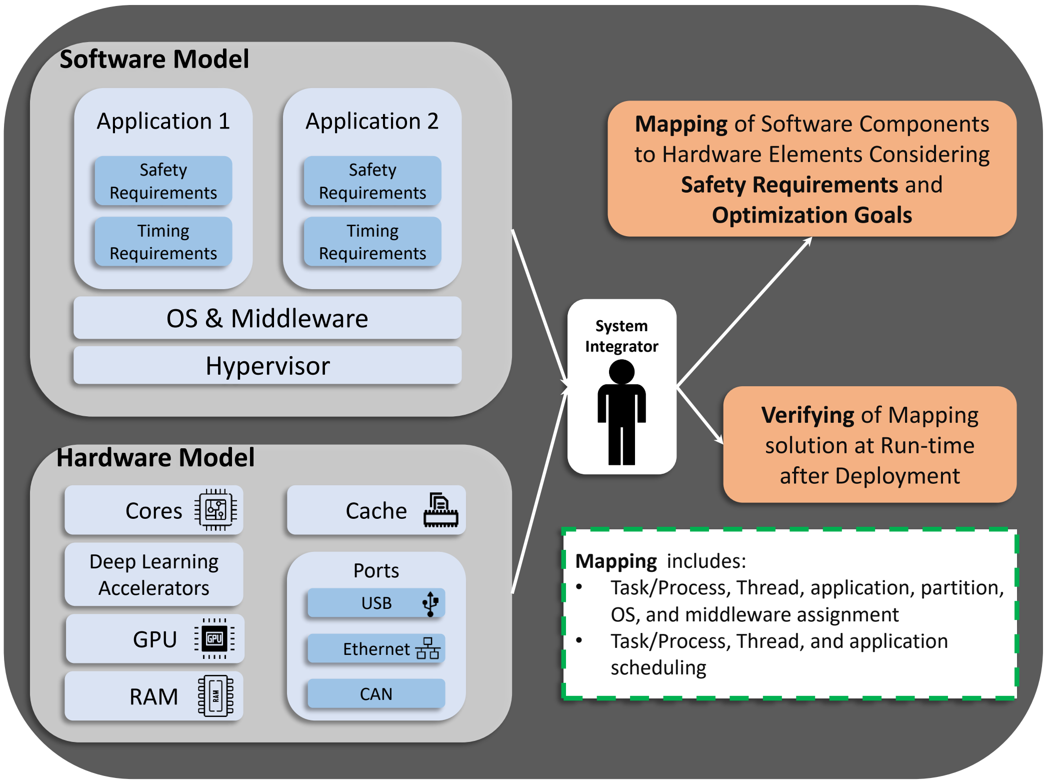

Software Model
Application 1
Safety Requirements
Timing Requirements
Application 2
Safety Requirements
Timing Requirements
OS & Middleware
Hypervisor
Hardware Model
Cores
Cache
Deep Learning Accelerators
Ports
USB
Ethernet
CAN
GPU
RAM
System Integrator
Mapping of Software Components to Hardware Elements Considering Safety Requirements and Optimization Goals
Verifying of Mapping solution at Run-time after Deployment
Mapping includes:
• Task/Process, Thread, application, partition, OS, and middleware assignment
• Task/Process, Thread, and application scheduling

Figure 3. The manual procedures have to be executed by a system integrator to map automotive software components to an automotive HPCU.

As explained in Sections 2.1 and 4, current and future E/E architectures encounter various challenges and bottlenecks including software integration and configuration for the HPCU. Hence, this paper focuses mainly on software integration and configuration for automotive embedded systems. It firstly expresses the existing approaches and methods for dynamic and static task mappings in multi-core processors considering various optimization goals. Secondly, it discusses the current design-time technologies and approaches for software integration, including model analysis, E/E architecture synthesis, configuration, model checking, and mapping.

3.1. *Task Mapping in Multi-Core Computing Units*

As discussed before, in this section the current mapping approaches and methods and studied optimization objectives for task mapping in multi-core platforms are illustrated.

3.1.1. Introduction to Task Mapping

Allocation of the tasks in the multi-core can be performed in design-time or run-time. There is no application running during design-time task allocation as opposed to run-time mapping, where the tasks can be assigned to different cores, while the system is running. When the application requirements are fairly deterministic, design-time mapping is preferred, whereas the run-time assignment should be used when the application requirements are changing in dynamic scenarios and there is a need for reassignment.

Multi-core architecture can consist of two types, homogeneous and heterogeneous, based on the application circumstances and user requirements. In homogeneous multi-core architecture, all the cores are identical (i.e., the cores have similar computing capacity and instruction set architecture (ISA)). However, heterogeneous architecture is a combination of various cores which are particularly designed, e.g., cores that provide high performance and low power consumption [33]. Various types of processing units can be integrated on a single chip by utilizing heterogeneous architecture, which ensures lower energy consumption and better flexibility [34]. Additionally, automotive applications increase the demand for heterogeneous architecture due to different contexts and needs. In terms of task mapping complexity, a homogeneous processor requires less work than a heterogeneous one as it has identical cores. Furthermore, homogeneous multi-cores do not need task analysis based on core properties for task execution in contrast to heterogeneous multi-core architecture.

3.1.2. Mapping Techniques

Static or design-time mapping utilizes 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 requirements for the applications as well as the hardware. In other words, the mapping problem, including changing application/hardware requirements, cannot be solved dynamically with design-time methodology. Solutions with higher quality can be acquired in the design-time mapping problem rather than the run-time mapping problem due to less limitation of the computational power.

Design-Time Mapping: There are several design-time mapping algorithms such as graph-theoretic algorithms, mathematical programming algorithms, and heuristic-based algorithms. In graph-theoretic algorithms, the application is partitioned into separate tasks (can be allocated to the cores for execution) to use the fundamental parallelism [33,35]. This approach includes various theory methods including Levelized Weight Timing [36], Shortest tree [37], Max-Min [38], hyper-graph [35], A* [36], and Kahn Process networks [39].

Another design-time approach to solve the mapping problem is mathematical programming, where the requirements are transformed into mathematical inequalities. After that, by utilizing different mathematical programming solutions including mixed integer linear programming (MILP) [40], Branch and bound [41], constraint programming (CP) [42], and integer linear programming (ILP) [43], the inequalities are solved. In the mathematical programming approach, the optimal solution is always guaranteed as long as the complexity of the mapping problem does not become NP. In the case of the NP problem, heuristic-based algorithms can be used.

The last design-time approach is heuristic-based algorithms. As mentioned before, with increasing the problem complexity and proximity to an NP problem (e.g., increasing the number of cores and complexity of the application requirements), finding the optimal solution in the desired time by utilizing mathematical programming algorithms is infeasible. Therefore, heuristic algorithms are presented to discover a solution which may not be the best of all the solutions to this problem or may approximate the exact solution in a more faster and efficient fashion. They classify alternatives in search algorithms at each branching step based on current information to decide which branch to go after. Heuristic algorithms can be divided into Population-Based and Single Solutions. For example, the greedy randomized adaptive search procedure (GRASP) [44], simulated annealing (SA) [45], and Tabu search [38], which utilize iterative search methods, are considered as the Single solution algorithms. In addition, there are several other heuristic-based algorithms which are categorized as Population-based including genetic algorithm (GA) [46], ant colony optimization (ACO) [47], and particle swarm optimization (PSO) [48].

Run-Time Mapping: In this type of mapping, the tasks' assignments to various cores are performed when the applications are executing. The required time to find a feasible and optimal solution plays a critical role in this mapping methodology. According to [33,34], run-time mapping can be divided into two approaches: On-the-fly mapping and Hybrid mapping. When task allocation is completely online or performed during application execution (i.e., without using offline or design-time knowledge) this is interpreted as On-the-fly mapping. While in Hybrid mapping, the task planner uses the offline/design-time mapping result to perform task assignments dynamically. This approach takes advantage of both run-time and design-time techniques. Similar to design-time, there are three well-known algorithms which can be utilized for run-time mapping such as Greedy [49], Feedback control theoretic, and Heuristic-based algorithms [33].

3.1.3. Optimization Parameters in Mapping

To improve the quality of task assignment in multi-core processors, either in static or dynamic mapping, some key requirements must be considered.

Performance: The design of multi-core processors has been becoming more and more complex with the increasing number of applications as well as their requirements. Various design-time and run-time approaches have been developed to optimize the performance of mapping (i.e., reducing task execution time during task allocation [43] or increasing CPU utilization). The performance of a multi-core system can be assessed by measuring execution time, latency, response time, and throughput as the evaluation metrics [30].

Power Consumption: As electric vehicles are becoming more popular and most of autonomous cars will be electric-based, saving electric power has a significant impact on providing an optimized E/E system for the vehicle. As a result, reducing energy consumption during application mapping in multi-core computational units plays a pivotal role in mapping optimization [50]. Based on [51], minimizing cache miss results in energy reduction in a system by 76 percent.

Reliability: Another important parameter that must be considered as an optimization goal in multi-core computing units is reliability. ISO 26262 has introduced different ASILs to ensure the reliability of the system [2]. System reliability can be evaluated using the mean time to failure (MTTF), mean time between failures (MTBF), and mean time to repair (MTTR) as failure metrics [52]. For example, system reliability is improved in [53] utilizing the SA approach. This paper calculates the MTTF by using the temperature variation of cores. Based on this definition, task allocation to the cores is performed in such a way that MTTF is minimized.

There are several works regarding the static mapping problem comprising various optimization objectives that have utilized the ILP method. For instance, [40,43,54,55] have studied the design-time mapping problem in homogeneous multi-core architecture using the ILP method while optimizing their solutions considering performance, energy consumption, and temperature as the optimization parameters. Authors in [56] utilized the same method to solve mapping for a heterogeneous architecture, including different optimization goals such as execution time, reliability, and temperature. Furthermore, [45,53,57,58] studied the mapping problem of homogeneous architecture in design-time using heuristic-based approaches including SA, ACO, and GA while optimizing the final solution based on response time, reliability, and energy consumption.

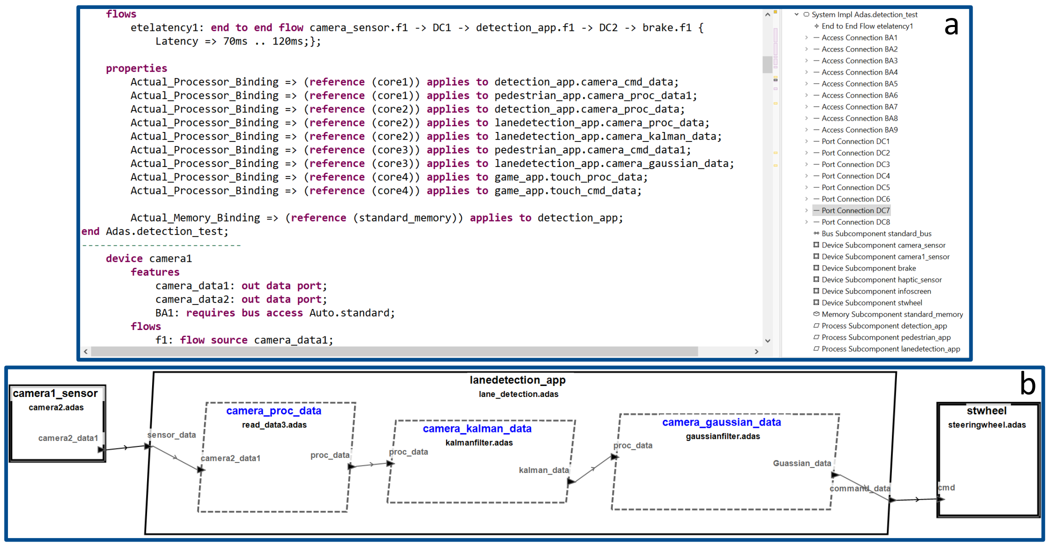
Several pieces of research have been completed to assign tasks to multi-core processors dynamically. Ref. [59–61] worked on the heterogeneous architecture to perform run-time mapping using the ILP method. They also optimized their solution, considering performance and power consumption as the optimization goals. While [62] utilized the heuristic method (i.e., Age Balancer Heuristics) to execute dynamic mapping on heterogeneous architecture while aiming to minimize computation energy and improve reliability.

Therefore, with an increase in demand for higher efficiency of mapping in terms of the aforementioned optimization parameters, new challenges have appeared in multi-core processors. These challenges can include thermal management in integrated circuits (ICs), machine learning approaches to perform efficient mapping, and quality of service (QoS) in multi-core architecture [33,34].

3.2. Technologies for Software Integration and Configuration in Design Process

As mentioned before, software integration and configuration for future vehicles will be considered a challenge using manual integration. In this section, existing technologies for software integration, focused on model analysis, model checking and validation based on requirements, and mapping problem, in the embedded systems are explained in detail. Moreover, each technology is analyzed based on problem attributes and various design metrics, DSE approaches, optimization algorithms, and safety-related and optimization attributes.

OSATE (Open Source AADL Tool Environment): This is a powerful open-source tool that creates AADL (architecture analysis and design language) models using a syntax-aware text editor and synchronized graphical editor (see Figure 4). OSATE is an Eclipse-based tool and comprises modeling elements for aerospace and the automotive systems using AADL language [63,64].



```
    flows
        etelatency1: end to end flow camera_sensor.f1 -> DC1 -> detection_app.f1 -> DC2 -> brake.f1 {
            Latency => 70ms .. 120ms;};

    properties
        Actual_Processor_Binding => (reference (core1)) applies to detection_app.camera_cmd_data;
        Actual_Processor_Binding => (reference (core1)) applies to pedestrian_app.camera_proc_data1;
        Actual_Processor_Binding => (reference (core2)) applies to detection_app.camera_proc_data;
        Actual_Processor_Binding => (reference (core2)) applies to lanedetection_app.camera_proc_data;
        Actual_Processor_Binding => (reference (core2)) applies to lanedetection_app.camera_kalman_data;
        Actual_Processor_Binding => (reference (core3)) applies to pedestrian_app.camera_cmd_data1;
        Actual_Processor_Binding => (reference (core3)) applies to lanedetection_app.camera_gaussian_data;
        Actual_Processor_Binding => (reference (core4)) applies to game_app.touch_proc_data;
        Actual_Processor_Binding => (reference (core4)) applies to game_app.touch_cmd_data;

        Actual_Memory_Binding => (reference (standard_memory)) applies to detection_app;
end Adas.detection_test;
--------------------------
    device camera1
        features
            camera_data1: out data port;
            camera_data2: out data port;
            BA1: requires bus access Auto.standard;
        flows
            f1: flow source camera_data1;
```

System Impl Adas.detection_test
- End to End Flow etelatency1
- Access Connection BA1
- Access Connection BA2
- Access Connection BA3
- Access Connection BA4
- Access Connection BA5
- Access Connection BA6
- Access Connection BA7
- Access Connection BA8
- Access Connection BA9
- Port Connection DC1
- Port Connection DC2
- Port Connection DC3
- Port Connection DC4
- Port Connection DC5
- Port Connection DC6
- Port Connection DC7
- Port Connection DC8
- Bus Subcomponent standard_bus
- Device Subcomponent camera_sensor
- Device Subcomponent camera1_sensor
- Device Subcomponent brake
- Device Subcomponent haptic_sensor
- Device Subcomponent infoscreen
- Device Subcomponent stwheel
- Memory Subcomponent standard_memory
- Process Subcomponent detection_app
- Process Subcomponent pedestrian_app
- Process Subcomponent lanedetection_app

Figure 4. The AADL text editor (**a**) to synchronized graphical editor (**b**) in OSATE framework.

AADL is a modeling language that supports early and repeated analyses of a system's architecture concerning performance-critical properties through an extendable notation, a tool framework, and precisely specified semantics. The language utilizes formal modeling concepts for the description and analysis of application system architectures in terms of distinct components and their interactions. It includes abstractions of software (e.g., process and thread), computational hardware (e.g., processor, bus, device, and memory), and system components for determining and analyzing automotive, aerospace, and real-time embedded systems, and investigating the performance capabilities of the designed system, for instance, data-flow analysis (i.e., collecting information about the set of values computed at different points in the designed model/system). It also provides mapping of software components (e.g., a process) into computational hardware elements, for example, a processor. AADL is especially effective for model-based analysis and specification of complex real-time embedded systems [65].

By using the OSATE tool, the user can model a system (e.g., an ADAS system) including the hardware and software down to the application-level (see Figure 4). For instance, the threads used for each application can be modeled, including their properties such as period, compute execution time, million instruction per second (MIPS) budget, and reference processor. The OSATE checks the model created by the user in terms of syntax issues regarding the AADL text and violations in properties definition for each specified component. Moreover, various model analyses can be performed using this framework comprising a flow latency check including end-to-end flow latency computation, scheduling analysis (such as scheduling bound threads, i.e., processor utilization report, binding and scheduling threads, i.e., thread binding report, and rate monotonic priority assignment), budget analysis (comprising analyzing bus load, power requirements, resource allocations, computer resource budgets, and calculating the total weight), and safety analysis comprising fault tree analysis (FTA), functional hazard assessment (FHA), fault impact analysis, failure mode effect analysis (FMEA), and checking unhandled faults. In addition, various semantic checks or functional integration analyses can be performed using this framework such as checking binding constraints, connection binding consistency, port connection consistency, etc. Furthermore, this tool has different code generation capabilities utilizing various plugins, e.g., Ocarina, and is capable of importing models from MATLAB and Simulink into the OSATE [66–68].

However, OSATE does not use or cover any DSE approaches, e.g., solving mapping problems for multi-core automotive computing units and, consequently, it supports no optimization [64]. In addition, it covers a limited number of safety attributes, as illustrated before.

ArcheOpterix: As mentioned before, finding an acceptable architecture design is a challenging task for software and system architects, considering quality and functional requirements in the architecture design phase. ArcheOpterix is an open-source Eclipse-based tool that contributes to simplifying the task using evaluation techniques, a DSE approach, and optimization heuristics for AADL specifications. The framework supports modeling of software components and communication among software components, ECUs, buses, and services. Moreover, the tool can optimize the deployment of software components to ECUs considering design constraints and optimization objectives, including redundancy allocation and cost [69]. This tool can specify the uncertain information relevant to system parameters and, therefore, search for the most optimal and robust candidate architecture. In addition, a list of the most appropriate optimization algorithms, comprising 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, can be provided by ArcheOpterix so that the user can choose the most suitable one [70].

Figure 5 shows the high-level architecture of the ArcheOpterix framework. As shown, it consists of various modules, of which the most important parts are explained in the following [70].

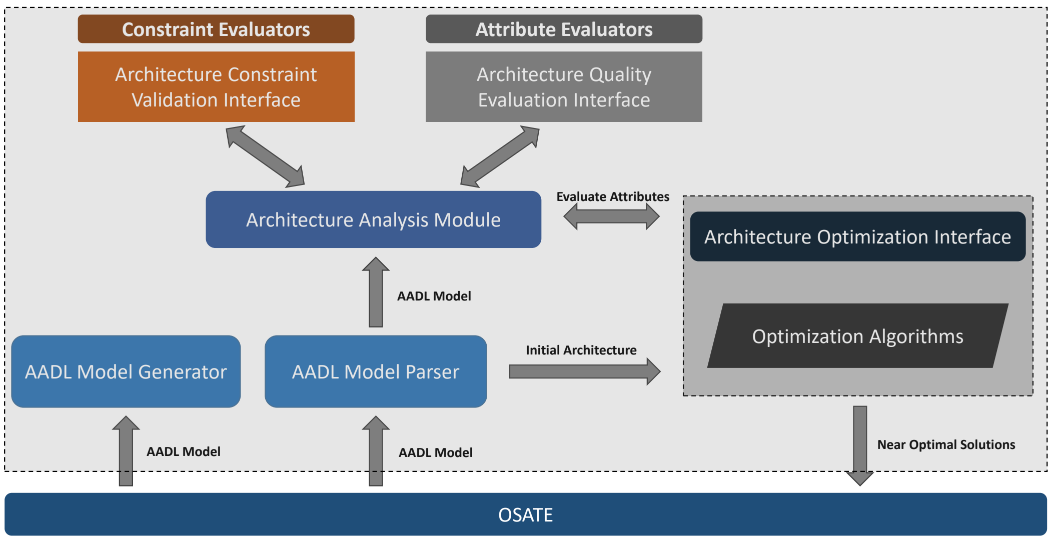


Figure 5. The architecture of ArcheOpterix framework.

- AADL Model Parser: This interprets and extracts system descriptions from an AADL specification coming from the OSATE tool. The module can access AADL elements such as components, services, buses, etc. The extracted parameters are sent to the Architecture Analysis Module, as an input, which supports the two interfaces for analyzing the model comprising Architecture Constraints Validation and Architecture Quality Evaluation Interface (see Figure 5).
- Architecture Constraints Validation Interface: As displayed in Figure 5, it provides a plug-in point for Constraint Evaluator modules that check a given architecture for constraint satisfaction.

- Architecture Quality Evaluation Interface: In this part, various quality evaluation functions can be taken into account. In ArcheOpterix, the Attribute Evaluator module performs quality evaluation functions, which can be extended for evaluated features. Current integrated features in ArcheOpterix are Service Reliability, Data Transmission Reliability, and Communication Overhead.
- Architecture Optimization Interface: This provides an opportunity to add new optimization algorithms to the framework. The current tool comprises Exact Algorithms, Genetic Algorithms, and Ant Colony Optimization [70].

However, this tool has several limitations in the context of mapping, architecture synthesis, and software integration for automotive platforms. First of all, the framework is outdated and is not well-documented for use. It does not support mapping analysis and solving the mapping problem for multi-core architecture, and there is no focus on multi-core computing platforms for automotive applications. In addition, the safety-related attributes regarding ISO 26262 are not covered in this tool except for reliability, and the framework itself supports no model checking and model analysis. Additionally, the covered optimization objectives by ArcheOpterix are restricted, including cost, communication overhead, and data transmission reliability; furthermore, the tool has a limited number of architectural elements.

PerOpteryx: This is another open-source framework for feature configuration and clustering during the design stage in the software domain. Authors in [71] claimed that this approach can contribute to finding optimal solutions for software architecture based on predefined requirements and constraints while applying multi-objective evolutionary optimization to software architectures modeled with the Palladio Component Model. In such a case, software architects can select the most suitable architecture for their situation. This approach provides software architecture solutions based on different quality attributes such as performance, cost, and reliability using the DSE method.

PerOpteryx automatically creates architecture candidates based on several degrees of freedom of component-based software architectures and afterward evaluates and optimizes these architecture candidates according to the specified requirements. This approach was validated by applying two different architecture models comprising a business reporting system and an industrial control system.

However, this framework has no support for mapping analysis or DSE approach for task mapping i.e., finding a mapping solution for assigning, e.g., processes into cores integrated into automotive high-performance computing units, as meeting all safety and non-safety requirements. Further, it has limited elements which can be used in the software architecture, and there is no model checking or analysis integrated into this approach. More importantly, automotive safety parameters have not been defined in this open-source framework, and PerOpteryx is outdated and suffers from a lack of proper documentation.

MechatronicUML: As far as modern technical systems are elaborate, including reconfigurable mechatronic systems where most control and reconfiguration functionality is identified in the software, several requirements have to be satisfied to apply the model-driven development approach to these types of systems. Therefore, an open-source framework, based on the Eclipse framework, to model software/hardware components, define constraints, and verify the defined models, comprising the constraints using a model checker, namely UPPAAL, for the embedded systems is presented in [72,73]. This tool is a model-based approach and tries to bring model-based design formal analysis to the mechatronic domain. This tool supports the DSE approach to find the solution based on predefined constraints in the model; moreover, it provides software reconfiguration in such a way that this function allows the user to design a system so that the system adapts automatically at runtime according to the changing environment. This framework has the capability of C code generation, and the designed models can be simulated using MATLAB Simulink, Modelica, or the Functional Mock-up Interface (FMI) [68,74,75].

Nonetheless, this tool has limitations based on our criteria for mapping and software integration in the automotive domain. Firstly, mapping analysis and DSE related to mapping problem in computing units are not covered by this framework, and MechatronicUML supports no optimization. Furthermore, no safety-related attributes were considered in this tool for modeling, analysis, and solving, and this approach does not focus on multi-core or high-performance computing unit modeling and analysis. In addition, this tool is not updated and has no active community or proper documentation.

App4MC: As mentioned before, the automotive industry increasingly utilizes multi- and many-core systems to deal with ADAS and self-driving functionalities due to a considerable number of applications that require high computational power for processing. APP4MC is an open-source Eclipse platform that concentrates on performance simulation regarding mostly scheduling and timing analysis in multi-core platforms using a model-based development approach (see Figure 6) [76,77]. The hardware and software elements can be modeled, including different properties such as the processor type, connection type among various specified modules, OS schedulers, and task properties such as execution time and deadline. In addition, different timing constraints with respect to the tasks, OS schedulers, and mapping constraints (i.e., assigning tasks and schedulers to a specific core) can be defined, visualized, checked, and finally validated using this tool.

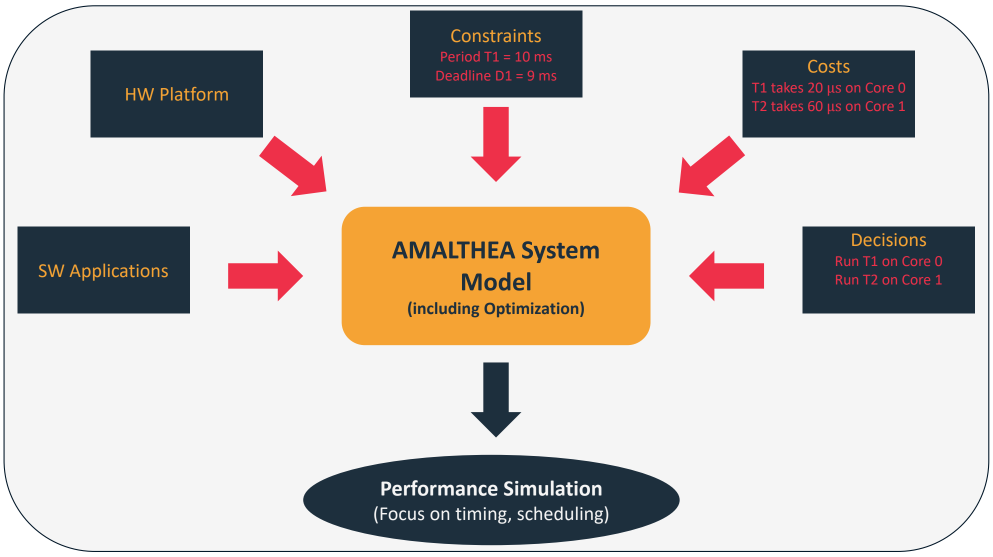
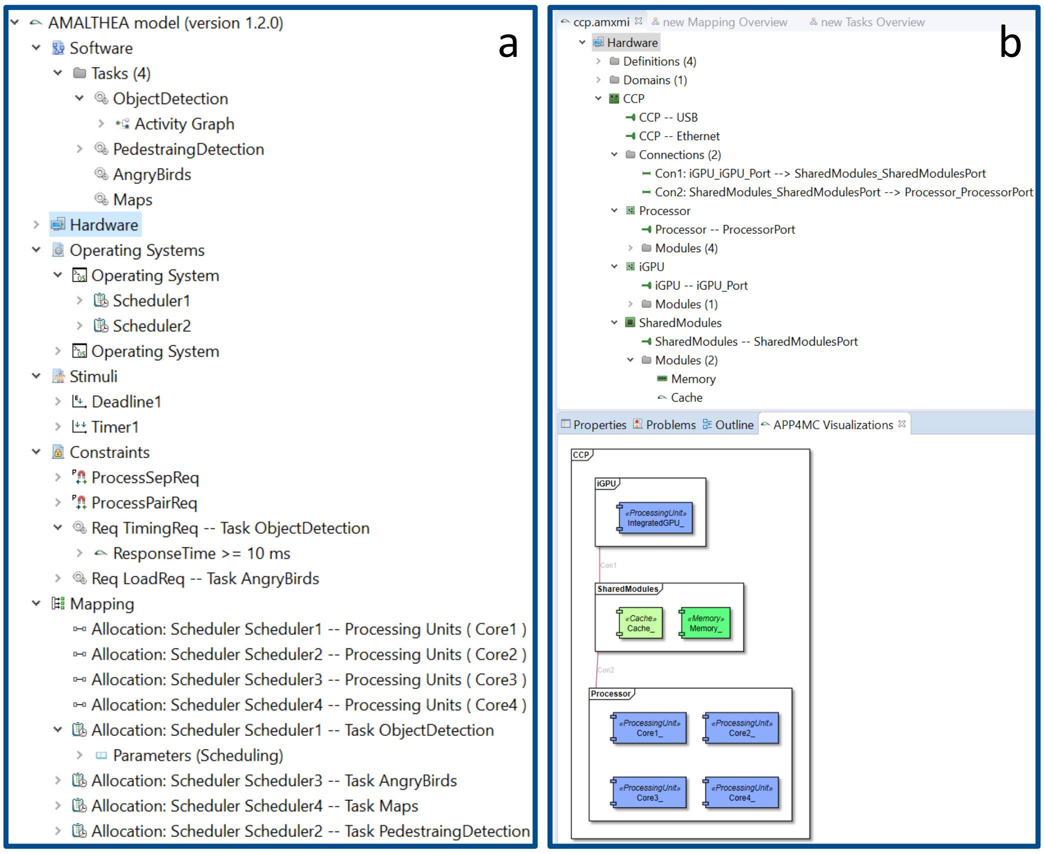


Figure 6. The APP4MC architecture.

Moreover, the definition of the hardware and software model and constraints and the whole model can be simulated (i.e., using different graphs e.g., Gantt chart and tables to present the result) while considering different optimization goals comprising load balancing, energy consumption, and memory mapping. Figure 7 shows an example of a modeled automotive system in APP4MC including tasks, hardware, OS, Stimuli, constraints, and mapping. For instance, in the mapping part, schedulers can be assigned to the cores and tasks (see Figure 7a). In Figure 7b the hardware model, including a processor (comprising four cores), integrated GPU (iGPU), shared modules such as cache and memory, and communication among these components, is visualized using the AAP4MC visualization feature.



```
a
AMALTHEA model (version 1.2.0)
  Software
    Tasks (4)
      ObjectDetection
        Activity Graph
      PedestraingDetection
      AngryBirds
      Maps
  Hardware
  Operating Systems
    Operating System
      Scheduler1
      Scheduler2
    Operating System
  Stimuli
    Deadline1
    Timer1
  Constraints
    ProcessSepReq
    ProcessPairReq
    Req TimingReq -- Task ObjectDetection
      ResponseTime >= 10 ms
    Req LoadReq -- Task AngryBirds
  Mapping
    Allocation: Scheduler Scheduler1 -- Processing Units ( Core1 )
    Allocation: Scheduler Scheduler2 -- Processing Units ( Core2 )
    Allocation: Scheduler Scheduler3 -- Processing Units ( Core3 )
    Allocation: Scheduler Scheduler4 -- Processing Units ( Core4 )
    Allocation: Scheduler Scheduler1 -- Task ObjectDetection
      Parameters (Scheduling)
    Allocation: Scheduler Scheduler3 -- Task AngryBirds
    Allocation: Scheduler Scheduler4 -- Task Maps
    Allocation: Scheduler Scheduler2 -- Task PedestraingDetection
```

```
b
ccp.amxmi    new Mapping Overview    new Tasks Overview
Hardware
  Definitions (4)
  Domains (1)
  CCP
    CCP -- USB
    CCP -- Ethernet
    Connections (2)
      Con1: iGPU_iGPU_Port --> SharedModules_SharedModulesPort
      Con2: SharedModules_SharedModulesPort --> Processor_ProcessorPort
    Processor
      Processor -- ProcessorPort
      Modules (4)
    iGPU
      iGPU -- iGPU_Port
      Modules (1)
    SharedModules
      SharedModules -- SharedModulesPort
      Modules (2)
        Memory
        Cache
Properties    Problems    Outline    APP4MC Visualizations
```

Figure 7. An automotive HW and SW system model including timing and mapping constraints (**a**), and visulization of the HW model (**b**) in APP4MC framework.

However, this framework does not contribute to automating the mapping of various tasks to various HW elements, for instance, core, including the safety and non-safety requirements (i.e., DSE for mapping problem). APP4MC only analyzes and simulates the task mapping but not solving. Furthermore, it is limited in the covered safety attributes and optimization goals, and there are a considerable number of E/E architecture elements which are not considered by AAP4MC.

Autofocus3: Another open-source and Eclipse-based tool uses a model-based development approach to synthesize E/E architectures. This framework supports architecture modeling from requirements to code generation for embedded systems. Additionally, the tool can simulate the designed model including its defined constraints and check and validate the model. It utilizes domain-specific modeling language to formalize an exploration problem and is capable of calculating end-to-end latency and schedule synthesis using the DSE method; moreover, optimization algorithms, such as binary search, have been integrated into this tool to apply the defined optimization objectives, including timing and communication load, to the explored solution [78,79].

Autofocus3 does not cover the solving approach for the mapping problem in the multi-core platforms considering functional and non-functional requirements to automate mapping configuration and there is no mapping analysis for these platforms. In addition, it supports a limited number of safety attributes, only ASIL level, and optimization goals for E/E architecture synthesis.

Clafer: An approach that combines structural modeling with behavioral formalism to contribute to the mapping of feature configurations to component configurations or model templates. Calfer allows capturing feature models (variability), component models, and discrete control models (automata) in a single unified syntax and semantics. The language part is built on top of first-order logic with quantifiers over basic entities (for modeling structures) combined with linear temporal logic (for modeling behavior). This approach provides DSE for feature modeling considering timing as a problem attribute, and it supports model analysis. Furthermore, it covers multi-objective optimization for the discovered solution comprising mass, end-to-end latency, and cost [80].

However, Clafer supports no model checking, mapping analysis, or DSE for mapping problems in the embedded systems. The architectural elements are limited; moreover, there are no safety-related attributes covered by Clafer for model analysis. This approach has not given any considerations to the modeling and analysis of multi-core computing units. Furthermore, this approach is not well-documented and is outdated.

AAOL Framework: This model-based approach is a constraint-based E/E architecture optimization framework utilizing a domain-specific language. The supporting tool uses the DSE method to find the optimal solution for deployment problem considering design constraints, e.g., memory capacity and applying multi-objective optimization mechanism, and the applied goals are cost and weight [81].

However, this tool has several limitations. It is limited in architectural elements regarding software and hardware level, e.g., it does not define any software application parameter. It covers no DSE for mapping problem in multi-core computing units, and it supports no model analysis and checking. In addition, it only takes ASIL level as a safety-related parameter into account for architecture synthesis, and it only covers a few optimization objectives. This tool is also outdated and not well documented.

Assist: Aviation electronics (avionics) are sophisticated and distributed systems aboard an airplane. The complexity of these systems is continuously growing as an increasing number of functionalities are realized in software. Using multi-core processors allows multiple functions in one hardware while meeting all safety requirements, which results in performance improvement of the system which can be a significant breakthrough in the aviation industry. A model-based approach was introduced in [82], namely Assist, to solve and optimize mapping problems (i.e., deployment of a safety-critical application on the avionics hardware) for distributed systems in the aviation domain. It uses the DSE mechanism to find optimal mapping solutions while considering optimizations objectives, including resource usage, weight, and cost. In addition, this Eclipse-based tool can create a periodic schedule for real-time tasks and ensure a deterministic timing behavior [83].

This framework, however, has some limitations. The specified architectural elements are extremely limited in terms of hardware and software level, and also its focus is on the aviation domain rather than automotive and E/E architecture. Moreover, it only supports redundancy as a safety-relevant attributes based on ISO 26262 for mapping problem. Additionally, Assist does not support model checking and model analysis, and the number of defined optimization goals is limited.

Deepcompass Framework: Designing embedded systems for multiprocessor platforms requires early prediction and the balancing of multiple system quality attributes. The authors of [84] present a DSE framework for component-based software systems that allows an architect to gain insight into a space of possible design alternatives with further evaluation and comparison of these alternatives. This framework supports the design of multiple alternatives for software and hardware architectures, and it is capable of performing model analysis, mapping analysis, and model validation. Furthermore, it uses the DSE approach for finding the suitable SW/HW architecture alternatives while taking multiple optimization goals into account, comprising cost, throughput, and resource utilization.

Nonetheless, it covers no automotive-related elements and multi-core computing units attributes. In addition, there is no DSE for the mapping problem, and it does not include any safety-related attributes. The specified optimization goals are also extremely limited. This open-source tool is outdated and is not well documented.

SCALL: This is a prototype tool that uses an allocation method to provide deployment solutions for system architects in the design phase using DSE [85]. In addition, it supports multi-objective DSE, including heterogeneous component allocation, bandwidth, and communication cost, by utilizing heuristics and AHP approaches to support systems architects in complex allocation decisions in early design stages.

However, this approach does not consider model analysis, model checking, or mapping analysis. There is no coverage regarding DSE for mapping problem in multi-core computing units and no safety-relevant constraints and requirements, and the automotive architectural elements have not been defined in this framework. Moreover, it does no optimization and is an outdated tool.

AQOSA: The authors in [86] introduce AQOSA (Automated Quality-driven Optimization of Software Architecture) toolkit to facilitate the design of software architecture in advanced component-based software development. It integrates modeling technologies, performance analysis techniques, and advanced evolutionary multi-objective optimization algorithms to improve non-functional properties of systems in an automated manner. It enables the modeling of software components and the mapping of feature configurations to component configurations or model templates. Additionally, it supports the DSE method, including multi-objective optimization (such as data flow latency, processor usage, and architecture cost), and the architect can easily design the initial architecture in OSATE [64] utilizing AADL language [65] and then import it into AQOSA framework. For software architecture modeling, AQOSA integrates ROBOCOP [87] (Robust Open Component-Based Software Architecture for Configurable Devices Project) modeling language (see Figure 8).

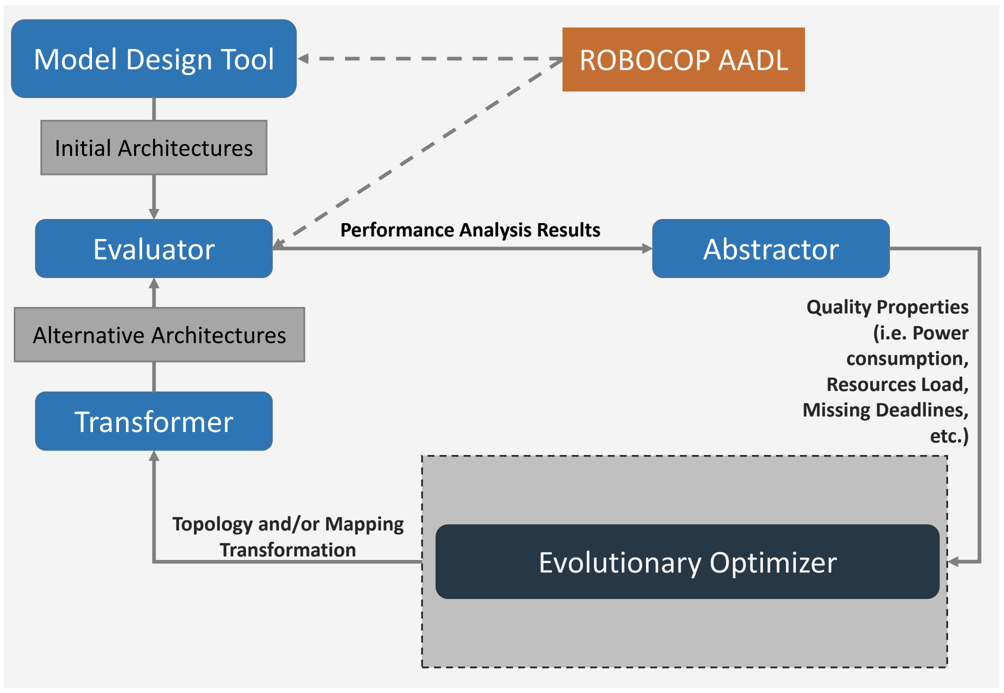

Model Design Tool
ROBOCOP AADL
Initial Architectures
Evaluator
Performance Analysis Results
Abstractor
Alternative Architectures
Quality Properties (i.e. Power consumption, Resources Load, Missing Deadlines, etc.)
Transformer
Topology and/or Mapping Transformation
Evolutionary Optimizer

Figure 8. The working scheme of the AQOSA toolkit.

AQOSA itself involves a few architectural elements; moreover, it does not examine mapping analysis and DSE for mapping problems, nor does it pay attention to the multi-core platforms as well. Furthermore, this tool provides no model checking functionality and comprises restricted optimization targets. This framework has no proper documentation and is outdated.

SQuAT-Vis: This is a tool that can be plugged into software architecture optimization approaches and allows architects to investigate results [88] to have an optimal software architecture satisfying quality-attribute requirements. This tool uses the DSE approach, including the optimization to explore the optimal design solution for the software clustering problem.

However, this framework does not include mapping problem as a DSE problem, model checking, or mapping analysis for multi-core processing units. In addition, it has limited optimization goals comprising response time and CPU utilization, and it supports no safety-related attributes such as exploration parameters during software clustering.

Several other research approaches have been illustrated about software integration and architecture synthesis related to the automotive domain in recent years. For example, the author in [89], proposed optimized reconfiguration of the industrial automation systems. The author used the DSE approach to compute optimal architectural configurations of control applications through specifying constraints and optimization goals. Similarly, a framework was presented in [27] to provide architecture modeling for automotive systems. The authors claimed that their framework facilitates system integration while utilizing optimization approaches. In addition, it considers validation for different design metrics including reliability and timing.

In the following tables, the studied problem, the DSE type, optimization algorithms and attributes, and safety-relevant attributes regarding each of the aforementioned open-source technologies are presented.

Table 1 presents the problem type (e.g., mapping, deployment, model checking, model analysis, etc.) that each technology tried to solve. In addition, it goes through problem attributes or DSE items such as resource usage, scheduling, task response time and, finally, it illustrates which DSE method was utilized in each of them. As explained before, there are various optimization attributes for designing automotive-related embedded systems, of which the most important ones are considered in Table 2. Table 2 presents the coverage result of the optimization parameters by the various approaches discussed above as well as expressing the used optimization algorithms, e.g., genetic algorithm. In addition, as satisfying safety requirements during automotive software configuration is extremely critical, Table 2 also covers the safety-related attributes considered by each approach, including ASIL level, reliability, FFI, redundancy, etc. Reliability, as mentioned before, is an optimization parameter and a safety-relevant element in the embedded system, calculated based on failure rate, to ensure the system reliability [50]. Cost is interpreted as the design expenses in such a way that the number of used components in the system can be decreased. Latency and execution time are parameters that improve the system performance, e.g., reducing task latency during task allocation. Energy consumption, as explained in Section 3.1.3, is another major optimization parameter in embedded systems. To utilize the CPU, and memory of the system in an optimized way, CPU utilization and memory usage are studied, and these two parameters contribute to the improvement of the system performance.

Table 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

Table 1. *Cont.*

E/E Configurator	Problem	Problem Attributes	Design Space Exploration
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
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

Table 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
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

Table 2. *Cont*.

E/E Configurator	Optimization Algorithms	Safety-Related Attributes	Optimization Attributes
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, 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

Furthermore, there are several commercial tools, developed by various companies, which contribute to E/E architecture design and automotive software integration and configuration [90]. The most relevant are explained in the following.

PreeVision: A commercial tool for model-based development of distributed, embedded systems in the automotive industry. It offers comprehensive functions for classic and service-oriented architecture construction and all aspects of an E/E system, including requirements engineering, AUTOSAR, software and communication design, and wiring harness evolution [91]. The integrated and model-based approach helps complex tasks to remain straightforward and controllable. It also supports the tried-and-tested system engineering principles of abstraction, decomposition, and reuse, and can serve as the engineering backbone. It enables parallel work on a shared database from multiple locations [92]. It supports the design of E/E architecture platforms used for different vehicles [93]; moreover, it provides design and evaluation of components, signal routing, model consistency checks, and functional safety analysis.

MotionWise: By using this commercial platform users can integrate, test, validate, and schedule any number of components and applications, helping them to reduce development, testing, and validation complexity and to ensure that essential safety and mission-criticality requirements can be met in both single or multi-SoC environments for automotive-related platforms. This tool abstracts the hardware and OS, creating a unified management environment out of heterogeneous elements [94].

Volcano Vehicle Systems Architect (VSA): This is a commercial Eclipse-based platform enabling generation of design environment for E/E systems [95,96]. It supports

SW architecture design, including defined SW components and compositions and HW architecture design, i.e., defining ECUs, networks, sensors, and actuators. VSA covers the full AUTOSAR metamodel and its formats, and it has the capability of automatic code generation [91]. It includes mapping analysis (connecting software components with ECUs and system signals), topology and communication design, and model validation. Moreover, it supports the bi-directional exchange of AUTOSAR XML files for software components and compositions with MATLAB and Simulink [68].

ASCET-DEVELOPER: This is a commercial model-based software targeting the automotive domain which is built on an Eclipse platform [97]. It assists system architects in creating high-performance, safe and secure embedded software with low overheads. As it has safety certifications such as ISO26262 ASIL-D, it will be appropriate for safety-critical software development [2]. The model analysis, including graphical and textual specifications and model validation, is supported by this commercial framework. Furthermore, it supports automatic C code generation from a designed model and provides unit test capability; moreover, toolchain integration can be supported in such a way that providing different interfaces and a standardized file exchange format makes it easy to integrate the tool into a development process and toolchain.

Autosar Builder: Another commercial tool for the design, configuration, and simulation of E/E systems following Autosar standards [98,99], it is built on the Eclipse platform with the Autosar development environment (Artop). Autosar Builder framework supports the development, verification, and validation of E/E components and the corresponding embedded software in the automotive field, and also system descriptions on the application level. In addition, it includes graphic visualizations and diagrams to make it easier to develop complex architectures. It also provides the possibility of simple integration with third-party tools.

SymTA/S: This is a commercial framework for analyzing the performance and optimizing the real-time embedded systems supporting heterogeneous architectures. SymTA/S is utilized for budgeting, scheduling verification, and optimization for processors, ECUs, communication buses, and networks. Timing and scheduling analysis of distributed embedded architectures is supported by this tool such as calculation of worst-case execution time (WCET). It enables unique end-to-end timing analysis and visualization; furthermore, it can plan and optimize the designed system by defining multi optimization objectives and its integration concepts and determine its reliability and safety while using DSE approach [100,101].

ChronSIM/ChronVAL: This is a tool for timing analysis of automotive systems. This commercial framework makes use of formal verification methods to analyze the real-time capability of safety-critical embedded systems. In addition, end-to-end analysis of distributed functions can be performed using the DSE approach. It provides graphical validation and simulation of the systems' timing requirements; it also supports multi-objective optimization such as resource utilization and response time objectives [102,103].

Although the above-described commercial frameworks support various features regarding E/E architecture synthesis, they have some limitations too. None of these platforms include mapping analysis (except VSA and ASCET-DEVELOPER), solving the mapping problem utilizing the DSE method for multi-core computing units, and considering the safety-relevant attributes based on ISO 26262. Additionally, model checking capability is not covered in the above-mentioned frameworks except for the PreeVision, VSA, and ASCET-DEVELOPER. Only three of them provide optimization for their synthesis results, such as ASCET-DEVELOPER, SymTA/S, and ChronSIM/ChronVAL, although they comprise a limited number of objectives. None of the platforms consider a wide range of architectural elements in terms of hardware and software level in E/E architecture configuration.

4. Future Research Areas

There is a demand to simplify the integration process for automotive OEMs and third-party developers. In other words, simplifying the offline software integration process by semi-automating and automating the integration of the safety-critical applications in next-generation automotive central compute platforms. As illustrated in Section 3, designing the configurations and mappings for the HPCUs, while fulfilling all safety requirements and considering optimization goals, is a challenging and time-consuming task. Furthermore, as the number of applications and their requirements increase, exploring a mapping solution becomes NP.

Current manual and offline integration processes are not ready for the needs of next-generation automotive HPCUs. As studied in this paper, there are various frameworks and approaches which contribute to semi-automate the integration and mapping process for embedded systems and HPCUs. However, none of these studies or platforms fully automate the integration and mapping process for multi-core computing units. In addition, there are a significant number of safety requirements based on automotive safety standards which have not been discussed in the previously mentioned studies. Moreover, as illustrated in Figure 3, deploying the mapping solution on an HPCU plays a critical role in evaluating the performance of the solution in run-time in terms of predefined optimization objectives in design-time and study the run-time behavior of the OS and applications to improve the mapping outcome created by the design-time framework. More importantly, verifying all constraints and requirements, which are defined in the design-time approach, after solution deployment on the HW in the run-time is another major criterion to ensure the functionality of the created configuration solution in the running phase.

After the DSE method is utilized to find the optimal mapping solution, there is a probability that the user receives an infeasible solution as the output of the solver due to conflicts in the constraints system. Therefore, proposing an approach to provide a capability to identify the source of the violation in the constraints system of the whole system model is extremely important for determining the feasibility of the solution. Otherwise, when the number of constraints increases, finding the violated constraints will be extremely challenging and time-consuming without any clues from which constraint equation they stem. None of the above-discussed frameworks or studies support this capability. Thus, we state the following research questions for the E/E architecture synthesis and the automotive software integration for HPCUs focused on the mapping based on the above-noted explanation.

- How to facilitate and automate the assignment of HPCU resources to safety-critical applications while verifying the satisfaction of the specified safety requirements in the design phase to compute a verified and optimized mapping configuration considering the predetermined optimization objectives?
- How to verify the fulfillment of the specified requirements (particularly safety-critical ones) after deployment of the derived configuration to the HPCU at run-time?
- How to evaluate the performance of the calculated mapping configuration at run-time focused on the specified optimization goals in design-time?
- How to discover the source of the conflict among defined constraints in our system while using the DSE approach to find the optimal solution in case of an infeasible solution?

5. Conclusions

In this paper, we first presented the evolution of car E/E architecture during the last few years and illustrated the current bottlenecks of E/E architecture as well as the major technologies for the future of vehicle architecture comprising software architecture of the high-performance computing unit (HPCU). Moreover, we expressed the challenges and technologies related to automotive software integration and deployment to the HPCU, including task mapping, software frameworks, and approaches for software configuration in the design process.

The current approaches and techniques for static and dynamic task mapping in multi-core processors using the design space exploration (DSE) method were discussed. In addition, the current optimization parameters in task mapping to boost the quality of the task assignment were identified. We went through the current existing technologies and approaches, dividing them into open-source and commercial, relevant to modeling, model analysis and checking, and solving the mapping problem, including the optimization goals for vehicle E/E architecture and multi-core processors. We depicted the strengths and limitations of each framework while we compared the open-source technologies by providing two tables covering various attributes such as the problem attribute, DSE method, safety-related attributes, optimization parameters, etc., which have been covered by each. Finally, we proposed our research questions as future research areas based on our analysis and study focused on the mapping of the software elements to automotive HPCUs which can be utilized by other researchers. There are several other topics, such as description languages and verification platforms for the synthesis of the E/E systems, which are relevant to the content of this paper; however, they are out of the scope of this paper.

Author Contributions: Conceptualization, H.A., M.H.F. and A.K.; methodology, H.A.; software, H.A.; validation, H.A.; formal analysis, H.A., M.H.F. and A.K.; investigation, H.A.; resources, H.A.; data curation, H.A.; writing—original draft preparation, H.A.; writing—review and editing, H.A. and M.H.F.; visualization, H.A.; supervision, H.A., M.H.F. and A.K.; project administration, H.A. and A.K.; funding acquisition, A.K. and H.A. All authors have read and agreed to the published version of the manuscript.

Funding: This work is part of the project, namely KI-FLEX, (project number 16ES1027) which is funded by German Federal Ministry of Education and Research (BMBF) within the framework of the guidelines on promoting research initiatives in the field of AI-based electronic solutions for safe autonomous driving.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. ISO. Safety Of The Intended Functionality (SOTIF). 2019. Available online: https://www.iso.org/standard/70939.html (accessed on 16 September 2021).
2. ISO 26262-1:2018. 2018. Available online: https://www.iso.org/standard/68383.html (accessed on 30 August 2021).
3. Freddie Holmes, J.H.; Moreton, J. Zonal E/E Architectures the Cornerstone of Future Mobility Development. 2021. Available online: https://www.automotiveworld.com/articles/zonal-e-e-architectures-the-cornerstone-of-future-mobility-development (accessed on 30 September 2021).
4. Broy, M. Challenges in automotive software engineering. In Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, 20–28 May 2006; pp. 33–42.
5. Askaripoor, H.; Farzaneh, M.H.; Knoll, A. A Model-Based Approach to Facilitate Design of Homogeneous Redundant E/E Architectures. In Proceedings of the 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), Indianapolis, IN, USA, 19–22 September 2021; pp. 3426–3431. [CrossRef]
6. Bandur, V.; Selim, G.; Pantelic, V.; Lawford, M. Making the Case for Centralized Automotive E/E Architectures. *IEEE Trans. Veh. Technol.* **2021**, *70*, 1230–1245. [CrossRef]
7. Askaripoor, H.; Farzaneh, M.H.; Knoll, A. A Platform to Configure and Monitor Safety-Critical Applications for Automotive Central Computers. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–4. [CrossRef]
8. Apostu, S.; Burkacky, O.; Deichmann, J.; Doll, G. Automotive Software and Electrical/Electronic Architecture: Implications for OEMs. 2019. Available online: https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/automotive-software-and-electrical-electronic-architecture-implications-for-oems (accessed on 10 October 2021).
9. Butzkamm, C.; Brand, K. E/E Architecture in the HARRI Innovation Platform. *Atzelectroni. Worldw.* **2020**, *15*, 18–24. [CrossRef]
10. Zerfowski, D.; Lock, A. *Functional Architecture and E/E-Architecture–A Challenge for the Automotive Industry*; Internationales Stuttgarter Symposium; Springer: Berlin/Heidelberg, Germany, 2019; pp. 909–920.
11. Jiang, S. *Vehicle e/e Architecture and Its Adaptation to New Technical Trends*; Technical report; SAE Technical Paper: Warrendale, PA, USA, 2019.
12. Shavit, M.; Gryc, A.; Miucic, R. *Firmware Update over the Air (FOTA) for Automotive Industry*; Technical report; SAE Technical Paper: Warrendale, PA, USA, 2007.
13. Wang, J.; Liu, J.; Kato, N. Networking and communications in autonomous driving: A survey. *IEEE Commun. Surv. Tutor.* **2018**, *21*, 1243–1274. [CrossRef]
14. Singer, S. The Car Will Become a Data Center. *Atzelectron. Worldw.* **2020**, *15*, 8–13. [CrossRef]
15. Navale, V.M.; Williams, K.; Lagospiris, A.; Schaffert, M.; Schweiker, M.A. (R) evolution of E/E architectures. *SAE Int. J. Passeng.-Cars-Electron. Electr. Syst.* **2015**, *8*, 282–288. [CrossRef]

16. Sommer, S.; Camek, A.; Becker, K.; Buckl, C.; Zirkler, A.; Fiege, L.; Armbruster, M.; Spiegelberg, G.; Knoll, A. RACE: A Centralized Platform Computer Based Architecture for Automotive Applications. In Proceedings of the 2013 IEEE International Electric Vehicle Conference (IEVC), Santa Clara, CA, USA, 23–25 October 2013; pp. 1–6. [CrossRef]
17. Sangiovanni-Vincentelli, A.; Di Natale, M. Embedded System Design for Automotive Applications. *Computer* **2007**, *40*, 42–51. [CrossRef]
18. Brunner, S.; Roder, J.; Kucera, M.; Waas, T. Automotive E/E-architecture enhancements by usage of ethernet TSN. In Proceedings of the 2017 13th Workshop on Intelligent Solutions in Embedded Systems (WISES), Hamburg, Germany, 12–13 June 2017; pp. 9–13.
19. Shapiro, D. The Future of Automotive is Software-defined. *Atzelectron. Worldw.* **2020**, *15*, 72–72. [CrossRef]
20. Knoll, A.; Buckl, C.; Kuhn, K.J.; Spiegelberg, G. The RACE Project: An Informatics-Driven Greenfield Approach to Future E/E Architectures for Cars. In *Automotive Systems and Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2019; pp. 171–195.
21. Dall, C.; Nieh, J. KVM/ARM: the design and implementation of the linux ARM hypervisor. *ACM Sigplan Not.* **2014**, *49*, 333–348. [CrossRef]
22. Desai, A.; Oza, R.; Sharma, P.; Patel, B. Hypervisor: A survey on concepts and taxonomy. *Int. J. Innov. Technol. Explor. Eng.* **2013**, *2*, 222–225.
23. Khan, A.; Zugenmaier, A.; Jurca, D.; Kellerer, W. Network virtualization: a hypervisor for the Internet? *IEEE Commun. Mag.* **2012**, *50*, 136–143. [CrossRef]
24. Pelliccione, P.; Knauss, E.; Heldal, R.; Ågren, S.M.; Mallozzi, P.; Alminger, A.; Borgentun, D. Automotive architecture framework: The experience of volvo cars. *J. Syst. Archit.* **2017**, *77*, 83–100. [CrossRef]
25. Askaripoor, H.; Shafaei, S.; Knoll, A.C. A Flexible Scheduling Architecture of Resource Distribution Proposal for Autonomous Driving Platforms. In Proceedings of the VEHITS, Online Streaming, 28–30 April 2021; pp. 594–599.
26. Askaripoor, H.; Farzaneh, M.H.; Knoll, A. Considering Safety Requirements in Design Phase of Future E/E Architectures. In Proceedings of the 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vienna, Austria, 8–11 September 2020; Volume 1, pp. 1165–1168. [CrossRef]
27. Zheng, B.; Liang, H.; Zhu, Q.; Yu, H.; Lin, C.W. Next generation automotive architecture modeling and exploration for autonomous driving. In Proceedings of the 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Pittsburgh, PA, USA, 11–13 July 2016; pp. 53–58.
28. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Resource and Throughput Aware Execution Trace Analysis for Efficient Run-Time Mapping on MPSoCs. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2016**, *35*, 72–85. [CrossRef]
29. Sahu, P.K.; Chattopadhyay, S. A survey on application mapping strategies for network-on-chip design. *J. Syst. Archit.* **2013**, *59*, 60–76. [CrossRef]
30. Mehrara, M.; Jablin, T.; Upton, D.; August, D.; Hazelwood, K.; Mahlke, S. Multicore compilation strategies and challenges. *IEEE Signal Process. Mag.* **2009**, *26*, 55–63. [CrossRef]
31. Xie, G.; Zeng, G.; Liu, Y.; Zhou, J.; Li, R.; Li, K. Fast functional safety verification for distributed automotive applications during early design phase. *IEEE Trans. Ind. Electron.* **2017**, *65*, 4378–4391. [CrossRef]
32. Xie, G.; Li, Y.; Han, Y.; Xie, Y.; Zeng, G.; Li, R. Recent advances and future trends for automotive functional safety design methodologies. *IEEE Trans. Ind. Informatics* **2020**, *16*, 5629–5642. [CrossRef]
33. Gupta, M.; Bhargava, L.; Indu, S. Mapping techniques in multicore processors: current and future trends. *J. Supercomput.* **2021**, *77*, 9308–9363. [CrossRef]
34. Singh, A.K.; Shafique, M.; Kumar, A.; Henkel, J. Mapping on multi/many-core systems: survey of current and emerging trends. In Proceedings of the 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), Austin, TX, USA, 29 May–7 June 2013; pp. 1–10.
35. Deveci, M.; Kaya, K.; Uçar, B.; Çatalyürek, Ü.V. Hypergraph partitioning for multiple communication cost metrics: Model and methods. *J. Parallel Distrib. Comput.* **2015**, *77*, 69–83. [CrossRef]
36. 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 Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, NM, USA, 26–30 April 2004; p. 110.
37. Bokhari, S.H. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. Softw. Eng.* **1981**, *SE-7*, 583–589. [CrossRef]
38. 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. *J. Parallel Distrib. Comput.* **2001**, *61*, 810–837. [CrossRef]
39. Castrillon, J.; Tretter, A.; Leupers, R.; Ascheid, G. Communication-aware mapping of KPN applications onto heterogeneous MPSoCs. In Proceedings of the DAC Design Automation Conference, San Francisco, CA, USA, 3–7 June 2012; pp. 1262–1267.
40. Niemann, R.; Marwedel, P. An algorithm for hardware/software partitioning using mixed integer linear programming. *Des. Autom. Embed. Syst.* **1997**, *2*, 165–193. [CrossRef]
41. Hu, J.; Marculescu, R. Energy-and performance-aware mapping for regular NoC architectures. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2005**, *24*, 551–562.

42. Bhatti, Z.W.; Miniskar, N.R.; Preuveneers, D.; Wuyts, R.; Berbers, Y.; Catthoor, F. Memory and communication driven spatio-temporal scheduling on MPSoCs. In Proceedings of the 2012 25th Symposium on Integrated Circuits and Systems Design (SBCCI), Brasilia, Brazil, 30 August–2 September 2012; pp. 1–6.
43. Kaida, J.; Hieda, T.; Taniguchi, I.; Tomiyama, H.; Hara-Azumi, Y.; Inoue, K. Task mapping techniques for embedded many-core socs. In Proceedings of the 2012 International SoC Design Conference (ISOCC), Jeju Island, Korea, 4–7 November 2012; pp. 204–207.
44. Pascual, J.A.; Miguel-Alonso, J.; Lozano, J.A. Optimization-based mapping framework for parallel applications. *J. Parallel Distrib. Comput.* **2011**, *71*, 1377–1387. [CrossRef]
45. Giannopoulou, G.; Stoimenov, N.; Huang, P.; Thiele, L. Mapping mixed-criticality applications on multi-core architectures. In Proceedings of the 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6.
46. Gan, Z.; Zhang, M.; Gu, Z.; Zhang, J. Minimizing energy consumption for embedded multicore systems using cache configuration and task mapping. In Proceedings of the 2016 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Chengdu, China, 13–15 October 2016; pp. 328–334.
47. Ferrandi, F.; Lanzi, P.L.; Pilato, C.; Sciuto, D.; Tumeo, A. Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems. *IEEE Trans. -Comput.-Aided Des. Integr. Circuits Syst.* **2010**, *29*, 911–924. [CrossRef]
48. Xu, X.X.; Hu, X.M.; Chen, W.N.; Li, Y. Set-based particle swarm optimization for mapping and scheduling tasks on heterogeneous embedded systems. In Proceedings of the 2016 Eighth International Conference on Advanced Computational Intelligence (ICACI), Chiang Mai, Thailand, 14–16 February 2016; pp. 318–325.
49. Carvalho, E.; Calazans, N.; Moraes, F. Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs. In Proceedings of the 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07), Porto Alegre, Brazil, 28–30 May 2007; pp. 34–40.
50. Xie, G.; Peng, H.; Li, Z.; Song, J.; Xie, Y.; Li, R.; Li, K. Reliability enhancement toward functional safety goal assurance in energy-aware automotive cyber-physical systems. *IEEE Trans. Ind. Inf.* **2018**, *14*, 5447–5462. [CrossRef]
51. Peress, Y. Multi-core Design and Memory Feature Selection Survey. Available online: https://ww2.cs.fsu.edu/~peress/publications/AreaSurvey.pdf (accessed on 30 August 2021).
52. Das, A.; Kumar, A.; Veeravalli, B. Communication and migration energy aware task mapping for reliable multiprocessor systems. *Future Gener. Comput. Syst.* **2014**, *30*, 216–228. [CrossRef]
53. Huang, L.; Yuan, F.; Xu, Q. Lifetime reliability-aware task allocation and scheduling for MPSoC platforms. In Proceedings of the 2009 Design, Automation & Test in Europe Conference & Exhibition, Nice, France, 20–24 April 2009; pp. 51–56.
54. Ding, H.; Liang, Y.; Mitra, T. Shared cache aware task mapping for WCRT minimization. In Proceedings of the 2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC), Yokohama, Japan, 22–25 January 2013; pp. 735–740.
55. Coskun, A.K.; Rosing, T.S.; Whisnant, K.A.; Gross, K.C. Temperature-aware MPSoC scheduling for reducing hot spots and gradients. In Proceedings of the 2008 Asia and South Pacific Design Automation Conference, Seoul, Korea, 21–24 March 2008; pp. 49–54.
56. Girault, A.; Zarandi, H.R. Erpot: A quad-criteria scheduling heuristic to optimize execution time, reliability, power consumption and temperature in multicores. *IEEE Trans. Parallel Distrib. Syst.* **2019**, *30*, 2193–2210.
57. Hartman, A.S.; Thomas, D.E.; 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, Scottsdale, AZ, USA, 24–29 October 2010; pp. 145–154.
58. Das, A.; Kumar, A.; Veeravalli, B.; Bolchini, C.; Miele, A. Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In Proceedings of the 2014 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6.
59. Kinsy, M.A.; Devadas, S. Algorithms for scheduling task-based applications onto heterogeneous many-core architectures. In Proceedings of the 2014 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA USA, 9–11 September 2014; pp. 1–6.
60. Lee, S.; Ro, W.W. Workload and variation aware thread scheduling for heterogeneous multi-processor. In Proceedings of the 18th IEEE International Symposium on Consumer Electronics (ISCE 2014), Jeju, Korea, 22–25 June 2014; pp. 1–2.
61. Liu, G.; Park, J.; Marculescu, D. Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In Proceedings of the 2013 IEEE 31st International Conference on Computer Design (ICCD), Asheville, NC, USA, 6–9 October 2013; pp. 54–61.
62. Bolchini, C.; Carminati, M.; Mitra, T.; Muthukaruppan, T.S. Combined on-line lifetime-energy optimization for asymmetric multicores. In Proceedings of the 2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), Storrs, CT, USA, 19–20 September 2016; pp. 35–40.
63. Feiler, P. *The Open Source AADL Tool Environment (OSATE)*; Technical report; Carnegie Mellon University Software Engineering Institute. 2019. Available online: https://resources.sei.cmu.edu/asset_files/Presentation/2019_017_001_635851.pdf (accessed on 30 August 2021).
64. Welcome to OSATE. 2021. Available online: https://osate.org/ (accessed on 11 November 2021).

65. Feiler, P.H.; Gluch, D.P.; Hudak, J.J. *The Architecture Analysis & Design Language (AADL): An Introduction*; Technical report; Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst: Pittsburgh, PA, USA, 2006.
66. Introduction to Ocarina Plugin. 2021. Available online: https://ocarina.readthedocs.io/en/latest/introduction.html (accessed on 10 November 2021).
67. Hugues, J.; Zalila, B.; Pautet, L.; Kordon, F. From the prototype to the final embedded system using the Ocarina AADL tool suite. *ACM Trans. Embed. Comput. Syst. (TECS)* **2008**, *7*, 1–25. [CrossRef]
68. MATLAB. *Version 7.10.0 (R2010a)*; The MathWorks Inc.: Natick, MA, USA, 2010.
69. Aleti, A.; Bjornander, S.; Grunske, L.; Meedeniya, I. ArcheOpterix: An extendable tool for architecture optimization of AADL models. In Proceedings of the 2009 ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Vancouver, BC, USA, 16 May 2009; pp. 61–71.
70. Meedeniya, I.; Buhnova, B.; Aleti, A.; Grunske, L. Reliability-driven deployment optimization for embedded systems. *J. Syst. Softw.* **2011**, *84*, 835–846. [CrossRef]
71. Koziolek, A.; Koziolek, H.; 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–ISARCS, Boulder, CO, USA, 20–24 June 2011; pp. 33–42.
72. Behrmann, G.; David, A.; Larsen, K.G.; Håkansson, J.; Pettersson, P.; Yi, W.; Hendriks, M. Uppaal 4.0. In Proceedings of the Quantitative Evaluation of SysTems (QEST), University of California, Riverside, CA, USA, 11–14 September 2006.
73. Burmester, S.; Giese, H.; Tichy, M. Model-driven development of reconfigurable mechatronic systems with mechatronic UML. In *Model Driven Architecture*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 47–61.
74. Fritzson, P.; Engelson, V. Modelica—A unified object-oriented language for system modeling and simulation. In *European Conference on Object-Oriented Programming*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 67–90.
75. Schäfer, W.; Wehrheim, H. Model-driven development with mechatronic uml. In *Graph Transformations and Model-Driven Engineering*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 533–554.
76. Höttger, R.; Krawczyk, L.; Igel, B. Model-based automotive partitioning and mapping for embedded multicore systems. International Conference on Parallel, Distributed Systems and Software Engineering. *Citeseer* **2015**, *2*, 888.
77. Höttger, R.; Mackamul, H.; Sailer, A.; Steghöfer, J.P.; Tessmer, J. APP4MC: Application platform project for multi-and many-core systems. *Inf. Technol.* **2017**, *59*, 243–251. [CrossRef]
78. Aravantinos, V.; Voss, S.; Teufl, S.; Hölzl, F.; Schätz, B. AutoFOCUS 3: Tooling Concepts for Seamless, Model-based Development of Embedded Systems. *ACES-MB&WUCOR@ MoDELS* **2015**, *1508*, 19–26.
79. Voss, S.; Eder, J.; Hölzl, F. Design Space Exploration and its Visualization in AUTOFOCUS3. Available online: http://ceur-ws.org/Vol-1129/paper33.pdf (accessed on 30 August 2021).
80. Juodisius, P.; Sarkar, A.; Mukkamala, R.R.; Antkiewicz, M.; Czarnecki, K.; Wasowski, A. Clafer: Lightweight modeling of structure, behaviour, and variability. *arXiv* **2018**, arXiv:1807.08576.
81. Kugele, S.; Pucea, G.; Popa, R.; Dieudonné, L.; Eckardt, H. On the deployment problem of embedded systems. In Proceedings of the 2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), Austin, TX, USA, 21–23 September 2015; pp. 158–167.
82. Hilbrich, R.; Dieudonné, L. Deploying Safety-Critical Applications on Complex Avionics Hardware Architectures. Available online: https://www.scirp.org/html/1-9301646_31297.htm?pagespeed=noscript (accessed on 10 October 2021).
83. Hilbrich, R.; Behrisch, M. Experiences gained from modeling and solving large mapping problems during system design. In Proceedings of the 2017 Annual IEEE International Systems Conference (SysCon), Montreal, QC, Canada, 24–27 April 2017; pp. 1–8.
84. Bondarev, E.; Chaudron, M.R.; de Kock, E.A. Exploring performance trade-offs of a JPEG decoder using the DeepCompass framework. In Proceedings of the 6th International Workshop on Software and Performance, Buenes Aires, Argentina, 5–8 February 2007; pp. 153–163.
85. Švogor, I.; Carlson, J. SCALL: Software component allocator for heterogeneous embedded systems. In Proceedings of the 2015 European Conference on Software Architecture Workshops, Dubrovnik/Cavtat, Croatia, 7–11 September 2015; pp. 1–5.
86. Li, R.; Etemaadi, R.; Emmerich, M.T.; Chaudron, M.R. An evolutionary multiobjective optimization approach to component-based software architecture design. In Proceedings of the 2011 IEEE Congress of Evolutionary Computation (CEC), New Orleans, LA, USA, 5–8 June 2011; pp. 432–439.
87. Bondarev, E.; Chaudron, M. A process for resolving performance trade-offs in component-based architectures. In *International Symposium on Component-Based Software Engineering*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 254–269.
88. Frank, S.; van Hoorn, A. SQuAT-Vis: Visualization and Interaction in Software Architecture Optimization. In *European Conference on Software Architecture*; Springer: Berlin/Heidelberg, Germany, 2020; pp. 107–119.
89. 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, Copenhagen, Denmark, 14–19 October 2018; pp. 180–185.
90. Waszecki, P.; Lukasiewycz, M.; Masrur, A.; Chakraborty, S. How to engineer tool-chains for automotive e/e architectures? *ACM Sigbed Rev.* **2013**, *10*, 6–15. [CrossRef]

91. Fürst, S.; Mössinger, J.; Bunzel, S.; Weber, T.; Kirschke-Biller, F.; Heitkämper, P.; Kinkelin, G.; Nishikawa, K.; Lange, K. AUTOSAR–A Worldwide Standard is on the Road. In Proceedings of the 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, Germany, 7–8 October 2009; Volume 62, p. 5.
92. Vector. PREEvision. 2021. Available online: https://www.vector.com/de/en/products/products-a-z/software/preevision/#c1789 (accessed on 30 September 2021).
93. Schäuffele, J. *E/e Architectural Design and Optimization using Preevision*; Technical report; SAE Technical Paper; SAE: Warrendale, PA, USA, 2016.
94. TTTechAuto. MotionWise. 2021. Available online: https://www.tttech-auto.com/products/safety-software-platform/motionwise (accessed on 20 October 2021).
95. The Community for Open Innovation and Collaboration: The Eclipse Foundation. Available online: http://www.eclipse.org/ (accessed on 29 September 2021).
96. Volcano Vehicle Systems Architect (VSA). Available online: https://www.mathworks.com/products/connections/product_detail/volcano-vehicle-systems-architect.html (accessed on 1 October 2021).
97. ASCET-DEVELOPER. Available online: https://www.etas.com/en/products/ascet-developer.php (accessed on 25 September 2021).
98. Simulation Toolset: Autosarbuilder—Dassault Systèmes. Available online: https://www.3ds.com/products-services/catia/products/autosar-builder/ (accessed on 9 October 2021).
99. Simulation Toolset: Autosarbuilder. Available online: https://www.3ds.com/fileadmin/Welcome_to_AUTOSAR_Builder_2020x.pdf (accessed on 10 October 2021).
100. Henia, R.; Hamann, A.; Jersak, M.; Racu, R.; Richter, K.; Ernst, R. System level performance analysis–the SymTA/S approach. *IEE-Proc.-Comput. Digit. Tech.* **2005**, *152*, 148–166. [CrossRef]
101. Hamann, A.; Henia, R.; Racu, R.; Jersak, M.; Richter, K.; Ernst, R. Symta/s-symbolic timing analysis for systems. In Proceedings of the WIP Proc. Euromicro Conference on Real-Time Systems 2004 (ECRTS'04). Citeseer, Catania, Italy, 30 June–2 July 2004; pp. 17–20.
102. Anssi, S.; Albers, K.; Dörfel, M.; Gérard, S. chronval/chronsim: A tool suite for timing verification of auto-motive applications. Embedded Real Time Software and Systems (ERTS2012). Available online: https://hal.archives-ouvertes.fr/hal-02191852/document (accessed on 20 December 2021).
103. ChronVALWorst-Case Timing Analysis. 2021. Available online: https://www.inchron.com/chronval/ (accessed on 10 October 2021).