

FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO



Embedded Runtime software analysis and optimization for automotive industry

Gonalo Marinho dos Santos

Mestrado em Engenharia Eletrot cnica e de Computadores

Supervisor: Daniel Gouveia Costa

October 21, 2023

Resumo

Com o progresso da tecnologia e o desenvolvimento de sistemas cada vez mais complexos, há uma procura crescente destes produtos para satisfazer uma gama alargada de requisitos. Estes aspetos são, sem surpresa, sensíveis em sistemas críticos de segurança, como os da indústria de *software* para automóveis.

A criação do *AUTomotive Open System ARchitecture* (AUTOSAR) foi um esforço de colaboração para resolver estas preocupações, estabelecendo uma norma industrial e mundial. Esta iniciativa visa facilitar o desenvolvimento de metodologias e soluções de *software* compatíveis na indústria automóvel. No entanto, apesar dos avanços que esta arquitetura trouxe aos engenheiros nos últimos anos, há espaço para melhorias no que diz respeito à forma como se espera que os veículos e esta estrutura se comportem em cenários emergentes.

Neste contexto, esta dissertação fez um levantamento do estado da arte nesta área, focando-se na contextualização da norma AUTOSAR, da sua arquitetura e de conceitos essenciais. A revisão da literatura efetuada e os seus resultados, que situam a posição científica desta *framework*, serão explicados em pormenor. As contribuições científicas diretas, que analisaram os problemas de implementação mais comuns, as preocupações de segurança e os casos de utilização emergentes, culminaram num artigo científico aceite para publicação na *IEEE Xplore Digital Library*.

Além disso, foram discutidas e testadas várias implementações e soluções compatíveis com AUTOSAR. Este trabalho foi realizado na empresa *Continental Engineering Services* (CES) Porto com o apoio da sua equipa no âmbito de *Advanced Driver-Assistance Systems* (ADAS). Foram estudadas e trabalhadas várias configurações de *software* relativas à troca de comunicações de um sensor de câmara frontal. Houve um interesse particular da equipa do CES Porto em encontrar o método de leitura mais eficiente suportado pelo AUTOSAR. Neste sentido, foram aplicados vários procedimentos e ferramentas de última geração utilizados na indústria.

Em vez de descobrir uma única resposta, o estudo resultante produziu vários métodos de receção de sinais do sensor da câmara, compatíveis com AUTOSAR, que eram desconhecidos na CES Porto. Estas técnicas foram analisadas e comparadas consoante as suas características técnicas e o seu tempo de execução. Para este efeito, foram também criados métodos de teste automatizados que pudessem ser facilmente instituídos em ambiente empresarial por membros da CES Porto nos seus projetos. Todas estas contribuições tiveram como objetivo melhorar a eficiência da aquisição de dados e o desenvolvimento de soluções de *software*.

Palavras-Chave: ADAS, AUTOSAR, Sistemas Automóveis, ECU, Sistemas Embebidos, Arquitetura de *Software*.

Abstract

With the progression of technology and the development of evermore intricate systems, there is a rising demand for these products to satisfy an expanding range of requirements. These aspects are unsurprisingly sensitive in safety-critical systems such as those in the automotive-related software industry.

The creation of the AUTomotive Open System ARchitecture (AUTOSAR) was a collaborative effort to address these concerns by establishing an industry and worldwide standard. This initiative aims to facilitate the development of compliant methodologies and software solutions within the automotive industry. However, despite the advances this architecture has brought to engineers over recent years, there is room for improvements concerning how vehicles and this framework are expected to behave in emerging scenarios.

In this context, this dissertation surveyed the state-of-the-art in this area, focusing on contextualizing the AUTOSAR standard, its architecture, and essential concepts. The performed literature review and its results, which situate the scientific standing of this framework, will be thoroughly explained. The direct scientific contributions, which reviewed the most common implementation issues, security concerns, and emerging use cases, culminated in a scientific article that was accepted to be published in the *IEEE Xplore Digital Library*.

Furthermore, several AUTOSAR-compliant implementations and solutions were discussed and tested. This endeavor was done at Continental Engineering Services (CES) Porto company with support from its team in the context of Advanced Driver-Assistance Systems (ADAS). Several software configurations were studied and worked on concerning the communication exchange of a front camera sensor. There was a particular interest by the team at CES Porto in finding the most efficient reading method supported by AUTOSAR. Within this scope, several state-of-the-art procedures and tools used in the industry were employed.

Instead of discovering a sole answer, the resulting study yielded several AUTOSAR-compliant reception methods of signals from the camera sensor that were unknown to CES Porto. These techniques were analyzed and compared according to their technical characteristics and execution time. For this purpose, automated test methods were equally created that could be easily introduced in a corporate setting by members of CES Porto in their projects. All of these contributions aimed to improve the efficiency of data acquisition and the development of software solutions.

Keywords: ADAS, AUTOSAR, Automotive Systems, ECU, Embedded Systems, Software Architecture.

Acknowledgments

The completion of this dissertation was the culmination of a long journey in my higher education in the Electrical and Computer Engineering field. Hence, I thank those who supported me in this odyssey and encouraged me to overcome any obstacle.

A special thanks to my supervisor at the Faculdade de Engenharia da Universidade do Porto (FEUP), Professor Daniel Costa, who was always available to advise and guide me in this work. His unwavering support, motivation, and dedication contributed immensely to exceeding all expectations behind this dissertation and its success.

To the Continental Engineering Services (CES) Porto team for the atmosphere and friendliness with which they received me at their facilities and work environment. Especially to the Advanced Driver-Assistance Systems (ADAS) team members and my supervisors, Ricardo Esteves and Tiago Gomes, who, during this dissertation, as far as possible, made themselves available to help me, sometimes to the detriment of their personal work.

To my family, in particular, my parents and grandparents, whom I thank for providing me with this academic path. My brothers Henrique, Eduardo, Daniela, and all my friends, for the patience, understanding, and steadfast loyalty they have always shown, especially in my absences that this long trail required.

Gonalo Marinho dos Santos

“A true master is an eternal student”.

Master Yi

Contents

1	Introduction	1
1.1	Context	1
1.2	Goals	3
1.3	Structure	3
2	Background and Fundamental Aspects	5
2.1	Automotive Industry Pre-AUTOSAR	5
2.2	AUTOSAR <i>Classic Platform</i>	6
2.2.1	General Overview of the <i>Application Layer</i> and RTE	7
2.2.2	Components and Modules of the <i>BSW Layer</i>	9
2.3	AUTOSAR <i>Adaptive Platform</i>	13
3	Literature Review	15
3.1	Methodology	15
3.2	Development Issues	16
3.2.1	AUTOSAR Overheads	17
3.2.2	Multi-Core ECUs	18
3.2.3	<i>Intra-ECU Communication</i>	18
3.2.4	<i>Inter-ECU Communication</i>	20
3.3	Security	21
3.3.1	Cryptography	21
3.3.2	High-Layered Specifications	22
3.3.3	Safety Mechanisms	22
3.4	Use Cases	23
3.4.1	Advanced Driver-Assistance Systems	23
3.4.2	Vehicle-to-Vehicle and Vehicle-to-Infrastructure	24
3.5	Summarizing the Results	25
4	Hardware and Software Architecture	27
4.1	AUTOSAR Methodology	27
4.2	AUTOSAR Workflow	29
4.3	Workbench and Workstation	30
4.4	Control Unit Communication Stack Topology	30
5	Software Implementations	35
5.1	<i>Explicit</i> and <i>Implicit</i> Communication	35
5.2	ARXML Configurations	36
5.2.1	Rte_Read	37

5.2.2	Rte_IRead	40
5.2.3	Rte_DRead	41
5.2.4	Rte_Receive	43
5.2.5	Comparison Between “Signals” and “Signal Groups”	45
6	Examinations, Results, and Discussion	47
6.1	Test Environment	47
6.2	Execution Time Measurements	49
6.3	Discussion	51
7	Conclusions and Future Work	55
A	Reviewed “Research Works”	59
B	ARXML Configurations and Source Code	71
C	Script Files and Example of Extracted Data	73
	References	77

List of Figures

1.1	Benefits of the Usage of AUTOSAR	2
2.1	AUTOSAR Consortium	6
2.2	AUTOSAR Layered Software Architecture	6
2.3	AUTOSAR SWCs and Ports	7
2.4	VFB Model for a <i>Sender-Receiver</i> Communication Topology	8
2.5	AUTOSAR <i>Internal Behaviors</i> and <i>Runnables</i>	9
2.6	AUTOSAR Architecture with Expanded <i>BSW Layer</i>	10
2.7	AUTOSAR Architecture with Expanded <i>BSW Layer Stacks</i>	10
2.8	AUTOSAR Architecture with Expanded <i>Communication Stack</i>	12
2.9	Deployment of the <i>Classic Platform</i> and <i>Adaptive Platform</i>	13
3.1	Distribution of the Sources of the “Research Works”	16
3.2	Mapping of 1 <i>Sender</i> & 1 <i>Receiver</i>	19
3.3	Mapping of Multiple <i>Receivers</i>	19
3.4	Mapping of Multiple <i>Senders</i>	19
3.5	AUTOSAR Methodology	20
3.6	Camera used in ADAS Projects at CES Porto	23
3.7	Networking in Modern Cars	24
4.1	AUTOSAR Methodology for OEMs and <i>Tier 1 Suppliers</i>	28
4.2	Generic AUTOSAR Workflow for <i>Tier 1 Suppliers</i>	28
4.3	AUTOSAR Workflow and Toolkit	29
4.4	Architecture of the Workbench	30
4.5	Data Flow associated with Transmitted Signals	31
4.6	Software Architecture of the Modules within the “Control Unit” SWC	32
5.1	<i>Explicit</i> Communication	35
5.2	<i>Implicit</i> Communication	35
5.3	ARXML Hierarchy of the “SWC.arxml” file with <code>Rte_Read</code> API Function . . .	38
5.4	ARXML Hierarchy of the “Mapping.arxml” file with <code>SystemSignalGroup</code> . .	39
5.5	ARXML Hierarchy of the “Signals.arxml” file with <code>SystemSignalGroup</code> . . .	39
5.6	ARXML Hierarchy of the “SWC.arxml” file with <code>Rte_IRead</code> API Function . .	41
5.7	ARXML Hierarchy of the “Signals.arxml” file without <code>SystemSignalGroup</code> .	42
5.8	ARXML Hierarchy of the “SWC.arxml” file with <code>Rte_DRead</code> API Function . .	42
5.9	ARXML Hierarchy of the “Mapping.arxml” file with <code>SystemSignal</code>	43
5.10	ARXML Hierarchy of the “SWC.arxml” file with <code>Rte_Receive</code>	44
5.11	ARXML Hierarchy of the “Signals.arxml” file without <code>SystemSignalGroup</code> .	44

6.1	Envisioned Strategy for Data Gathering	47
6.2	RunTime Window	48
6.3	Measured Execution Times in Run-Time	49
6.4	Priority Level Hierarchy	50
6.5	Measured Execution Times in Run-Time with Protection against Interruptions . .	51

List of Tables

5.1	<i>Sender-Receiver</i> API Functions	36
5.2	RTE Status and Error Codes	40
6.1	Metrics of the Execution Times in Run-Time	49
6.2	Metrics of the Execution Times in Run-Time with Protection against Interruptions	51
A.1	Sources and References of the Reviewed “Research Works”	59
A.2	Authors and Year of Publication of the Reviewed “Research Works”	64
A.3	Classification of the Reviewed “Research Works”	66
A.4	Distribution of the AUTOSAR Platform Present in the Reviewed “Research Works”	68

List of Listings

2.1	Example Source Code of a <i>Runnable</i>	9
5.1	Generated <code>Rte_Read</code> API Function.	39
5.2	Generated <code>Rte_IRead</code> and <code>Rte_IStatus</code> API Functions	41
5.3	Generated <code>Rte_DRead</code> API Function.	43
5.4	Generated <code>Rte_Receive</code> API Function.	44
5.5	Generated <code>Rte_Read</code> API Function without “Signal Groups”.	46
B.1	ARXML Specification of a <i>Runnable</i> with <code>DataReceivePointByArgument</code> Attribute.	71
B.2	Code Snippet of <code>Runnable_ControlUnit</code> with <code>Rte_Read</code> API Function. . .	71
C.1	<code>Measurements.cmm</code> Script File.	73
C.2	Extracted <code>RunTime</code> Window.	74
C.3	VBA Script File for Data Extraction.	74
C.4	VBA Script File for Minimum, Maximum and Average Metrics.	74

Abbreviations

ADAS	Advanced Driver-Assistance Systems
API	Application Programming Interface
ARA	AUTOSAR Run-time for Adaptive applications
ARXML	AUTOSAR eXtensible Markup Language
AUTOSAR	AUTomotive Open System ARchitecture
BSW	Basic SoftWare
CAN	Controller Area Network
CANif	Controller Area Network Interface
CAL	Crypto Abstraction Library
CES	Continental Engineering Services
COM	COMmunication
CPU	Central Processing Unit
CSM	Crypto Service Manager
DC	Direct Current
E2E	End-to-End
ECU	Electronic Control Unit
Euro-NCAP	European New Car Assessment Programme
FEUP	Faculdade de Engenharia da Universidade do Porto
FIFO	First-In-First-Out
IDE	Integrated Development Environment
I-PDU	Interaction layer Protocol Data Unit
I-PDU ID	Interaction layer Protocol Data Unit Identifier
I/O	Input/Output
IoT	Internet of Things
LiDAR	Light Detection And Ranging
MAC	Message Authentication Code
OEM	Original Equipment Manufacturer
OSEK	Offene Systeme und deren schnittstellen für die Elektronik im Kraftfahrzeug
PDU	Protocol Data Unit
PDuR	Protocol Data unit Router
RTE	Run-Time Environment
SWC	SoftWare Component
V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VBA	Visual Basic for Application
VFB	Virtual Function Bus
XML	eXtensible Markup Language

Chapter 1

Introduction

This introductory chapter contextualizes the dissertation, concisely presenting the AUTomotive Open System ARchitecture (AUTOSAR) and identifying the proposed core objectives. In addition, a brief description of the organization of this document is presented.

1.1 Context

The advancement of current systems for automotive-embedded applications has been conscious of safety and real-time requirements in progressively competitive and demanding surroundings. Such solutions, like modern self-driving cars and their growth in international markets, have heightened complexity and computational needs [1]. Analogously, a continuous necessity for augmented efficiency and productivity lingers in the development process of such systems.

Until the introduction of AUTOSAR in 2003, automotive-embedded software development predominantly centered around the hardware specifications of the end product. This procedure led to several necessary iterations of the design and development process of these projects. For instance, producing software-related logic in the context of a camera sensor would yield a solution with low reusability and high development expenses. The establishment of the AUTOSAR partnership was a direct response to address this inefficient and tedious approach. Its primary objective is to ensure that programs are compatible across different hardware platforms [2].

The main focus of this joint operation was primarily directed towards Electronic Control Units (ECUs) in the automotive industry. ECUs are central to modern automotive-embedded systems and oversee and manage specific functions or subsystems within a vehicle. Subsequently, a software architecture framework, organized into multiple layers, was created to facilitate the manufacturing of adaptable and robust ECU programs. The abstraction between what is hardware and what is software is one of its paramount objectives, ushering in a cutting-edge era of vehicle development. In particular, this initiative has successfully brought together key practitioners in the industry and has also aided in tackling the current challenges faced by the mobility sector [3]. Consequently, products within this consortium exhibit enhanced scalability, quality, reliability, and

lifespan. Moreover, a worldwide standard was introduced to govern the software and methodology advancements of open electrical engineering system architectures [4].

The expected benefits of the AUTOSAR standard have been discussed in recent years, being a subject of study in various reports and research works. In one of them, the perceived advantages when adopting AUTOSAR were surveyed through a web-based questionnaire [5]. This particular survey was targeted to end-users, of which 51 responded to the question “Which are the benefits of using AUTOSAR?”, with the answers shown in Figure 1.1.

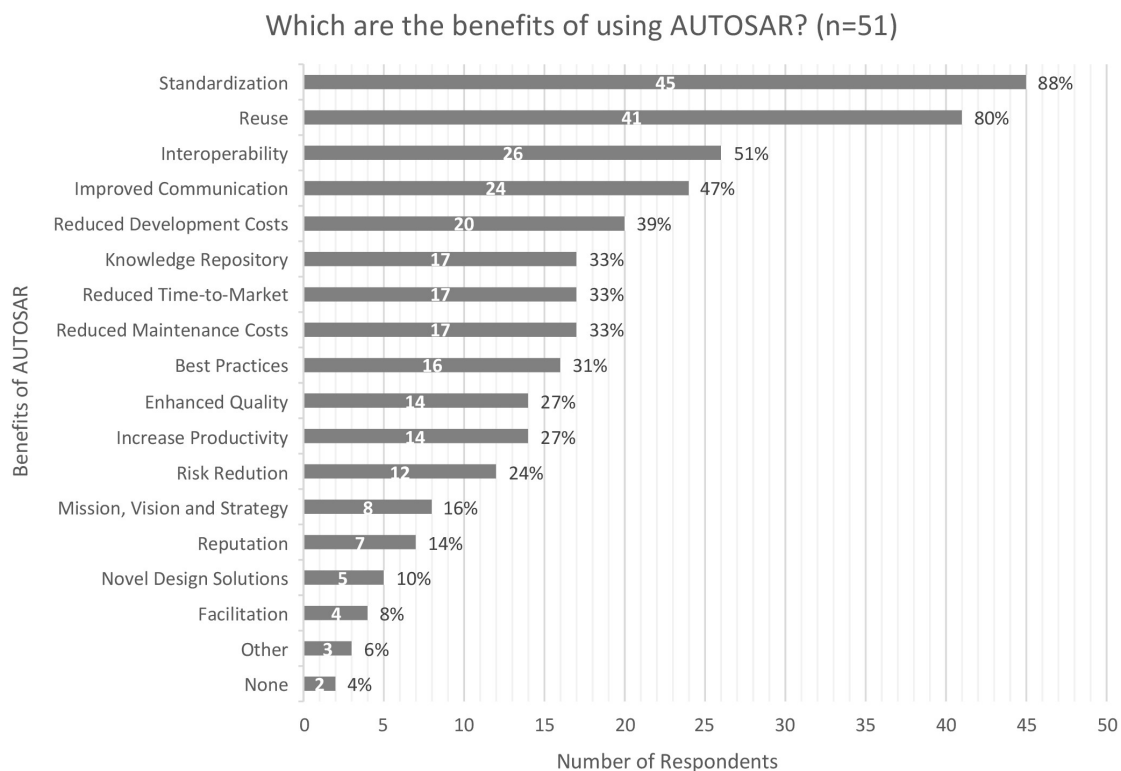


Figure 1.1: Benefits of the usage of AUTOSAR, adapted from [5].

It was in this paradigm that the dissertation was developed. The workplace in which this work took place, Continental Engineering Services (CES) Porto, is part of the *Continental Group*. This company is one of the nine *Core Partners* founders responsible for technical contributions to the standard. Besides the *Core Partners*, the AUTOSAR consortium is divided into several tiers and membership levels, each with a specific role. It comprises *Premium Members* that also aid in progressing the standard and *Associated Members* with access to pre-release documentation. There are also supporting roles, such as the *Development Members* and *Attendees*, that help in maintaining instructions and guidelines of the partnership with outside institutions [3].

1.2 Goals

The main aim of this dissertation is to discuss and test different AUTOSAR-compliant implementations and solutions that can help companies and researchers in the embedded systems field. In order to aid in this endeavor, the dissertation was worked on with support from a team at CES Porto that specializes in developing Advanced Driver-Assistance Systems (ADAS) solutions.

More specifically, software-related modifications of a front camera sensor, were studied and worked on. Besides this camera being European New Car Assessment Programme (Euro-NCAP) compliant, it has cutting-edge features like *Traffic Light Recognition*, *Continuous Lane Centering Assist*, and *Emergency Lane Keeping* [6].

This undertaking was done with the prospect of uncovering AUTOSAR-compliant reception methods of retrieving signals from the camera sensor. Additionally, there was interest in categorizing these techniques according to their advantages and disadvantages.

A particular emphasis was placed on evaluating the communication efficiency established by these different AUTOSAR implementations. The specific modules that were the focus of this work, more notably the concept of a *Receiver* paradigm, will be highlighted during the course of this document.

Furthermore, developing effective methods of measuring the obtained results that could help achieve these goals was also one of the objectives. The team at CES Porto was particularly interested in discovering the most optimal reading method that AUTOSAR supported.

Simultaneously, AUTOSAR constitutes a crucial step towards the development of a new age of vehicles. Therefore, the actual applicability of this concept in a more interconnected and ubiquitous world was worth questioning. Hence, a parallel goal was to analyze the progress of AUTOSAR over its lifespan and simultaneously address significant challenges and development prospects that have emerged in recent years. Ultimately, the aim was to consolidate the uncovered findings into a comprehensive scientific article. Key topics were mainly addressed, specifically common implementation issues, security concerns, and emerging use case scenarios when positioning AUTOSAR in contemporary demands.

1.3 Structure

This document consists of seven chapters structured to aid the reader in understanding each one of the concepts and phases that the dissertation process went through. Since this dissertation is inserted in a corporate setting, the naming of the components and modules of the camera ECU will be abstracted in order to keep confidentiality.

As the name indicates, the current Chapter 1 exposes a contextualization of the problem, the proposed objectives, and the structure of the dissertation document.

Chapter 2 provides essential and theoretical knowledge about the AUTOSAR framework, architecture, and analogous notions. These topics serve to assist the reader in grasping the engineering and scientific domain in which this work is positioned.

Chapter 3 encompasses the performed literature review, including its methodology and objectives. Additionally, the chapter discusses the outcomes attained through the literature review and the subsequent composition of a scientific article.

Chapter 4 offers more profound insights into AUTOSAR concepts while presenting the hardware and software architecture of the camera ECU and the toolkit that was handled.

Chapter 5 describes the development of the practical work that was done, with particular attention devoted to explaining each step undertaken and the rationale behind them.

Chapter 6 discusses the implemented solutions, noting the various functionality tests and debugging strategies performed to validate them.

The final Chapter 7 presents the key conclusions reached, taking stock of the fulfillment of the proposed objectives and envisaging steps for possible future work and improvements.

Chapter 2

Background and Fundamental Aspects

This chapter describes AUTOSAR theoretical concepts essential to the proposed objectives of the dissertation. The contextualization of the state of the automotive industry prior to AUTOSAR, the presentation of its layered architecture, and its independent platforms will be addressed. There will be a particular emphasis on some AUTOSAR notions and supported communication modules, as these were critical perceptions studied during the dissertation. A bottom-up approach was employed in the explanation of this hierarchy in order to aid the reader in understanding these notions.

2.1 Automotive Industry Pre-AUTOSAR

Picking up the snippet in Section 1.1, pre-AUTOSAR Original Equipment Manufacturers (OEMs) prior to 2003 had to rely on separate hardware and software tools to produce ECUs. Whenever there was a change in the selected hardware, it was necessary to go through multiple iterations of the software design and development phases. This strong dependence on hardware resulted in the creation of ECUs that had limited reusability and incurred significant development costs. Back then, the amalgamation of all these factors emphasized the lack of a defining standard [4].

A group of German automotive companies made an initial effort to tackle these concerns by developing the automotive software platform Offene Systeme und deren schnittstellen für die Elektronik im Kraftfahrzeug (OSEK). OSEK translates to “Open Systems and their Interfaces for the Electronics in Motor Vehicles” and provides standardized interfaces that bring together hardware and software development, allowing for the reuse of components. This solution, coupled with the continuous demand for increased system complexity in the automotive industry, paved the way for the current idea known as AUTOSAR. Currently, AUTOSAR divides itself into two primary platforms: the original *Classic Platform* and the younger *Adaptive Platform* [4].

Nowadays, the AUTOSAR consortium retains a global presence, encompassing numerous entities and organizations in the automotive industry, totaling 186 as of 2015. Some of its more iconic members and *Core Partners* include *BMW Group*, *Bosch*, *Continental Group*, *Daimler-Chrysler*,

Ford Motor, PSA Peugeot Citroën, Siemens VDO, Toyota Motor Corporation, and Volkswagen [3, 5]. Figure 2.1 portrays some of the participants of this global partnership.

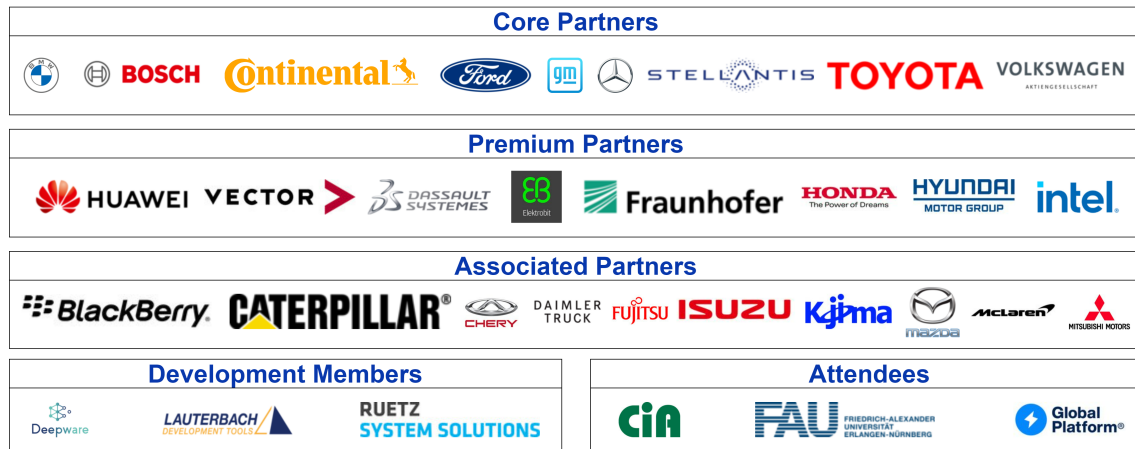


Figure 2.1: AUTOSAR Consortium, adapted from [7].

2.2 AUTOSAR Classic Platform

The goal of the AUTOSAR software architecture was to establish itself as a standard that simplifies the development of modular and interconnected cyber-physical components by abstracting the underlying hardware elements. The first platform that the standard devised was the *Classic Platform* seen in Figure 2.2. This solution encompasses several layers, each offering specific services and organized modularly to accommodate software modules. Each element of this intrinsic and complex hierarchy fulfills a critical role in delivering the component abstraction promise of AUTOSAR [4].

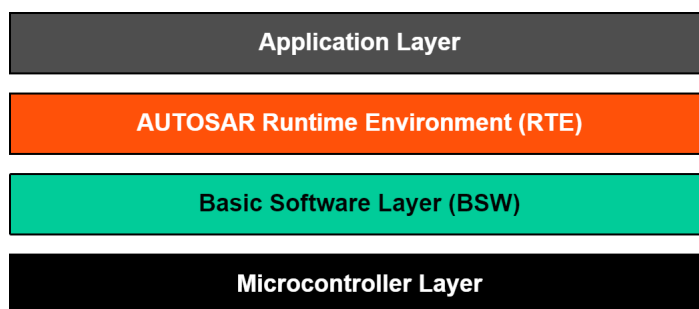


Figure 2.2: AUTOSAR Layered Software Architecture, adapted from [8].

Firstly, all physical ECU-related hardware is grouped in the lower-most layer. Despite not being an AUTOSAR-exclusive layer, the *Microcontroller Layer* represents how the AUTOSAR architecture views and positions hardware-related concepts. Above this, the *Basic Software (BSW) Layer* maintains modules for specific ECU-related requirements and other generic AUTOSAR components. They are responsible for upholding ECU abstraction and are comprised of intricate

device drivers, services, operating systems, and communication protocols. The Run-Time Environment (RTE) aids the *BSW Layer* in ensuring a distinct separation between the hardware and software elements of an AUTOSAR application. The RTE facilitates the information network linking components in the *Application Layer* and permits communication between AUTOSAR Software Components (SWCs) and BSW modules. The highest layer, identified as the *Application Layer*, encompasses all application-related software in the format of SWCs [8, 9].

2.2.1 General Overview of the *Application Layer* and RTE

SWCs make up the basis of an AUTOSAR application and are created inside an ECU instance according to the *SWC Template*. The *SWC Template* serves as a combination of guidelines defined by AUTOSAR that a software developer must follow in structuring and characterizing the contents of a SWC while inventing AUTOSAR-compliant systems. The RTE requires the information outlined in this template. Therefore, a SWC needs to specify its composition (*Hierarchical Structure*), the communication syntax (*Ports and Interfaces*), scheduling and method invocation details (*Internal Behavior*), and memory consumption and execution times (*Implementation*). Among these, the communication syntax dictates the gateways in which SWC data is transferred. Consequently, the RTE is created separately for each ECU instance in the *Application Layer* to accurately determine the presence and absence of associated components [8, 9, 10].

Alongside the RTE, the Virtual Function Bus (VFB) establishes the network scheme for SWCs and BSW modules, such as operating system access. Additionally, it specifies an interface for an access point that sends and receives data to a specific SWC. This communication interface can be either a *Sender-Receiver* or a *Client-Server*, as seen in Figure 2.3.

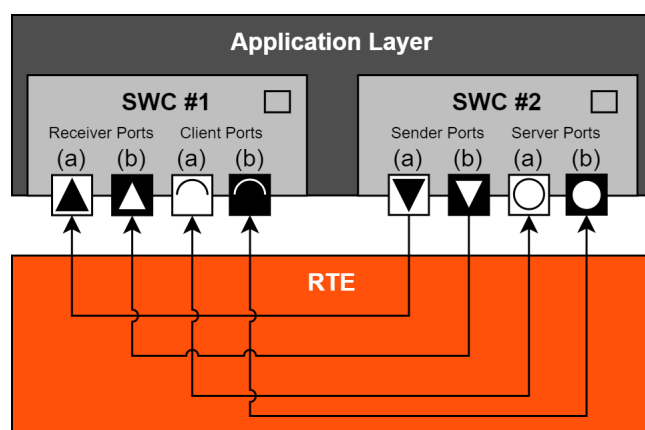


Figure 2.3: AUTOSAR SWCs and Ports.

Sender-Receiver paradigms comprise unidirectional data exchanges between SWCs, where a *Sender* provides information to a *Receiver* that, in turn, receives and processes the data. In contrast, *Client-Server* schemes encompass request-response-based communication between SWCs, where a *Client* requests a specific service or operation towards a *Server* [11].

These communication patterns are defined through the configuration of AUTOSAR *Ports*. Each AUTOSAR *Port* is associated with a specific SWC and serves as a door frame that states a future occurrence of a yet undefined type of communication. There are three types of AUTOSAR *Ports*, the *P-Port* that provides information, the *R-Port* which receives information, and the *PR-Port* that does both of these actions. Additionally, *Ports* can be specified as *Service Ports*, which deal with information from the lower layers and not directly from other SWCs. They possess a proportionate graphical representation to normal *Ports* (a) but with inverted colors (b).

With these gateways defined, the *AUTOSAR Interface* dictates the method of communication expected to occur and the structure of the *data elements* anticipated to be transported over the VFB [8, 11]. These are akin to programming variables and can vary from simple `integers` to more complex formats, such as `arrays`.

Independent of the hardware topology of the ECUs, the VFB represents a fundamental idea that allows the advancement of the system design, consequently making RTE the core of the ECU architecture. Figure 2.4 exhibits the AUTOSAR VFB specification for a communication topology with three distinct SWCs. It represents one *Sender* that retains an output *P-Port* in contact with two *Receivers*, each with an input *R-Port* [8, 9, 10].

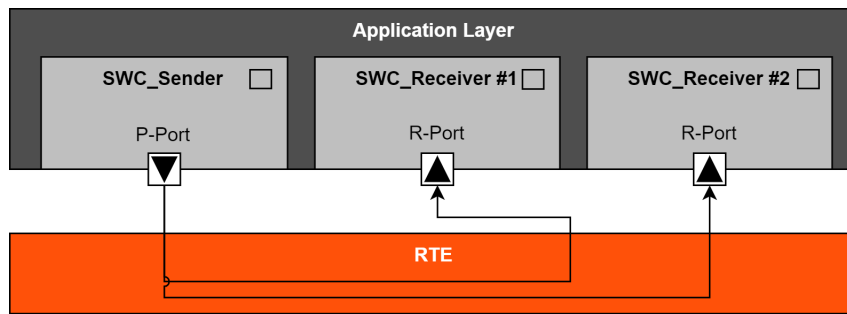


Figure 2.4: VFB Model for a *Sender-Receiver* Communication Topology, adapted from [10].

It is equally worthy of mentioning another key concept, *Runnable Entities (Runnables)*, that comprises the executable segment of an AUTOSAR SWC. They are akin to functions produced in C/C++ programming languages and comprise the fundamental piece of code in an AUTOSAR SWC. Created inside the *Internal Behavior* of the SWC, these must be mapped by the RTE, while respecting specific rules, into a task of the operating system during the configuration of the ECU. They are later executed concurrently and cyclically via the activation of *RTEEvents*, depending on the wanted behavior of the SWC. AUTOSAR defines several types of *RTEEvents* that can be used in conjunction with *Runnables*. Some of the more prevalent to mention include the *TimingEvent*, *BackgroundEvent*, and *DataReceivedEvent*. The *TimingEvent* activates *Runnables* periodically based on the amount specified in the SWC, e.g., a 10 millisecond interval [12]. The *BackgroundEvent* is similar to *TimingEvent*, however, it is delegated to *Runnables* that perform low-priority background operations. The *DataReceivedEvent* triggers *Runnables* whenever signals in a *Sender-Receiver* communication topology are received.

Using a cyclic or periodic execution model for the activation and execution of *Runnables* is

one of the most critical aspects of AUTOSAR. This repetitive and time-triggered approach allows deterministic and predictable behavior, efficient resource utilization, and synchronized software components for real-time systems [12]. A graphical representation of the composition of tree different *Runnables* is depicted in Figure 2.5.

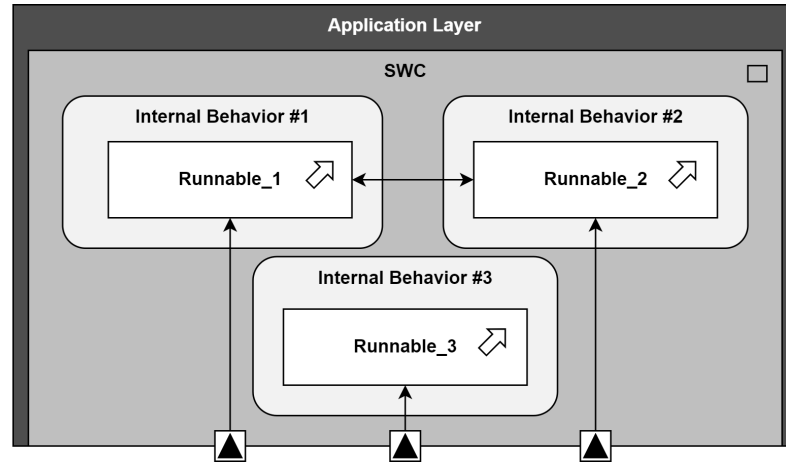


Figure 2.5: AUTOSAR *Internal Behaviors* and *Runnables*.

An example of an implementation code of a *Runnable* that retrieves a *data element* from an input *R-Port* and formulates a *data element* to an outgoing *P-Port* is displayed in Listing 2.1. As observed, it operates Application Programming Interface (API) functions from the RTE to gather and send the *data elements*. The VFB handles these sequences of instructions, while the RTE deals with the context and environment in which they will run. The creation of ports, *AUTOSAR Interfaces*, the *Internal Behavior*, and *Runnables* is done at the *Application Layer* and then linked with RTE via API functions [8, 10, 13].

```

1 void SetPorts_Runnable() {
2
3     int data_element_1;
4
5     Rte_Read_SWC_R_Port_Data_Element_1 (data_element_1);
6
7     data_element_1 = data_element_1 + 1;
8
9     Rte_Write_SWC_P_Port_Data_Element_1 (data_element_1);
10
11 }

```

Listing 2.1: Example Source Code of a *Runnable*, adapted from [10].

2.2.2 Components and Modules of the BSW Layer

The aforementioned layered structure of Figure 2.2 can be further divided, regarding the BSW, into the *Service Layer*, the *ECU Abstraction Layer*, and the *Microcontroller Abstraction Layer*, as displayed in Figure 2.6.

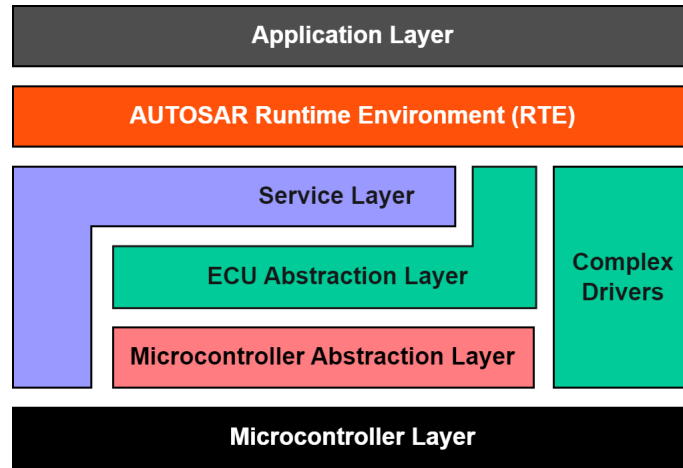


Figure 2.6: AUTOSAR Architecture with Expanded *BSW Layer*, adapted from [8].

The *Microcontroller Abstraction Layer* abstracts specialized controllers on the ECU using drivers. These drivers provide interfaces to the upper *ECU Abstraction Layer* to make it possible for various microcontrollers to be used in a broader context. The *ECU Abstraction Layer* aims at abstracting the location of the controller to the uppermost layers, as this information is inconsequential for them. Therefore, this layer constitutes a hardware-independent interface and driver for generating upper-layer software. Components in this layer are coupled via *AUTOSAR Interfaces* that use the VFB for outgoing and incoming communications from the *ECU Abstraction Layer* and specific sensor and actuator SWCs. The *Service Layer* offers services for AUTOSAR accessible applications via *Standardized AUTOSAR Interfaces* that AUTOSAR SWCs use. These services are tied to the RTE through these interfaces that allow SWCs to attach to outbound *P-Ports* or to send inputs to *R-Ports* [4, 8, 10].

Additionally, the BSW layers can be deepened into different orthogonal stacks, from left to right, that relate to the functionality that the software provides, as presented in Figure 2.7. In total, there exists the *System Stack*, *Memory Management Stack*, *Communication Stack*, *Input/Output (I/O) Stack* and the *Complex Driver Stack*, each with specific and distinct modules [14].

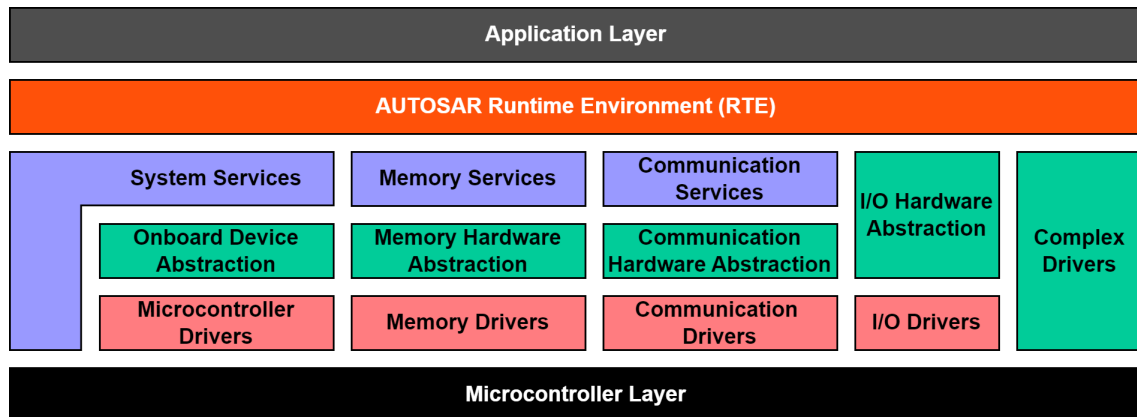


Figure 2.7: AUTOSAR Architecture with Expanded *BSW Layer Stacks*, adapted from [8].

The *Microcontroller Drivers* offers drivers for *Software Modules* and internal peripherals of the *Microcontroller Layer* below. As the name suggests, the *Onboard Device Abstraction* abstracts from ECU onboard devices by encompassing drivers not visible as sensors or actuators. The *System Services* offers functions and modules for the application, like real-time operating system and ECU status supervision. These three comprehend the *System Stack*, which supports standardized services, library functions, and distinct ECU services [4, 8, 15].

The *Memory Drivers*, *Memory Hardware Abstraction*, and *Memory Services* make up the *Memory Management Stack* that contributes with the standardized access to non-volatile memory. This stack enables the allocation of memory by each software application component, independently of whether the memory of the ECU is internal or external. More specifically, the *Memory Hardware Abstraction* gives access to both internal and external memory devices through the same interface [4, 8].

The *Communication Drivers*, *Communication Hardware Abstraction*, and *Communication Services* incorporate the *Communication Stack*, which gives standardized access to the vehicle network system. Due to this stack, SWCs can contact one another via the RTE independently of their respective ECUs. In essence, two components on the same ECU will interact directly, otherwise, the *Communication Stack* is used for different ECUs. In this manner, the dependency of the lower layers is eliminated as a result of the communication network management and unified interface provided by the *Communication Services*. This abstraction allows applications to be designed in the topmost layer, oblivious to the protocol in use [4, 8, 16].

Right of the aforementioned stack, the *I/O Stack*, comprised of *I/O Drivers* and *I/O Hardware Abstraction*, yield standardized access to ECU onboard peripherals like sensors and actuators. For instance, the *I/O Hardware Abstraction* gives I/O access to the uppermost layers without the *Service Layer*. The absence of a *Service Layer* is solved with special SWCs called *Sensor/Actuator Software Components* that are deployed on an ECU. The former is used to analyze the state of the sensors, while the latter controls the actuators of the system. The interfaces provided by either one of the modules of the *I/O Hardware Abstraction* or SWCs are called *AUTOSAR Interfaces* [4, 8].

Although the *Complex Driver Stack* is not an actual stack, and AUTOSAR does not provide these drivers, they set up a bypass of the *BSW Layer*. This feature is employed by application components provided by the manufacturers that, because of performance criteria, want direct access to the hardware devices on the ECU. However, using non-standard modules, which hardware-dependent *Runnables* can utilize to reach the *Microcontroller Layer*, causes a loss of abstraction which is one of the founding objectives of the standard [8, 15].

Figure 2.8 highlights some modules within the AUTOSAR software architecture that are part of the *Communication Stack*. These were utilized in the communication topology that was studied in the context of this dissertation and will be emphasized throughout this document.

The Controller Area Network (CAN) vehicle bus is one of the communication protocols AUTOSAR supports. The CAN Driver module is situated in the *Communication Drivers* and offers hardware access and a hardware-independent API to the upper layers [16, 17].

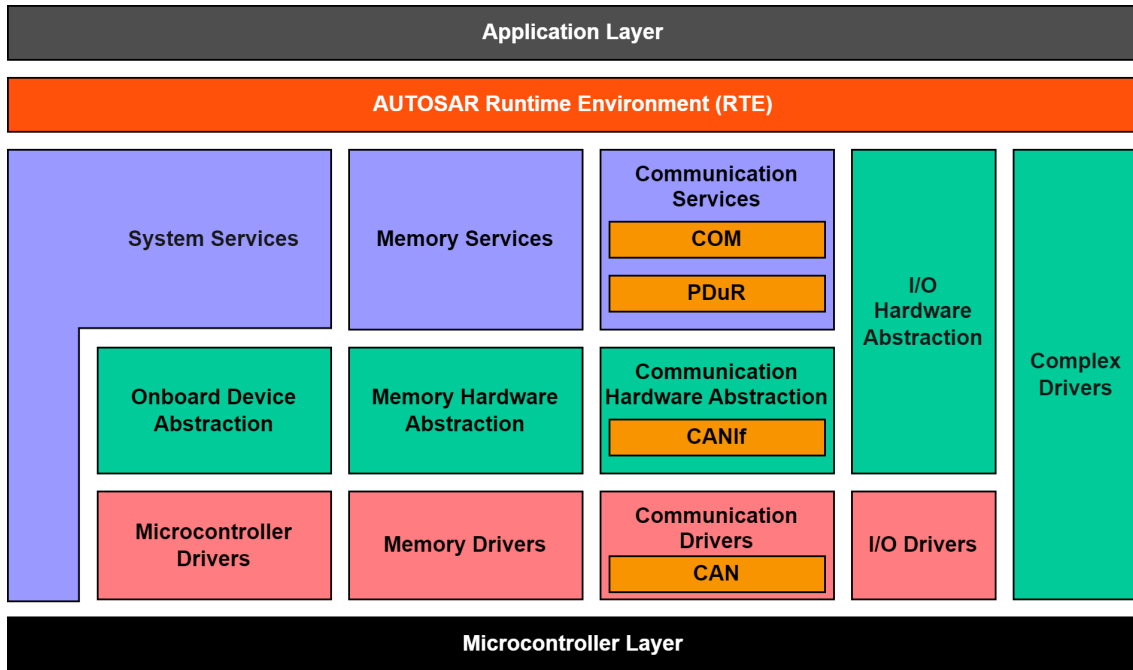


Figure 2.8: AUTOSAR Architecture with Expanded *Communication Stack*, adapted from [15].

The *Communication Hardware Abstraction* receives data from this module, such as transmissions, through the Controller Area Network Interface (CANIf). As the name suggests, this module is the interface between the lower-layer drivers and the upper-layer services. For instance, it controls data flow requirements and transmission confirmation for modules in the *Communication Services*. Additionally, it converts signals into Protocol Data Units (PDU) consisting of an identifier, signal data, and coupled length [16, 18].

When reaching the *Communication Services*, PDUs are converted into Interaction layer Protocol Data Units (I-PDUs). These frames contain a data buffer, the associated length, and a specific Interaction layer Protocol Data Unit Identifier (I-PDU ID). Then, the Protocol Data unit Router (PDuR) uses the I-PDU ID to ascertain the destination of the respective I-PDU. The PDuR uses static routing paths tables and an internal engine to forward the I-PDU to the intended modules [16, 19].

The AUTOSAR COMMunication specification (COM), based on the ISO 17356-4:2005 defined by OSEK, is one of the possible targets. This module specifies a uniform communication environment for ECU application software and is responsible for converting I-PDUs to AUTOSAR signals. This operation entails packing AUTOSAR signals to I-PDUs in the case of transmission and unpacking I-PDUs in the case of reception. For instance, received I-PDUs are converted to RTE-compatible signals that are then conveyed to the SWCs in the *Application Layer* [4, 8, 15, 16, 20].

2.3 AUTOSAR Adaptive Platform

Although the dissertation work was done on top of the AUTOSAR *Classic Platform*, it is equally worth mentioning and presenting the AUTOSAR *Adaptive Platform*.

Introduced in 2017, this framework emerged as the answer AUTOSAR gave to the emergence of contemporary concepts such as the Internet of Things (IoT) and ADAS. This solution enables the development and management of applications that deal with intensive computing tasks and vast volumes of data. Some examples include in-vehicle infotainment, connectivity, and automated driving. Hence, the *Adaptive Platform* seeks to create the necessary conditions and environment for engaging with non-traditional AUTOSAR applications [21, 22].

This object-oriented solution shares the goal of the AUTOSAR *Classic Platform* in offering independent design and development of ECU applications. Besides sharing some concepts, this framework introduced enhancements and incorporated additional functionalities not available in the *Classic Platform* at the time of its foundation. Among these, updates-over-the-air, communication with back-end systems, and dynamic deployment of applications stand out as important features. For instance, the exchange between local and remote systems and services can be established through its programming interface, the AUTOSAR Run-time for Adaptive applications (ARA) [23].

Figure 2.9 illustrates an example of an hybrid deployment of both *Platforms*, (“CP” for *Classic Platform* and “AP” for *Adaptive Platform*) on an automotive vehicle.

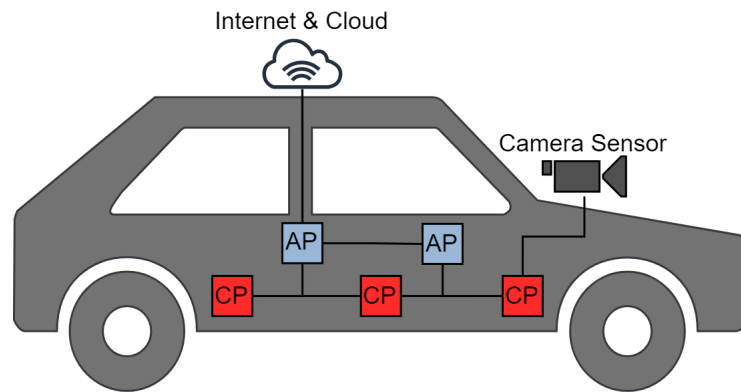


Figure 2.9: Deployment of the *Classic Platform* and *Adaptive Platform*, adapted from [24].

As observed, ECUs with traditional responsibilities, such as those in front camera sensors, still rely on the *Classic Platform*. These can then be paired with ECUs using the *Adaptive Platform* better prepared to perform high-performance computing operations and communications with external back-end systems. Thus, instead of seeking to replace its older brother, the *Adaptive Platform* looks to enable the creation of integrated systems based on more dynamic and intelligent ECUs.

Chapter 3

Literature Review

This chapter presents the performed literature review, highlighting its methodology, goals, results, and contributions. Initially, some common development issues were explored to understand potential hurdles and solutions regarding this framework that could impact the development phase with the team at CES Porto. Subsequently, AUTOSAR concepts were correlated and evaluated with the challenges that entail some emerging scenarios. The gathered and processed information was compiled into a survey article, which aimed to describe AUTOSAR concerning its implementation issues, challenges, and promising applications. Therefore, some of the most notable works uncovered will be highlighted and discussed.

3.1 Methodology

Initially, it was decided that the literature review would be conducted with search criteria around conference papers, journal articles, and technical reports. These will be collectively referred to as “research works” throughout this document. Afterward, it was decided to label the retrieved documents into three primary classifications: *Development*, *Security*, and *Use Cases*. This taxonomy was chosen because it involves areas that represent current demands in both academic and industry fields. Additionally, it was decided that the *Use Cases* label would be applied to “research works” that fit both *Development* and *Security*.

The selection and analysis of the “research works” was manually conducted in order to ensure that they would be themed around primary keywords, such as “AUTOSAR”, “ECU”, “Security”, and “AUTOSAR Case Studies”. Additionally, due to the nature of the proposed objectives of the dissertation, “ADAS” and “Real-Time Scheduling” were considered secondary keywords. Although not used in the context of this dissertation, the combination of the keywords mentioned above yields the ((“AUTOSAR” AND “ECU” AND “Security” AND “AUTOSAR Case Study”) AND (“ADAS” OR “Real-Time Scheduling”)) search string. Likewise, the number of citations of each document was not considered.

Moreover, the definition of the scope of this study was another crucial decision taken into account when drafting the aim of the literature review. More specifically, a time frame of almost 20

years was chosen in order to outline the scientific advancements made in the AUTOSAR industry. In the end, the state-of-the-art retrieved documents from 2004 to 2022.

Finally, the literature review returned 60 works, of which 60% originate from *IEEE Xplore*. The remainder was derived, in descending order, from *Springer Link*, *Semantic Scholar*, *ScienceDirect*, *ACM Digital Library*, *MDPI*, *SAE International*, *HAL*, *Hasso Plattner Institute*, and *CORE*, as can be seen in Figure 3.1. Among these, *Hasso Plattner Institute*, a German technology institute, produced detailed and precise documents surrounding AUTOSAR-associated notions. These technical reports played a fundamental role in understanding and grasping concepts regarding the AUTOSAR architecture and modules.

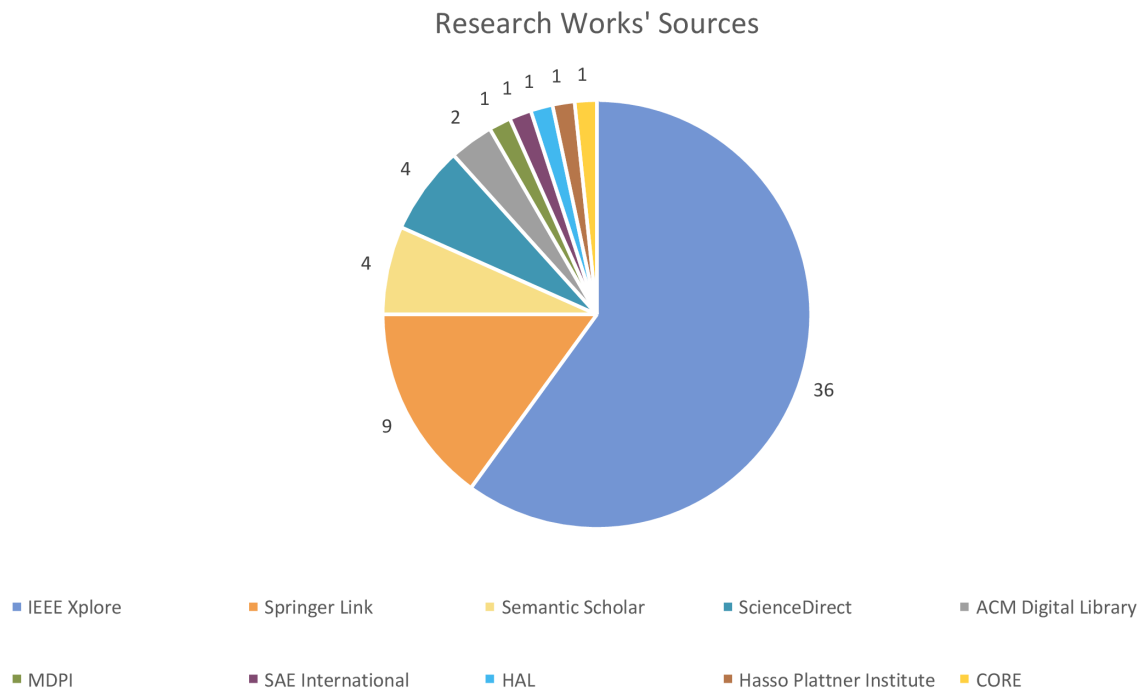


Figure 3.1: Distribution of the Sources of the “Research Works”.

The following sections address the most prominent “research works” of the aforementioned classifications. Some of these concepts were explored and summarized in the published survey article. However, there was attention to expanding these ideas in the dissertation document that, due to constraints around submission guidelines, could not be explored in greater detail in the published survey article [25].

3.2 Development Issues

This section emphasizes some common development issues uncovered during the literature review regarding the *Development* classification. Originally, this criterion was intended to anticipate potential obstacles and solutions when working with AUTOSAR. However, these findings were not

applicable throughout the context of the work done during the implementation phase of the dissertation. Nonetheless, they evolved into common development issues at the forefront of the evolution of AUTOSAR-compliant ECU creation.

3.2.1 AUTOSAR Overheads

The design phase of software-based solutions is one of the most crucial stages in developing AUTOSAR applications. Unfortunately, as explored by Chauhan et al. [26], developers usually overlook the potential impact caused by overheads when constructing AUTOSAR-compliant solutions. If left unattended, this negligence can cause a downgrade of efficiency in the environment where these products are usually deployed, real-time automotive systems. With the help of a third-party open-source tool for schedulability analysis, these authors sought to quantify and heed this problem. Ultimately, four unique types of overheads were estimated as explained below:

- **Operating System Overheads:** Operations that supervise task execution and device drivers that can potentially cause a higher-priority task to preempt (stall) a lower-priority task.
- **Communication Overheads:** Execution time associated with RTE API functions that rely on the communication type and mapping of *Runnables*.
- **Other Service Overheads:** Execution time corresponding to non-communication related RTE API functions.
- **Runnable Overheads:** Time consumed when performing a *Runnable* as a part of a task. This type can be further divided into four sub-types as described underneath:
 - **Pre-Runnable Event:** Execution time between the instant that the task hears the event to run the code and the moment the RTE API function receives the flag to start the *Runnable*.
 - **Pre-Runnable:** The duration that the RTE API function takes to execute the *Runnable*.
 - **Post-Runnable Event:** The span between the point when the *Runnable* finishes its code and the instant the RTE API function signals the task.
 - **Post-Runnable:** Remaining time the leftover task code needs before receiving another event and restarting this cycle.

Afterward, Chauhan et al. [26] employed the “Modeling and Analysis Suite for Real Time Applications” toolset to estimate the effects caused by AUTOSAR overheads in an application. In order to simulate the behavior in real-time automotive systems, a test bench composed of an ECU that controlled four lights based on buttons and switches was used. The tests performed by these authors yielded a surge of 10% regarding Central Processing Unit (CPU) utilization. Consequently, it was uncovered that more overheads existed in tests with a larger number of *Runnables* and tasks. Moreover, solutions with a task period identical to the system timer generated fewer schedulable applications. These findings warrant attention since unexpected delays can lead to perilous situations in real-time automotive systems.

3.2.2 Multi-Core ECUs

Back in 2003, when AUTOSAR was first idealized, single-core ECUs occupied the spotlight in the automotive industry. Since then, this dynamic has changed with the growth in interest of companies in migrating towards multi-core ECUs. This switch is attributed, for instance, to the growing need and offer of improved driving experiences. Products that deal with these concepts need the computational power that multi-core ECUs solutions offer. Nonetheless, transitioning towards higher processing capacity can cause obstacles in obtaining an efficient AUTOSAR-based ECU design.

This current challenge is the theme of the work developed by Gupta et al. [27], which focuses on increasing the efficiency of a dual-core processor. They aimed to reduce the communication cost between *Runnables* by addressing their mapping into cores as a way to inhibit the adverse effects that this procedure can cause. For instance, tasks and *Runnables* are periodic, which means the system will suffer adversely if any *Runnable* misses its deadline.

Their proposed solution consists of initially gathering the data of each *Runnable* from the application. Following this, the *Port* data of the SWCs is retrieved, as well as additional information from the ECU relative to tasks and *Runnables* dependency. Moreover, hardware-dependent *Runnables* are allocated to the main core while the remainder is kept in the second core. Afterward, the operating system and RTE are set up, and the CPU load of the cores is calculated.

Ultimately, the combination of these steps resulted in the distribution of the CPU load between the two cores. However, it also induced a more sizable amount of combined CPU load than the total CPU load observed on a single-core ECU.

3.2.3 Intra-ECU Communication

Section 2.2.1 presents *Runnables* as a critical building block in the construction of AUTOSAR applications. With the aid of the RTE, developers need to map these components into operating system tasks. This process is carried out at the ECU design phase and must respect specific AUTOSAR guidelines. Long et al. [13] ascertain that AUTOSAR proposes only a few cardinal rules for this practice. Hence, their work focuses on improving this apparent ambiguity in the context of *Intra-ECU communication*. This topology entails data exchange between several *Runnables* inside an ECU, which these authors sought to optimize.

After going through AUTOSAR-related documentation, these authors identified three major guidelines as the cornerstones delineated by the standard. These rules vary according to the communication behavior of the corresponding *Runnable* and are here presented:

1. *Runnables* can be mapped to a basic task or an extended task if they possess permissions to access, read and/or write data (`DataReadAccess` and `DataWriteAccess` attributes). These are addressed as *Category 1 Runnable Entities* by AUTOSAR and need to be executed in finite time. Extended tasks represent basic tasks that may enter a waiting state if they call a *RTEEvent* via a `WaitEvent`.

2. *Runnables*, that do not uphold the previously mentioned privileges are referred to as *Category 2 Runnable Entities*. They are usually mapped to extended tasks if these include *WaitPoints*.
3. *Category 2 Runnable Entities* that call upon a server and wait for it to return a response at least once are mapped to extended tasks.

These core values cannot be infringed if a developer wishes to produce AUTOSAR-compliant solutions. Afterward, Long et al. [13] enhanced these core values into distinct directives described bellow, with some displayed in Figures 3.2, 3.3, and 3.4:

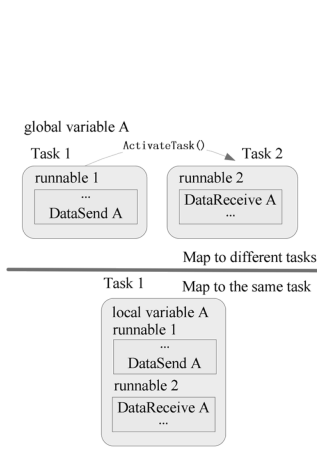


Figure 3.2: Mapping of 1 *Sender* & 1 *Receiver*, retrieved from [13].

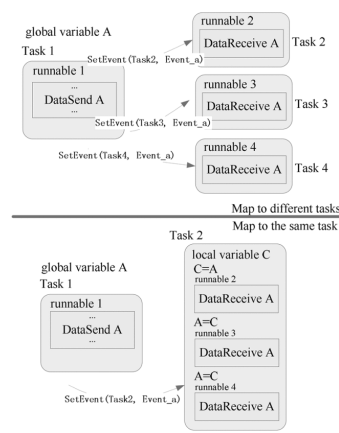


Figure 3.3: Mapping of Multiple *Receivers*, retrieved from [13].

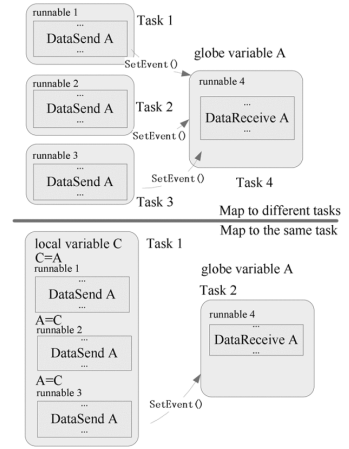


Figure 3.4: Mapping of Multiple *Senders*, retrieved from [13].

1. Instead of being allocated into separate tasks, both the *Sender* and *Receiver Runnables* should be mapped to a sharing basic task to reduce task-switching, as illustrated in Figure 3.2. This rule is valid for *Senders* that have only one *DataSendPoint* and none *WaitPoints*, and *Receivers* that possess a single *DataReceivePoint* and at most one *WaitPoint*.
2. In the case of two or additional *Runnables* that possess a single *DataReceivePoint* for the same *data element*, it is advised to map them into a sole task, as depicted in Figure 3.3. This approach will also optimize task-switching and promote data consistency. Data inconsistency can occur in *Intra-ECU communications* during the access of global variables by *Runnables* that keep distinct priority levels.
3. Akin to the previous rule, two or more *Category 1 Runnable Entities* that carry a single *DataReadAccess* for the same *data element* should be plotted to an individual basic task.
4. Similar to the second rule, two or more *Runnables* that send data to a sharing *Receiver* can be mapped into a singular basic task, as portrayed in Figure 3.4. These *Runnables* merely have

a single `DataSendPoint`, are triggered by cyclic `TimingEvents` *RTEEvents*, possess zero `WaitPoints`, or use the `DataWriteAccess` attribute.

5. *Category 1 Runnable Entities* that have set the `canBeInvokedConcurrently` attribute and are *Servers* validate the mapping of both the *Client* and *Server Runnables* into the same task.
6. Time delays can be avoided by mapping to various tasks more than two *Category 2 Runnable Entities* that do not share the same `WaitPoints`.

Finally, the authors evaluated these rules on a workbench that represented a vehicle wheel control application. The obtained results produced a lower task-switching with zero delay time in mappings that followed these guidelines.

3.2.4 Inter-ECU Communication

Besides defining a comprehensive multi-layered software architecture, AUTOSAR also outlines a design flow with a set of AUTOSAR eXtensible Markup Language (ARXML) files. This textual format is based on the eXtensible Markup Language (XML) and is the stepping stone in developing AUTOSAR applications. The AUTOSAR methodology lays out a development process that starts with the system configuration, which includes the mapping of the SWCs to the ECUs. The remaining steps include the extraction of ECU-specific data, ECU configuration, and generation of the ECU executable. These last two steps are responsible for the module configuration and code creation, compiling, and linking of the BSW and RTE modules. This approach, presented in Figure 3.5, outlines the work-product flow based on initial mapping decisions, such as whether or not components can share the same ECU [2, 28].

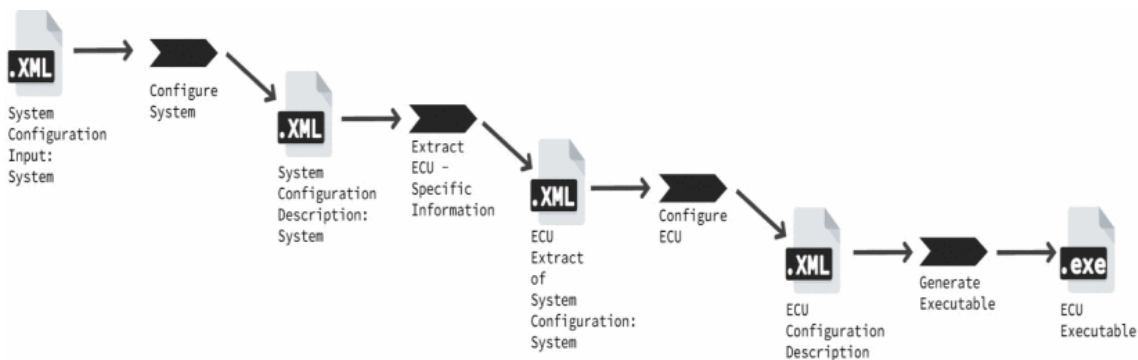


Figure 3.5: AUTOSAR Methodology, retrieved from [2].

With this in mind, Peng et al. [29] proposed a solution to optimize *Inter-ECU communication* while mapping SWCs to ECUs. These authors focused on addressing communication costs and resource demands. Firstly, ECU-dependent components and ECU-independent components are segmented and rebuilt. This process is implemented according to the necessity of

allocating components on the same ECU. Following this, ECU-dependent components and specific hardware-related ECU-dependent components are mapped. Lastly, the remaining ECUs are optimally mapped with a genetic algorithm that the authors proposed.

Subsequently, Peng et al. [29] created four separate applications of vehicle light control and automated mechanical transmission. The use case that implemented the explored design procedure achieved the most promising results of all the tested deployments.

3.3 Security

IoT is at the forefront of the current evolution of software-based solutions and is typically associated with a greater amount of data exchange. Therefore, it is necessary to respect security and safety questions when developing increasingly automated and self-reliant products. Consequently, this classification was established to unveil how the AUTOSAR standard responds to these lingering challenges.

3.3.1 Cryptography

Automotive-related products cover a large portion of international markets, being distributed and branded as cutting-edge systems with endless new and improved features. Unfortunately, the development of these products is not solely noted by potential customers. The competitive nature of this industry warrants attention against potential attackers that seek to exploit flaws and gaps in the characteristics offered by these commodities. This paradigm is especially crucial if developers and companies intentionally or unintentionally fail to anticipate and safeguard their systems.

Murvy et al. [30] sought to investigate if AUTOSAR was aware of these dangers and explore whether the standard supported cryptography-related concepts. These authors discovered that the consortium was mindful of external threats and offered developers several protection specifications. For instance, the Crypto Abstraction Library (CAL) and Crypto Service Manager (CSM) are among the most recent additions to the standard. These provide security mechanisms for software applications. CAL consists of an independent library that depends on software implementations, while CSM is part of the *System Services* in the BSW and can be either software or hardware dependent.

These security modules offer specifications for deploying well-known and standard cryptographic algorithms on AUTOSAR applications. Hash functions, Message Authentication Codes (MAC), symmetric and asymmetric encryption, and digital signatures are some of these methods. Additionally, randomness-related operations handled in the cybersecurity domains are also present, such as key generation, derivation, exchange, and wrapping interfaces.

Unfortunately, Murvy et al. [30] ascertain that, although AUTOSAR supports these mechanisms, the standard does not disclose which primitives are the most adequate for distinct use cases. Instead, AUTOSAR decides to shoulder this choice to developers during the design phase of the system. Finally, they verify that adding security measures to ECUs entails considerable resource and computational costs.

3.3.2 High-Layered Specifications

As explained in the previous section, any unintentional or intentional negligence in securing safety-critical and real-time systems is ill-advised. This concern was the key topic addressed in the work authored by Bernardeschi et al. [31]. They engaged with the notion that modern automotive systems leave open several exposed access points in order to achieve an ever-increasing amount of features related to enhanced driving experiences. Additionally, they argue that AUTOSAR delegates most of its security mechanisms to the BSW while overlooking safety aspects at the *Application Layer*. Subsequently and in line with the previous section, these authors also noted an apparent lack of guidelines that developers would need to comply with in order to secure the communications of a system effectively.

With this in mind, Bernardeschi et al. [31] drafted two approaches in order to confront these challenges, with the aid of “Rational Rhapsody”, a modeling tool with the capability to define extensions for AUTOSAR. Firstly, they devised a trust specification that considered the effort required to execute an attack on SWCs. This solution is beneficial, for instance, for developers mapping SWCs to ECUs. Secondly, they formulated a security requirement specification focused on the confidentiality and integrity of a particular communication. Finally, these authors developed a Python script that was proficient in augmenting an existing AUTOSAR project with these characteristics. Thus, developers can wield this tool to improve their projects while avoiding potential oversights at the higher layers.

3.3.3 Safety Mechanisms

CAN messages are one of the plethora of existent automotive communications supported by the AUTOSAR standard. As expected, these messages are vulnerable to the attempts of external aggressors that want to misuse this protocol to launch attacks. Hence, Nasser et al. [32] investigated methods used by the AUTOSAR End-to-End (E2E) module that could disrupt this particular protocol. E2E is a group of specifications that aim to ensure the integrity and reliability of data communication within an automotive system. For instance, it offers end-to-end data protection in scenarios where information is transmitted between different ECUs [33].

This solution was paired up with *Zero-ID Flood* and *Resource Exhaustion* attack types in the study conducted by these authors. As the name suggests, *Zero-ID Flood* attempts to overwhelm the constant communication of the CAN bus with messages that have invalid or non-existent message IDs. Similarly, a *Resource Exhaustion* attack aims to paralyze a target system with a large volume of CAN requests within a limited time frame in order to degrade its performance.

Nasser et al. [32] uncovered that the defense level against *Zero-ID Flood* attack fluctuated based on the access level granted to the CAN bus in question. E.g., the system would struggle to defend against an attack if the targeted CAN bus was directly accessible or shared with a malicious ECU. Under other conditions, a gateway ECU could oversee and detect the intrusion, aiding in mitigating its effects. Additionally, it was uncovered that it was theoretically possible to reduce and eliminate the threat of a *Resource Exhaustion* attack. However, this type of attack

highlighted potential vulnerabilities in the current protection mechanisms offered by AUTOSAR. Finally, these authors advise automotive systems to adopt alternative preventative measures.

3.4 Use Cases

One of the more crucial aspects covered by the compiled survey article was the discussion of the discovered “research works” that envision the *Use Cases* classification. Current and emerging scenarios were paired up with the AUTOSAR concept and standard in order to evaluate the compatibility of these endeavors. The written scientific article concerning integrating this solution with the Smart Cities idea was also derived from this classification.

3.4.1 Advanced Driver-Assistance Systems

As pointed out in Section 1.2, the dissertation was developed in conjunction with a team CES Porto in the context of ADAS-based applications. These products focus on improving road safety offered to drivers of automotive vehicle systems. E.g., collision mitigation, adaptive cruise control, and lane departure warning are some of the solutions present in this domain. These features are undeniably associated with safety-critical functions because any degradation in their performance and reliability may lead to unexpected and adverse outcomes.

As expected, current cutting-edge sensors, such as Light Detection And Ranging (LiDAR), short to long-range radar, ultrasonic detectors, and cameras, like the one presented in Figure 3.6, are frequently combined in ADAS products. They are connected with ECUs, for instance, via either CAN, FlexRay, or Ethernet communication protocols sharing similar data flows as the one explained in the latter part of Section 2.2.2.



Figure 3.6: Camera used in ADAS Projects at CES Porto, retrieved from [6].

The development process of these applications starts with ensuring the gathering of data from numerous sensors positioned across the vehicle. Afterward, the retrieved data are processed in order to accurately arbitrate the most proper course of action and computationally take said decision. These stages must occur in run-time and within a limited time frame since these devices are expected to be deployed in automotive vehicles [34].

Vulić et al. [34] explored the timing effects caused by an AUTOSAR-compliant implementation of the previously mentioned chain of events. In the end, they concluded that the layered

nature of the AUTOSAR architecture added overheads that needed to be combated while defining the interfaces of connecting modules. An added delay time is not surprising since these products require significant computational resources to precisely analyze and perceive the surrounding environment under various weather conditions. Nonetheless, these findings highlight the necessity for AUTOSAR to account for timing constraints in current emerging scenarios.

3.4.2 Vehicle-to-Vehicle and Vehicle-to-Infrastructure

The Car-to-X communication domain revolves around data exchange between vehicular and outside systems. This field is divided into two significant topologies, Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I). As their name suggests, V2V surrounds information networks from two or more vehicles, while V2I encompasses correspondence between automotive systems and external frameworks [35]. Figure 3.7 portrays a broad depiction of these ideas in contemporary vehicles.

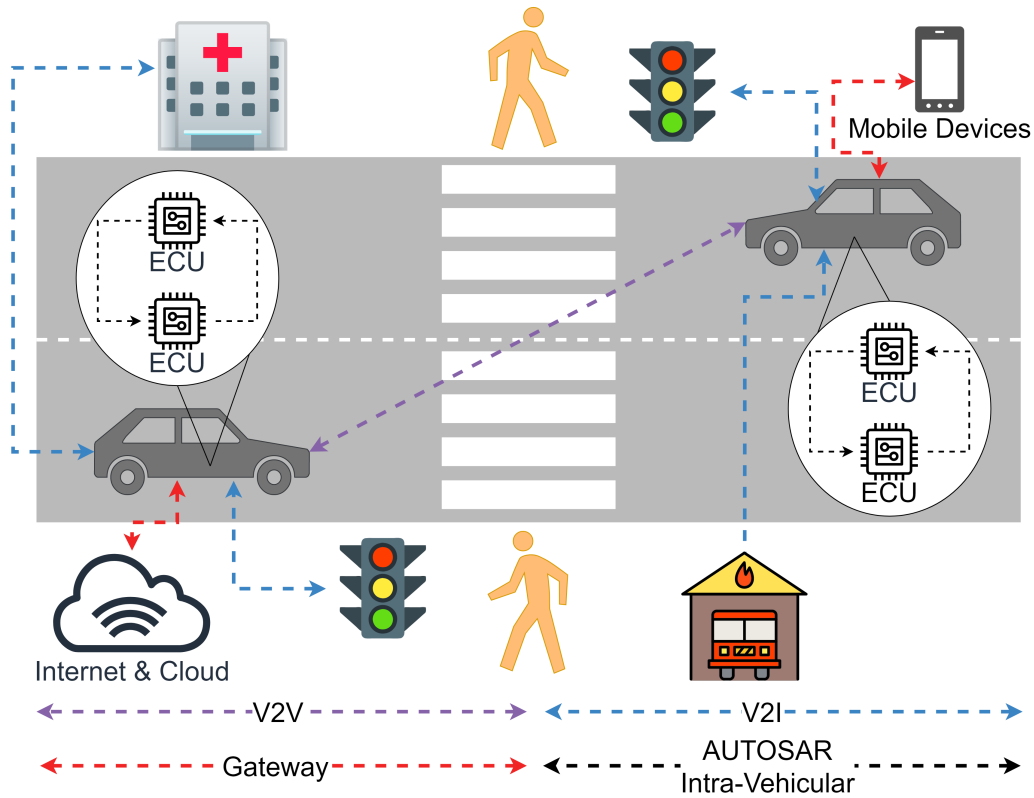


Figure 3.7: Networking in Modern Cars, adapted from [36].

As alluded to by other works uncovered in the literature review, CAN and FlexRay are among the majority of the communication protocols that AUTOSAR applications employ. These network domains aim at establishing *Intra-Vehicle* communication such as *Inter-ECU* and *Intra-ECU communication*. Consequently, Fürst and Bechter [37] deliberated employing V2V and V2I notions

in AUTOSAR-compliant solutions. They explicitly argued that the AUTOSAR *Adaptive Platform* was the most in line with meeting these emerging requirements.

3.5 Summarizing the Results

This section summarizes the performed literature review, disclosing the gathered “research works” and discussing the achieved results. All the cataloging performed in the literature review can be consulted in Appendix A.

Table A.1 conglomerates all the uncovered “research works”, identifying the indexing source and appropriate reference. Furthermore, these works are linked in Table A.2 to their respective authors and year of publication. Subsequently, Table A.3 categorizes the recovered documents according to the *Development*, *Security*, and *Use Cases* classifications. Lastly, Table A.4 highlights whether these documents revolve around concepts of the AUTOSAR *Classic* or *Adaptive Platform*.

As mentioned, the collected data was compiled into the “AUTOSAR in the Smart Cities Era: Current Developments and Research Trends” survey article. The main discussion and conclusions that this literature review yielded concerning the integration of AUTOSAR applications with Smart Cities are well summarized in the document [25]. Nonetheless, some observations still remained that were left out in the final version.

Firstly, the technical reports and a portion of the “research works” focused only on contextualizing the AUTOSAR framework. These documents are usually highly technical in nature and rarely address some concerns and drawbacks that AUTOSAR has or may have. On the other side of the spectrum, conference papers and journal articles merely provide a brief presentation of this tool. Their authors opt to delve into these concepts in minuter detail when addressing a potential hurdle that their proposed solution aims at solving. Some of the themes uncovered centered around potential improvements in the employment and design of the existing architecture and surrounding timing and computational necessities of AUTOSAR applications.

Furthermore, most uncovered “research works” were categorized in the *Development* classification. It was also unveiled that most of the studies delved into concepts related to the *Classic Platform*. These two observations can be explained by the relatively young age of the *Adaptive Platform*. The *Classic Platform* has had more than a decade to cement itself in the automotive industry, especially in accommodating the demands of the leading traditional automotive suppliers. These requirements are typically focused on the functionality of ECU projects. Hence, it is unsurprising that aspects related to security and emerging scenarios take the backstage. Nonetheless, the literature review also revealed interest and migration attempts within the industry in order to face and acknowledge these questions.

Additionally, it is worth mentioning a few reviewed “research works” that stand out among their peers. Firstly, Singh et al. [38] propose a hybrid strategy based on the AUTOSAR *Adaptive Platform* that uses elements of the *Classic Platform* via APIs. This unique approach was the only one that directly employed concepts of both platforms and may pave the way for future solutions.

Moreover, Shaout et al. [39], even though they did not directly mention AUTOSAR, contextualize the ADAS concept in-depth and were considered in the literature review in order to understand the evolution of this modern idea. Finally, it was impossible to precisely determine which AUTOSAR platform was being studied in some documents. Hence, questions arose if these “research works” addressed topics that were shared by both AUTOSAR *Adaptive Platform* and *Classic Platform* or if we were observing an oversight by the respective authors.

As a final remark, the written scientific paper was published in the IEEE Metro Automotive 2023 conference “2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)”, held in Modena, Italy. This paper was successfully presented and received the “Best Paper Presented By A Young Researcher” award.

Chapter 4

Hardware and Software Architecture

The literature review in Chapter 3 unveiled that AUTOSAR defines a methodology for the system and application designers to follow during the development process. In this context, this chapter will present the AUTOSAR methodology and workflow and correlate it with the architecture of the camera ECU regarding both hardware and software. Likewise, the toolkit used for the development of the different implementations in order to achieve the proposed requirements will be detailed. Furthermore, some AUTOSAR concepts regarding the studied communication topology will be explored in more detail. These constitute vital notions, referenced in Chapter 5, associated with a “Control Unit” present in the camera ECU. This specific module was the target of the majority of the development process of the dissertation.

4.1 AUTOSAR Methodology

As explored in Section 3.2.4, AUTOSAR conveys the sequence of steps a designer should take in order to develop AUTOSAR-compliant solutions. This design process was one of the core and earliest concepts tackled in the dissertation and is depicted in Figure 4.1. In the context of automotive-related projects, the *AUTOSAR Methodology* is a critical component that divides the development process between members of the AUTOSAR consortium, particularly OEMs and *Tier 1 Suppliers*.

Firstly, OEMs define the system architecture, requirements, specifications, and configurations of the intended ECUs. Additionally, they delineate the wanted hardware topology and the communication matrix by characterizing which sensors are to be deployed and which signals and messages will be exchanged. This information is then conveyed to the *Tier 1 Suppliers* in a textual format in ARXML files through the *AUTOSAR System Description*. *Tier 1 Suppliers* can be responsible for developing solutions for either an individual or a multitude of sensors, depending on their relationship and trust with the OEM. In particular, projects at CES Porto usually entail the construction of products for one specific sensor.

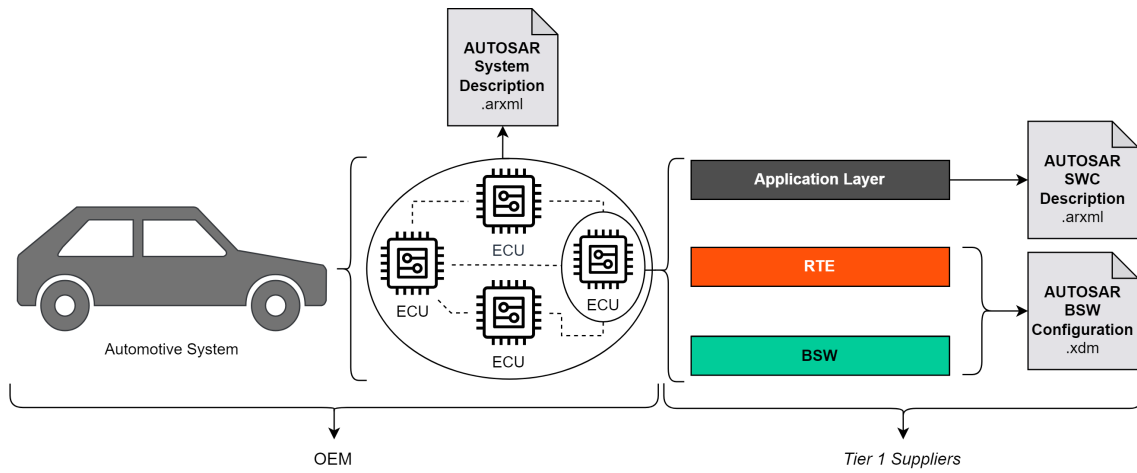


Figure 4.1: AUTOSAR Methodology for OEMs and *Tier 1 Suppliers*.

Afterward, *Tier 1 Suppliers* are then delegated to produce an AUTOSAR-compliant solution with the aid of the *AUTOSAR System Description* file. The development of ECU software configurations and SWCs architecture will be done based on the expected requirements presented in the file. For example, *Tier 1 Suppliers* will create *Runnables* with their respective means of access, such as *Ports*, *Interfaces*, triggering events, and relationships.

This procedure is divided into the different AUTOSAR architecture layers and modules, leading to the creation of additional textual files, as seen in Figure 4.2.

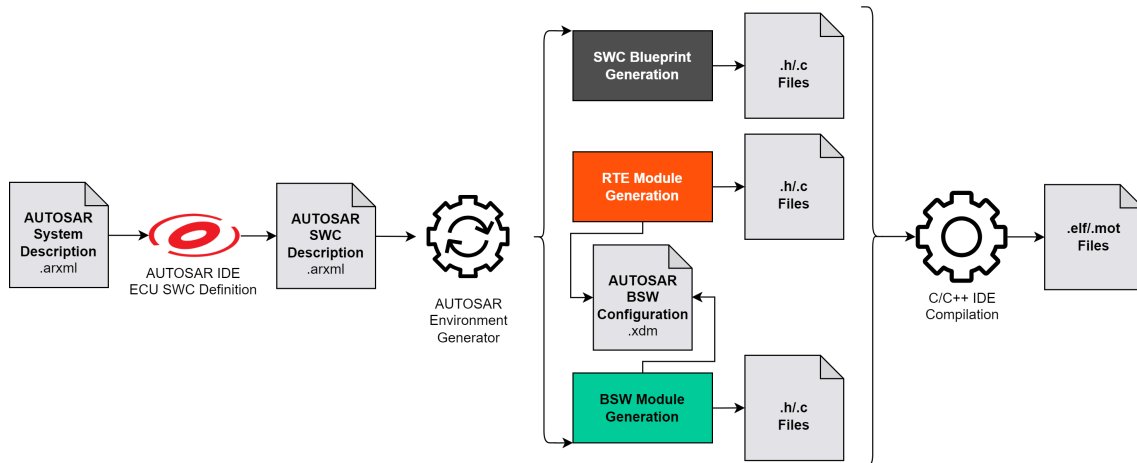


Figure 4.2: Generic AUTOSAR Workflow for *Tier 1 Suppliers*.

For instance, application-level configurations will be compiled in ARXML files creating an *AUTOSAR SWC Description*. After that, an automated AUTOSAR environment generator must be employed to configure RTE and BSW grade components. This tool selects as input the files mentioned above, such as *AUTOSAR System Description* and *AUTOSAR SWC Description*. *Tier 1 Suppliers* can then configure modules present in these lower layers, generate them, and save the end result in a *AUTOSAR BSW Configuration* file [7, 28]. Furthermore, *.h and *.c files

are produced, where *Tier 1 Suppliers* can complete the intended AUTOSAR-compliant solution. Additionally, blueprints of the functions of the *Runnables* of the SWC are equally created for the developer to implement the wanted software logic manually. Finally, the resulting project is then linked and compiled into `*.elf` mappings and `*.mot` executables to be used by the ECU system.

4.2 AUTOSAR Workflow

The dissertation focused on the work delegated to the *Tier 1 Suppliers*, starting with the definition of ARXML files regarding the interaction between a *Runnable* and the RTE. This initial deed and subsequent steps in developing different implementations are portrayed in Figure 4.3, alongside the needed tools.

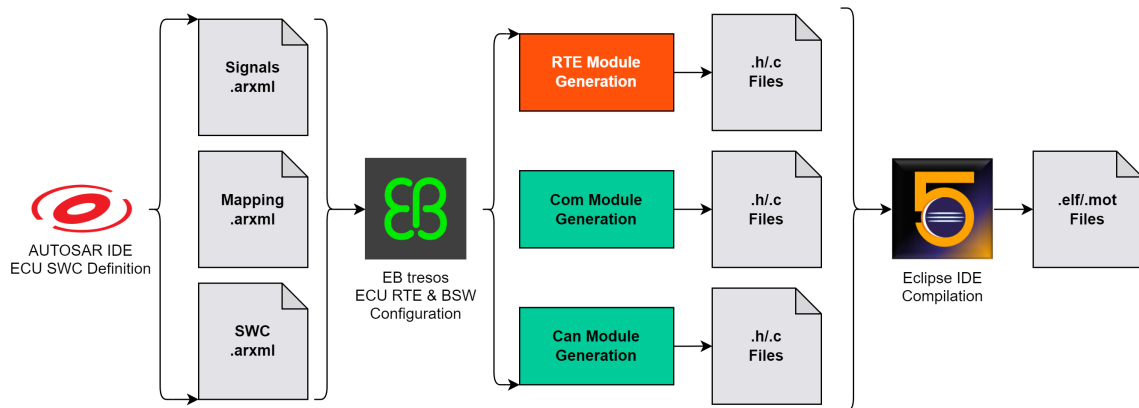


Figure 4.3: AUTOSAR Workflow and Toolkit.

An AUTOSAR-compliant Integrated Development Environment (IDE) was used in describing three separate ARXML files: “Signals.arxml”, “Mapping.arxml”, and “SWC.arxml”. Chapter 5 will further explore these files and their contents.

The next step of the AUTOSAR Methodology revolved around processing the ARXML files through a tool capable of generating AUTOSAR-compliant code. The output code defined the ECU behavior regarding the RTE and BSW and was created in `*.h` and `*.c` files. The engine employed for this task was the EB tresos Autocore, which allowed configurations of each module comprehended inside these layers. Each module is generated separately and then linked together using this tool [40]. As explained in Chapter 2, the RTE, COM, and CAN software modules were the most affected in this context. In this case, the blueprints of the functions of the *Runnables* of the SWC were not generated as they were already present in the camera ECU base project.

The final step was the development of the intended application logic via manual writing with the aid of a C/C++ programming language compatible IDE. Usually, the written code, situated in the *Application Layer*, will use generated API functions created in the preceding phase to communicate with the lower layers. At the end of this process, generated code and source code were compiled and linked into `*.elf` mappings and `*.mot` executables. An Eclipse-based IDE was used for this phase.

4.3 Workbench and Workstation

In conjunction with the software tools mentioned above, a hardware workbench was used in order to test the developed implementations on the camera ECU. The workbench comprises a multitude of state-of-the-art procedures and tools used in the industry. It encompasses the camera connected to a remotely controlled LABPS3005DN Direct Current (DC) power supply (a) [41]. The camera is also linked to a VN1610 CAN Interface (b) that simulates the transmission and reception of signals and messages [42]. This way, the developed implementations could be evaluated as if the camera were deployed in an automotive vehicle. An LA-3500 PowerDebug Module is likewise connected (c) with the job of flashing the camera ECU with `*.elf` mappings and `*.mot` executables [43]. These three components were supervised via a remote PC (d) accessed by the workstation (e) used to develop the software aspect specified before. The software modifications were worked on in a Git domain (f) in order to ease development and track the performed changes. Figure 4.4 presents a diagram of this architecture.

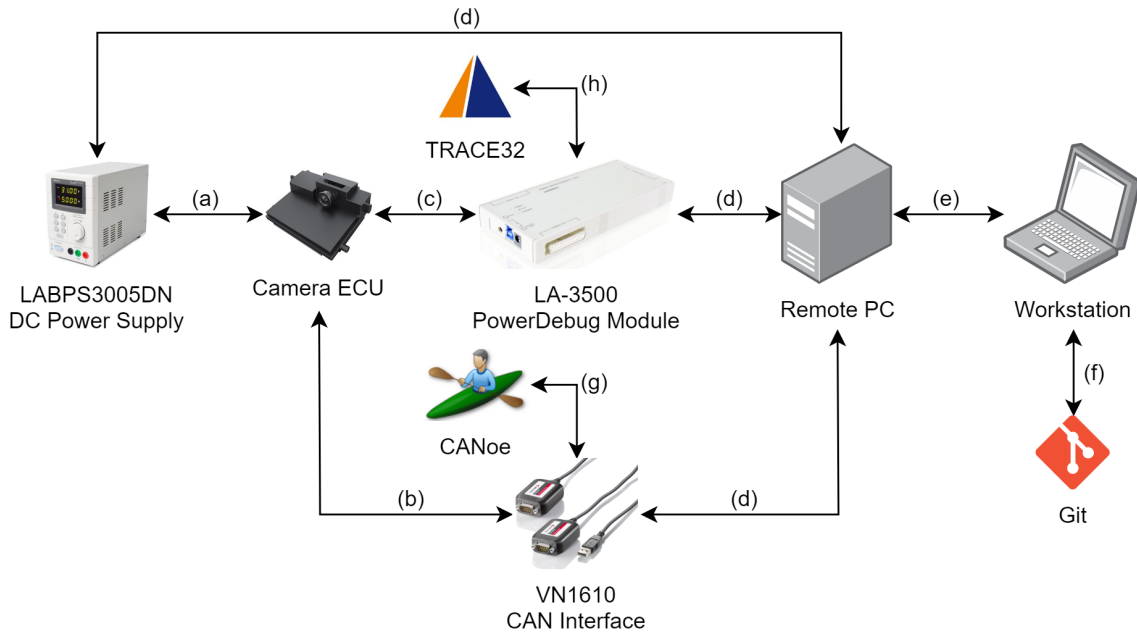


Figure 4.4: Architecture of the Workbench.

Additional software tools were used alongside the already described methodology and toolkit. The CANoe (g) controlled the signals and messages forwarded and received with the VN1610. In parallel, the TRACE32 PowerView (h) flashed and debugged the camera ECU via the LA-3500 [44, 45].

4.4 Control Unit Communication Stack Topology

Beforehand, it is substantial to remember that one of the goals of this dissertation was to study and evaluate the efficiency of several software configurations of AUTOSAR. The main area of

interest was the employment of communications between components in the *Application Layer* and the RTE. In this manner, AUTOSAR-compliant reading methods available in the standard were analyzed and classified according to their advantages and disadvantages.

For this endeavor, a version of the camera ECU base project was made available by CES Porto, specifically, a *Runnable* inserted in a SWC. These modules were modified to investigate how communication with the RTE could be established. The SWC in question was a “Control Unit” responsible for receiving and transmitting several signals and messages while doing proportionate conversion operations. The base project already defined this data structure in the provided *AUTOSAR System Description* ARXML file.

The “Control Unit” is the first SWC of a plethora of SWCs that make up the architecture of the camera ECU. This component receives information from the physical bus of the camera ECU, processes the retrieved data, and forwards it to other SWCs. The inverse is also applied, being responsible for relaying data from SWCs through the lower layers to the physical bus. Therefore, this component occupies a critical position and role in the hierarchy of the camera ECU.

Figure 4.5 shows the data flow between a *Runnable* present in the “Control Unit” SWC and the physical bus. This specific *Runnable* is tasked with implementing reading methods capable of obtaining the signals emitted by the lower layers and interpreting them. The road of AUTOSAR communication modules and architecture layers that this information goes through is also shown.

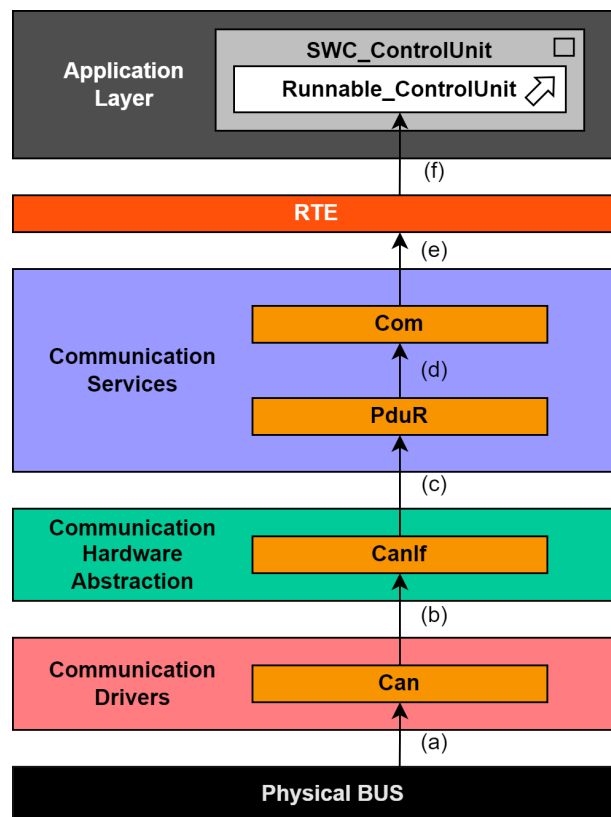


Figure 4.5: Data Flow associated with Transmitted Signals.

As explained in Section 2.2.2, signals sent by the physical bus must pass along several AUTOSAR modules until the *Application Layer* is reached. The preceding unidirectional example represents how signals reach a *Runnable* in the “Control Unit” SWC, referred to herein as “Runnable_ControlUnit” and “SWC_ControlUnit”. This data flow involves intrinsic configurations and interactions between several modules that help achieve a high abstraction level of the AUTOSAR architecture.

Firstly, the camera sensor transmits raw signals in a frame format via the CAN bus protocol to the CAN Driver (a). Then, the CAN Driver evaluates the received data frame, extracting relevant information to be forwarded to the CANIf (b). Afterward, the CANIf converts this information into AUTOSAR PDUs and forwards them to the PDuR (c). Subsequently, the PDuR receives the PDUs, transforms them into I-PDUs, and then reroutes them into the appropriate destination, which in this case is the COM module (d). Consequently, the COM receives I-PDUs signals from the PDuR, remodels them into RTE-compatible signal buffers, and delivers them to the RTE (e). Finally, the RTE delivers this data to the receiving “Runnable_ControlUnit” of the “SWC_ControlUnit” (f).

The “Runnable_ControlUnit” represents the *Receiver* aspect of a *Sender-Receiver* topology with *Inter-ECU communication*. Figure 4.6 illustrates the AUTOSAR-compliant software architecture of the “Runnable_ControlUnit” and “SWC_ControlUnit” in question. It also reminds the ARXML files that were configured to develop the reading methods. More specifically, a SWC with a singular *Runnable* was used to analyze the dynamic of 20 signals conveyed between the *Application* and RTE layers.

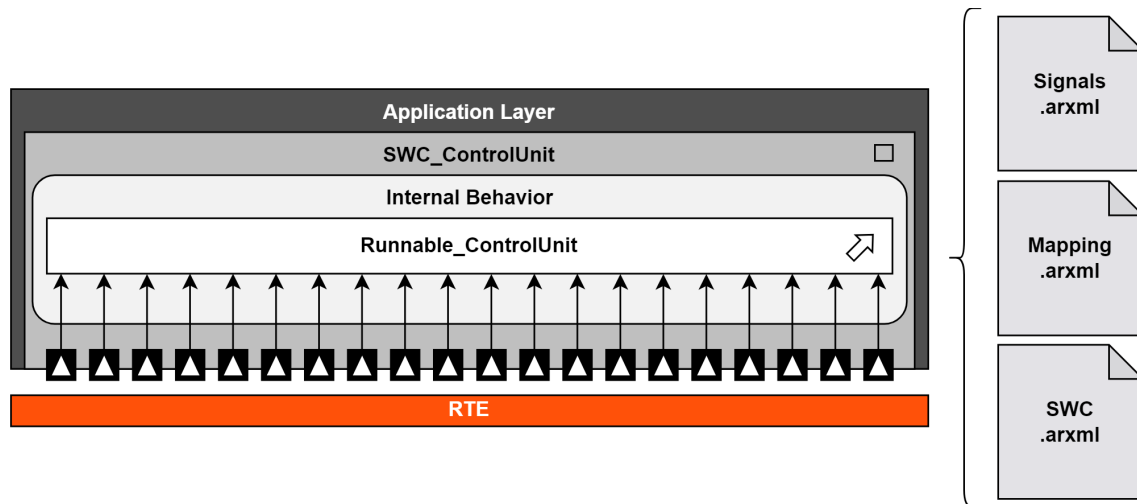


Figure 4.6: Software Architecture of the Modules within the “Control Unit” SWC.

In this particular case, information coming from the RTE is received by 20 *Service R-Ports*, as they are expected to obtain information from the lower layers and not directly from other SWCs. Then, *AUTOSAR Interfaces* convey this data exchange in the format of *data elements* to the “Runnable_ControlUnit”. Consequently, developers can process received camera sensor signals at the *Application Layer*. The next chapter will address the AUTOSAR-compliant reading methods explored and applied in this stage.

The opposite data flow of transmitting signals to the physical bus is tasked to other *Runnables* present in the “SWC_ControlUnit”. With this in mind, it was decided to focus exclusively on the reception aspect of *Inter-ECU communications*. This decision was considered within the context and goals of the dissertation. The motivating reasons were the inherent complexity associated with working with the AUTOSAR framework and the initial goal of centralizing the study on exploring reading methods. Hence, *Sender*-related concepts were abstracted by operating with the existing structure of the base project.

Chapter 5

Software Implementations

This chapter will explore alternative reading methods related to *Inter-ECU communication* between SWCs that the AUTOSAR *Sender-Receiver* paradigm supports. Their associated characteristics and the way they are implemented will be addressed. The differences between *Explicit* and *Implicit* communication, which AUTOSAR divides *Sender-Receiver* architectures, will be highlighted. Consequently, the following subsections will explain the explored reading methods regarding their purpose and generation process.

5.1 *Explicit* and *Implicit* Communication

Upon analyzing the referenced AUTOSAR documentation, it was uncovered that the standard specifies two types of communication for *Sender-Receiver* architectures, *Explicit* and *Implicit*. The difference between these topologies lies in how *Runnables* access *data elements* coming from the RTE, as shown in Figures 5.1 and 5.2 [12, 46].

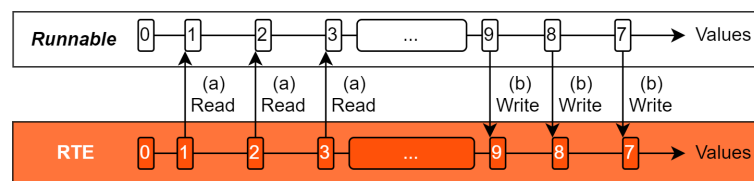


Figure 5.1: *Explicit* Communication.

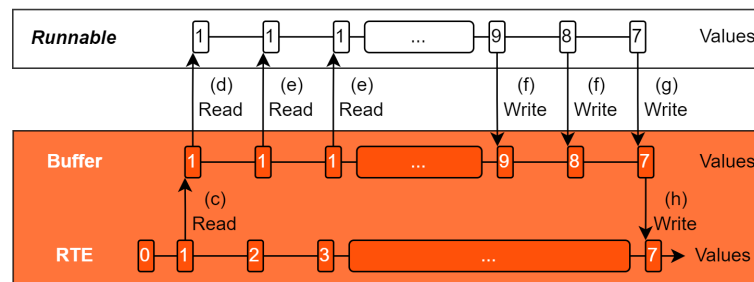


Figure 5.2: *Implicit* Communication.

In *Explicit*, *Runnables* can directly retrieve and transmit RTE data using appropriate reading (a) and writing (b) methods. Contrarily, in *Implicit*, the data recovered from the RTE is placed in an intermediate buffer before the *Runnable* starts (c). After this, the *Runnable* can extract the information preserved within the buffer (d). Even if the *Runnable* requests more read operations (e), these will return the deposited initial value. A similar logic applies if we are writing data (f), where the buffer only communicates the utmost written value (g) to the RTE after the *Runnable* has ended (h).

Within these two paradigms, AUTOSAR defines several API functions that request information from the RTE in the format of either individual or grouped signals. Table 5.1 structures these reading methods according to their communication type.

Table 5.1: *Sender-Receiver* API Functions.

<i>Sender-Receiver</i>	<i>Explicit</i>	Rte_Read
		Rte_DRead
		Rte_Receive
		Rte_IrvRead
	<i>Implicit</i>	Rte_IRead
		Rte_IReadRef
		Rte_IrvIRead
		Rte_IrvIReadRef

Four of these eight techniques were selected to be explored and implemented, more specifically, the `Rte_Read`, the `Rte_IRead`, the `Rte_DRead`, and the `Rte_Receive`. Their definitions, uniqueness, and characteristics defined by the AUTOSAR standard were compelling motivations. For instance, the `Rte_IReadRef` was not examined due to its AUTOSAR definition being too similar to the `Rte_IRead`. Additionally, the documentation offered by the standard did not specify sufficient information concerning its generation. The remaining API functions are associated with *Inter-Runnable* data, used to share information between *Runnables* inside a SWC. These were not considered because the scope of this study dealt with information coming from the lower layers.

5.2 ARXML Configurations

The automatic generation of these API functions depends on the needed specifications of ARXML meta-classes. These are defined through the AUTOSAR methodology in ARXML files, as was explained in Section 4.1. The three following files were configured:

- **“Signals.arxml”**: *AUTOSAR System Description* file provided by an OEM to CES Porto. The messages that the camera sensor will manage are specified here. They can be outlined as `SystemSignal` or `SystemSignalGroup`, referred to herein as “signal” and “signal

group”. A “signal” represents an individual *data element* within an AUTOSAR system, while a “signal group” describes a collection of “signals”. From a programming language perspective, these classifications can be seen as analogous to C/C++ programming variables, e.g., a sole `integer` and a `struct`. The original file provided by the OEM entailed a “signal group” topology.

- **“SWC.arxml”**: File belonging to the *AUTOSAR SWC Description*. All specifications concerning the “SWC_ControlUnit” such as the “Runnable_ControlUnit”, *AUTOSAR Interfaces*, *Service R-Ports*, *Internal Behavior*, and internal RTE-compatible reading methods are done in this file.
- **“Mapping.arxml”**: The bridge between the “Signals.arxml” and the “SWC.arxml” files. It links the *Service R-Ports* and “signals” or “signal groups”. Associating the appropriate “signals” or “signal groups” with the relevant *Service R-Ports* enables the correct transmission and reception of data within the AUTOSAR system. It equally belongs in the *AUTOSAR SWC Description*.

Even a slight change in the ARXML configurations leads to the creation and modification of a multitude of RTE related `*.h` and `*.c` files. However, nearly all of this dissertation was focused on the `Rte.c` and `Rte_SWC.h` files, as these were the most critical for the study in question.

Some ARXML configurations behind these modules are shared between each use case, e.g., the creation of *Runnable* instances. Hence, attention was devoted to expressing these acute concepts clearly, concisely, and coherently. For instance, each AUTOSAR ARXML meta-class contains a plethora of attributes, such as the name of the class (`ShortName`), that developers can fulfill. The diagrams presented in this chapter will highlight the most impactful for understanding the developed configurations.

Once the wanted API functions were successfully generated, the C/C++ source code of the “Runnable_ControlUnit” function blueprint could be modified. The newly discovered reading methods were integrated, more specifically, the reception of various signals derived from 20 separate messages. The incorporation of the different generated API functions differed according to the existence of individual “signals” and “signal groups”.

Unfortunately, the ARXML configurations and modified source code cannot be disclosed since this dissertation was done in a corporate environment, which entails confidentiality concerns from the company. Nonetheless, brief abstracted examples of how these modifications were done are shown in Appendix B.

5.2.1 Rte_Read

The `Rte_Read` API function requests information with *data semantics* from the RTE. *Data semantics* means the *data element* in question is “last-is-best”. That is, the value is consistently the latest returned from the RTE [11, 12, 46]. Figure 5.3 displays the configurations made in ARXML at the SWC level.

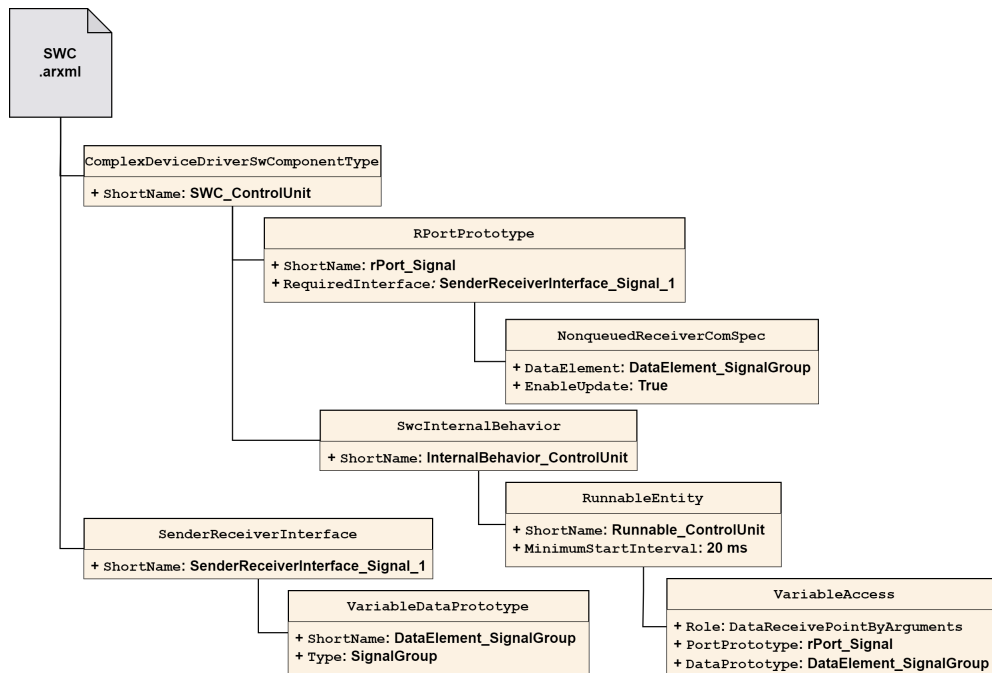


Figure 5.3: ARXML Hierarchy of the “SWC.arxml” file with Rte_Read API Function.

Firstly, the `ComplexDeviceDriverSwComponentType` class creates the SWC instance “SWC_ControlUnit”. Alongside this step, the `SenderReceiverInterface` class defines the *AUTOSAR Interface* for each “signal group”. Then, the `VariableDataPrototype` couples *data elements* to the structure and type of the wanted “signal groups” stored in the “Signals.arxml” file. This disposition is depicted in ARXML textual format in Listing B.1.

Afterward, the corresponding *R-Ports* are built with the `RPortPrototype` and linked with the previously mentioned *AUTOSAR Interfaces*. Additionally, a `NonqueuedReceiverComSpec` class was created in order to monitor if the value of the matching *data element* has been updated with the `enableUpdate` attribute. This behavior is concurrent with the *Explicit* topology and allows efficient data handling for developers in the *Application Layer*.

At the same time, the *Internal Behavior* and *Runnable* components are defined with the `SwcInternalBehavior` and `RunnableEntity` classes. Lastly, a `VariableAccess` class instance is created for a “signal group” with the `dataReceivePointByArgument` attribute. This last action is responsible for creating the `Rte_Read` API function. Taking account the requirements of the camera ECU, a 20 millisecond interval `minimumStartInterval` is defined in order to ensure that the *Runnable* is not initiated again before the conclusion of its previous execution.

Moreover, as shown in Figure 5.4, these “signal groups” are mapped in the “Mapping.arxml” file with the `SenderReceiverToSignalGroupMapping`.

The `SenderReceiverToSignalGroupMapping` class uses `VariableDataPrototype` and `SenderRecRecordElementMapping` classes to map a *Sender-Receiver* communication *data element* to the respective “signal group” and *R-Port*.

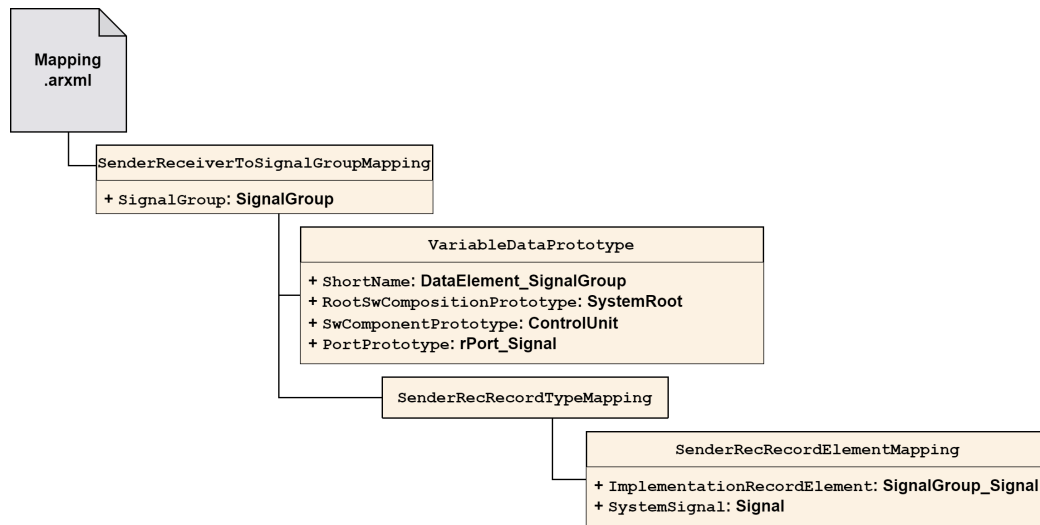


Figure 5.4: ARXML Hierarchy of the “Mapping.arxml” file with SystemSignalGroup.

Lastly, the provided “Signals.arxml” file creates the SystemSignalGroup and associated SystemSignal instances, as represented in Figure 5.5. The data type of the corresponding ‘signals’ is also defined with the SwDataDefPropsConditional, SwDataDefProps, and ImplementationDataType classes.

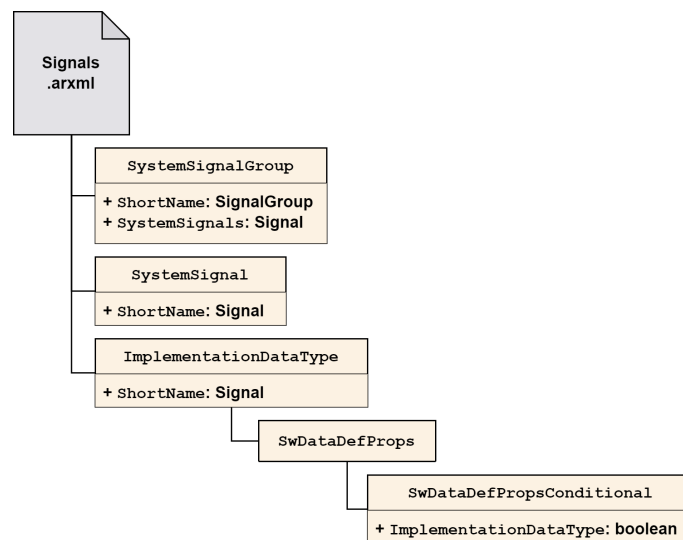


Figure 5.5: ARXML Hierarchy of the “Signals.arxml” file with SystemSignalGroup.

Consequently, these ARXML specifications resulted in the generation of the corresponding API function in the Rte.c file, as displayed in Listing 5.1. It is also noteworthy to mention that subsequent API functions will share some of these ARXML specifications.

```

1 FUNC(Std_ReturnType, RTE_CODE)
    Rte_Read_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup (P2VAR(
        SignalGroup, AUTOMATIC, RTE_APPL_DATA) data)
2 {
  
```

```

3  Std_ReturnType Rte_Status = RTE_E_OK;
4
5  Rte_ReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup_Start(data);
6  Rte_IntLock();
7  Rte_ReceiveBuffer_SignalGroup_isUpdated = FALSE;
8  *data = Rte_ReceiveBuffer_SignalGroup_value;
9  Rte_Status = Rte_ReceiveBuffer_SignalGroup_status;
10 Rte_IntUnlock();
11 Rte_ReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup_Return(data);
12
13 return Rte_Status;
14 }

```

Listing 5.1: Generated Rte_Read API Function.

As observed, this function updates a global pointer with the received “signal group” data and returns a status code that indicates the state of the communication with the RTE. Table 5.2 discloses the several status and error codes that AUTOSAR defines for the reception side for *Sender-Receiver Inter-ECU communication*.

Table 5.2: RTE Status and Error Codes, adapted from [12, 46].

Status Code	Value	Definition
RTE_E_OK	0	No error occurred
RTE_E_INVALID	1	Invalid data received
RTE_E_LOST_DATA	64	Incoming data has been partly lost
RTE_E_COM_STOPPED	128	COM service is not available
RTE_E_TIMEOUT	129	No data was received and no other error was detected
RTE_E_LIMIT	130	An internal RTE limit was exceeded
RTE_E_NO_DATA	131	No data was received
RTE_E_NEVER_RECEIVED	133	No data was received since system start
RTE_E_UNCONNECTED	134	<i>R-Port</i> is not connected
RTE_E_COM_BUSY	141	COM service is already being used by another transmission/reception

The generated Rte_Read function is then integrated into the “Runnable_ControlUnit” function blueprint. Listing B.2 displays an abstracted version of this arrangement for one particular “signal group”. An analogous procedure was performed for all 20 messages, only differing concerning the specific characteristics of the API function in question.

5.2.2 Rte_IRead

The succeeding explored API function was the Rte_IRead with the configurations displayed in Figure 5.6. In order to implement this reading method, a dataReadAccess attribute of the VariableAccess class was defined in the ARXML model. The remainder configurations detailed in the preceding subsection were unaltered. However, the enableUpdate option in the NonqueuedReceiverComSpec class, defined in the *R-Port*, was disabled. This step was necessary in order to comply with the *Implicit* topology to which this API function belongs.

These changes generated code in the Rte_SWC.h and Rte.c files which, as perceived in Listing 5.2, updates a pointer with the received “signal group”.

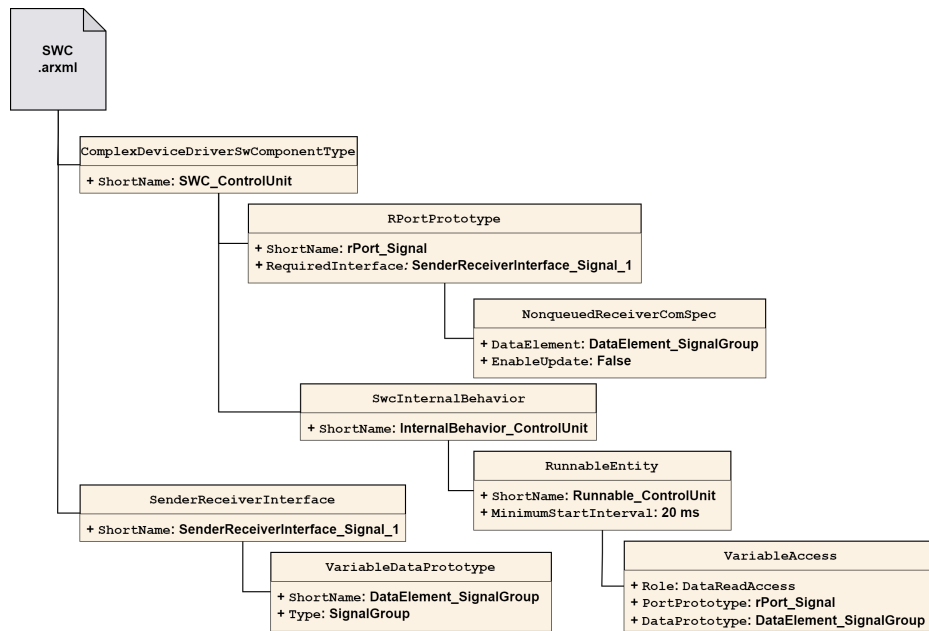


Figure 5.6: ARXML Hierarchy of the “SWC.arxml” file with Rte_IRead API Function.

```

1 #define Rte_IRead_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup()
2 (
3   Rte_IReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup_Start((
4     P2CONST(SignalGroup, AUTOMATIC, RTE_APPL_DATA)) &Rte_Inst_SWC_ControlUnit->
5     SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup->value),
6     Rte_IReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup_Return((
7       P2CONST(SignalGroup, AUTOMATIC, RTE_APPL_DATA)) &Rte_Inst_SWC_ControlUnit->
8       SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup->value),
9     (P2CONST(SignalGroup, AUTOMATIC, RTE_APPL_DATA)) &Rte_Inst_SWC_ControlUnit->
10     SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup->value
11 )
12
13 #define Rte_IStatus_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup()
14 Rte_Inst_SWC_ControlUnit->SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup
15   ->status

```

Listing 5.2: Generated Rte_IRead and Rte_IStatus API Functions

Several additional intermediate buffers, concordant with the *Implicit* topology, were equally created. Furthermore, the Rte_IStatus function was generated alongside the Rte_IRead. This API obtains the status and error code, akin to the return value of the Rte_Read. These two functions were employed to maintain as much coherence as possible regarding the logic between the various implementations.

5.2.3 Rte_DRead

The Rte_DRead was the subsequent API function that was studied. Similarly to previous implementations, an attribute of the VariableAccess class had to be defined in the ARXML model,

in this case, `dataReceivePointByValue`.

Regrettably, it was discovered that this solution was only compatible with primitive *data elements* (integer, boolean, and other types up to 32 bits). This limitation meant that the original “signal group” configuration present in the base project was not supported. This constraint was overcome by eliminating the `SystemSignalGroups` defined in the *AUTOSAR System Description* file, as depicted in Figure 5.7. Likewise, the remainder configurations in the “SWC.arxml” had to be changed since we dealt with individual *data elements*, one per “signal”. Another impediment discovered was the incompatibility of the occurrence of the `NonqueuedReceiverComSpec` inside the `RPortPrototype`. Hence, this class was removed, as observed in Figure 5.8.

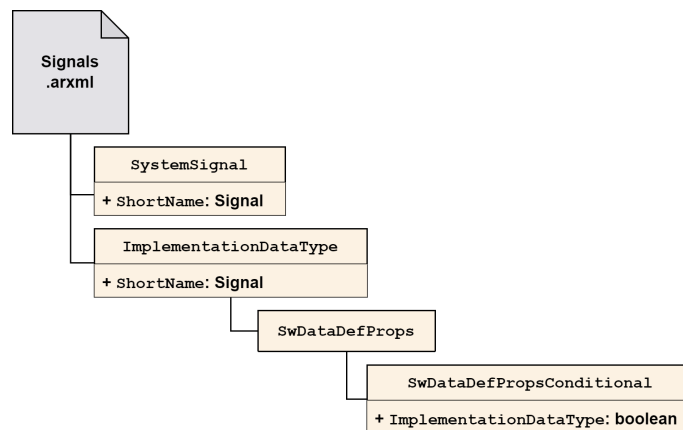


Figure 5.7: ARXML Hierarchy of the “Signals.arxml” file without `SystemSignalGroup`.

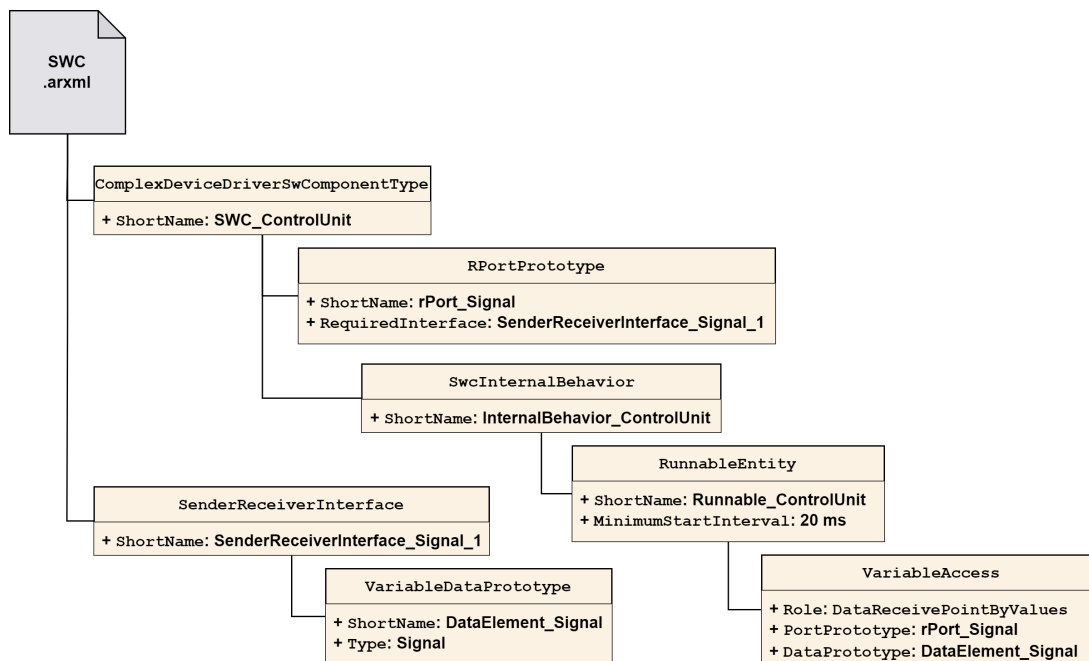


Figure 5.8: ARXML Hierarchy of the “SWC.arxml” file with `Rte_DRead` API Function.

Furthermore, the `SenderReceiverToSignalGroupMapping` class was also switched in

the “Mapping.arxml” file for the `SenderReceiverToSignalMapping` class in order to directly link the `SystemSignals`, as modeled in Figure 5.9.

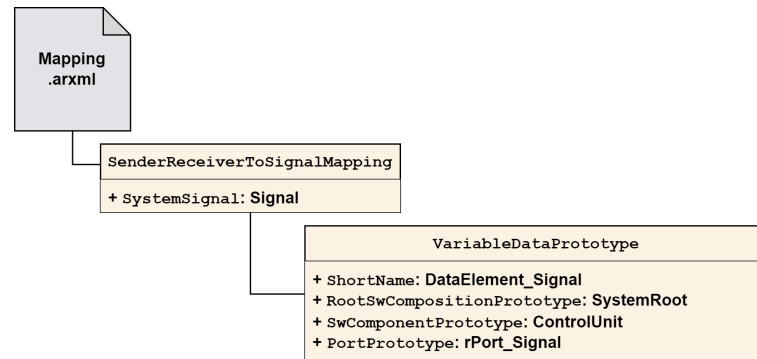


Figure 5.9: ARXML Hierarchy of the “Mapping.arxml” file with `SystemSignal`.

These changes generated differences in files related to the COM module and the `Rte.c` file, with the latter shown in Listing 5.3.

```

1 FUNC(Signal, RTE_CODE) Rte_DRead_SWC_ControlUnit_Runnable_ControlUnit_rPort_Signal
   (void)
2 {
3   Signal data;
4
5   Rte_DReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_Signal_Start(data);
6   (void) Com_ReceiveSignal(ComConf_ComSignal_Signal, &data);
7   Rte_DReadHook_SWC_ControlUnit_Runnable_ControlUnit_rPort_Signal_Return(data);
8
9   return data;
10 }
  
```

Listing 5.3: Generated `Rte_DRead` API Function.

In contrast to other described API functions, the `Rte_DRead` directly returns the “signal” information associated with the *data element*. Woefully, it does not convey any information regarding the status of the RTE.

5.2.4 `Rte_Receive`

The last API function analyzed was the `Rte_Receive`, which adopts a queue file following a First-In-First-Out (FIFO) logic for individual “signals” [11, 12, 46]. A FIFO topology follows the principle of processing data in the order of arrival to ensure the preservation of the initial assortment. The latest data received are added to the end of the associated queue, whereas the oldest data are processed at the front.

Coincidentally, as described in the `Rte_Read`, the `dataReceivePointByArgument` attribute of the `VariableAccess` class is used, as exhibited in Figure 5.10. This API function does not support “signal groups”, so the treatment mentioned in the `Rte_DRead` was also performed. The `NonqueuedReceiverComSpec` class defined in the *R-Ports* was replaced since

this reading method utilizes a queue paradigm. In its place, a `QueuedReceiverComSpec` class was added, where the queue size was also outlined. Additionally, the `swImplPolicy` attribute of the “signal” in question had to be set to `Queued` in the “Signals.arxml” file, as seen in Figure 5.11.

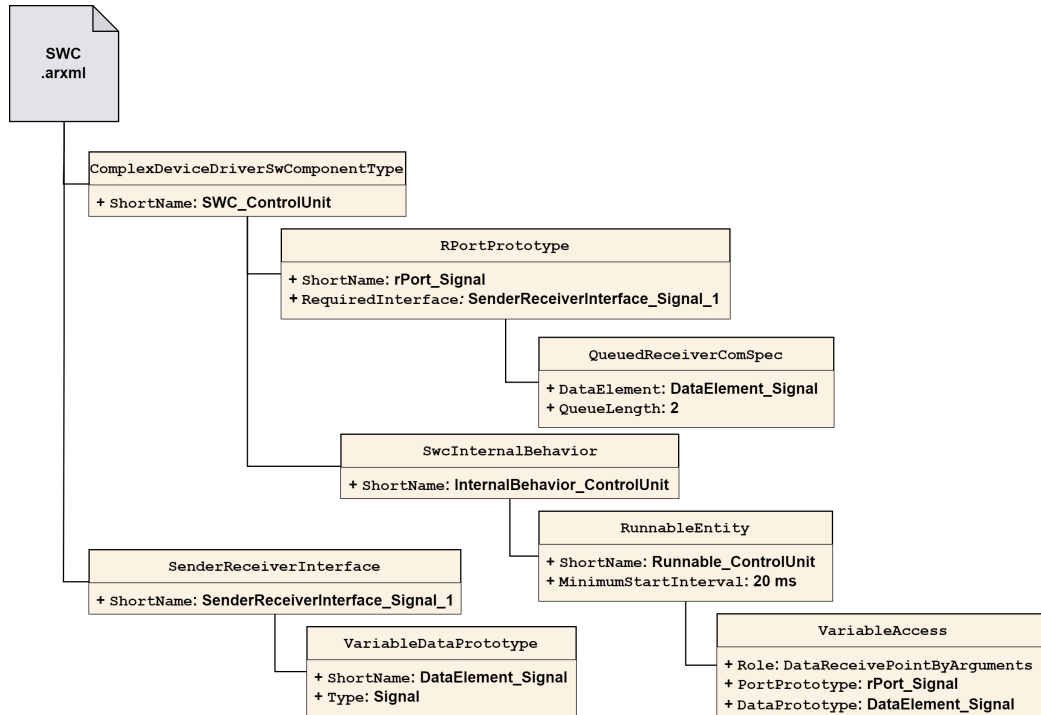


Figure 5.10: ARXML Hierarchy of the “SWC.arxml” file with `Rte_Receive`.

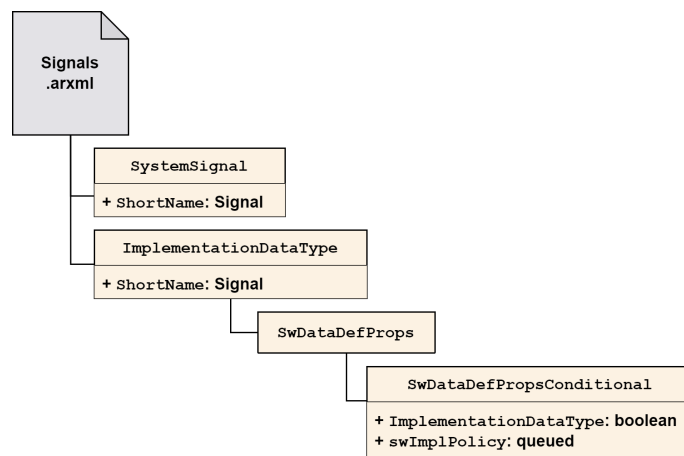


Figure 5.11: ARXML Hierarchy of the “Signals.arxml” file with `Queued` attribute.

Similar to the implementations mentioned above, these configurations generated files related to the COM module and the `Rte.c` file. In the latter, several intermediate buffers were created to store the queues of each “signal”. Listing 5.4 shows the generated code that, as visible, is strikingly similar to the `Rte_Read`.

```

1 FUNC (Std_ReturnType, RTE_CODE)
    Rte_Receive_SWC_ControlUnit_Runnable_ControlUnit_rPort_Signal (P2VAR(Signal,
    AUTOMATIC, RTE_APPL_DATA) data)
2 {
3     Std_ReturnType Rte_Status = RTE_E_OK;
4
5     Rte_ReceiveHook_SWC_ControlUnit_rPort_Signal_Start (data);
6     Rte_IntLock();
7
8     if (Rte_ReceiveQueue_Buffer_Signal.head == Rte_ReceiveQueue_Buffer_Signal.tail)
9     {
10         Rte_Status = RTE_E_NO_DATA;
11     }
12     else
13     {
14         if (Rte_ReceiveQueue_Buffer_Signal.ldcReader != Rte_ReceiveQueue_Buffer_Signal.
            ldcWriter)
15         {
16             Rte_Status |= RTE_E_LOST_DATA;
17             Rte_ReceiveQueue_Buffer_Signal.ldcReader = Rte_ReceiveQueue_Buffer_Signal.
            ldcWriter;
18         }
19         Atomics_ThreadFence();
20         *data = Rte_ReceiveQueue_Buffer_Signal.queue[Rte_ReceiveQueue_Buffer_Signal.
            head % 2U];
21         Atomics_ThreadFence();
22         Rte_ReceiveQueue_Buffer_Signal.head = ((Rte_ReceiveQueue_Buffer_Signal.head + 1
            U) % 4U);
23     }
24     Rte_IntUnlock();
25     Rte_ReceiveHook_SWC_ControlUnit_rPort_Signal_Return (data);
26
27     return Rte_Status;
28 }

```

Listing 5.4: Generated Rte_Receive API Function.

5.2.5 Comparison Between “Signals” and “Signal Groups”

After analyzing the steps needed to generate the preceding API functions and the subsequent results, questions and interest began to appear regarding the impact of the message design specified in the *AUTOSAR System Description* file. What impacts could a `SystemSignal` class-based design entail compared to a `SystemSignalGroup` class-based design?

For instance, a `SystemSignal` class-based style drastically increases the number of necessary reading methods inside a *Runnable* compared to a `SystemSignalGroup` class-based pattern. Conversely, a `SystemSignalGroup` class-based system congregates several *data elements*.

While this option reduces the number of API function invocations, these buffers can entail unnecessary information for a specific use case. E.g., a “signal group” of 10 “signals” is an inefficient memory usage for a developer that is merely interested in the information concerning one or two “signals”.

In this manner, it was decided to develop a fifth implementation, which would encompass the generation of the `Rte_Read` API function. However, it would employ individual `SystemSignal` class “signals” mapped with the `SenderReceiverToSignalMapping` class. The resulting generated code in Listing 5.5 shows the different output. This case study will be referred to herein as `Rte_Read_WS`.

```

1 FUNC(Std_ReturnType, RTE_CODE)
    Rte_Read_SWC_ControlUnit_Runnable_ControlUnit_rPort_Signal_1 (P2VAR(Signal,
    AUTOMATIC, RTE_APPL_DATA) data)
2 {
3     Std_ReturnType Rte_Status = RTE_E_OK;
4
5     Rte_ReadHook_SWC_ControlUnit_rPort_Signal_Start(data);
6     (void)Com_ReceiveSignal(ComConf_ComSignal_Signal, data);
7     Rte_ReadHook_SWC_ControlUnit_rPort_Signal_Return(data);
8
9     return Rte_Status;
10 }

```

Listing 5.5: Generated `Rte_Read` API Function without “Signal Groups”.

Similarly to the `Rte_DRead` and the `Rte_Receive`, this modification yielded several file changes concerning the COM module. This behavior was attributed to the performed switch from “signals” to “signal groups”. Unfortunately, those differences could not be explored in greater detail because of the abstraction underlined in the generation process.

Chapter 6

Examinations, Results, and Discussion

After completing the implementation steps, tests were performed in order to validate them and record run-time performance metrics regarding the execution time. Hence, this chapter presents the idealized test environment, achieved results, and discussions surrounding the developed solutions. The original goal was to categorize the studied reading methods according to their advantages and disadvantages with automated and effective techniques of measuring the obtained results. Additionally, there was a particular interest by the team at CES Porto in discovering an optimal AUTOSAR-compliant reading method.

6.1 Test Environment

A strategy and approach had to be devised in order to quantify and evaluate the impact of the various implementations that were developed. After thoughtful deliberation and assessment of the available tools, it was decided to use the TRACE32 debugger device to extract run-time measurements regarding the execution time. Besides being used to flash the camera ECU, this framework provides various tools for software developers to analyze and debug their implementations in real-time. Furthermore, it allows the development of PRACTICE .cmm script files for automated studies, tests, and data export [47]. The exported data were thoroughly analyzed with the Microsoft Excel tool with Visual Basic for Application (VBA) scripts. Figure 6.1 illustrates the mentioned approach.

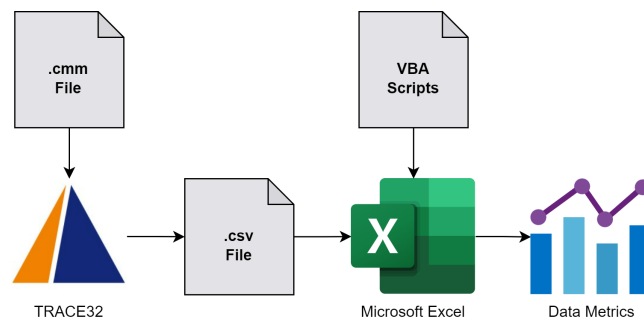
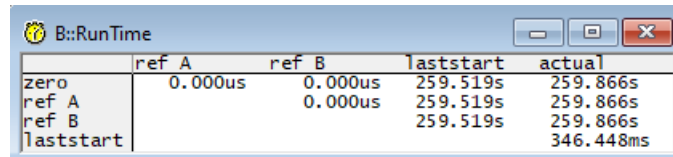


Figure 6.1: Envisioned Strategy for Data Gathering.

The first step was guaranteeing that the information the camera ECU received aligned with the data delivered via the CANoe simulation. Therefore, TRACE32 was used in order to verify that the data yielded by the various reading methods were congruent with the values of the messages being transmitted by the CANoe. Consequently, with the confirmation that the API functions were operating correctly, a procedure had to be devised so as to measure the execution time of each implementation.

After consulting TRACE32-related documentation, an action plan was devised based on employing *breakpoints*. *Breakpoints* are a debugging feature that allows developers to pause the execution of a program at a specific point in the code. The program will stop running when it reaches the designated instruction in which the *breakpoint* was set. Additionally, TRACE32 supports the RunTime window that can obtain the time the code takes to execute, as seen in Figure 6.2. Likewise, TRACE32 supports data export to .csv files with the PRinTer.FILE command [48, 49, 50].



	ref A	ref B	laststart	actual
zero	0.000us	0.000us	259.519s	259.866s
ref A		0.000us	259.519s	259.866s
ref B			259.519s	259.866s
laststart				346.448ms

Figure 6.2: RunTime Window.

In its default configuration, the TRACE32 debug driver monitors the internal state of the CPU 10 times/second while it is running. It is proficient in checking the values of CPU registers, memory locations, and other relevant information to track CPU-related activity. Hence, *breakpoints*, RunTime, and PRinTer.FILE command were combined as a means to evaluate the cyclic and periodic execution model into which the camera ECU is inserted [50].

As a result, it was decided to place *breakpoints* in the “Runnable_ControlUnit” function at the beginning and end. When the code hits the first *breakpoint*, the RunTime will measure the time it took for the “Runnable_ControlUnit” to be activated within the *Runnable* hierarchy of the base project. Subsequently, the program is resumed with the Go command. After the second *breakpoint* is reached, the measured laststart timestamp is extracted as this value represents the time difference between the start and finish of the “Runnable_ControlUnit” function.

With this acquired knowledge, the Measurements.cmm script file displayed in Listing C.1 was created in order to obtain real-time measurements. After discussing a quality metric that could represent an adequate analyzable sample size, it was decided that 500 samples would be considered for each implementation.

Moreover, it was discovered that the measurement error could be reduced by setting the SYStem.POLLING command to CONTinuous. This setting allows the debug driver to constantly monitor the state of the CPU in intervals of 1 millisecond. The readings obtained with this configuration were more accurate than the original disposition. Additionally, it was necessary to use the WAIT and STATE.RUN commands to ensure the program is resumed only after the first *breakpoint*

is reached. Finally, the script file writes the contents obtained by the `RunTime` command when the code stops at the second *breakpoint* and appends them to a specified `.csv` file path [51, 52, 53].

The `.csv` files were manually combined in separate `.xslm` files, where the results for each individual API function could be evaluated. Unfortunately, the `PRinTer.FILE` only supports data writing concerning the entirety of the `RunTime` window, not individual specified values. For instance, `.csv` files were created with the data displayed in Listing C.2. Similarly, the associated inaccuracy of these measurements could not be quantified due to TRAC32-related limitations.

Thus, VBA scripts were used in order to extract the wanted `laststart` timestamp measurements provided by the `RunTime` window. Afterward, the measured execution times were evaluated with the `MAX`, `MIN`, and `AVERAGE` functions in order to draw metrics regarding the maximum, minimum, and average values. These scrips are exhibited in Listings C.3 and C.4.

6.2 Execution Time Measurements

Table 6.1 shows the computed calculations, while Figure 6.3 illustrates the “box and whisker” graphical representation concerning the measured run-time values for each API function. This depiction presents the respective data distribution, highlighting the average values and their spread.

Table 6.1: Metrics of the Execution Times in Run-Time.

	Rte_Read	Rte_IRead	Rte_DRead	Rte_Receive	Rte_Read_WS	Units
MAX	1.121	2.089	0.991	1.555	1.912	ms
MIN	0.520	0.518	0.518	0.727	0.506	ms
AVERAGE	0.651	0.784	0.637	0.952	0.790	ms

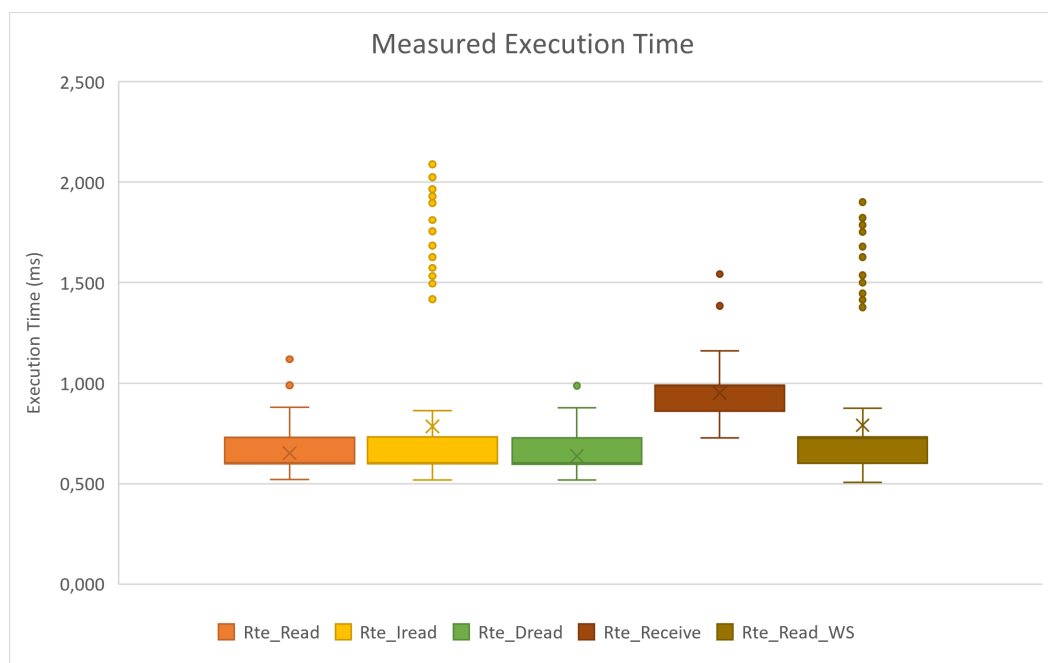


Figure 6.3: Measured Execution Times in Run-Time.

The spread (size of the box) of the obtained results was very similar, with most of the 500 measured samples fitting in this distribution. The average values differ in each reading method, which can allude to the respective technical characteristics. The `Rte_Receive` API stands out among its peers, with higher average and overall values.

However, as observed by the presence of a few outliers, there is a considerable fluctuation in the measured times between the minimum and maximum values. This behavior was unexpected since the tests were performed under uniform conditions. In the end, this performance was attributed to the effect of other *Runnables* that exist within the architecture into which this study was conducted.

Consequently, it was discovered that besides having a cyclic and periodic execution model, *Runnables* possess different priority levels. This hierarchy is established the moment the *Runnable* is mapped to a task of the operating system and comprises a critical element of the requirements of AUTOSAR projects. Figure 6.4 shows the position of the task where the “Runnable_ControlUnit” is mapped within the priority level hierarchy of the camera ECU. As observed, the task in question is preempted (interrupted) whenever the occurrence of a higher-priority task. This image was verified handling an internal tool that is property of *Continental Group* parent company.

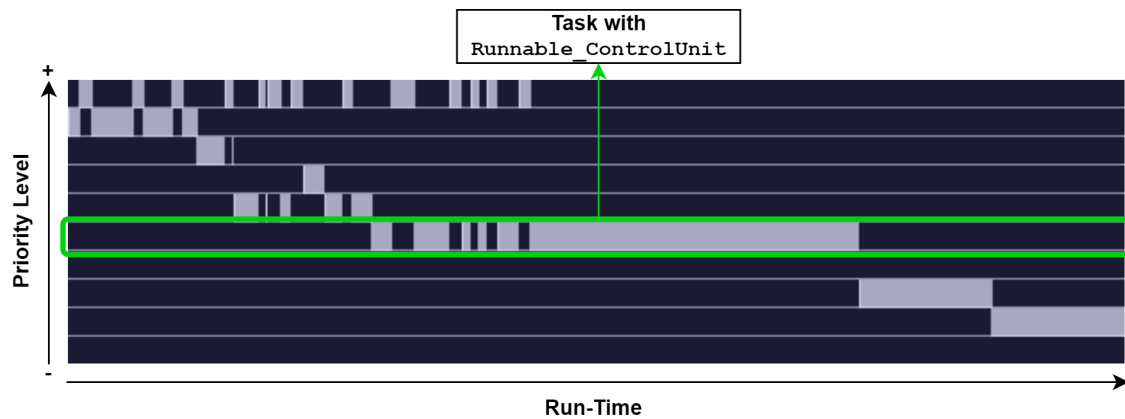


Figure 6.4: Priority Level Hierarchy.

With this in mind, it was decided to include a second case study. The aim was to minimize external factors and aspects that could affect this analysis, as observed in the previously presented data. Hence, the retrieved run-time measurements would yield a more thorough overview and understanding of the effects of the implemented API functions.

For this endeavor, attention was diverted to reduce the occurrence of preemption by temporally turning off all interruptions. After investigating ways of implementing this idea, it was discovered that AUTOSAR supported functions that could carry out this plan. Therefore, the `SuspendAllInterrupts` and `ResumeAllInterrupts` functions were employed at the beginning and end of the “Runnable_ControlUnit” function. This arrangement was made in line with the logic that was applied to the positioning of the *breakpoints*. These functions are inherited in AUTOSAR from OSEK and protect code sections from higher-priority interruptions that might occur [54].

In this case, we obtained more constant, coherent, and aligned values, as observed in Table 6.2 and Figure 6.5.

Table 6.2: Metrics of the Execution Times in Run-Time with Protection against Interruptions.

	Rte_Read	Rte_IRead	Rte_DRead	Rte_Receive	Rte_Read_WS	Units
MAX	0.244	0.246	0.243	0.524	0.245	ms
MIN	0.213	0.213	0.213	0.236	0.214	ms
AVERAGE	0.219	0.227	0.226	0.344	0.230	ms

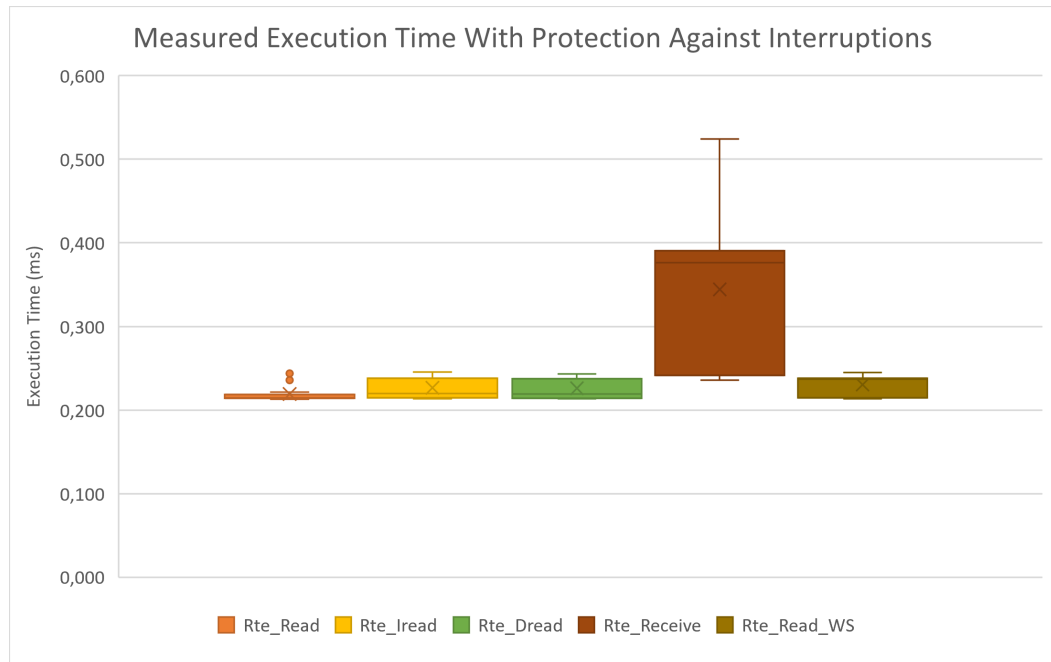


Figure 6.5: Measured Execution Times in Run-Time with Protection against Interruptions.

The adopted strategy was able to compress the spread and range of the retrieved statistics, significantly decreasing the occurrence of outliers. However, the `Rte_Receive` still sticks out from the rest of the API functions. Additionally, the closeness of the measured average values for these remaining API functions was unanticipated. It was predicted that, similarly to the first use case, this data would be unique for each reading method. Hence, there were concerns about the ability to draw conclusions based on this approach.

6.3 Discussion

The conducted analysis and measurements of this study provided essential findings concerning each API function that was worked on and the overall AUTOSAR architecture. The values obtained under normal run-time conditions with and without the preemption caused by higher-priority tasks enabled us to get a comprehensive view of the behavior of AUTOSAR applications and systems. The effects caused by preemption were apparent when comparing both use cases,

highlighting the importance of mapping *Runnables* to the correct operating system task, as this directly impacts the execution time of the specific *Runnable*.

The usage of the `SuspendAllInterrupts` and `ResumeAllInterrupts` functions was effective in its intended objective of eliminating the previously mentioned interruptions. Surprisingly, the results obtained in the second use case were unexpected, as these values were remarkably similar to each other, except for the `Rte_Receive` use case. This closeness was not expected and led us to ascertain that the initial aim that CES Porto had in finding an API function that could be considered the best and most optimal could not be scientifically proven in the context of this study. On one hand, the results yielded in the first use case can help us perceive the behavior of each API function. However, the results retrieved in the second use case raised questions about whether drawing conclusions regarding this initial classification was scientifically correct.

Nonetheless, the execution time observed for the `Rte_IRead` and `Rte_Receive` in the first use case can be explained by the fact that both API functions utilize intermediate buffers. A heavier reliance on intermediate buffers causes an increase in the occurrence of reading and writing operations, which entails higher timing costs. In addition, the difference between these two can be explained as `Rte_Receive` experiences a more significant overhead since we deal with individual “signals”, while `Rte_IRead` still includes the “signal groups” logic. This behavior is verified by the metrics extracted concerning the fifth implementation, `Rte_Read` API function with `SystemSignals` (`Rte_Read_WS`). The impact of a broader number of reading requests between the *Application* and the RTE layers that entails this design warrants attention from developers.

With this in mind, each API function has specific practical and technical advantages and disadvantages that a designer should consider when developing AUTOSAR applications.

`Rte_DRead` is the most straightforward implementation to retrieve data from the RTE. However, this option sacrifices information in the format of status and error codes that can be valuable for a developer. Additionally, it is incompatible with the `NonqueuedReceiverComSpec` and `SystemSignalGroup` classes, which offer efficient data handling.

Subsequently, `Rte_Read` addresses this gap at the price of a slight increase in execution time and can be considered the most balanced of all the API functions studied, a “jack of all trades, master of none”.

`Rte_IRead` guarantees that the message value is fixed during the execution of the *Runnable*. Thus, it may be appealing in applications where *Runnables* access the RTE more than once to get information. If more readings concerning a specific “signal” exist, the designer may require this value to be static in order to guarantee no data loss. Furthermore, concurrence-related problems can be minimized by implementing an *Implicit* communication between two ECUs.

Finally, based on the measurements obtained in both case studies, we can verify that the `Rte_Receive` is the most costly concerning execution time. Having said that, this API may be productive if the information is transmitted at a cyclic time shorter than the cyclic time of the *Runnable* in question. The adoption of a queue topology could potentially ensure a reduction in the loss of information that can happen in this use case.

Hence, the employment of each API function should be done according to the intended requirements of the wanted application in which the reading method will be employed. This delegation must also consider evaluating the compatibility with “signals” and “signal groups”.

Chapter 7

Conclusions and Future Work

The work that resulted from this dissertation achieved considerable findings, advancements, and contributions. The literature review and the work developed with the CES Porto team fulfilled academy and industry-related objectives.

The AUTOSAR consortium and standard were introduced, contextualized, and explored, particularly emphasizing its *Classic Platform*. As seen, AUTOSAR presents itself as a multilayered framework prime for exploration with intrinsic levels of complexity and potential, built with ambitious and noble goals. Among these, the simplification of the design and development process of automotive embedded software stands out. The partnership behind this global initiative occupies a significant role in the contemporary automotive industry.

Furthermore, a comprehensive literature review explored the scientific advancements in the AUTOSAR industry over the past 20 years. A survey article was produced and published, yielding several insights regarding the relationship between AUTOSAR-related concepts and future Smart City requirements. The resulting document highlighted current and future challenges in the automotive industry and was submitted at the “2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)” that occurred in Modena, Italy. The document was accepted, presented, and awarded with the “Best Paper Presented By A Young Researcher”. Currently, the article is waiting to be published in *IEEE Xplore*, where the scientific results will be publicly available.

However, the findings unveiled in the literature review revealed certain areas where technical optimization could be improved. Additionally, answers to security and emerging demands were remarked as needing improvement. Moreover, despite its 20-year history of intending to simplify design and development processes, AUTOSAR-related documentation is viewed by several authors as extensive, disorganized, and convoluted. Regardless of these limitations, this tool shows excellent potential, dedication to improvement, and capacity to rise to these challenges in future scenarios.

The practical work achieved with the ADAS-team at CES Porto provided invaluable insights concerning the AUTOSAR standard and architecture. Notably, an extensive mapping of the core

elements related to the structure and operation of the RTE and the *Application Layer* was accomplished. The discovery that this framework supported several distinct reading methods was crucial knowledge beyond the expectations and understanding of CES Porto. The ability to quantify the execution time of an implementation and temporarily halt the effect of preemption were also innovative concepts introduced to the company. All these endeavors were done with the prospect of aiding the optimization of the development process and data gathering of current and future projects.

The research that entailed the idealization of the action plan in order to successfully test, validate, and gather meaningful metrics highlighted several features and techniques unknown to the team at CES Porto. Working with state-of-the-art procedures and tools used in the industry was a unique opportunity for this dissertation. In particular, TRACE32 proved a farther versatile tool than initially expected and compatible with creating automated test environments for the developed implementations. Although the developed strategy still needs some efficiency and fine-tuning in preparing its integration into the company, this solution paved the way for increasing productivity in their projects.

The previously mentioned drawback of the official documentation of the AUTOSAR standard was one of the more significant obstacles faced during this dissertation. The way that official AUTOSAR-related documentation is written as specifications and requirements instead of guidelines accentuated the learning curve demanded in this dissertation. The years-long experience of the team at CES Porto in working with this standard in the automotive industry was invaluable in overcoming and easing this learning process. With this in mind, an in-depth and easy-to-read dissertation document that can serve readers interested in this area was produced.

Contrary to initial expectations, this study disclosed that it is impossible to state that a single API function can be deemed the best or superior. Each examined reading method possesses distinct technical traits that a designer must consider. Advantages and disadvantages arise when looking at the architecture and requirements of a project, and an optimized AUTOSAR architecture may involve utilizing multiple API functions rather than relying solely on one. The implications of an architecture based on “signals” or “signal groups” were also unfamiliar to the team at CES Porto and should be equally considered.

Future work regarding the literature review will entail the consideration of the previously mentioned unused search string and the number of citations of each “research works”. Other classifications and criteria could also be possibilities. Moreover, the attained results could be strengthened through graph-based analysis, utilizing different types of metadata.

Additionally, there is a desire to incorporate viewpoints from industry practitioners. These perspectives would provide valuable insights into how users who work daily with AUTOSAR perceive the tool. These testimonials may be gathered through open-ended or multiple-choice questionnaires. Additionally, direct inquiries within academic communities would aid in gauging their familiarity with this standard. This approach aims to understand the perspective of the target audience on the connection between AUTOSAR and upcoming challenges and requirements.

Forthcoming work regarding the practical study performed with the CES Porto team would focus on continuing the research of the AUTOSAR architecture, its framework, and related concepts. The inclusion of the reading methods that were left out in this study would mark the first step. Subsequently, the performed investigation would be extended from the *Receiver* to the *Sender* perspective. This way, the performed study would encompass all the AUTOSAR-supported reading methods of *Sender-Receiver* architectures.

Moreover, the continuation of upgrading the automated tools for the gathering of data would also be a paramount concern. At the end phase of the dissertation, it was discovered that TRACE32 supported a Python-based API. This discovery could potentially lead to migrating a combined `.comm` and VBA script strategy towards an easy-to-use solution supported on Python. The finished product would improve data acquisition efficiency and produce a tool that members of CES worldwide could easily use in their projects. With the development of a more robust recovery method, new metrics besides the execution time could also be procured.

Appendix A

Reviewed “Research Works”

Table A.1: Sources and References of the Reviewed “Research Works”.

Title	Source	Reference
A Framework for Real-time Automotive Applications to Multicore Platform in Perspective of AUTOSAR	IEEE Xplore	[27]
A hierarchical modeling method for AUTOSAR software components	IEEE Xplore	[55]
A Novel AUTOSAR-Compliant Approach to Develop Resolver as Complex Device Driver for Electric Vehicle Inverter	Springer Link	[56]
A Survey on the Benefits and Drawbacks of AUTOSAR	IEEE Xplore	[5]
A Taxonomy of Attack Mechanisms in the Automotive Domain	ScienceDirect	[57]
A Tour of Secure Software Engineering Solutions for Connected Vehicles	Springer Link	[35]
Advanced Driver Assistance Systems - Past, present and future	IEEE Xplore	[39]
An Approach to Optimize Intra-ECU Communication Based on Mapping of AUTOSAR Runnable Entities	IEEE Xplore	[13]
An efficient scheduling of AUTOSAR runnables to minimize communication cost in multi-core systems	IEEE Xplore	[58]
Application lifecycle management in automotive Adaptive AUTOSAR example	IEEE Xplore	[23]

Approximate optimal AUTOSAR software components deploying approach for automotive EE system	Springer Link	[1]
Automated Generation and Integration of AUTOSAR RTE Configurations	IEEE Xplore	[28]
Automation Test Method for Automotive Embedded Software Based on AUTOSAR	IEEE Xplore	[59]
Automotive Cybersecurity	Springer Link	[60]
AUTomotive Open System ARchitecture - An Industry-Wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures	SAE International	[3]
AUTOSAR – A Worldwide Standard is on the Road	Semantic Scholar	[61]
AUTOSAR – the Standardized Software Architecture	Springer Link	[62]
AUTOSAR Architecture Based Kernel Development for Automotive Application	Springer Link	[9]
AUTOSAR BSW in real life: A summary of the last years starting and putting projects into production	HAL	[63]
AUTOSAR Communication Stack	Hasso Plattner Institute	[16]
AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform	IEEE Xplore	[37]
AUTOSAR migration from existing automotive software	IEEE Xplore	[64]
AUTOSAR Runtime Environment and Virtual Function Bus	Semantic Scholar	[10]
AUTOSAR Software Architecture	Semantic Scholar	[8]
Defending AUTOSAR Safety Critical Systems Against Code Reuse Attacks	ACM Digital Library	[65]
Deployment optimization for AUTOSAR system configuration	IEEE Xplore	[29]
Design and Implementation Procedure for na Advanced Driver Assistance System Based on na Open Source AUTOSAR	MDPI	[4]

Design optimization for AUTOSAR models with preemption thresholds and mixed-criticality scheduling	ScienceDirect	[66]
Development of an AUTOSAR Compliant Cryptographic Library on State-of-the-Art Automotive Grade Controllers	IEEE Xplore	[30]
Development of Autonomous Car—Part I: Distributed System Architecture and Development Process	IEEE Xplore	[67]
Development of Autonomous Car—Part II: A Case Study on the Implementation of an Autonomous Driving System Based on Distributed Architecture	IEEE Xplore	[68]
Digital Cockpit in AUTOSAR Adaptive Context	IEEE Xplore	[22]
Driver Monitoring implementation in Adaptive AUTOSAR environment	IEEE Xplore	[69]
End-to-end timing analysis of cause-effect chains in automotive embedded systems	ScienceDirect	[70]
Evolution of Long-Term Industrial Meta-Models – An Automotive Case Study of AUTOSAR	IEEE Xplore	[71]
Exploiting AUTOSAR Safety Mechanisms to Launch Security Attacks	Springer Link	[32]
Extending AUTOSARs Counter-Based Solution for Freshness of Authenticated Messages in Vehicles	IEEE Xplore	[72]
How Timing Interfaces in AUTOSAR can Improve Distributed Development of Real-Time Software.	Semantic Scholar	[73]
Implementing Adaptive AUTOSAR Diagnostic Manager with Classic Diagnostics as APIs	IEEE Xplore	[38]
Improving Secure Coding Rules for Automotive Software by Using a Vulnerability Database	IEEE Xplore	[74]
Modeling and Development of AUTOSAR Software Components	IEEE Xplore	[2]

Modeling and Generation of Secure Component Communications in AUTOSAR	ACM Digital Library	[75]
Modeling the effects of AUTOSAR overheads on application timing and schedulability	IEEE Xplore	[26]
Multisource Software on Multicore Automotive ECUs—Combining Runnable Sequencing With Task Scheduling	IEEE Xplore	[76]
Optimizing the task allocation step for multi-core processors within AUTOSAR	IEEE Xplore	[77]
Overview of End-to-End Event Chain in Advanced Driver-Assistance Software following AUTOSAR	IEEE Xplore	[34]
Parallel execution of AUTOSAR legacy applications on multicore ECUs with timed implicit communication	IEEE Xplore	[78]
Paving the Way for Connected Cars with Adaptive AUTOSAR and AGL	IEEE Xplore	[79]
Review on AUTOSAR MemStack	IEEE Xplore	[80]
Security and privacy in vehicular communications: Challenges and opportunities	ScienceDirect	[36]
Security Concerns for Automotive Communication and Software Architecture	IEEE Xplore	[81]
Study of Security and Effectiveness of DoIP in Vehicle Networks	IEEE Xplore	[82]
Time and memory tradeoffs in the implementation of AUTOSAR components	IEEE Xplore	[83]
Timing modeling and analysis for AUTOSAR-based software development - a case study	IEEE Xplore	[84]
Timing Simulation of Interconnected AUTOSAR Software-Components	IEEE Xplore	[85]
Towards a communication-efficient mapping of AUTOSAR runnables on multi-cores	IEEE Xplore	[86]
Using AUTOSAR High-Level Specifications for the Synthesis of Security Components in Automotive Systems	Springer Link	[31]

Verifying data secure flow in AUTOSAR models	Springer Link	[87]
Verifying data secure flow in AUTOSAR models by static analysis	CORE	[88]
Work-in-progress: towards industry strength mapping of AUTOSAR automotive functionality on multicore architectures	IEEE Xplore	[89]

Table A.2: Authors and Year of Publication of the Reviewed “Research Works”.

Authors	Year	Reference
Priyanshi Gupta et al.	2019	[27]
Ruyi Wu et al.	2010	[55]
Sakshi Sharma et al.	2021	[56]
Silverio Martínez-Fernández et al.	2015	[5]
Irdin Pekaric et al.	2021	[57]
Antonia Bertolino et al.	2018	[35]
Adnan Shaout et al.	2011	[39]
Rongshen Long et al.	2009	[13]
Hamid Reza Faragardi et al.	2014	[58]
Mia Stepanović et al.	2018	[23]
Zheng Ran et al.	2017	[1]
Shawn Smith et al.	2022	[28]
Huichoun Moon et al.	2009	[59]
Ashish Jadhav	2021	[60]
Harald Heinecke et al.	2004	[3]
Simon Fürst et al.	2009	[61]
Stefan Bunzel	2011	[62]
R. S. Sandhya Devi et al.	2019	[9]
Matthieu Courier et al.	2019	[63]
Johannes Gosda et al.	2009	[16]
Simon Fürst, Markus Bechter	2016	[37]
Daehyun Kum et al.	2008	[64]
Nico Naumann	2009	[10]
Robert Warschofsky	2009	[8]
Ahmad M K Nasser et al.	2019	[65]
Wei Peng et al.	2010	[29]
Jaeho Park et al.	2019	[4]
Zhao Qingling et al.	2017	[66]
Pal-Stefan Murvay et al.	2016	[30]
Kichun Jo et al.	2014	[67]
Kichun Jo et al.	2015	[68]
Mila Kotur et al.	2018	[22]
Milan Đokić et al.	2018	[69]
Matthias Becker et al.	2017	[70]
Darko Durisic et al.	2014	[71]
Ahmad M K Nasser et al.	2017	[32]

Thomas Rosenstatter et al.	2019	[72]
Kai Richter et al.	2008	[73]
Gaurav Singh et al.	2018	[38]
Ryo Kurachi et al.	2018	[74]
Valentin T. Popović et al.	2019	[2]
Cinzia Bernardeschi et al.	2017	[75]
Manish Chauhan et al.	2017	[26]
Aurélien Monot et al.	2012	[76]
Andreas Sailer et al.	2013	[77]
Marko Vulić et al.	2019	[34]
Sebastian Kehr et al.	2015	[78]
S. Aust	2018	[79]
P Shanthi, Shriya Tatachar	2021	[80]
Cesar Bernardini et al.	2017	[36]
Huafeng Yu et al.	2016	[81]
V W Ajin et al.	2016	[82]
Alberto Ferrari et al.	2009	[83]
Kay Klobedanz et al.	2010	[84]
Matthias Krause et al.	2007	[85]
Hamid Reza Faragardi et al.	2013	[86]
Cinzia Bernardeschi et al.	2016	[31]
Cinzia Bernardeschi et al.	2018	[87]
Cinzia Bernardeschi et al.	2017	[88]
Cosmin Avasalcăi et al.	2017	[89]

Table A.3: Classification of the Reviewed “Research Works”.

Classification			Reference
<i>Development</i>	<i>Security</i>	<i>Use Cases</i>	
X		X	[27]
X			[55]
X			[56]
X			[5]
	X	X	[57]
	X	X	[35]
X			[39]
X			[13]
X		X	[58]
X		X	[23]
X			[1]
X			[28]
X			[59]
	X	X	[60]
X			[3]
X			[61]
X			[62]
X			[9]
X			[63]
X			[16]
X		X	[37]
X			[64]
X			[10]
X			[8]
	X		[65]
X			[29]
X		X	[4]
X			[66]
	X		[30]
X		X	[67]
X		X	[68]
X		X	[22]
X		X	[69]
X			[70]
X			[71]

	X		[32]
	X		[72]
X			[73]
X		X	[38]
	X		[74]
X			[2]
	X		[75]
X			[26]
X		X	[76]
X			[77]
X		X	[34]
X		X	[78]
X		X	[79]
X		X	[80]
	X	X	[36]
	X	X	[81]
	X		[82]
X			[83]
X			[84]
X			[85]
X			[86]
	X		[31]
	X		[87]
	X		[88]
X		X	[89]

Table A.4: Distribution of the AUTOSAR Platform Present in the Reviewed “Research Works”.

<i>AUTOSAR Classic</i>	<i>AUTOSAR Adaptive</i>	Reference
X		[27]
X		[55]
X		[56]
X		[5]
X		[57]
-	-	[35]
-	-	[39]
X		[13]
X		[58]
	X	[23]
X		[1]
X		[28]
X		[59]
X		[60]
X		[3]
X		[61]
X		[62]
X		[9]
X		[63]
X		[16]
	X	[37]
X		[64]
X		[10]
X		[8]
-	-	[65]
X		[29]
	X	[4]
X		[66]
X		[30]
X		[67]
X		[68]
	X	[22]
	X	[69]
X		[70]
X		[71]
-	-	[32]

-	-	[72]
X		[73]
X	X	[38]
X	X	[74]
X		[2]
X		[75]
X		[26]
X		[76]
X		[77]
X		[34]
X		[78]
	X	[79]
X		[80]
-	-	[36]
X		[81]
-	-	[82]
X		[83]
X		[84]
X		[85]
X		[86]
X		[31]
X		[87]
X		[88]
X		[89]

Appendix B

ARXML Configurations and Source Code

```
1 <RUNNABLE-ENTITY>
2   <SHORT-NAME>Runnable_ControlUnit</SHORT-NAME>
3   <MINIMUM-START-INTERVAL>0.02</MINIMUM-START-INTERVAL>
4   <DATA-RECEIVE-POINT-BY-ARGUMENTS>
5     <VARIABLE-ACCESS>
6       <SHORT-NAME>VariableAccess_Signal</SHORT-NAME>
7       <ACCESSED-VARIABLE>
8         <AUTOSAR-VARIABLE-IREF>
9           <PORT-PROTOTYPE-REF DEST="R-PORT-PROTOTYPE">rPort_Signal</
PORT-PROTOTYPE-REF>
10          <TARGET-DATA-PROTOTYPE-REF DEST="VARIABLE-DATA-PROTOTYPE">
DataElement_SignalGroup</TARGET-DATA-PROTOTYPE-REF>
11          </AUTOSAR-VARIABLE-IREF>
12        </ACCESSED-VARIABLE>
13      </VARIABLE-ACCESS>
14    </DATA-RECEIVE-POINT-BY-ARGUMENTS>
15 </RUNNABLE-ENTITY>
```

Listing B.1: ARXML Specification of a *Runnable* with *DataReceivePointByArgument* Attribute.

```
1 #include "Rte.c"
2 #include "Rte_SWC.h"
3
4 // Runnable Function which handles conversion operations of signals
5 void Runnable_ControlUnit(void)
6 {
7   // Status and Error Code
8   Std_ReturnType ui8_retVal;
9
10  // Data Elements used
11  struct
```

```
12 {
13     SignalGroup SignalGroup_1;
14     ...
15 } data;
16
17 //Invocation of the \texttt{Rte\_Read} API
18 ui8_retVal = Rte_Read_SWC_ControlUnit_Runnable_ControlUnit_rPort_SignalGroup_1
19             (&data.SignalGroup_1);
20
21 // Verification of the RTE Communication
22 if (RTE_E_OK == ui8_retVal)
23 {
24     //conversion operation with data.SignalGroup_1
25     ...
26 }
27 }
```

Listing B.2: Code Snippet of Runnable_ControlUnit with Rte_Read API Function.

Appendix C

Script Files and Example of Extracted Data

```
1 ; Variables for limiting the number of samples that are gathered
2 PRIVATE &counter
3 PRIVATE &counterlimit
4
5 &counter=0.
6 &counterlimit=500.
7
8 ; Re-directs the printer output to a file path and appends new data to the existing
   file
9 PRinTer.FILE C:\Measurement_APIFunction.csv /Append
10
11 ; Reduces the uncertainty and increases the precision of the RunTime Counter
12 SYStem.POLLING.CONtInuous
13
14 WHILE (&counter<&counterlimit)
15 (
16   ; Starts the program execution
17   Go
18
19   ; Writes the given arguments to the AREA window
20   PRINT "1ST Go"
21
22   ; Wait until program stops at the breakpoint
23   WAIT !STATE.RUN()
24   (
25     ; Starts the program execution again after the first breakpoint
26     Go
27
28     PRINT "2nd Go"
29
30     ; Exports the contents of the RunTime to the previous mentioned file path
31     WinPrint.RunTime
```

```

32 )
33
34 ; Increments the samples counter
35 &counter=&counter+1.
36 )

```

Listing C.1: Measurements.cmm Script File.

```

1 B::RunTime
2      *ref A      ref B      laststart  actual
3 zero      *      0.000us      0.000us      259.519s      259.866s
4 ref A      *              0.000us      259.519s      259.866s
5 ref B      *              259.519s      259.866s
6 laststart*              346.448ms

```

Listing C.2: Extracted RunTime Window.

```

1 Sub Extract()
2
3 Dim i As Integer
4
5 i = 1
6 y = 1
7
8 Do While i < 5000
9     If IsEmpty(Worksheets("Raw").Cells(i, 1).Value) = False Then
10         Worksheets("Raw").Cells(i, 1).Select
11         Selection.TextToColumns , xlDelimited, xlDoubleQuote, True, True, , , True,
            False
12         If Worksheets("Raw").Cells(i, 1).Value = "laststart*" Then
13             Worksheets("Final").Cells(y, 1).Value = Worksheets("Raw").Cells(i, 2).
            Value
14             y = y + 1
15         End If
16     End If
17     i = i + 1
18 Loop
19
20 End Sub

```

Listing C.3: VBA Script File for Data Extraction.

```

1 Sub Convert()
2
3 Application.DisplayAlerts = False
4
5 Dim i As Integer
6
7 i = 1
8
9 Do While i < 5000

```

```
10 If IsEmpty(Worksheets("Final").Cells(i, 1).Value) = False Then
11     If Application.IfError(Application.Search("u", Worksheets("Final").Cells(i,
12         1).Value), 0) = 0 Then
13         Worksheets("Final").Cells(i, 1).Select
14         Selection.TextToColumns , xlDelimited, xlDoubleQuote, , , , , True, "
15         m"
16         Worksheets("Final").Cells(i, 2).Value = "ms"
17     ElseIf Application.IfError(Application.Search("u", Worksheets("Final").
18         Cells(i, 1).Value), 0) <> 0 Then
19         Worksheets("Final").Cells(i, 1).Select
20         Selection.TextToColumns , xlDelimited, xlDoubleQuote, , , , , True, "
21         u"
22         Worksheets("Final").Cells(i, 1).Value = Application.Convert(Worksheets
23         ("Final").Cells(i, 1).Value, "us", "ms")
24         Worksheets("Final").Cells(i, 2).Value = "us"
25     End If
26 End If
27 i = i + 1
28 Loop
29
30 Worksheets("Final").Cells(4, 10).Value = Application.Max(Range("A:A"))
31 Worksheets("Final").Cells(4, 11).Value = "ms"
32 Worksheets("Final").Cells(5, 10).Value = Application.Min(Range("A:A"))
33 Worksheets("Final").Cells(5, 11).Value = "ms"
34 Worksheets("Final").Cells(6, 10).Value = Application.Average(Range("A:A"))
35 Worksheets("Final").Cells(6, 11).Value = "ms"
36
37 Application.DisplayAlerts = True
38
39 End Sub
```

Listing C.4: VBA Script File for Minimum, Maximum and Average Metrics.

References

- [1] Zheng Ran, Hua Yan, Huimin Zhang, and Yun Li. Approximate optimal autosar software components deploying approach for automotive e/e system. *International Journal of Automotive Technology*, 18(6):1109–1119, Dec 2017. doi:[10.1007/s12239-017-0108-3](https://doi.org/10.1007/s12239-017-0108-3).
- [2] Valentin T. Popović, Marko Vulić, Aleksandra Davidović, and Ivan Kaštelan. Modeling and development of autosar software components. In *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pages 313–316, 2019. doi:[10.1109/ISCT.2019.8901035](https://doi.org/10.1109/ISCT.2019.8901035).
- [3] Harald Heinecke, K P Schnelle, H Fennel, Jürgen Bortolazzi, Lennart Lundh, Jean Leflour, J L Mate, Kenji Nishikawa, and Thomas Scharnhorst. Automotive open system architecture - an industry-wide initiative to manage the complexity of emerging automotive e/e-architectures. In *Convergence International Congress & Exposition On Transportation Electronics*, 2004.
- [4] Jaeho Park and Byoung Wook Choi. Design and implementation procedure for an advanced driver assistance system based on an open source autosar. *Electronics*, 8, 2019. doi:[10.3390/electronics8091025](https://doi.org/10.3390/electronics8091025).
- [5] Silverio Martínez-Fernández, Claudia Ayala, Xavier Franch, and Elisa Nakagawa. A survey on the benefits and drawbacks of autosar. In *2015 First International Workshop on Automotive Software Architecture (WASA)*, pages 19–26, 05 2015. doi:[10.1145/2752489.2752493](https://doi.org/10.1145/2752489.2752493).
- [6] Continental Automotive. Multi function mono camera - MFC500. Accessed December 26th, 2022. URL: <https://www.continental-automotive.com/en-gl/Passenger-Cars/Autonomous-Mobility/Enablers/Cameras/Mono-Camera>.
- [7] AUTOSAR. Partners autosar. Accessed October 27th, 2022. URL: <https://www.autosar.org/about/partners>.
- [8] Robert Warschofsky. Autosar software architecture. *Hasso-Plattner-Institute für Softwaresystemtechnik: Potsdam, Germany*, 2009.
- [9] R. S. Sandhya Devi, P. Sivakumar, and R. Balaji. Autosar architecture based kernel development for automotive application. In Jude Hemanth, Xavier Fernando, Pavel Lafata, and Zubair Baig, editors, *International Conference on Intelligent Data Communication Technologies and Internet of Things (ICICI) 2018*, pages 911–919, Cham, 2019. Springer International Publishing.
- [10] Nico Naumann. Autosar runtime environment and virtual function bus. *Hasso-Plattner-Institute für IT-Systems Engineering: Potsdam, Germany*, 2009.

- [11] AUTOSAR. Virtual functional bus. Technical report, AUTOSAR, 2022. Accessed February 15th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_EXP_VFB.pdf.
- [12] AUTOSAR. Specification of rte software. Technical report, AUTOSAR, 2022. Accessed February 14th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_RTE.pdf.
- [13] Rongshen Long, Hong Li, Wei Peng, Yi Zhang, and Minde Zhao. An approach to optimize intra-ecu communication based on mapping of autosar runnable entities. In *2009 International Conference on Embedded Software and Systems*, pages 138–143, 2009. doi:10.1109/ICCESS.2009.63.
- [14] Vector Informatik. Autosar classic | vector. Accessed October 27th, 2022. URL: <https://www.vector.com/int/en/know-how/autosar/autosar-classic/>.
- [15] AUTOSAR. Layered software architecture. Technical report, AUTOSAR, 2022. Accessed December 12th, 2022. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.
- [16] Johannes Gosda. Autosar communication stack. *University Potsdam, HPI*, 2009.
- [17] AUTOSAR. Specification of can driver. Technical report, AUTOSAR, 2022. Accessed February 20th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_CANDriver.pdf.
- [18] AUTOSAR. Specification of can interface. Technical report, AUTOSAR, 2022. Accessed February 20th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_CANInterface.pdf.
- [19] AUTOSAR. Specification of pdu router. Technical report, AUTOSAR, 2022. Accessed February 16th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_PDURouter.pdf.
- [20] AUTOSAR. Specification of communication. Technical report, AUTOSAR, 2022. Accessed February 16th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/CP/AUTOSAR_SWS_COM.pdf.
- [21] AUTOSAR. History autosar. Accessed October 27th, 2022. URL: <https://www.autosar.org/about/history>.
- [22] Mila Kotur, Marko Dragojević, Gordana Velikić, and Ilija Bašičević. Digital cockpit in autosar adaptive context. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 1–4, 2018. doi:10.1109/ICCE-Berlin.2018.8576209.
- [23] Mia Stepanović, Jelena Jovičić, Goran Stupar, and Marko Kovačević. Application lifecycle management in automotive: Adaptive autosar example. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 1–4, 2018. doi:10.1109/ICCE-Berlin.2018.8576248.
- [24] AUTOSAR. Explanation of adaptive platform design. Technical report, AUTOSAR, 2022. Accessed March 2nd, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/AP/AUTOSAR_EXP_PlatformDesign.pdf.

- [25] Gonalo M. d. Santos and Daniel G. Costa. Autosar in the smart cities era: Current developments and research trends. In *2023 IEEE International Workshop on Metrology for Automotive (MetroAutomotive)*, pages 164–169, 2023. doi:10.1109/MetroAutomotive57488.2023.10219095.
- [26] Manish Chauhan, Rodolfo Pellizzoni, and Krzysztof Czarnecki. Modeling the effects of autosar overheads on application timing and schedulability. In *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2017. doi:10.1145/3061639.3062287.
- [27] Priyanshi Gupta, N.P. Singh, and Geetha Srinivasan. A framework for real-time automotive applications to multicore platform in perspective of autosar. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*, pages 706–709, 2019. doi:10.1109/RTEICT46194.2019.9016689.
- [28] Shawn Smith and Mohammed A.S. Khalid. Automated generation and integration of autosar rte configurations. In *2022 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 141–144, 2022. doi:10.1109/CCECE49351.2022.9918435.
- [29] Wei Peng, Hong Li, Min Yao, and Zheng Sun. Deployment optimization for autosar system configuration. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–189–V4–193, 2010. doi:10.1109/ICCET.2010.5485642.
- [30] Pal-Stefan Murvay, Alexandru Matei, Cristina Solomon, and Bogdan Groza. Development of an autosar compliant cryptographic library on state-of-the-art automotive grade controllers. In *2016 11th International Conference on Availability, Reliability and Security (ARES)*, pages 117–126, 2016. doi:10.1109/ARES.2016.60.
- [31] Cinzia Bernardeschi, Gabriele Del Vigna, Marco Di Natale, Gianluca Dini, and Dario Varano. Using autosar high-level specifications for the synthesis of security components in automotive systems. In Jan Hodicky, editor, *Modelling and Simulation for Autonomous Systems*, pages 101–117, Cham, 2016. Springer International Publishing.
- [32] Ahmad M. K. Nasser, Di Ma, and Sam Lauzon. Exploiting autosar safety mechanisms to launch security attacks. In Zheng Yan, Refik Molva, Wojciech Mazurczyk, and Raimo Kantola, editors, *Network and System Security*, pages 73–86, Cham, 2017. Springer International Publishing.
- [33] AUTOSAR. E2e protocol specification. Technical report, AUTOSAR, 2022. Accessed February 16th, 2023. URL: https://www.autosar.org/fileadmin/standards/R22-11/FO/AUTOSAR_PRS_E2EProtocol.pdf.
- [34] Marko Vulić, Valentin T. Ponović, Aleksandra Davidović, and Ivan Kaštelan. Overview of end-to-end event chain in advanced driver-assistance software following autosar. In *2019 IEEE 23rd International Symposium on Consumer Technologies (ISCT)*, pages 308–312, 2019. doi:10.1109/ISCT.2019.8900984.
- [35] Antonia Bertolino, Antonello Calabro’, Felicita Di Giandomenico, Giuseppe Lami, Francesca Lonetti, Eda Marchetti, Fabio Martinelli, Ilaria Matteucci, and Paolo Mori. A tour of secure software engineering solutions for connected vehicles. *Software Quality Journal*, 26(4):1223–1256, Dec 2018. doi:10.1007/s11219-017-9393-3.

- [36] Cesar Bernardini, Muhammad Rizwan Asghar, and Bruno Crispo. Security and privacy in vehicular communications: Challenges and opportunities. *Vehicular Communications*, 10:13–28, 2017. doi:<https://doi.org/10.1016/j.vehcom.2017.10.002>.
- [37] Simon Fürst and Markus Bechter. Autosar for connected and autonomous vehicles: The autosar adaptive platform. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W)*, pages 215–217, 2016. doi:[10.1109/DSN-W.2016.24](https://doi.org/10.1109/DSN-W.2016.24).
- [38] Gaurav Singh, Narayan Kamath, and R.K. Sharma. Implementing adaptive autosar diagnostic manager with classic diagnostics as apis. In *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 894–898, 2018. doi:[10.1109/ICCONS.2018.8662958](https://doi.org/10.1109/ICCONS.2018.8662958).
- [39] Adnan Shaout, Dominic Colella, and S. Awad. Advanced driver assistance systems - past, present and future. In *2011 Seventh International Computer Engineering Conference (ICENCO'2011)*, pages 72–82, 2011. doi:[10.1109/ICENCO.2011.6153935](https://doi.org/10.1109/ICENCO.2011.6153935).
- [40] Elektrobit. Classic autosar basic software (bsw). Accessed March 1st, 2023. URL: <https://www.elektrobit.com/products/ecu/eb-tresos/bsw/>.
- [41] Velleman Group. Labps3005dn velleman group. Accessed February 20th, 2023. URL: <https://www.velleman.eu/products/view/?id=439082>.
- [42] Vector Informatik. Vn1600 - network interfaces with usb for can / can fd, lin, k-line, j1708 and io. Accessed February 20th, 2023. URL: <https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn16xx/>.
- [43] Lauterbach GmbH. Powerdebug module usb 3.0 | la-3500. Accessed February 20th, 2023. URL: <https://www.lauterbach.com/products/LA-3500>.
- [44] Vector Informatik. Simulation with canoe. Accessed March 1st, 2023. URL: <https://www.vector.com/int/en/products/products-a-z/software/canoe/simulation/>.
- [45] Lauterbach GmbH. Debugging over usb | lauterbach trace32. Accessed March 1st, 2023. URL: <https://www.lauterbach.com/products/software/debugging-via-usb>.
- [46] O. Scheid. *AUTOSAR Compendium - Part 1: Application and RTE*. AUTOSAR - Compendium Series. CreateSpace Independent Publishing Platform, Bruchsal, August 2015.
- [47] Lauterbach GmbH. Practice script language user's guide. Technical report, Lauterbach GmbH, 2023. Accessed April 10th, 2023. URL: https://www2.lauterbach.com/pdf/practice_user.pdf.
- [48] Lauterbach GmbH. Training basic debugging. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://repo.lauterbach.com/pdf/training_debugger.pdf.
- [49] Lauterbach GmbH. General commands reference guide r. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://www2.lauterbach.com/pdf/general_ref_r.pdf.

- [50] Lauterbach GmbH. Powerview command reference. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://repo.lauterbach.com/pdf/ide_ref.pdf.
- [51] Lauterbach GmbH. General commands reference guide s. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://www2.lauterbach.com/pdf/general_ref_s.pdf.
- [52] Lauterbach GmbH. Practice script language reference guide. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://repo.lauterbach.com/pdf/practice_ref.pdf.
- [53] Lauterbach GmbH. General function reference. Technical report, Lauterbach GmbH, 2023. Accessed April 12th, 2023. URL: https://www2.lauterbach.com/pdf/general_func.pdf.
- [54] OSEK. Osek/vdx operating system. Technical report, OSEK, 2005. Accessed April 13th, 2023. URL: <https://www.irisa.fr/alf/downloads/puaut/TPNXT/images/os223.pdf>.
- [55] Ruyi Wu, Hong Li, Min Yao, Jinbo Wang, and Yuhao Yang. A hierarchical modeling method for autosar software components. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–184–V4–188, 2010. doi:10.1109/ICCET.2010.5485641.
- [56] Sakshi Sharma, Bhanu Pratap, Ajit Safai, and Raghav Ankur. *A Novel AUTOSAR-Compliant Approach to Develop Resolver as Complex Device Driver for Electric Vehicle Inverter*, pages 157–166. Springer Singapore, Singapore, 2021. doi:10.1007/978-981-15-8045-1_16.
- [57] Irdin Pekaric, Clemens Sauerwein, Stefan Haselwanter, and Michael Felderer. A taxonomy of attack mechanisms in the automotive domain. *Computer Standards & Interfaces*, 78:103539, 2021. doi:<https://doi.org/10.1016/j.csi.2021.103539>.
- [58] Hamid Reza Faragardi, Björn Lisper, Kristian Sandström, and Thomas Nolte. An efficient scheduling of autosar runnables to minimize communication cost in multi-core systems. In *7th International Symposium on Telecommunications (IST'2014)*, pages 41–48, 2014. doi:10.1109/ISTEL.2014.7000667.
- [59] Huichoun Moon, Gwanghun Kim, Yeongyun Kim, Seokkyoo Shin, Kyoung Kim, and Sang-gui Im. Automation test method for automotive embedded software based on autosar. In *2009 Fourth International Conference on Software Engineering Advances*, pages 158–162, 2009. doi:10.1109/ICSEA.2009.32.
- [60] Ashish Jadhav. *Automotive Cybersecurity*, pages 101–114. Springer International Publishing, Cham, 2021. doi:10.1007/978-3-030-59897-6_6.
- [61] Simon Fürst, Jürgen Mössinger, Stefan Bunzel, Thomas Weber, Frank Kirschke-Biller, Peter Heitkämper, Gerulf Kinkel, Kenji Nishikawa, and Klaus Lange. Autosar – a worldwide standard is on the road. In *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*, 2009.

- [62] Stefan Bunzel. Autosar - the standardized software architecture. *Informatik-Spektrum*, 34(1):79–83, Feb 2011. doi:[10.1007/s00287-010-0506-7](https://doi.org/10.1007/s00287-010-0506-7).
- [63] Matthieu Courrier, Helen Clergeau, Alain Calvy, Philippe Favrais, and Michel Lecat. Autosar bsw in real life: A summary of the last years starting and putting projects into production. In *Embedded Real Time Software and Systems (ERTS2012)*, Toulouse, France, Feb 2012.
- [64] Daehyun Kum, Gwang-Min Park, Seonghun Lee, and Wooyoung Jung. Autosar migration from existing automotive software. In *2008 International Conference on Control, Automation and Systems*, pages 558–562, 2008. doi:[10.1109/ICCAS.2008.4694565](https://doi.org/10.1109/ICCAS.2008.4694565).
- [65] Ahmad Nasser and Di Ma. Defending autosar safety critical systems against code reuse attacks. In *Proceedings of the ACM Workshop on Automotive Cybersecurity*, AutoSec ’19, page 15–18, New York, NY, USA, 2019. Association for Computing Machinery. doi:[10.1145/3309171.3309172](https://doi.org/10.1145/3309171.3309172).
- [66] Qingling Zhao, Zonghua Gu, and Haibo Zeng. Design optimization for autosar models with preemption thresholds and mixed-criticality scheduling. *Journal of Systems Architecture*, 72:61–68, 2017. Design Automation for Embedded Ubiquitous Computing Systems. doi:<https://doi.org/10.1016/j.sysarc.2016.08.003>.
- [67] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car—part i: Distributed system architecture and development process. *IEEE Transactions on Industrial Electronics*, 61(12):7131–7140, 2014. doi:[10.1109/TIE.2014.2321342](https://doi.org/10.1109/TIE.2014.2321342).
- [68] Kichun Jo, Junsoo Kim, Dongchul Kim, Chulhoon Jang, and Myoungho Sunwoo. Development of autonomous car—part ii: A case study on the implementation of an autonomous driving system based on distributed architecture. *IEEE Transactions on Industrial Electronics*, 62(8):5119–5132, 2015. doi:[10.1109/TIE.2015.2410258](https://doi.org/10.1109/TIE.2015.2410258).
- [69] Milan Đokić, Stefan Nićetin, Gordana Velikić, and Tihomir Anđelić. Driver monitoring implementation in adaptive autosar environment. In *2018 IEEE 8th International Conference on Consumer Electronics - Berlin (ICCE-Berlin)*, pages 1–4, 2018. doi:[10.1109/ICCE-Berlin.2018.8576255](https://doi.org/10.1109/ICCE-Berlin.2018.8576255).
- [70] Matthias Becker, Dakshina Dasari, Saad Mubeen, Moris Behnam, and Thomas Nolte. End-to-end timing analysis of cause-effect chains in automotive embedded systems. *Journal of Systems Architecture*, 80:104–113, 2017. doi:<https://doi.org/10.1016/j.sysarc.2017.09.004>.
- [71] Darko Durisic, Mirosław Staron, Matthias Tichy, and Jörgen Hansson. Evolution of long-term industrial meta-models – an automotive case study of autosar. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 141–148, 2014. doi:[10.1109/SEAA.2014.21](https://doi.org/10.1109/SEAA.2014.21).
- [72] Thomas Rosenstatter, Christian Sandberg, and Tomas Olovsson. Extending autosar’s counter-based solution for freshness of authenticated messages in vehicles. In *2019 IEEE 24th Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 1–109, 2019. doi:[10.1109/PRDC47002.2019.00012](https://doi.org/10.1109/PRDC47002.2019.00012).

- [73] Kai Richter, Nico Feiertag, Michael Rudorfer, Oliver Scheickl, and Christoph Ainhauser. How timing interfaces in autosar can improve distributed development of real-time software. In *GI Jahrestagung*, pages 662–667, 01 2008.
- [74] Ryo Kurachi, Hiroaki Takada, Masato Tanabe, Jun Anzai, Kentaro Takei, Takaaki Iinuma, Manabu Maeda, and Hideki Matsushima. Improving secure coding rules for automotive software by using a vulnerability database. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–8, 2018. doi:[10.1109/ICVES.2018.8519496](https://doi.org/10.1109/ICVES.2018.8519496).
- [75] Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, and Dario Varano. Modeling and generation of secure component communications in autosar. In *Proceedings of the Symposium on Applied Computing, SAC '17*, page 1473–1480, New York, NY, USA, 2017. Association for Computing Machinery. doi:[10.1145/3019612.3019682](https://doi.org/10.1145/3019612.3019682).
- [76] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multi-source software on multicore automotive ecus—combining runnable sequencing with task scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, 2012. doi:[10.1109/TIE.2012.2185913](https://doi.org/10.1109/TIE.2012.2185913).
- [77] Andreas Sailer, Stefan Schmidhuber, Michael Deubzer, Martin Alfranseder, Matthias Mucha, and Jürgen Mottok. Optimizing the task allocation step for multi-core processors within autosar. In *2013 International Conference on Applied Electronics*, pages 1–6, 2013.
- [78] Sebastian Kehr, Eduardo Quiñones, Bert Böddeker, and Günter Schäfer. Parallel execution of autosar legacy applications on multicore ecus with timed implicit communication. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015. doi:[10.1145/2744769.2744889](https://doi.org/10.1145/2744769.2744889).
- [79] S. Aust. Paving the way for connected cars with adaptive autosar and agl. In *2018 IEEE 43rd Conference on Local Computer Networks Workshops (LCN Workshops)*, pages 53–58, 2018. doi:[10.1109/LCNW.2018.8628558](https://doi.org/10.1109/LCNW.2018.8628558).
- [80] P Shanthi and Shriya Tatachar. Review on autosar memstack. In *2021 Asian Conference on Innovation in Technology (ASIANCON)*, pages 1–4, 2021. doi:[10.1109/ASIANCON51346.2021.9544540](https://doi.org/10.1109/ASIANCON51346.2021.9544540).
- [81] Huafeng Yu and Chung-Wei Lin. Security concerns for automotive communication and software architecture. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 600–603, 2016. doi:[10.1109/INFOCOMW.2016.7562147](https://doi.org/10.1109/INFOCOMW.2016.7562147).
- [82] V W Ajin, Lekshmy D Kumar, and James Joy. Study of security and effectiveness of doip in vehicle networks. In *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*, pages 1–6, 2016. doi:[10.1109/ICCPCT.2016.7530357](https://doi.org/10.1109/ICCPCT.2016.7530357).
- [83] Alberto Ferrari, Marco Di Natale, Giacomo Gentile, Giovanni Reggiani, and Paolo Gai. Time and memory tradeoffs in the implementation of autosar components. In *2009 Design, Automation & Test in Europe Conference & Exhibition*, pages 864–869, 2009. doi:[10.1109/DATE.2009.5090783](https://doi.org/10.1109/DATE.2009.5090783).
- [84] Kay Klobedanz, Christoph Kuznik, Andreas Thuy, and Wolfgang Mueller. Timing modeling and analysis for autosar-based software development - a case study. In *2010 Design,*

- Automation & Test in Europe Conference & Exhibition (DATE 2010)*, pages 642–645, 2010. doi:[10.1109/DATE.2010.5457125](https://doi.org/10.1109/DATE.2010.5457125).
- [85] Matthias Krause, Oliver Bringmann, Andre Hergenhan, Gokhan Tabanoglu, and Wolfgang Rosentiel. Timing simulation of interconnected autosar software-components. In *2007 Design, Automation & Test in Europe Conference & Exhibition*, pages 1–6, 2007. doi:[10.1109/DATE.2007.364638](https://doi.org/10.1109/DATE.2007.364638).
- [86] Hamid Reza Faragardi, Björn Lisper, and Thomas Nolte. Towards a communication-efficient mapping of autosar runnables on multi-cores. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–5, 2013. doi:[10.1109/ETFA.2013.6648168](https://doi.org/10.1109/ETFA.2013.6648168).
- [87] Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, and Maurizio Palmieri. Verifying data secure flow in autosar models. *Journal of Computer Virology and Hacking Techniques*, 14(4):269–289, Nov 2018. doi:[10.1007/s11416-018-0317-y](https://doi.org/10.1007/s11416-018-0317-y).
- [88] Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, and Maurizio Palmieri. Verifying data secure flow in autosar models by static analysis. In *1st International Workshop on FORMAL methods for Security Engineering*, pages 704–713, 01 2017. doi:[10.5220/0006288707040713](https://doi.org/10.5220/0006288707040713).
- [89] Cosmin Avasalcui, Dhanesh Budhrani, and Paul Pop. Work-in-progress: towards industry strength mapping of autosar automotive functionality on multicore architectures. In *2017 International Conference on Compilers, Architectures and Synthesis For Embedded Systems (CASES)*, pages 1–2, 2017. doi:[10.1145/3125501.3125623](https://doi.org/10.1145/3125501.3125623).