

Functional-safety analysis of ASIL decomposition for redundant automotive systems

Citation for published version (APA):

Frigerio, A. (2022). *Functional-safety analysis of ASIL decomposition for redundant automotive systems*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Eindhoven University of Technology.

Document status and date:

Published: 21/04/2022

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Functional-safety analysis of ASIL decomposition for redundant automotive systems

PROEFONTWERP

ter verkrijging van de graad van doctor aan de Technische Universiteit
Eindhoven, op gezag van de rector magnificus prof.dr.ir. F.P.T. Baaijens, voor
een commissie aangewezen door het College voor Promoties in het openbaar te
verdedigen op donderdag 21 april 2022 om 11.00 uur

door

Alessandro Frigerio

geboren te Biella, Italië

Dit proefontwerp is goedgekeurd door de promotor en de samenstelling van de commissie is als volgt:

voorzitter:	prof.dr.ir. M. Mischi
1 ^e promotor:	prof.dr. K.G.W. Goossens
copromotor:	dr.ir. H.G.H. Vermeulen (NXP Semiconductors)
leden:	prof.dr.ir. P.H.N. de With
	prof.dr. M.G.J. van den Brand
	prof.dr.ir. J.-P. Katoen (RWTH Aachen)
	prof.dr.ir. J.P.M. Voeten

Het onderzoek dat in dit proefontwerp wordt beschreven is uitgevoerd in overeenstemming met de TU/e Gedragscode Wetenschapsbeoefening.

Functional-safety analysis of ASIL decomposition for redundant automotive systems

Alessandro Frigerio

Doctorate committee:

prof.dr. K.G.W. Goossens	Eindhoven University of Technology, promotor
dr.ir. H.G.H. Vermeulen	NXP, copromotor
prof.dr.ir. M. Mischi	Eindhoven University of Technology, chairman
prof.dr.ir. P.H.N. de With	Eindhoven University of Technology
prof.dr. M.G.J. van den Brand	Eindhoven University of Technology
prof.dr.ir. J.-P. Katoen	RWTH Aachen
prof.dr.ir. J.P.M. Voeten	Eindhoven University of Technology

This work was supported by the TU/e Impuls program, a strategic cooperation between NXP Semiconductors and the Eindhoven University of Technology. This research was supported through PENTA project HIPER 181004.

© Copyright 2021, Alessandro Frigerio

All rights reserved. Reproduction in whole or in part is prohibited without the written consent of the copyright owner.

Cover idea by I.R.L.

Printed by ProefschriftMaken || www.proefschriftmaken.nl

A catalogue record is available from the Eindhoven University of Technology Library.
ISBN: 978-90-386-5487-4

Abstract

The capabilities of the electronic system in cars have grown over the course of the years. The limitations of mechanical systems led to the widespread introduction of Electronic Control Units (ECUs) in the 1970s. Today, a commercial vehicle uses more than a hundred ECUs. With a combination of ECUs, software, sensors, and actuators the electronic system enables Advanced Driving Assistance Systems (ADASs) or even Autonomous Vehicles (AVs).

Complex functionality used in a safety-critical context is followed by a growing number of requirements, especially related to reliability. Safety requirements are a key factor in the development of a vehicle, especially when the driver responsibilities are shifted from the human to the electronic system. However, the road towards fully automated fail-operational vehicles is still long: while nowadays the prototypes are being built and tested, design and processes standardization is still required. The automotive system architecture is not standardized, and the manufacturers develop new automotive systems based on the experience of the designers, availability of components, and pre-existing techniques.

With this work, we discuss the modelling of automotive systems to qualify and quantify different implementations with safety as the main design goal, following the recommendation of current functional safety standards such as ISO26262.

Our first contribution is a three-layer model to describe the application, the resources, and the physical aspects related to the automotive system. We focus on safety-oriented designs, in which redundancy is used for fail-operational systems. An explicit notation for elements that define redundancy is used: two elements delimit redundant parts of the system, defined as splitter and merger, which have a safety-oriented functionality. We develop a framework in which an automotive system can be modelled and evaluated on multiple parameters: cost, application failure probability, total cable length, and application functional and communication loads.

Our second contribution is the development of model transformations which, based on the ISO26262 standard Automotive Safety Integrity Level (ASIL) decomposition, allow the introduction of redundancy in selected parts of the modelled system. After each transformation, the obtained system can be re-evaluated and compared by using the calculated parameters. We generate fault trees from the system description to perform a Common Cause Fault (CCF) analysis to validate the independence required by redundant parts of the system.

As a third contribution, we model and perform the analysis on typical automotive systems. To validate our transformation techniques we use automotive use-cases and compare our method with results obtained with a manual design process. We utilize our framework to analyse the different automotive architecture topologies that are foreseen to become the most prevalent in future vehicles. An extensive analysis of domain-based and zone-based architectures is performed to identify the strengths and weaknesses of the two topologies. The impact of redundancy is tested with the automated techniques introduced in our tools to understand in which scenarios the manufacturer should opt for one or the other architecture topology.

Finally, we discuss how mixed-critical applications, which are common in automotive systems, impact automotive architectures. Isolation is required, and to obtain it either physical separation or virtualization techniques are used. Each technique comes with a different cost, and we can quantify them in our framework. We calculate the effects of these two isolation techniques on example applications in which mixed-criticality and redundancy are present.

Contents

Abstract	i
1 Introduction	1
1.1 Advanced driving assistance systems and autonomous vehicles . . .	1
1.2 System requirements for ADASs and AVs	2
1.3 SAE automation levels	6
1.4 Functional safety	7
1.5 From fail-safe to fail-operational architectures	9
1.6 Problem statement and contributions	10
1.7 Thesis outline	12
2 Background and related work	15
2.1 Introduction	15
2.2 Automotive functional safety standards	16
2.2.1 ISO 26262	16
2.2.2 ASIL decomposition	20
2.2.3 ISO/PAS 21448 - SOTIF	22
2.2.4 UL4600	24
2.3 Faults and safety mechanisms	24
2.4 Fault tolerant schemes	28
2.5 Architecture safety patterns	29
2.6 Resource sharing and fail-silent systems	31
3 Model and functional-safety analysis of automotive systems	35
3.1 Introduction and related work	35
3.2 Three-layer model	39
3.2.1 Application layer	41
3.2.2 Resource layer	45
3.2.3 Physical layer	48
3.2.4 Mapping: inter-layer connections	49
3.2.5 Analysable input graphs	50
3.3 The splitter and the merger concepts	52
3.3.1 Splitter implementations	54
3.3.2 Merger implementations	56

3.3.3	Splitter and merger in the safety-executive architecture pattern	58
3.3.4	Splitter and merger and virtual resources	59
3.4	Quantitative analysis of the model	62
3.4.1	Fault tree generation and failure probability calculation	62
3.4.2	Static fault tree analysis	71
3.4.3	Approximation of the fault tree	72
3.4.4	Cost calculation	74
3.4.5	Communication cable length calculation	75
3.4.6	Functional and communication loads calculation	76
3.5	Conclusions	77
4	Model transformations to introduce redundancy	79
4.1	Introduction	79
4.2	Bottom-up ASIL decomposition	80
4.2.1	Transformation 1: substitution	80
4.2.2	Transformation 2: connect consecutive redundant patterns	84
4.2.3	Transformation 3: separation of communication elements mapped to multiple resources or locations	87
4.2.4	Transformation 4: reduce the number of resources in the resource layer with optimized mapping	89
4.2.5	Transformation 5: collapse consecutive communication nodes	91
4.3	Application-oriented transformation process	92
4.3.1	Selection of the application node to substitute	94
4.3.2	Modification of the resource layer and mapping of the new nodes	94
4.3.3	Mapping of the new resources to the physical layer	103
4.4	Resource-oriented transformation process	104
4.5	Experiments	104
4.5.1	Simple application example	105
4.5.2	Use case: ECOTWIN II platooning system	115
4.6	Discussion of the transformation process	124
4.7	Conclusions	126
5	Automotive electrical and electronic architectures	127
5.1	Introduction and related work	127
5.2	Architecture topologies and definitions	131
5.3	Modeling and analysis of architecture topologies with fail-silent assumption	132
5.3.1	Impact of redundancy on different topologies	137
5.3.2	Discussion of the architecture topologies variations	150
5.4	Analysis of redundant architecture topologies without fail-silent assumption	153
5.4.1	Illustrative system implementation - application and resource layers	155

5.4.2	Evaluation of the four implementations	158
5.5	Conclusions	162
6	Conclusion and future directions	165
6.1	Conclusions	165
6.2	Future directions and possible extensions of the framework	167
	Bibliography	171
	List of abbreviations	187
	Terminology	189
	List of publications	191
A	Three-layer model recap	193
B	Input format	197
C	EcoTwin lateral control modelled graphs	203
D	Mapping of applications to architecture topologies	209

1

Introduction

1.1 Advanced driving assistance systems and autonomous vehicles

Modern high-end vehicles have more than 100 Electronic Control Units (ECUs) [32] to provide advanced functionality. The vehicle Electrical and Electronic (E/E) system executes various functions. While the previous generation of vehicles used electronics mostly for vehicle control, nowadays Advanced Driving Assistance Systems (ADASs) are common practice and will be even more used in future vehicles. The European Parliament established that some ADASs, e.g. lane-keeping assistant and warning of driver distraction, will be mandatory starting from 2022 [31]. To help the driver, an ADAS collects information from the surrounding environment via sensors or communication and uses actuators to either control the vehicle or inform the driver about a specific situation.

The automotive industry interests do not stop at assisting the driver: on the path to increase road safety and minimize the number of accidents, Autonomous Vehicles (AVs) are being developed. Many studies indicate that the main reason for road accidents is human error [65], which could be avoided by shifting the driving responsibility to the car electronic system. However, this leads to many technical and safety concerns, next to ethical discussions such as the *trolley problem* [173], where the AV must choose between two driving paths that both could lead to severe injuries or death of the passengers or other people involved.

To understand these concerns, we describe an Adaptive Cruise Control (ACC) system. We then use this example to describe the model that we propose in

Chapter 3. An ACC functionality is divided into two parts:

1. Detect the distance from the forward obstacle and compute the ego-vehicle current speed and acceleration;
2. Control the ego-vehicle dynamics to adjust the speed to the desired value while maintaining a safe distance from the forward obstacle.

For the first step, sensors such as cameras, radars, and LiDARs are used [98, 116, 172]. The environment, the road, and the obstacles can widely vary, but the ACC system must work safely in all conditions. For safety reasons, more than one sensor is used since each sensor has a different optimal working condition, and one or more sensors may not provide correct information to the system at a particular moment in time (e.g. due to occlusions) [69, 70, 124]. Sensor redundancy and sensor fusion reduce the risk of faults due to malfunctioning or misdetection. Moreover, wheel speed sensors, accelerometers, gyroscopes, and GPS systems are used [30, 131] to compute the ego-vehicle speed and acceleration. For the second step, the ACC system must calculate the necessary vehicle speed. It then requires direct access to the vehicle throttle and brake actuators to adjust the current vehicle speed.

The ACC operations must all be performed reliably: the forward obstacles must be detected in all driving conditions, and correct distance values must be measured; the ego-vehicle speed and acceleration must be obtained accurately; the necessary vehicle speed must be calculated without faults; and the vehicle actuators must receive correct and reliable signals.

While safety is a clear requirement, many technical aspects interest ADASs. The extensive use of sensors results in a vast amount of data, that not only must be processed by the ECUs but must also be transmitted via the In-Vehicle Network (IVN). The vehicle actuators require real-time inputs, meaning that an ADAS has a limited time to process the sensor information and provide the actuation signals. The space in the vehicle is limited, and the software applications are executed on embedded platforms that have constrained resources.

For a single ADAS such as the ACC system, the safety and technical requirements are strict and require verification. This is true for all the other electronic applications that are installed in a vehicle. When multiple applications interfere with each other, a system-level verification is required. The different systems can interfere directly via communication, or indirectly, for example by sharing sensors or other hardware resources. In the most complex vehicles, such as AVs, the system-level analysis to ensure system reliability and efficiency is one of the most complex parts of the development.

1.2 System requirements for ADASs and AVs

One of the main differences between modern ADASs and AVs from legacy vehicles is the use of a perception module. ADASs and AVs use perception sensors such

as cameras, radars, LiDARs, and ultrasonic sensors to reconstruct the surrounding environment. For example, Figure 1.1 shows the sensors used by a modern Volkswagen Golf for ADAS. Complex classifiers, usually neural-network-based as in [50, 62, 99, 163], are used for object detection, depth estimation, and other perception-based applications. Figure 1.2 shows the reconstructed front view of an NVIDIA autonomous vehicle prototype [145], in which pedestrians, other vehicles, traffic signs, and lanes are detected and classified.

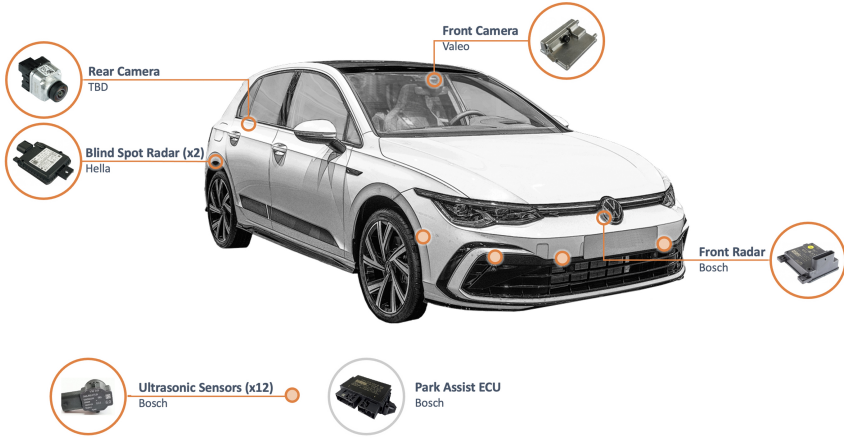


Figure 1.1: Perception sensors for ADASs mounted on a Volkswagen Golf 8 in 2021 [44], from System Plus Consulting.



Figure 1.2: Reconstructed front view from an AV prototype [145].

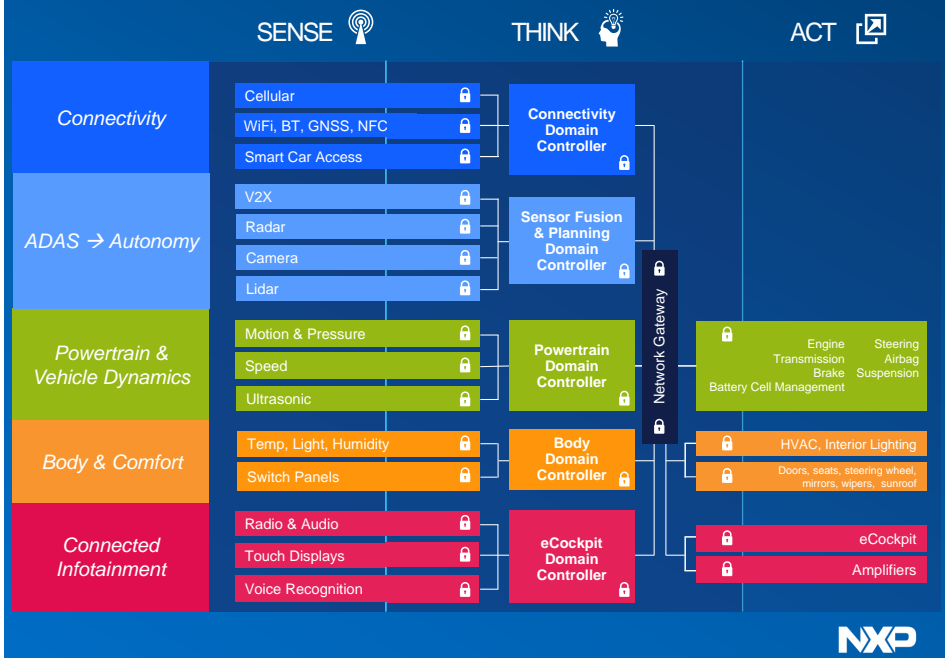


Figure 1.3: Sensors and actuators in automotive domains [112].

Other than perception sensors, many other sensors are used for advanced functionality. Figure 1.3 shows an overview of the classification of the different sensors based on the application domain they belong to. In this classification, we identify five application domains: connectivity, ADAS and autonomy, powertrain and vehicle dynamics, body and comfort, and connected infotainment. This classification is not standardized, and in literature it is possible to find different conventions [127].

Sensor redundancy is necessary for fail-operational automotive systems. In the example AV prototype from Figure 1.4, multiple sensors are used to cover the same environment areas but with different ranges and characteristics. Each of these sensors transfers data to computational platforms such as NVIDIA Drive PX2, NXP Bluebox, or dSPACE MicroAutoBox II. The data rates of perception sensors are high [37], and the software applications often require hardware accelerators to meet the real-time requirements. Because of the high data rates, part of the traditional in-vehicle networks such as CAN buses, LIN, and FlexRay are replaced by automotive Ethernet networks. At the same time, safety-critical signals such as actuator commands require real-time network capabilities, and often legacy IVNs are used for them. The combination of redundancy and the high number of sensors leads to complex IVNs and E/E architectures, e.g. Figure 1.5 shows

the cable harness of a Cruise AV test vehicle. To satisfy the different network requirements, the IEEE 802.1 working group is developing the Time-Sensitive Networking (TSN), a set of standards aiming to improve the standard Ethernet protocol, consisting of three main components: time synchronization, scheduling and traffic shaping, and selection of communication paths [53].

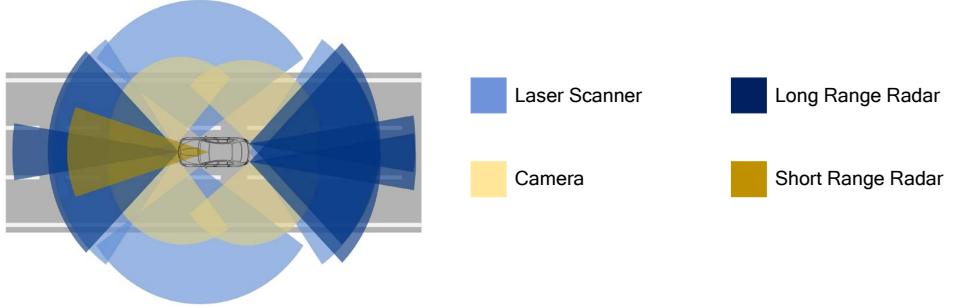


Figure 1.4: Perception sensors coverage in an AV prototype [2].

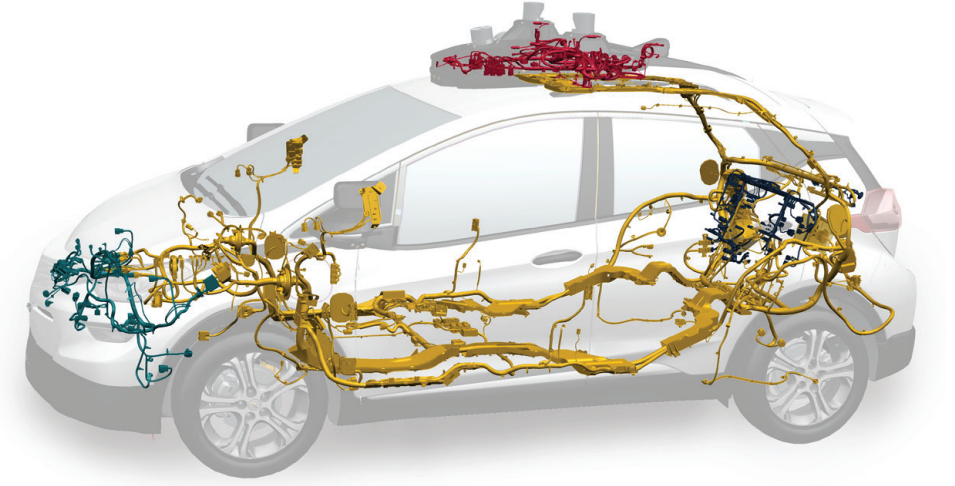


Figure 1.5: Cable harness of the Cruise AV test vehicle [60].

1.3 SAE automation levels

The Society of Automotive Engineers (SAE) has defined the intelligence level and automation capabilities of vehicles in the document J3016 [133]. Figure 1.6 shows the overview of the six automation levels.

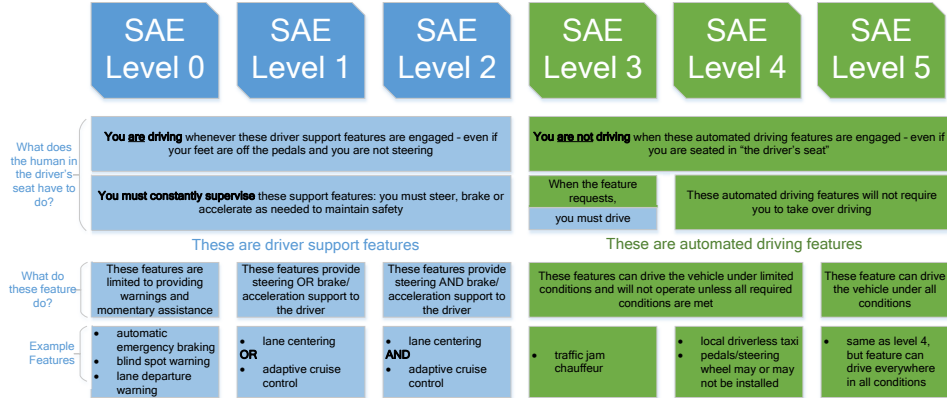


Figure 1.6: SAE J3016 levels of driving automation [133].

ADASs are usually categorized between levels 1 and 3, since they do not provide complete automation and still require a human driver to control the vehicle. The current commercial vehicles reach up to level 3 automation, in which the electronic system provides full control of the vehicle but may require a human driver to take control of the car in specific situations. These systems are *fail-safe*, meaning that in the presence of a fault, the system reaches a safe state. The shifting of driving responsibility from the driver to the electronic system pushed for new road regulations [165]. Levels 4 and 5 are considered fully AVs, in which no human driver is necessary. In the presence of a fault, a level 4 or 5 AVs can cope, either with a backup or a recovery mechanism, without human intervention. A system that provides full functionality even in the presence of a fault is defined as *fail-operational* [138,140]. Automotive safety-critical systems, of any automation level, often require fail-operational capabilities. For example, a brake-by-wire system, a safety-critical ADAS, requires fail-operational capabilities [152]. Figure 1.7 shows the fail-operational E/E brake-by-wire system of [152], in which redundant fail-silent units are used for the electric brake control module and sensors and actuators are connected with redundant CAN and FlexRay connections.

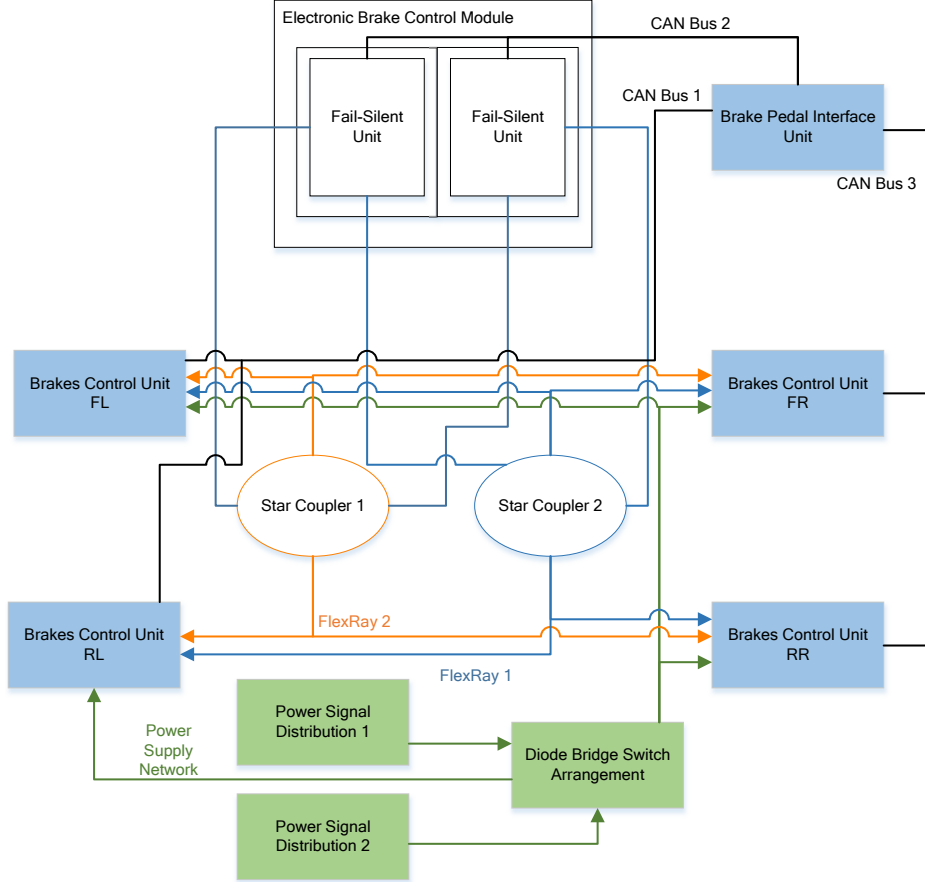


Figure 1.7: A fail-operational brake-by-wire E/E architecture. Reproduced from [152].

1.4 Functional safety

Functional Safety for automotive is defined in the ISO26262 *Road Vehicles - Functional Safety* standard [81] as “the absence of unreasonable risk due to hazards caused by malfunctioning behaviour of electrical/electronic systems”. The standard aims to eliminate the risk in electronic and electrical components in vehicles, which can result in injuries or damage to the overall health of people. It defines the Automotive Safety Integrity Level (ASIL), which is a risk-based parameter that determines the *severity*, *controllability*, and *probability of exposure* related to a specific Functional-Safety Requirement (FSR). Many safety-critical industries follow strict functional safety standards:

Table 1.1: Comparison of SIL levels of functional-safety standards.

FuSa Standard	Safety Levels (lowest to highest)				
IEC 61508	-	SIL 1	SIL 2	SIL 3	SIL 4
ISO26262	ASIL A	ASIL B	ASIL C	ASIL D	-
DO-178C	Level E	Level D	Level C	Level B	Level A
IEC 62304	Class A	Class B		Class C	
EN 50128	SSIL 0	SSIL 1	SSIL 2	SSIL 3	SSIL 4

- IEC 61508 *Functional Safety of Electrical/Electronic/Programmable Electronic Safety-related Systems* [79] is the main functional safety standard applicable to all industries. It defines the *safety life cycle*, a process that aims to discover design errors and omissions. It also defines the Safety Integrity Level (SIL), a parameter associated with a component that is based on the frequency and severity of hazards. It determines the safety requirements of each component and indicates its probability of failure.
- EN 50128 is the functional safety standard used in the rail industry that “specifies the process and technical requirements for the development of software for programmable electronic systems for use in railway control and protection applications” [51]. It also has a definition of SIL, different from the main standard.
- IEC 62304 focuses on medical devices, defining software safety classification to set the device requirements based on risk parameters [78].
- DO-178C *Software Considerations in Airborne Systems and Equipment Certification* is the primary functional safety standard for commercial software-based aerospace systems [52]. It defines the Design Assurance Level (DAL) based on the effect of a failure condition on the aircraft.

While all the SIL or the equivalent parameters are not interchangeable, in Table 1.1 we compare them to clarify the relationship between them.

In Chapter 2 we discuss in more detail the ISO26262 standard and its impact on automotive systems.

1.5 From fail-safe to fail-operational architectures

With increasing automation in the vehicle functionality, the responsibility of safely driving the vehicle shifts from the driver to the E/E system, as shown in Figure 1.8. The system switches from *fail-safe* requirements, in which the system can fail and reach a safe state (in this case by using the driver as the fallback mechanism), to *fail-operational* requirements, in which the system must function even in the presence of a fault.

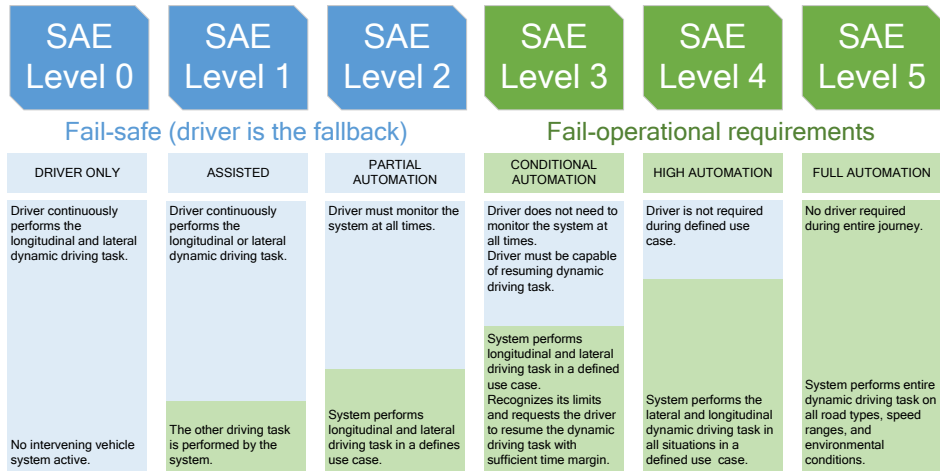


Figure 1.8: Driving responsibility in the different SAE automation levels. Reproduced from [162].

The failures in automotive systems can come from two main categories [38]: failures related to the *infrastructure* and failures related to the *vehicle components*. The first category is shown in Figure 1.9 and contains failures related to elements external to the vehicle, e.g. other road users or environmental conditions. The second category, shown in Figure 1.10, contains the failures that are related to the vehicle E/E architecture.

The infrastructure related failures can be mitigated with robust system design, e.g. using multiple sensors to collect information from the surroundings of the vehicle. For example, in case of adverse weather conditions (e.g. too much light pointing towards a camera) at least one of the redundant sensors can provide correct data, e.g. a radar. The more information the vehicle can collect from its surroundings, the lower the chance of failures related to the infrastructure can happen.

The vehicle component failures focus instead on what happens inside the vehicle: the hardware or software components can fail, or mechanical failures can happen. Similar to the previous scenario, robust and safety-oriented design

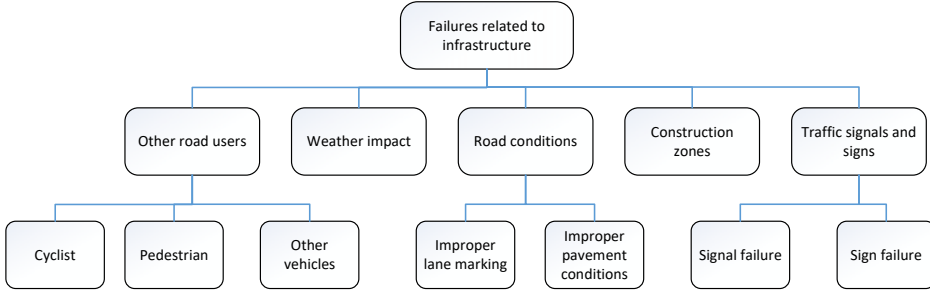


Figure 1.9: Possible failures related to transportation infrastructure. Reproduced from [38].

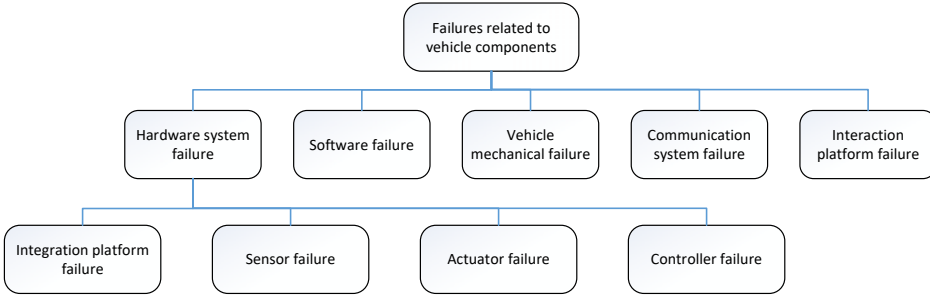


Figure 1.10: Possible failures related to AV components. Reproduced from [38].

is key to avoid these types of failures. Redundancy in the E/E architecture helps to provide fail-operational capabilities. In this work, we focus on how to add redundancy when needed and on how to analyse redundant automotive systems.

1.6 Problem statement and contributions

Developing an ADAS or an AV is a complex task. The many requirements, both safety and technical, result in an intricate design phase. Many developer teams working on different aspects of the vehicle are involved. With our research, we want to bridge the gap that exists between the development of the automotive electronic system and the functional safety analysis. In this thesis, we describe how to model and transform an automotive system to validate its cost and FSRs. The high-level research question in this thesis is: *A) How to model an automotive system B) so that we can evaluate quantitative architecture and functional safety metrics and C) so that we can introduce safety patterns and components iteratively. D) The model shall be proven on current and future or emerging automotive architecture.*

Our first contribution is a *model* that describes a complex automotive system such as ADASs (Chapter 3). The application, resources, and physical space are modelled each in a separate layer. The mapping between different layers establishes the relationships between them. An *evaluation framework* is developed to analyse a modelled system, quantifying high-level values such as cost, the failure probability of the applications, total functional load, total communication load, and total cable length. ASIL parameters are annotated in the model for traceability of the FSRs and are used to evaluate the parameters of interest.

As second contribution, we then define specific architectural elements that manage the redundant parts of the system: the *splitters* and *mergers* (Chapter 3). We analyse how these safety-oriented elements are implemented in existing automotive redundant systems, focussing on explicit design patterns used to achieve high system reliability. We propose *model transformations* that use these elements to introduce spatial redundancy (opposed to temporal redundancy, as discussed in Section 2.4). The goal is to achieve the required fail-operational status for ADAS and AV. The transformations are based on the ISO26262 ASIL decomposition technique, which is used to lower FSRs ASIL values (Chapter 4). The transformed model can be evaluated again in our framework to compare different solutions. The ASIL decomposition is verified with a *Common-Cause Fault (CCF) analysis* that analyses the independence of the obtained redundant parts of the system.

The many requirements of ADAS and AV have a great impact on the E/E architecture. While in previous-generation vehicles ad-hoc networks and ECUs were used, in modern systems more standardized solutions are needed. Current researchers anticipate Domain-based and Zone-based solutions as future-proof automotive architectures. As a third contribution, we use our evaluation framework and model transformations to analyse the two main architecture topologies in different conditions. We quantify the differences between the two topology types in the presence of redundant and mixed-critical applications. With our framework, we can guide the automotive system designer towards the correct architectural choice for a specific ADAS or AV system (Chapter 5).

As a final contribution, we analyse the effect of isolation techniques on the system-level analysis of an ADAS or AV system. Mixed-criticality can potentially lead to interferences, and non-critical applications can modify the behaviour of safety-critical ones if no isolation is present. Physical separation and virtualization techniques are analysed, and their impact is quantified in different architecture topologies scenarios (Chapter 5). Table 1.2 highlights the contributions of the thesis compared to the main research question.

Table 1.2: Contributions related to the research question.

Research question	Thesis chapter(s)	How
A) Modelling of automotive systems	3	Three-layer model of automotive systems with safety-oriented elements (Contrib. 1)
B) Quantitative evaluation of the system to allow comparison of different E/E implementations	3	Quantitative analysis of reliability, cost, physical space, and resource utilization (Contrib. 2)
C) Introduction of safety patterns and components in the system	4	Transformations of the system architecture to introduce redundancy (Contrib. 3)
D) Applicability to future architecture trends and case studies	4 and 5	Analysis of current and future automotive architecture topologies (Contrib. 4)

To validate our framework, all the contributions are supported by real-life or synthetic use cases. Our experiments provide insights for the development of safe ADASs and AVs. The results are system dependent, but our framework is a key point in the analysis of different implementations.

1.7 Thesis outline

The remainder of the thesis is organized as follows: Chapter 2 introduces the necessary background information related to the automotive safety standards, the safety life cycle, and the architecture safety patterns. Chapter 3 describes the details of the proposed three-layer model and introduces the *splitter* and *merger* components. Moreover it describes the quantified analysis that is performed in our framework. Chapter 4 characterizes the procedure for model transformations to introduce redundancy in the system. An example application and the EcoTwin II Platooning System [23] lateral control application are evaluated as use cases in this chapter. The work in Chapters 3 and 4 has been previously published in [55, 56]. The Domain-based and Zone-based EE architectures are defined and evaluated in Chapter 5. The impact of redundancy is quantified in each architecture scenario. Isolation techniques are evaluated in Chapter 5 as well, in which physical separation and virtualization are implemented in the previous-mentioned architecture topologies to quantify their respective impact. Chapter 5 was partially published

in [57,58]. Last, Chapter 6 concludes the thesis and provides an overview of future direction for safety-oriented ADASs and AVs system analysis.

2

Background and related work

2.1 Introduction

The research field of automotive systems is vast. In this thesis, we focus on the functional safety aspects defined in Chapter 1. Many safety-oriented techniques are used, e.g. the data stored in memories can be monitored by using Error-Correcting Code (ECC), Built-In Self-Test (BIST) capabilities, watchdog timers used to monitor the state of processors. At the architecture level instead, the most common technique to obtain highly reliable systems is redundancy [166]. For example in the avionic field, where the electronic system has many responsibilities in the control of the aircraft, multiple levels of redundancy are present [68, 86, 142, 154].

The automotive industry's goal is to obtain fail-operational systems, in which even in the case of a failure of part of the system, the full functionality can still be provided. In this scenario, a redundant system is necessary to substitute the failed part. However, simply duplicating the full system might not be the correct solution: the source of the failure can be present in a duplicated identical system as well, and if the two redundant parts are not independent the system reliability may not be improved [81]. Moreover, more components lead to a more complex system, where more sources for faults are present.

This is why we focus our attention on redundant automotive systems and the management of redundant elements. In this chapter, we provide the necessary background information to understand the modelling and analysis framework that is proposed in the following chapters. First, we introduce the de facto standards

used in the automotive industry for Advanced Driving Assistance System (ADAS) and Autonomous Vehicles (AVs). We then discuss the building of the Safety Case for a new system. The Automotive Safety Integrity Level (ASIL) decomposition technique of the ISO 26262 standard is described and is used in our model transformations to introduce redundancy in selected parts of the system in Chapter 4. We then describe general redundancy techniques and architecture patterns that are used in safety-critical automotive systems.

2.2 Automotive functional safety standards

2.2.1 ISO 26262

ISO 26262 “Road vehicles - functional safety” [81] is an automotive functional-safety standard derived from IEC61508 [79]. The standard is divided into 12 parts:

1. Vocabulary
2. Management of functional safety
3. Concept phase
4. Product development at the system level
5. Product development at the hardware level
6. Product development at the software level
7. Production, operation, service and decommissioning
8. Supporting processes
9. ASIL-oriented and safety-oriented analysis
10. Guidelines on ISO 26262
11. Guidelines on application of ISO 26262 to semiconductors
12. Adaptation of ISO 26262 to motorcycles

While part 1 defines the terminology used in the standards, parts 2 to 8 construct the safety life cycle of an automotive electronic system, as shown in Figure 2.1.

Figure 2.2 shows the multiple V-Models that are implemented in the standard: the design, validation, and testing is done in parts 4, 5, and 6, at the system, hardware, and software level respectively.

In part 3, during the concept phase, the *item* is defined. An *item* is a specific system, or combination of systems, to which the safety life cycle is applied. During the Hazard Assessment and Risk Analysis (HARA) step, a comprehensive set of hazards is identified for the item. Each hazard is classified according to three parameters: *severity*, *probability of exposure*, and *controllability*. The severity parameter is assigned based on the potential harm caused to each endangered person, including the driver or the passenger of the vehicle causing the hazardous event and other external endangered people such as cyclists or pedestrians. The probability of exposure is the probability of an event occurring during the operating time. The controllability parameter is based on the ability of the driver or other

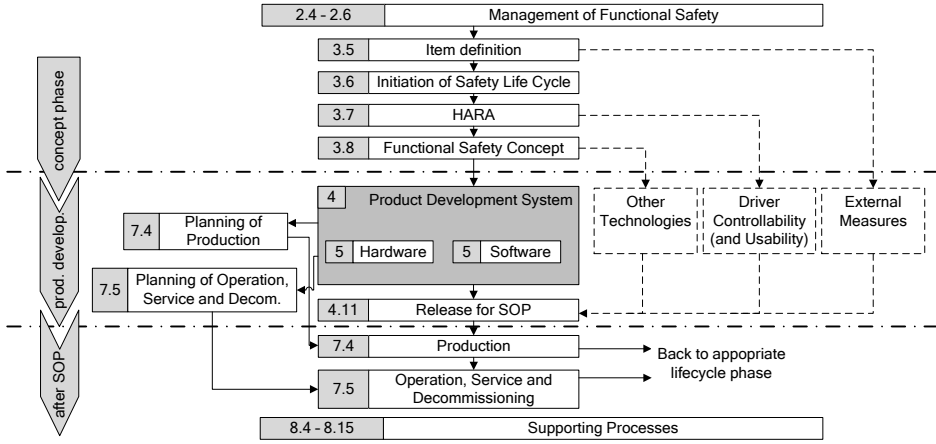


Figure 2.1: ISO 26262 safety lifecycle [81].

traffic participants to mitigate the hazard. Table 2.1 shows the values that can be assigned to each hazard and their description.

An Automotive Safety Integrity Level (ASIL) is determined for each hazardous event. Four ASILs are defined: ASIL A, ASIL B, ASIL C, and ASIL D, where ASIL A is the lowest safety integrity level and ASIL D the highest one. Additionally, the class Quality Management (QM) denotes no safety requirement in accordance with ISO 26262. Table 2.2 shows the ASIL definitions based on the three risk parameters. Note that the highest ASIL is assigned only in case of S3, E4, and C3 (maximum value of each risk parameter).

Once the hazardous event has been evaluated, a Safety Goal (SG) shall be determined. A SG is a top-level safety requirement for the item, expressed in terms of functional objectives. The ASIL determined for the hazardous event is inherited by the corresponding SG.

From the SGs, Functional-Safety Requirements (FSRs) are derived and allocated to the preliminary architectural *elements*. They inherit the ASIL of the SG, and if an FSR is related to multiple SGs, it will inherit the highest ASIL between them. The FSRs are furthermore refined into Technical Safety Requirements (TSRs), lower-level requirements allocated to hardware and software elements and so on. A summary of the structure of the safety requirements derived in the ISO 26262 standard is shown in Figure 2.3.

A comprehensive study on functional safety and the application of the ISO 26262 standard to automotive systems is found in [132]. In this work, the author proposes new models and tools to model the work flow of the safety lifecycle and to model safety analysis specifications, investigates safety patterns for safety-critical systems (see Section 2.5), and discusses the safety culture in automotive organizations. The holistic safety domain model presented in [132] focusses on

Table 2.1: Risk parameters assigned to hazards.

Severity (S)	
S0	No injuries
S1	Light and moderate injuries
S2	Severe and life-threatening injuries (survival probable)
S3	Life-threatening injuries (survival uncertain), fatal injuries
Probability of exposure (E)	
E0	Incredible
E1	Very low probability
E2	Low probability
E3	Medium probability
E4	High probability
Controllability (C)	
C0	Controllable in general
C1	Simply controllable
C2	Normally controllable
C3	Difficult to control or uncontrollable

Table 2.2: ASIL definition based on the risk parameters.

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

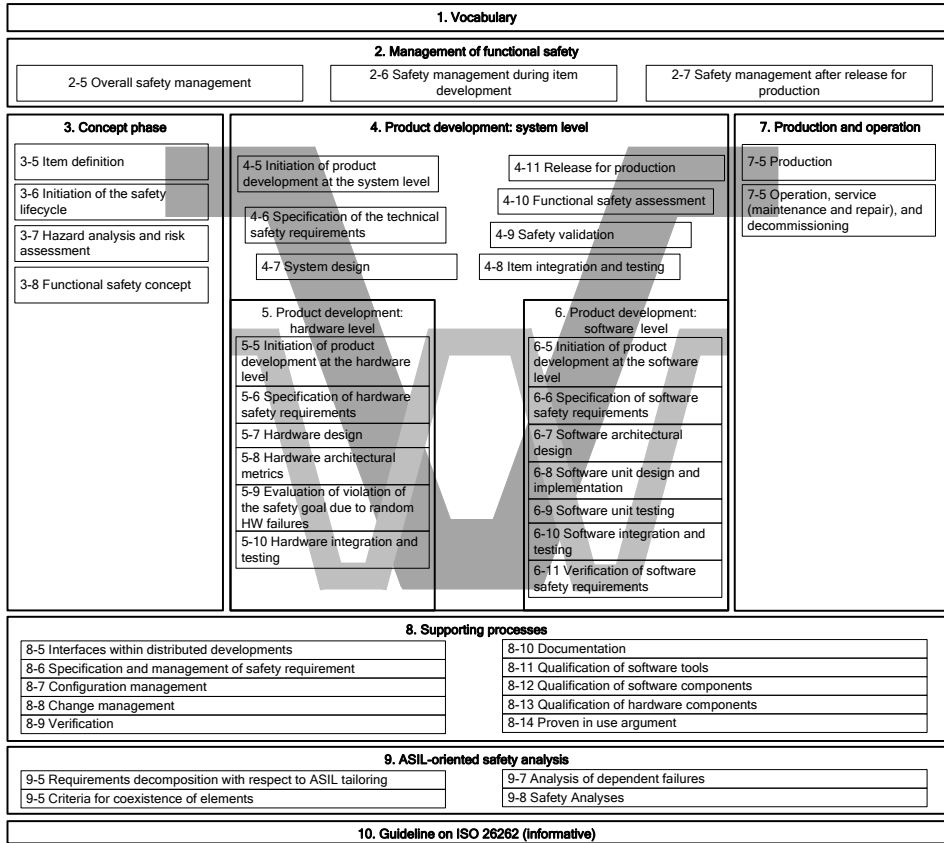


Figure 2.2: ISO 26262 structure based on a V-Model [81].

specifying system design and safety process aspects, and uses UML as the notation language. Moreover, it analyses the HARA process for the described system. In our model, we describe instead the interactions between the application, the hardware resources, and the physical space. We assume that the results of the HARA procedure are known. We then focus on an automated design processes to introduce redundancy in the system and describing different architecture topologies with the model, to understand the costs and benefits of future automotive trends.

Complementary to the ISO 26262 standard, the ASPICE [169] standard is the current standard for software best practices in the automotive industry. It addresses non-safety related concerns, such as cost and scheduling impacts on the system, and it focuses on the process implementation.

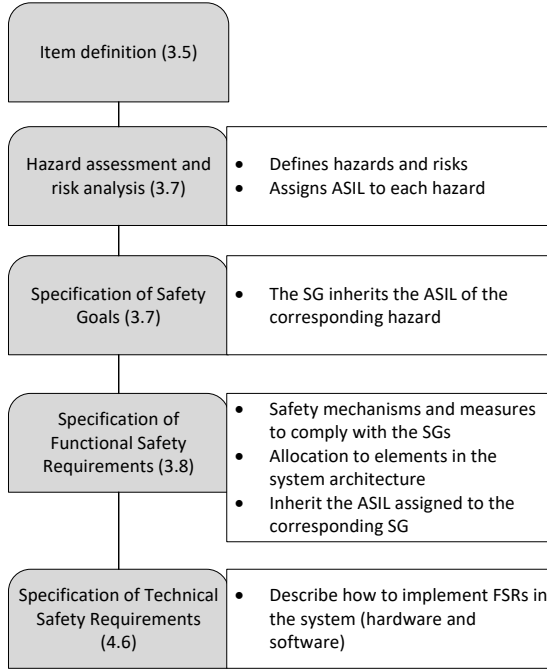


Figure 2.3: Summary of requirements specification in ISO 26262.

2.2.2 ASIL decomposition

A safety requirement can be decomposed into redundant safety requirements during the ASIL allocation process to allow the implementation of higher ASIL elements with lower ASIL ones. This procedure is referred to as *ASIL decomposition*. This technique can be applied to a safety requirement of any level (functional, technical, hardware, or software safety requirement). Benefits can be obtained by using ASIL decomposition in case of the existence of independent architectural elements (e.g. redundant resources with separate power supplies and separate communication networks), offering the opportunity to implement safety requirements redundantly and assign potentially lower ASIL to these redundant safety requirements. The elements must be independent, otherwise, the ASIL cannot be lowered.

Elements are independent when there is no dependent failure leading to a SG violation. To understand the meaning of this, we describe the two types of dependent failures that can occur. Dependent failures can either be Common-Cause Fault (CCF) or Cascading Fault (CF).

A CCF is a failure due to a single specific event that causes multiple elements to fail, as shown in Figure 2.4. The fail event leads to a fault in both elements A

and B, that cause their failures.

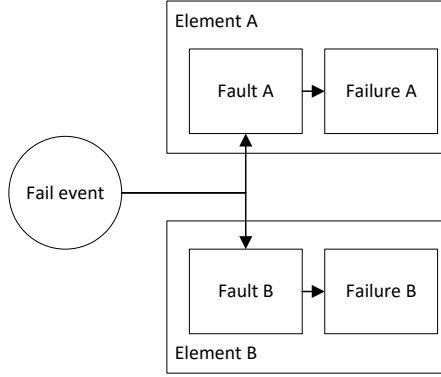


Figure 2.4: Common cause failure leads to the failure of elements A and B [41].

A CF is a failure that causes an element to fail, which in turns causes a successive element to fail, as shown in Figure 2.5

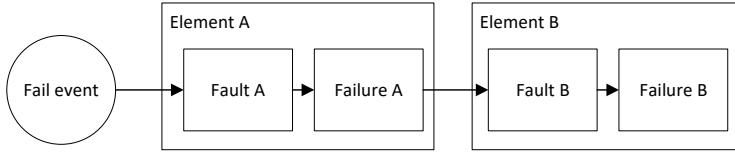


Figure 2.5: Cascade failure of elements A and B caused by a single fail event [41]

The standard defines *freedom from interference* when there is no CF that would lead to the violation of a SG between two or more elements. Figure 2.6 summarizes the conditions in which independence is achieved.

Figure 2.7 shows the ASIL decompositions permitted by the standard. For example, an ASIL D requirement can be decomposed into:

- An ASIL C and an ASIL A requirements;
- Two ASIL B requirements;
- An ASIL D and a QM requirements;

The ASIL decomposition can be applied recursively to already decomposed requirements, to further lower their ASIL. However, the standard specifies that the confirmation measurements, such as review of the safety, integration, and validation plans, must be applied in compliance with the ASIL of the SG, which

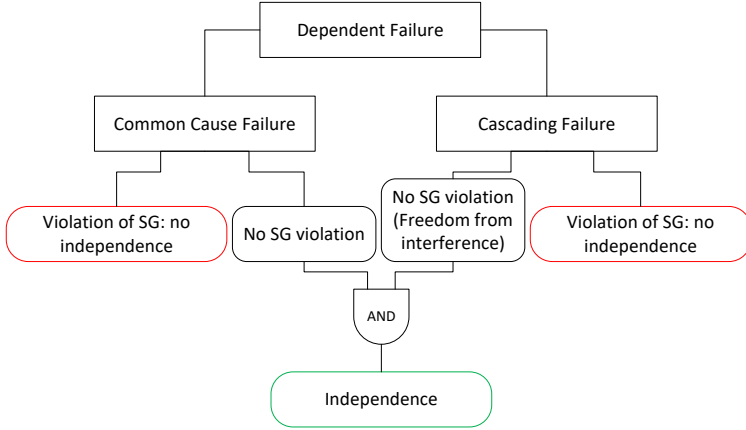


Figure 2.6: Independence is obtained with the absence of SG violations due to either CCF or CF [41].

corresponds to the ASIL of the safety requirement before the ASIL decomposition. This is why the original ASIL is annotated in the ASIL decomposition table of Figure 2.7, e.g. ASIL B(D).

In literature, we find different works related to ASIL decomposition. The authors of [171] discuss how to apply the technique correctly and common mistakes that are made in the industry when implementing redundancy, focusing on the independence requirements for redundant elements. The works [13, 14, 43, 105, 106, 118, 134] perform automatic ASIL allocation after the decomposition on a specific system, using different types of automated techniques to achieve correct allocation. Other works instead try to minimize cost functions by decomposing the safety requirements [175, 176]. Our experiments use concepts from related work, such as cost metrics related to the ASIL specifications, but use the ASIL decomposition technique in a different way. Instead of allocating the ASIL requirements on the different parts of an existing system, we modify the current implementation by adding redundancy elements. These elements are, by design, following the ASIL decomposition rules, and the decomposed FSR can be assigned to them. The new and redundant system is then evaluated in our analysis framework. In Chapter 4 we describe the model transformations that introduce these redundant elements.

2.2.3 ISO/PAS 21448 - SOTIF

While the focus of this thesis is on the ISO 26262 standard, it is worth mentioning additional functional safety standards related to ADAS and AV.

The ISO/PAS 21448 “Road vehicles - safety of the intended functionality” focuses on *the absence of unreasonable risk due to hazards resulting from functional insufficiencies of the intended functionality or by reasonably foreseeable misuse by*

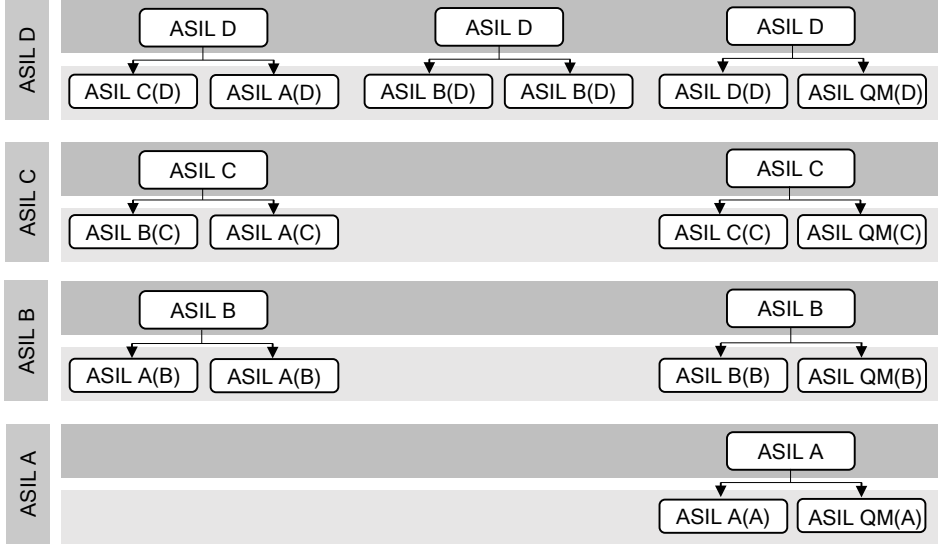


Figure 2.7: Possible ASIL decompositions from the ISO 26262 standard.

persons. A malfunction can occur even in absence of a fault in the architectural elements, and this scenario is not covered by the ISO 26262 standard. An example scenario is not detecting another vehicle on the road, which could cause fatal consequences [120].

In response, companies involved in AV research developed a set of rules for safe vehicle behaviour. Intel released the Responsibility-Sensitive Safety (RSS) rules that formalize the human notion of safe driving into five rules [59]:

1. Do not hit the car in front (longitudinal distance)
2. Do not cut in recklessly (lateral distance)
3. Right of way is given, not taken
4. Be cautious in areas with limited visibility
5. If a vehicle can avoid a crash without causing another one, it must

The implementation of this ruleset in the AV decision-making software leads to decisions that are safe and predictable. Each rule corresponds to specific calculations done by the software to plan a safe trajectory. For example, Figure 2.8 shows the distance required to maintain safe longitudinal distance from a vehicle moving in the same direction, where ρ is the response time, α_{max} is the maximum acceleration during the response time, v_r and v_f are the longitudinal velocities of the two cars, β_{min} and β_{max} are the minimum and maximum brake forces applied

by the vehicle. The RSS ruleset is implemented by various industry peers and standard organizations. For example, it is adopted in the Apollo autonomous driving platform [16, 77].

DEFINE SAFE LONGITUDINAL DISTANCE

$$d_{\min} = \left[v_r \rho + \frac{1}{2} \alpha_{\max} \rho^2 + \frac{(v_r + \rho \alpha_{\max})^2}{2\beta_{\min}} - \frac{v_f^2}{2\beta_{\max}} \right]_+$$

C_r
C_f
d_{min}

Figure 2.8: Minimum distance to maintain from the successive vehicle [144].

Similar to the RSS, NVIDIA promotes the Safety Force Field (SFF) [108, 109], to provide the basic mechanisms between actors on the road to avoid collisions.

In our work, we limit the research to ASIL-oriented analysis, and we do not further discuss the ADAS or AV intentions.

2.2.4 UL4600

While ISO 26262 and ISO/PAS 21448 expect a human driver as a fall-back safety mechanism (up to autonomy level 3), the UL4600 standard for the evaluation of autonomous products [164] has fully autonomous systems as a target [89]. It includes topics such as safety case construction, risk analysis, testing, validation, metrics and conformance assessment. It is intended as a supplement to the other standards, providing more detailed treatment regarding fully autonomous systems. Compliance with this standard includes other safety standards such as the previously mentioned two.

Figure 2.9 shows the landscape of the functional-safety standards and the tasks of an automotive electrical system that they influence [88].

2.3 Faults and safety mechanisms

In this section, we define what types of faults we analyse in this thesis, and what sort of safety mechanisms are used in automotive systems.

The ISO 26262 defines a *fault* as an *abnormal condition that can cause an element or an item to fail*. In part 5, the standard specifies different types of faults for a safety-oriented element [33]:

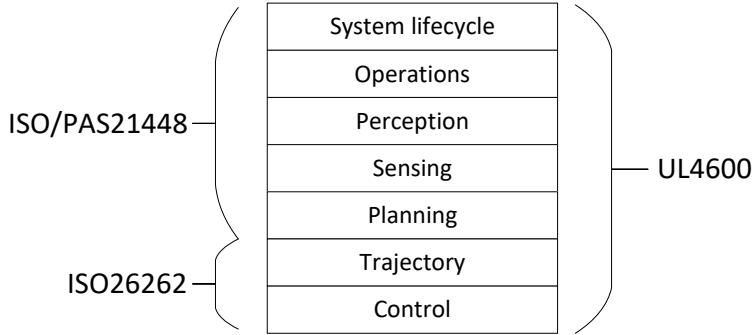


Figure 2.9: Overview of safety standards related to the automotive system [88].

- **Safe fault (λ_S):** the fault does not cause the safety-oriented element to fail. Either a safety mechanism protects the element from failure or the fault is harmless for the system.
- **Single-point fault (λ_{SPF}):** a fault that causes the failure of an element because no safety mechanism is present. In an ASIL C or D system, these faults are not tolerated.
- **Residual fault (λ_{RF}):** a fault that causes the failure of the safety-oriented element (same as *single-point fault*). However, in this scenario, a safety mechanism is present in a safety-oriented element but has a limited *coverage*. The *residual fault* is part of the faults that are not covered by the safety mechanism. In an ASIL C or D system, these faults are tolerated if their probability is low.
- **Multiple-point fault ($\lambda_{MPF,D}, \lambda_{MPF,L}$):** a fault that in combination with other independent faults leads to a multiple-point failure. The individual multiple-point fault does not cause the violation of the SG of the element by itself, but it does when it occurs in combination with the other independent faults it is linked to. A multiple-point fault can be *perceived*, in which case a safety mechanism detects the multiple-point fault and reacts, or *latent*, where no safety mechanism detects the fault (e.g. a failure in the safety mechanism itself which remains undetected and non-perceived until the related multiple-point faults occur).

The fault rate λ of a safety-oriented element is calculated by combining the failure rates related to each fault, according to Equation 2.1 [63]. Figure 2.10 summarizes the possible failure modes of a hardware element.

$$\lambda = \lambda_S + \lambda_{SPF} + \lambda_{RF} + \lambda_{MPF,D} + \lambda_{MPF,L} \quad (2.1)$$

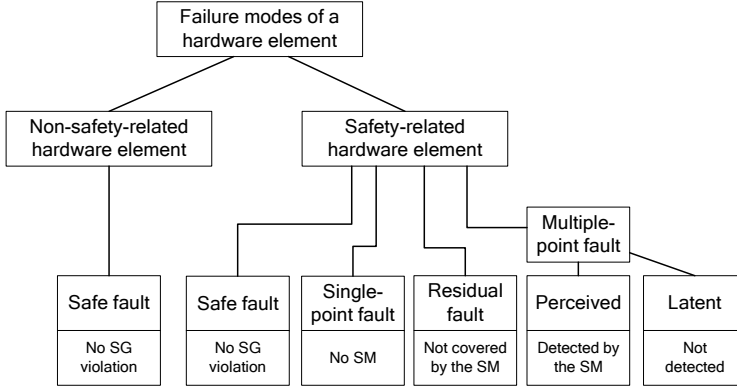


Figure 2.10: Summary of failure modes of a hardware element.

The fault rates are dependent on the safety mechanisms that are implemented in the system. A safety mechanism enhances the safety of electric and electronic systems. There are a variety of safety mechanisms that can be implemented in automotive systems, either in hardware or software form. We can categorize the safety mechanisms in [35]:

1. **Detect and correct:** a safety mechanism that monitors an element, and when a fault is identified, it can correct it. The fault does not turn into a failure of the element, functionality is maintained and the fault is not perceived by the other elements of the system in a *fail-operational* way.
2. **Detect and inhibit:** a safety mechanism that monitors an element, and when a fault is identified, it stops the element. The functionality is not maintained in this case and the fault is perceived. The fault becomes a failure of the element, but it is set to a defined inhibited state. The failure will not affect other elements of the system, avoiding e.g. babbling idiot scenarios [27] or byzantine failures [45], in a *fail-silent* way.

A safety mechanism is always formed by two parts: a detection and an actuation. Examples of safety mechanisms are:

- redundancy; e.g. lockstep [17, 155], triple modular redundancy [39, 177], redundant multithreading [129, 137];
- self-test; e.g. built-in self-test [47, 174], software-based self-test [19, 126], logic built-in self-test [121]), watchdogs [34, 158];
- error correcting codes [87, 143, 170].

The diagnostic coverage is one of the parameters that measures the effectiveness of a safety mechanism. It represents the percentage of faults that can occur in the monitored elements that are detected and solved by a safety mechanism [63, 107]. High diagnostic coverage leads to a lower residual fault rate λ_{RF} . The possibility of failure of a safety mechanism is included in the failure rate calculation as a multi-point fault: in the case of a simultaneous failure of the safety mechanism and a fault that is part of its diagnostic coverage, the system fails because of a multi-point fault. Multiple layers of safety mechanisms can exist: in the case of extremely safety-critical systems, additional safety mechanisms can monitor the first level safety mechanisms.

In this work, we focus on architectural safety mechanisms that are based on the use of redundancy and monitoring elements. We do not discuss hardware and software safety mechanisms further, however, we capture them in our model with the failure rates described in Chapter 4.

2.4 Fault tolerant schemes

As mentioned in the previous section, our focus on obtaining fault-tolerant and fail-operational automotive systems is on architectural safety mechanisms, and in particular on redundancy. The use of redundancy, if applied correctly, improves the system reliability. In a redundant system, the failure rate of a redundant application is not based only on the reliability of the components, but also on their architectural configuration [170].

Many different types of redundancy can be used, and we can distinguish between hardware, time, and information redundancy [91]:

- **Hardware redundancy** is divided into *static* (also called *passive*), *dynamic* (also called *active*), and *hybrid*. In static hardware redundancy the system masks faults by selecting correct outputs from the redundant parts [90]. In dynamic hardware redundancy, the system is reconfigured in the presence of a fault to switch to a fault-free spare element [115]. In hybrid redundancy both approaches are combined, the faults are masked and the system can be reconfigured [100].
- **Time redundancy** does not require additional hardware, but it can only detect and correct temporary faults. However, temporary faults happen more frequently than permanent faults [178], meaning that time redundancy can be a cost-efficient safety mechanism to reduce temporary-fault-induced failures, at the expense of performance because of the repeated executions. Time redundancy techniques are divided into *repeated execution* [104], *multiple sampling of outputs* at a different moment in time [54], and *diverse recomputation* on the same hardware [113].
- **Information redundancy** is based on error detecting and correcting codes, using fewer hardware resources compared to fully redundant elements. Upon detection of an error, the system can either correct it with the redundant information or roll back to a previous fail-safe state. These techniques are divided into *error detection* and *error correction* ones.

Moreover, when the functionality is executed more than once to compare its outputs, we encounter two scenarios:

1. **Homogeneous redundancy:** the functionality is replicated with identical redundant hardware and software elements. If these multiple replications are independent, the *availability* of the system is improved.
2. **Heterogeneous redundancy:** the functionality is replicated with different hardware and software elements. If the multiple replications are independent, both the system *availability* and *reliability* are improved since the failure modes of the multiple implementations are different.

In this work, we analyse redundancy at an architectural level, meaning that we consider hardware implementations that use multiple redundant elements. Safety mechanisms such as time and information redundancy are captured by the failure rates of the elements. For example, the failure rate of a memory that uses memory self-test for error detection and correction will be lower than a memory with no safety mechanism.

2.5 Architecture safety patterns

The ISO 26262 standard recommends to adhere to well-established architecture principles, also called architecture patterns. The concept of architecture pattern is also mentioned in the ISO/IEC/IEEE 42010 standard [80] as a way to describe and classify system designs. In the next paragraphs, we describe safety-related architecture patterns from the literature [9, 101, 135, 136], starting from a base channel and building up towards more complex and safe architectures.

A *channel*, shown in Figure 2.11, is a path via which data flows, usually from sensors to actuators. The inputs are processed in the input processing block, which translates sensors and signals from other systems to usable information. The data processing block analyses the input data to generate an output for the actuators. Finally, the output processing block translates the high-level output data to low-level control signals for the actuators. No architectural safety measure is present in the single-channel architecture.



Figure 2.11: Single channel functional view [8].

The first architecture pattern, the *protected single channel*, is shown in Figure 2.12. The actuation channel is formed by the processing channel of the single-channel architecture plus actuation monitoring and data validation blocks. In this non-redundant pattern, feedback and feed-forward loops are used to detect possible failures. When an unsafe fault is detected, the system output is not valid. No additional measure can be taken, other than shutting down this part of the system for a *fail-silent* behaviour towards the other applications. Moreover, a permanent failure that compromises one of the steps in the actuation channel cannot be recovered, since no backup is available.

The second architecture pattern is shown in Figure 2.13. This advanced pattern is called *safety executive*, in which a fail-safe processing channel is paired to the actuation channel. A safety coordinator monitors the two channels and selects which one to use to send the actuation control signals. The safety channel provides degraded functionality: simpler operations are executed to reach a safe state of

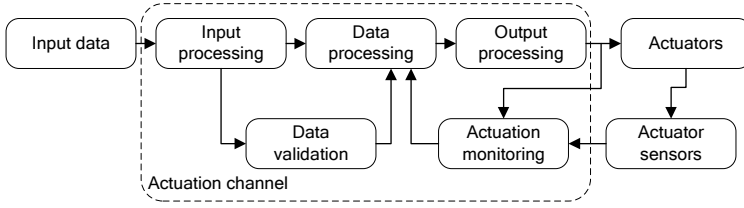


Figure 2.12: Protected single channel architecture pattern [8].

the vehicle, such as an emergency brake. In this case, the ASIL decomposition can be applied to decompose a safety requirement over the two channels. The safety executive module, which is a single point of failure element of the pattern, inherits instead the ASIL of the original SG. This system is *fail-safe* since the safety channel only provides degraded functionality, but it is not *fail-operational*.

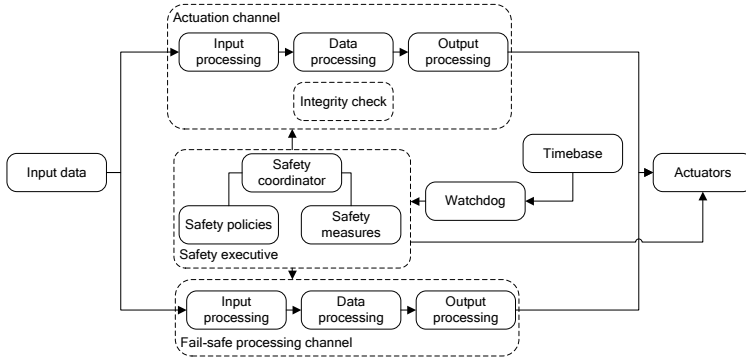


Figure 2.13: Safety executive architecture pattern [8].

The third architecture pattern, shown in Figure 2.14, is redundancy with full functionality, thus fail-operational. In case a failure is present in the primary channel, the fault detector can identify it and switch to the secondary channel. These two channels can be identical duplicates (same hardware and software) of a function or diverse implementations that use different hardware resources and different software. The fault detector and the switch are in common between the different channels, and inherit the ASIL of the original SG, while ASIL decomposition can be applied to the safety requirements related to the processing channel, to be decomposed over the multiple redundant channels. An example of the homogeneous redundancy is the M-out-of-N (MooN) voting [36], shown in Figure 2.15: the outputs of N duplicates of the system are compared and if M values coincide they are considered correct. The typical implementation of this is in the triple modular redundancy, a *2oo3* architecture pattern. Note that

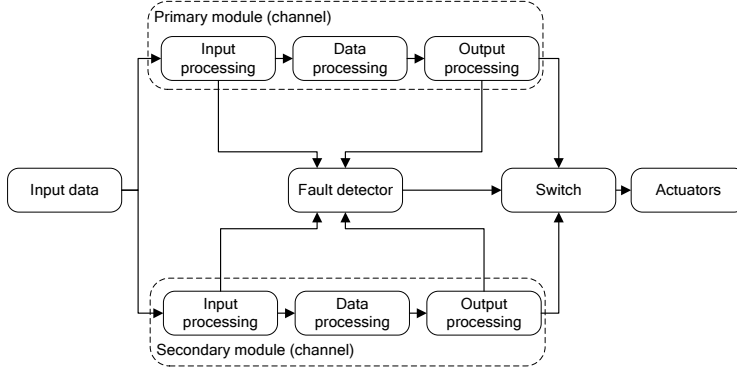


Figure 2.14: Homogeneous / Heterogeneous redundancy architecture pattern [8].

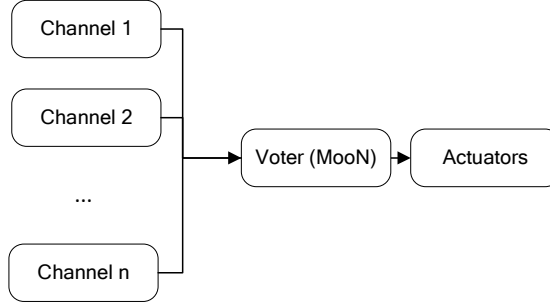


Figure 2.15: M-out-of-N architecture pattern.

in MooN patterns with homogeneous redundancy, the *availability* of the system increases, while the *reliability* is not impacted since it would require heterogeneous redundancy.

In Chapter 3 we use our proposed model to describe these and other architecture patterns, focussing on redundancy-related elements and their impact on the reliability of the system.

2.6 Resource sharing and fail-silent systems

As presented in Section 2.2.2, independence is a strict requirement for redundancy when applying the ASIL decomposition technique. However, CF and CCF are problems even when redundancy is not used. The hardware and software elements of the system can interfere with each other, and the failure of one of them could lead to the failure of others. For example, a function executed on a hardware resource could accidentally overwrite part of the memory space reserved for another

function, or it could lead to the failure of the hardware element itself. Another typical failure event is the *babbling idiot* [27], in which the element that fails continuously sends random data to its communication outputs, overloading the network and leading to the failure of the other connected parts of the system. When a failure of an element can interfere with other parts of the system, the element is not *fail-silent*. A fail-silent element recognizes that it is receiving the wrong information due to a fault, and either shuts down its operation or switches to a degraded mode to not propagate the fault further.

If multiple functions with different SGs are present, the failure of a non-fail-silent element can lead to the failure of an unrelated SG. Especially when the ASILs assigned to the SGs is different, a safety-critical SG can fail because of less critical elements, which follow a different and simpler process in terms of safety analysis. For this reason, interferences from other parts of the system must be avoided when developing safety-critical systems.

Since we focus on architectures and related design-choices, we discuss two techniques that avoid interferences due to the sharing of hardware elements by multiple functions. The simplest solution is to physically separate all the functions, removing all sorts of hardware sharing dependencies in the system. This includes not only sharing a resource by multiple software elements, but also resource-related dependencies, e.g. separate power supplies. We call this technique *physical separation*. To be independent, the different hardware elements are mapped to different physical locations and have a physical distance between them. With physical separation, the number of required hardware elements in advanced systems is high and it might not be feasible to install all of them in a single vehicle.

The second technique is *virtualization*. A middleware layer, such as a hypervisor, is used to safely share hardware elements and isolate different functions. In this way, fewer hardware elements are required in the system.

Hypervisors are classified into two types. Type 1 hypervisors are the most popular, they run directly on top of the hardware elements as shown in Figure 2.16a, they are also known as bare-metal hypervisors. Type 2 hypervisors run on top of an operating system, as shown in Figure 2.16b. The guest operative systems running in the Virtual Resources (VRs) created by the hypervisor do not have direct access to the hardware and access them via the virtualization mechanism. Further categorization is possible [167], but it is out of the scope of the current work.

Popular automotive hypervisors, such as QNX Hypervisor [25] and Green Hills INTEGRITY Multivisor [66], follow the ISO 26262 software code guidelines and are ASIL D certified.

When referred to communication elements, virtualization is used to separate different application domains and run multiple independent networks on the same physical network. This can provide individual Quality of Service (QoS) configurations in terms of bandwidth and latency, and isolation between the different virtual networks. For example, automotive Ethernet can use Virtual Local Area Networks (VLANs) [157]. VLAN-based isolation is performed on the data link layer, which

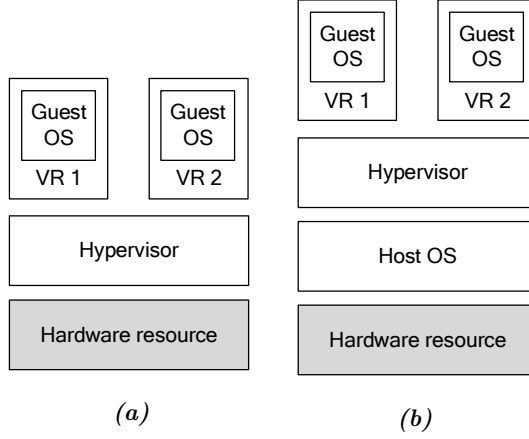


Figure 2.16: Type 1 (a) and type 2 (b) hypervisors.

may involve hardware elements, ensuring separation of the communication in the network. Other network virtualization mechanism are used, for example Virtual Controller Area Networks (VCANs) can be used to provide fault isolation, reserved bandwidth guarantees and small temporal interference [72]. Even wireless networks can be virtualized, for example automotive-oriented configuration of 5G network slices can be used for vehicle-to-everything communication [29].

In Chapter 5 we describe the effects of the two techniques on the automotive system in combination with redundancy. Interferences due to other dependencies, for example proximity to other failing hardware elements or sharing of a power supply, are not address in this work and will be considered in future work.

3

Model and functional-safety analysis of automotive systems

3.1 Introduction and related work

The introduction of Advanced Driving Assistance Systems (ADASs) and Autonomous Vehicles (AVs) will impact the Electrical and Electronic (E/E) architecture of vehicles. ADASs and AVs are complex cyber-physical systems, with many requirements related not only to performance, but also to safety, security, power efficiency, and cost. With our work, we model such a system to perform a quantitative evaluation that will guide the system architect to compare possible implementations. The system characteristics derive from multiple aspects of the vehicle related to the software applications, the hardware resources that execute the applications, the sensors that collect the environment data, the actuators that interact with the physical parts of the vehicle, and the communication infrastructure between them.

When describing these systems from a safety-oriented point of view, it is important to analyse the independence of system elements, as mentioned in the ISO 2626 standard, especially when focusing on redundant parts of the system. Only if independence is verified, Automotive Safety Integrity Level (ASIL) decomposition can be applied to the Functional-Safety Requirements (FSRs), hence the requirements on the separate elements can be lowered. The independence analysis involves many aspects of the vehicle. To perform it in an automated fashion, a model of the system must include a description of the relations between the application, the hardware elements, and physical characteristics of the environment

of the car. The standard recommends that each identified potential for dependent failure shall be evaluated in operational situations as well as in the different operating modes, and shall consider [81]:

1. hardware failures;
2. development faults, e.g. those related to item or element requirements, software design and implementation, hardware design and implementation;
3. manufacturing faults, e.g. those related to processes or procedures;
4. installation faults, e.g. those related to wiring routing, failures of adjacent items or elements;
5. repair faults, e.g. those related to processes or procedures;
6. environmental factors, e.g. temperature, vibration, pressure, pollution, Electro-Magnetic Interference (EMI);
7. failures of common external resources, e.g. power supply, input data;
8. stress due to specific situations, e.g. wear, ageing.

We analyse the aspects that are related to the implementation of the vehicle E/E architecture and the relationships between the applications (2 and 7), the hardware resources (1, 2, and 7), and the environment (6 and 8), while we do not focus on processes and procedures-related faults (3 to 5). In the literature, we identify various methods to describe an automotive system, some with safety-oriented, others with performance-oriented points of view.

For example, PTIDES [42] is a programming model that provides a framework to analyse time-synchronized distributed real-time embedded systems. In this model, the actors are event-driven and Worst-Case Execution Times are assigned to them. The Dataflow model of computation is also an event-based model, in which actors can be executed when all their inputs are available [94]. Compared to the PTIDES model, each actor has a fixed production and consumption rate, and it is possible to derive schedules for the actor's execution to analyse static performances of the system, such as maximum throughput [46]. PTIDES and Dataflow models are used for performance analysis of the applications. Our model uses directed (a)cyclic graphs as well, but instead of focusing on scheduling and performance, we perform a functional-safety analysis in which the connections between the nodes are used to identify the logical connections in the system and evaluate possible sources of faults for the applications.

Specific commercial tools are used to analyse the network performance, such as OMNeT++ [114]. This tool is a component-based C++ simulation library and framework, that allows real-time simulation, network emulation, etc. The faults can be simulated, but it is not possible to compute the failure probability of the system. The INET Framework [75] is a standard protocol model library of OMNeT++, which provides models and support for different communication protocols, from CAN to more recent Ethernet Time-Sensitive Networking (TSN).

In terms of functional safety analysis, commonly used models that describe automotive systems are Unified Modeling Language (UML) based models. UML is

a general-purpose modelling language that allows representing a system software architecture in diagram-based models. SysML [159] is an extension of the base UML model that adds hardware and system engineering concepts to it, allowing for performance analysis and requirements engineering [7, 20]. From these models additional safety analysis on the system can be performed, as in [93] where fault trees are synthesised from UML models for reliability analysis. Open-source tools such as Papyrus [48] and commercial ones such as IBM Rhapsody [74] fully support SysML and UML-based models. They provide the functionality to help accelerate the application of functional safety standards in industry by allowing tracking and documenting of Safety Goals (SGs) and FSRs, model-based simulation, and safety analysis. However, the models used in these tools can only be manually modified, compared to the automated transformations that we use in our framework with which we generate redundant elements and re-evaluate the new system for architecture exploration.

Many other commercial tools support functional safety analysis of automotive systems. For example, Vector PREEvision [1] can be used to implement the requirements of the ISO26262 standard, using a specific description language to describe the E/E architecture.

An example standardized automotive software architecture for safe automotive systems is the AUTomotive Open System Architecture (AUTOSAR) [10]. It is a development partnership created by major automotive industry Original Equipment Manufacturers (OEMs), suppliers, and tool and software vendors to combine these concepts into a platform fit for safety-critical automotive application development. The goal of the partnership is to standardize the software architectures of automotive Electronic Control Units (ECUs) and improve their performance, safety, and security. As shown in Figure 3.1, AUTOSAR uses a layered software architecture in which the application is separated from the hardware by a middleware layer, defined as Runtime Environment (RTE), that provides drivers and services. With the hardware abstraction provided by the Basic Software, the software components are modular and ECU-independent. The AUTOSAR framework provides an UML profile to describe AUTOSAR-based applications [11].

In our work, we decide to develop our lightweight system model inspired by the existing techniques and tools. Our goal is to analyse the reliability of the system. We want to evaluate systems that contain redundancy to enable the ISO26262 ASIL decomposition technique, and we want to calculate system-related parameters to compare different implementations. Moreover, we want to be able to modify the system by introducing redundancy in selected parts with automated system transformations, by applying the standard guidelines for independence (Chapter 4). With a custom model we can focus on these functional safety aspects, with the flexibility to introduce additional evaluation parameters or custom rules. Moreover, we can manipulate the model with custom transformation processes to introduce redundancy and re-evaluate the new system all in a single non-commercial framework. In comparison, other available options only provide

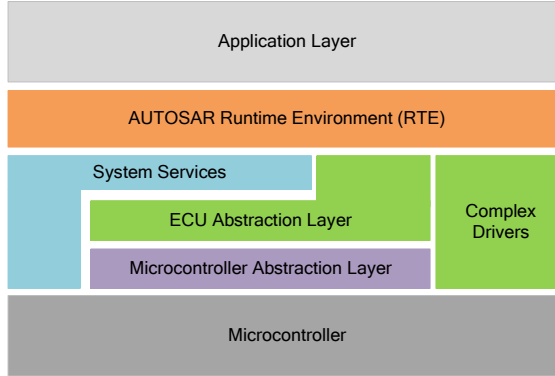


Figure 3.1: *AUTOSAR layered software architecture.*

evaluation of a described input system.

After modelling the system, we need to extract information that can be used in the design phase to optimize the design of an application, analyse its safety aspects and possible variations in the mapping of the application to the hardware. In literature, we find analyses of automotive systems performed from many points of view. For this work, we focus on the system-level safety-oriented analysis of an automotive system. System-level analysis of an automotive system includes many different aspects of the vehicle [97]. We calculate system-level parameters from the system model that help us compare different architecture choices and implementations, described in Section 3.4. In particular, we evaluate the failure probability, the cost, the total communication cable length, the functional load, and the communication load of a system.

In a *functional-safety analysis*, the system is tested and the FSRs are verified. In literature, the use of design patterns is suggested to simplify the functional safety analysis by using well-known methodologies, e.g. in [156] the authors suggest patterns such as the monitor-actuator pattern [8] to re-use available knowledge. In safety-critical systems, even well-known elements must be validated in the new context, but previous knowledge helps in reducing the effort for the new validation. Other methods to ensure the fulfilment of the FSRs use formal methods, e.g. [4, 22, 85, 102], or ensure that they are valid by design [156]. In our model, the applications are associated with FSRs and inherit their ASIL requirements. The model transformations that we use ensure that we satisfy the FSRs at design time.

Fault trees are a widely used instrument in functional safety analysis [92, 130] that logically connects the fault events to the different parts of the system. For example, in [40, 61] fault tree analysis is used to ensure that the failure probability of the system does not exceed the limits imposed by the ASIL requirements. Fault trees can also be used to derive the FSR, as in [141]. In our quantitative analysis, we generate the fault trees from the model and perform a *static fault tree analysis*

to calculate the *failure probability* of applications and compare non-redundant with redundant systems.

Many commercial and open source tools support this analysis [84], e.g. isograph FaulTree+ [82], HiP-HOPS [117], XFTA [3], or Storm Checker [71]. In our work, we use SCRAM [122] to calculate the failure probability of the system in a single-point in time, as described in Section 3.4.2.

In terms of *cost calculation* of a system, in literature, we find the allocation of the applications to the resources with safety constraints such as in [73, 161]. In our work instead, we calculate a bill of material type of cost on the resource layer, associated to the ASIL values of each resource, similarly to [14]. We analyse the cost of redundancy based on different cost metrics and model transformations that are used.

The cumulative length of electrical wires in modern cars is more than 4km [12]. The wiring harness is one of the three heaviest subsystems in many vehicles [123], weighting more than 70kg in a high-end car, comparable to an additional passenger. This factor impacts the vehicle fuel consumption or the usage of the battery charge. It is important to make design decisions that will minimize the total cable length, as in [96] with a combination of routing and cable size optimizations. However, redundancy goes in the opposite direction: more resources bring more communication cables that are used to connect the system, and more complex networks must be used. In our work, we will use the total communication cable length as one of the metrics to compare different solutions, as well as the total *functional and communication load* of an application.

Compared to the related work, in our framework, we can calculate the various parameters with a single model. We can modify the parameters metrics to better describe any automotive system, e.g. our cost metric is an input of the analysis framework that can be tailored to a specific scenario. Moreover, we can modify the system with automated model transformations, as we describe in Chapter 4, and re-evaluate it to compare different redundancy levels.

In the next section, we describe our three-layer model, that we use to describe automotive systems. In Section 3.3 we discuss how we model redundancy and how it can be implemented. In Section 3.4 we describe our analysis framework that calculates quantitative parameters for the modelled automotive system, and we conclude the chapter in Section 3.5. The proposed model has been presented in our publications [55–58].

3.2 Three-layer model

For this work, we chose to create a new system model that contains all the information necessary for our quantitative analysis. We introduce redundancy in the modelled system by following the ISO26262 ASIL decomposition rules. With the model and the tools that we developed, we modify and track the system characteristics, both concerning logical connections of the applications and hard-

ware, and for functional safety analysis. While other models exist that permit ASIL-oriented functional-safety analysis, our proposal is a lightweight model that allows us to focus on specific parts of the analysis and modify the system with the introduction of automated transformations that introduce redundancy following the ASIL decomposition requirements. In this section, we describe the layered structure of the model.

To describe the automotive system, in our model we separate it into three layers: application, hardware resources, and physical space. This allows us to separate the different requirements and specifications into multiple layers depending on their characteristics. Each layer contains information that is used in the analysis for the system reliability and cost parameters, which is described in Chapter 4. Redundancy is modelled with explicit elements, explained in detail in Section 3.3, and graph transformations are used to introduce redundancy in selected parts of the system, as explained in Chapter 4.

As an example system to highlight the capabilities of the proposed model and tools we use an Adaptive Cruise Control (ACC) System [168]: a cruising speed is selected by the driver and the speed of the vehicle is adjusted based on the desired speed and the distance from the preceding vehicles, which are detected via a radar system. The functional partitioning of this ADAS system is shown in Figure 3.2.

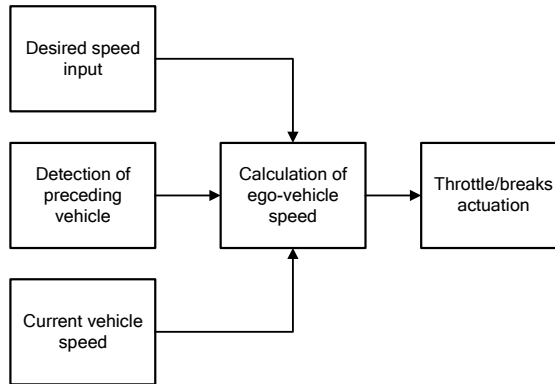


Figure 3.2: Functional partitioning of an ACC ADAS.

During the building of the Safety Case, as explained in Chapter 2, the results of the Hazard Assessment and Risk Analysis (HARA) are used to identify the Safety Goals that cover the possible hazardous events. One of the possible Safety Goals is the following:

- SG1. *The vehicle must maintain enough distance from the preceding vehicle to be able to execute an emergency brake at any cruising speed without a collision.*

The Safety Goal SG1 is then translated into multiple FSRs [81], for example:

FSR1. *The system should not transfer excess power to the wheels where it causes unintended acceleration.*

FSR2. *The system should not reduce the supplied power to the wheels where it causes unintended deceleration.*

In this scenario, *unintended acceleration* refers to violating the distance rule from the preceding vehicle. *Unintended deceleration* refers instead to not maintaining the desired vehicle speed when the system does not violate the distance rule. During the Failure Mode and Effect Analysis (FMEA) stage, the risk parameters (Exposure, Severity, Controllability) [81] are assigned to each of the FSRs, as shown in Table 3.1.

Table 3.1: Risk parameters assigned during the FMEA.

	Exposure	Severity	Controllability	ASIL
FSR1	E4	S3	C3	D
FSR2	E4	S3	C3	D

The two FSRs have the most critical levels for each of the risk parameters, resulting in the highest ASIL (ASIL D), as shown in Table 2.2. The FSRs are then refined into the Technical Safety Requirements (TSRs) [81], which inherit their ASIL requirements.

In the following sections, we describe the three layers of the model and show each layer for the ACC example.

3.2.1 Application layer

The application layer contains multiple fully connected directed graphs: each fully connected graph $G_a = (V_a, E_a)$ represents one application and contains a set of application nodes V_a and a set of edges E_a connecting them. In our description, the edges connecting the application nodes are only logical connections. They represent the data transfers between the nodes and do not contain information related to the communication between the source and the target nodes: this information is contained in explicit communication nodes. Cycles can exist in the application graph. They are often required e.g. by feedback loops in control applications.

When describing ADAS or AV related applications, the *sense-think-act* paradigm is generally used [148]. It is a common concept used originally in robotics, which separates an application into three main parts:

- a) *Sense*: an application always begins by collecting information about the surrounding environment or the vehicle status from one or more sensors.
- b) *Think*: the collected data is then processed. Different design approaches can be used to determine if it happens, for example, in a centralized architecture,

where a single module analyses the data, or in a distributed fashion, in which multiple modules analyse the different sensor data.

- c) *Act*: the final part of an application involves the actuators, which modify the state of the vehicle.

The application layer follows this paradigm. Sensor nodes do not have inputs (other than virtual splitters, Section 3.3). Actuator nodes do not have outputs (other than virtual mergers, Section 3.3).

One or more FSRs are linked to each graph G_a : the nodes V_a of a graph inherit the highest ASIL requirement between the related FSRs. In the ACC example, the FSR1 and FSR2 can be assigned to the same set of nodes. The FSRs are related in both cases to the detection of the preceding vehicle, the sensing of the current vehicle speed, the calculation of the desired vehicle speed, and the throttle and brakes actuation.

The application layer for the ACC example described in Section 3.2 is shown in Figure 3.3. In the picture the sensing nodes are yellow: the radars that provide information about the vehicles in front of the car, the in-vehicle sensors that provide current speed and acceleration information, and the Human-Machine Interface, in which the user selects the desired speed. The radar sensor is used to detect the objects in front of the vehicle, while the *Cruise Control* node performs data fusion between the objects detected by the radar and the in-vehicle sensors to obtain the signals required by the brakes and throttle actuators. All the nodes present in the graph inherit the ASIL D value from FSR1 and FSR2.

Since the goal of the model is to describe systems that include redundant elements for higher reliability, we now show an example of this: we assume that the detection of the objects in front of the vehicle in the ACC system is redundant. Two different radars with different characteristics (e.g. with different ranges) are used to detect preceding vehicles, and only one of the two input data is used at the time. Figure 3.4 shows how we model the application graph of such a (simplified) system.

In the redundant system, the radar sensors data is preprocessed and the preceding objects are identified (nodes *Object detect 1* and *2*). Note that the nodes *Object detect 1* and *Object detect 2* can have diverse implementations and are not necessarily duplicates. The *merger*, a redundancy-specific node, selects which output to forward to the central *Cruise Control* node. Here, the required speed is calculated and the correct actuator signals are generated.

We can apply the ASIL decomposition technique described in Chapter 2. The two paths formed by the radar, its data, the object detection step, and the detected objects, are redundant. We can apply the ASIL $D \rightarrow \text{ASIL A}(D) + \text{ASIL C}(D)$ decomposition of Figure 2.7 and lower the ASIL requirements on the two paths. Note: the system-level ASIL requirement (D) does not change with ASIL decomposition. The management of the redundancy performed by the safety-oriented splitter and merger nodes must be performed at the original ASIL requirement, so they maintain the ASIL D requirement [95].

Each node in the application layer has a list of properties. The properties are the following:

- *Node type*: a node can be a *sensor*, *actuator*, *functional*, *communication*, *splitter*, or *merger node*. This property allows to analyse the mapping of the node and ensuring that the used resource can fulfil the node's purpose. Splitter and merger nodes are described in more detail in Section 3.3.
- *ASIL requirement*: inherited from the original FSR, each node has an ASIL requirement. The possible values are QM, A, B, C, and D as seen in Table 2.2. Our *failure rate* metrics, that we describe in Section 4.5, are based on the ASIL requirement of the nodes.
- *ASIL of the SG*: the ASIL value of the original SG, to track the system-level requirement when applying ASIL decomposition.
- *Functional or communication load*: we assign a parameter (a positive integer number) to the functional or communication nodes that states their resource usage. In our analysis, the functional load represents the memory usage of the resource, while the communication load the bandwidth requirement of the node. In a different analysis, these values can represent other parameters such as the application execution time, the utilization of a specific resource, the periodicity of the communication signals, the latency of the communication, etc. The values that are assigned are tailored to the specific analysis

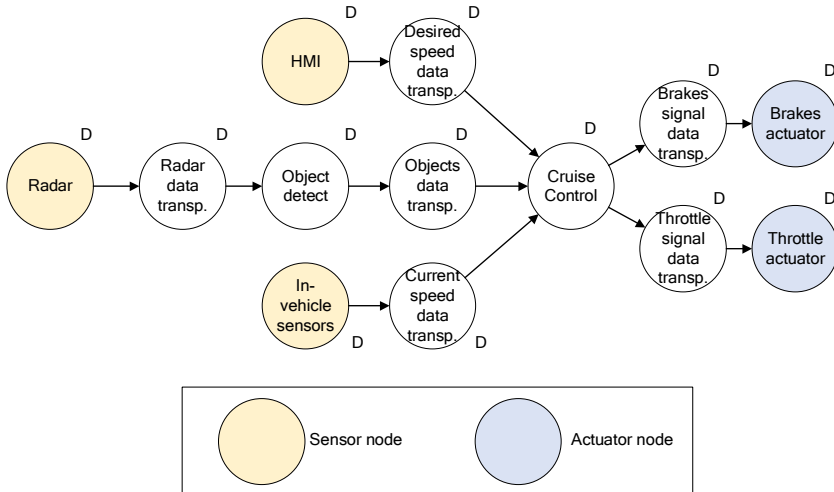


Figure 3.3: A simplified ACC application graph. All the nodes have ASIL D requirements.

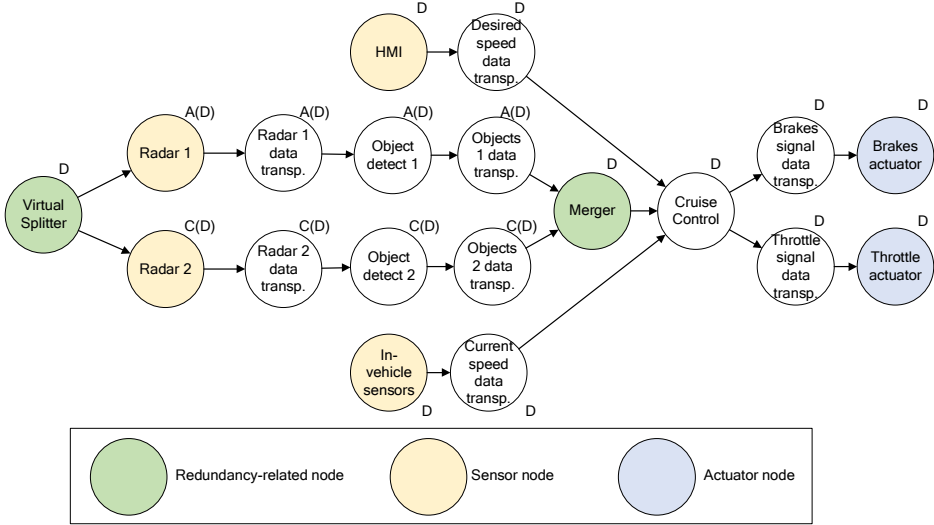


Figure 3.4: A simplified redundant ACC application graph. Part of the ASIL requirements are decomposed into A and C with the ASIL decomposition technique.

that is performed and can be either relative weights between the different application nodes, or can be absolute numbers obtained for example from timing analysis.

- *Failure rate*: a failure rate λ_{a-x} , expressed in failures per hour, is a positive number assigned to an application node x . In Chapter 2 we have identified the different failure modes for hardware elements: similarly, the failure of an application node can happen due to a Single-Point of Failure fault, a Multi-Point of Failure fault, or a Residual fault as shown in Figure 3.5. The value of the parameter is calculated as the sum of each of them, according to Equation 3.1 [81]. While not explicitly present in the model, using an application-level safety mechanism changes the values of the three failure rates, modifying the overall λ_{a-x} for application node x .

$$\lambda_{a-x} = \lambda_{aSPF-x} + \lambda_{aRF-x} + \lambda_{aMPF-x} \quad (3.1)$$

We define for later usage the *successor* and the *predecessor* of an application node:

- A node y is the *successor* of node x if $\exists e_a \in E_a$ so that $e_a = (x, y)$.
- A node y is the *predecessor* of node x if $\exists e_a \in E_a$ s.t. $e_a = (y, x)$.

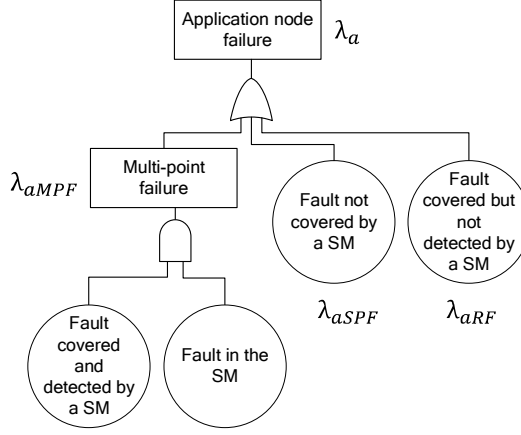


Figure 3.5: Faults leading to the failure of an application nodes related to the Safety Mechanisms.

3.2.2 Resource layer

The resource layer is a directed graph $G_r = (V_r, E_r, E_{dep})$. V_r is the set of physical and Virtual Resources (VRs). E_r is the set of edges that represents the logical directed connections between the resources. Physical wires, communication cables, wireless connections, and power lines are also part of the V_r set. In the case of bidirectional communication resources, one edge per direction is present in the graph. A resource can either be a physical resource or a virtual resource, as we describe later. E_{dep} is the set of resource-resource dependencies, which we use to model power dependencies and virtualization dependencies. Each dependency is captured with an edge $e_{dep} \in E_{dep} : V_r \rightarrow V_r$, and the resource-resource dependencies are *N-to-1*, since multiple resources can use the same power supply, or multiple VRs can be virtualized on the same physical resource. While multiple application graphs G_a can exist, there is only one resource graph G_r that contains all the physical resources and VRs used in the system on which the applications run. The graph G_r may be not fully connected.

Figure 3.6 shows the resource layer for the example of the ACC system in its redundant version. In this scenario, a single ECU executes both the object detection and the data fusion parts of the application. A hypervisor is used to obtain isolation between the redundant parts by providing VRs on the physical ECU resource. In our model, application nodes can be mapped either to the VRs or to the original physical resource. However, depending on the type of virtualization mechanism that is used, it may only be possible to run applications in the VRs.

Note that in our model, the resource ECU is not connected to the commu-

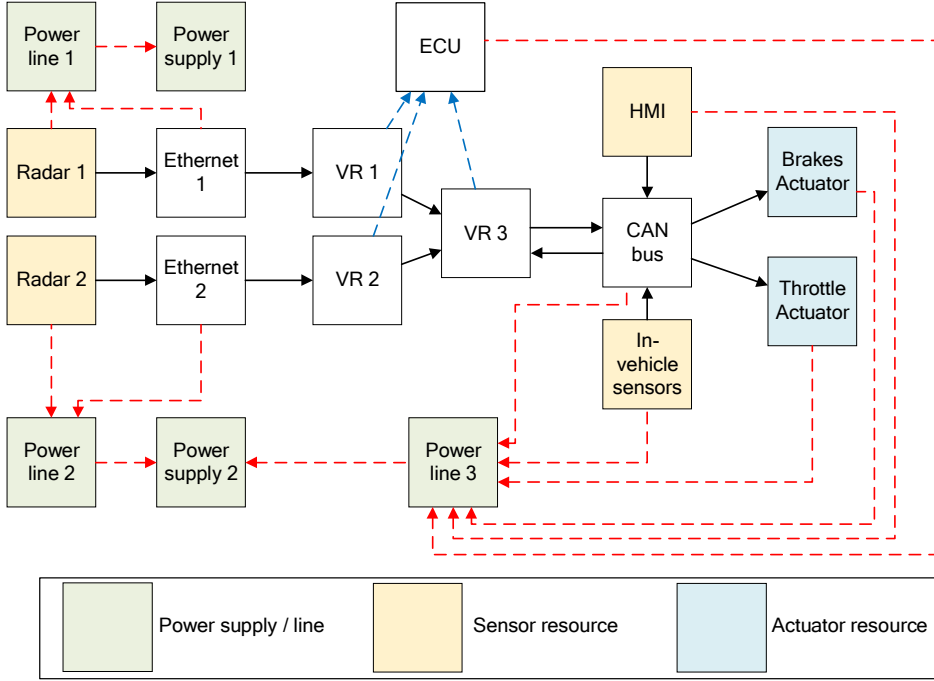


Figure 3.6: A simplified redundant ACC resource graph. The resource-resource dependency between VRs and the original resource is shown with the dashed blue arrows. The power network resource-resource dependency is shown with the dashed red arrows.

nication resources directly, but through the VRs *VR1*, *VR2*, and *VR3*. The communication ports of the physical resource are not modelled explicitly in the proposed model, but their number can be inferred by the connections of the VRs. In this way, we identify which VR has access to each communication resource.

VRs and the power system create resource-resource dependencies: each VR has a link to the physical resource that is running the virtualization mechanism, while each resource is connected to the power network to function. While the virtualization resource-resource dependency is a one-step connection, the power network is a multi-step, acyclic connection between a resource, the power lines, and the power supply. We model these two types of resource-resource dependencies, but any other shared resource can be modelled in the same way and is included in the later fault tree generation and Common-Cause Fault (CCF) analysis.

A hardware resource can be of one or multiple *types*:

- *Functional*: a resource that can process functional application nodes, like a processor or a controller;

- *Communication*: a resource that represents the different types of automotive cabled communication (LIN, CAN, FlexRay, MOST, Ethernet) or wireless connections. In case of multi-type resources, the communication type means that the resource has communication capabilities;
- *Sensor*: a resource that collects data from the physical environment, like a camera or a wireless receiver;
- *Actuator*: a resource that interacts with the physical environment by executing the desired operations, for example, the braking actuator;
- *Splitter*: a resource that can provide the necessary functionality for a split operation, described in detail in Section 3.3;
- *Merger*: a resource that can provide the necessary functionality for a merge operation, described in detail in Section 3.3;
- *Power Source*: a resource that provides the power supply for other resources, for example, a battery;
- *Power Line*: a resource that distributes the power supply to other resources.

Other than the resource types, the following properties are assigned to each resource:

- *ASIL specification*: the maximum ASIL that the resource can provide. This parameter is often referred to in the industry as an ASIL-X ready resource. Our *cost* and *failure rate* metrics, that we describe in Section 3.4.4 and Section 4.5 respectively, are based on the ASIL of the resources.
- *Maximum load*: the maximum functional or communication load that the resource can handle. The maximum load is a positive integer number. This parameter represents the maximum utilization of the resource and limits the number of nodes that can be implemented on a single resource.
- *Cost*: a cost parameter is attributed to each resource. The cost is a positive integer number. We consider this parameter as a bill of material cost. However, depending on the analysis that is performed, the cost can be related to other parameters, such as the development or manufacturing cost. It can be a relative value representing the proportions between the different resources or the absolute numbers obtained by an initial study. In Section 3.4.4 we present the cost metric that we used in this work. In the proposed framework, it is also possible to assign a separate cost value to each resource without following a specific metric.
- *Failure rate*: a failure rate λ_{r-y} , expressed in failures per hour, is a positive number assigned to a resource y . The failure rate of a resource is related

to the failure modes described in Chapter 2. Similarly to the λ_{a-x} , λ_{r-y} represents the possibility of system failure due to the occurrence of a Single-Point Fault, a Multi-Point Fault, or a Residual Fault. This value is used in Section 3.4.1 to calculate the failure probability of the system. We use a failure rate metric based on the ASIL specifications of the resources, as described in Section 4.5. Alternatively, each resource can be assigned a different failure rate.

- *Virtualized*: a boolean parameter is assigned to each resource to identify the utilization of a virtualization mechanism. Each VR is connected with a resource-resource dependency edge to the physical resource on which they are virtualized (N-to-1 connection, where N is the number of VRs virtualized on a resource). A VR does not have a power supply resource-resource dependency, but the physical resource that virtualizes it has one. In Figure 3.6 the resources VR1–3 are VRs created in the resource ECU by a hypervisor. Each virtual resource has all the properties of other resources (ASIL specification, cost, failure rate, etc.). In particular, when using VRs, the additional failure possibility of each VR present in the generated fault tree contributes to the failure probability of the system, while without virtualization only the failure probability of the physical resource is present.

While virtualization generally increases the cost and the failure probability of the system, it provides an isolated environment for the application nodes. With this mechanism, the application nodes that fail in a VR are isolated from nodes mapped to other VRs on the same physical resource. Virtualization forces a fail-silent behaviour between VRs.

As we did for the application nodes, we define the *successor* and *predecessor* of a resource:

- A resource b is the *successor* of resource a if $\exists e_r \in E_r$ s.t. $e_r = (a, b)$.
- A resource b is the *predecessor* of resource a if $\exists e_r \in E_r$ s.t. $e_r = (b, a)$.

We define *purely-communication resources* as resources which have only the *communication* type.

3.2.3 Physical layer

The physical layer is formed by a graph $G_p = V_p$, with no set of edges. We assume that every other location is reachable from any point. A location represents the physical point in the vehicle in which the hardware resources can be placed. In our abstraction, the physical volume in which the hardware resources can be placed is reduced to a single point in two dimensions. Two properties are assigned to the physical locations:

- *2D coordinates*: normalized between -1.00 and 1.00 , that identify the physical location as a relative position in the car.

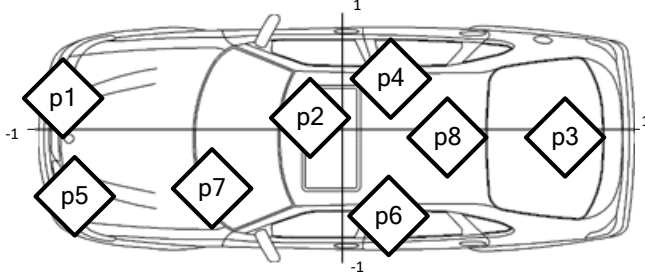


Figure 3.7: *A simplified physical layer.*

- *Failure rate:* a failure rate λ_{p-z} , expressed in failures per hour, is assigned to a location z . A failure rate is a positive number. This parameter is related to the environmental factors that are mentioned in the dependent fault analysis of the ISO26262 standard (e.g. temperature, vibrations, electromagnetic interferences). We categorize these faults as external faults, which may or may not be covered by a safety mechanism. λ_{p-z} represents the probability of a failure due to environmental conditions, that would lead to the failure of all the resources mapped to that location.

Figure 3.7 shows the example locations in which we can map the hardware resources of our simplified ACC application.

3.2.4 Mapping: inter-layer connections

The mapping functions $M_{ar} : V_a \rightarrow V_r$ and $M_{rp} : V_r \rightarrow V_p$ complete the model.

Each non-communication application node is mapped to only one resource. A communication node can be mapped to multiple communication resources, for example to cables and switch resources. Virtual splitters and virtual mergers, described in Section 3.3, are an exception, as they are application nodes that are not mapped to any resource: they represent elements external to the electronic system (e.g. redundant camera collecting data of the same part of the environment). Multiple application nodes can be mapped to the same resource, and, as discussed in Section 2.6, this can lead to interferences and high failure probability when the application nodes are not fail-silent.

Figure 3.8 shows the mapping of the application nodes to the resources in the ACC example.

Non-purely-communication resources are mapped to only one location, while purely communication resources, such as communication cables, can be mapped to multiple locations.

In our model, simultaneous mapping of different application nodes to the VRs and to the physical resource that runs the virtualization mechanism is allowed, as shown in Figure 3.9a. In practice, this possibility is limited by the type of

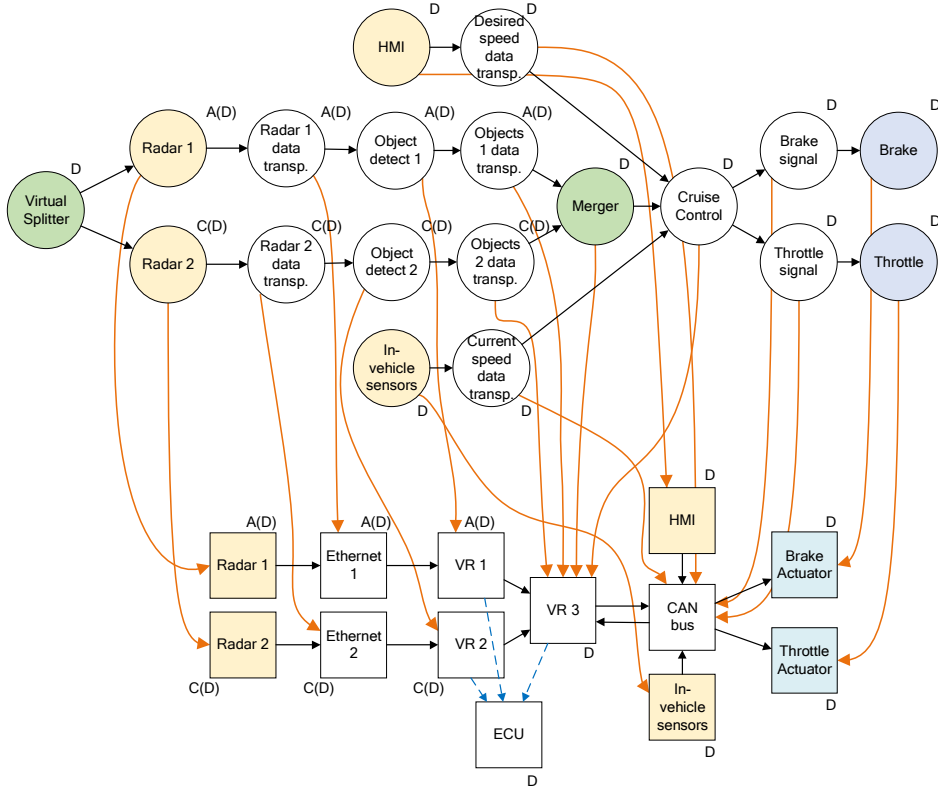


Figure 3.8: The application layer mapping to the resource layer in the ACC example.

virtualization mechanism that is used, which can either allow or not the use of non-virtualized parts of the resource.

When multiple nodes are mapped to the same VR, as shown in Figure 3.9b, they share it in the same way as they would share a physical resource: in the case they do not have a fail-silent behaviour, they will interfere with each other.

3.2.5 Analysable input graphs

To perform the functional safety analysis of the later sections and the model transformations of Chapter 4, the input model must follow these rules:

Each application node is reached by at least one sensor and reaches at least one actuator.

1. Applications must follow the sense-think-act paradigm, meaning that sensors and actuators do not have inputs or outputs respectively (other than virtual

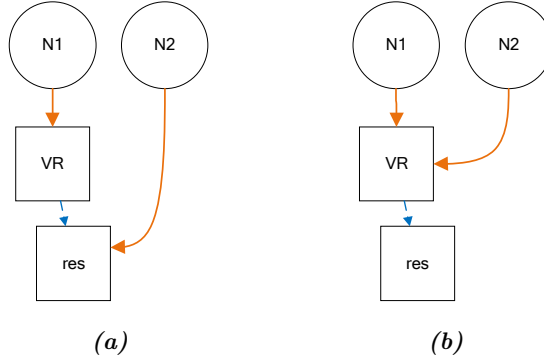


Figure 3.9: Application nodes mapped to a VR and the corresponding physical resource (a) and multiple application nodes mapped to a VR (b).

splitters or virtual mergers). Each application requires at least one sensor and one actuator. Accordingly, each resource graph has at least one sensor and one actuator resources.

2. Each functional, communication, splitter, and merger application node is reached by at least one sensor node and reaches at least one actuator node.
3. Predecessors and successors of functional, splitter, and merger nodes are communication nodes.
4. Predecessors and successors of functional, splitter, and merger resources are purely communication resources.
5. All the properties of the application, resource, and physical layers are defined between their boundaries.
6. The ASIL requirements of application nodes do not exceed the values of the ASIL specification of the resource the nodes are mapped to.
7. The sum of the functional and communication loads of the nodes mapped on a resource does not exceed its maximum utilization.
8. Each communication node is mapped to one or more consecutive resources. Consecutive resources are series of successive resources.
9. Each non-communication application node is mapped to one resource.
10. Virtual splitter and virtual merger application nodes are not mapped to any resource.
11. Predecessors (successors) of a node are mapped to a predecessor (successor) of the resource the node is mapped to, or to the same resource.

12. Each pure communication resource is mapped to one or more physical locations, while other resources are mapped to one location.
13. Each purely communication resource is mapped to one or more physical locations.
14. Each non-purely communication resource is mapped to one physical location.
15. VRs are mapped to the same physical location as the physical resource that runs the virtualization mechanism.
16. VRs have a resource-resource dependency with the physical resource, and each physical resource has a resource-resource dependency with at least one power line that leads to a power supply.

Before performing the analysis, our tools test these rules on the input graphs. In Appendix B we show how we implemented the input files to be read by our framework. The tools are implemented in python and use the graph-tool [119] library.

3.3 The splitter and the merger concepts

A variety of safety mechanisms are used to improve the reliability of safety-critical systems. However, we focus on redundancy to obtain fail-operational automotive systems. To describe redundancy in the model, we introduce the *splitter* and *merger* concepts, which are the explicit elements that identify the redundant parts of the system.

While it is possible to use temporal redundancy to improve reliability, e.g. by saving a copy of the initial data and executing a processing function twice on the same resource, we focus on spatial redundancy, in which separate functions are executed in parallel on independent resources. In our model, we consider temporal redundancy as a generic safety mechanism, that will influence the values of the failure rate λ_{a-x} since it is an application-related technique rather than an architecture solution. In all spatial redundancy patterns, we can identify two specific steps:

1. The initial data is sent along multiple redundant channels;
2. The outputs of the redundant channels are used to compute a single output value (e.g. by selecting only one of them, or by combining their values).

We define two application nodes that provide this functionality as *splitter* and *merger* nodes:

- A *splitter* node replicates its input data to its output ports.

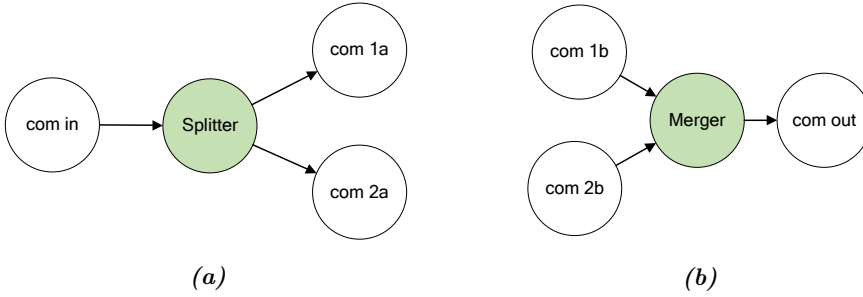


Figure 3.10: A splitter node (a) and a merger node (b) in an application graph.

- A *merger* node ensures that only correct data out of its redundant inputs is forwarded once.

We also define *virtual* splitters and mergers application nodes. In case multiple sensors collect the same information and transmit the data on parallel redundant paths and then only one of the outputs is used in later stages, we identify the split operation in the selection of the different input types. We model the application graph with a virtual splitter with the correct ASIL requirement, which allows us to identify the pattern *splitter - redundant paths - merger*, even if there is no explicit software or hardware component performing the split operation. This happens for example in the ACC system of Figure 3.4, in which the inputs of the two redundant paths are obtained by two different radar systems. Similarly, we can insert a virtual merger in case multiple actuators are providing the same function, e.g. an independent braking system [26].

Being able to identify the redundancy pattern in a graph is important to separate redundant parallel paths from paths that are just working in parallel without redundancy. For example, in Figure 3.4 the HMI and the in-vehicle sensors are in parallel to the object detection part of the application but are not redundant. Without the explicit splitter and merger nodes, it is not possible to understand only from the graph if the application has functional parallelism or redundancy.

The split and merge operations can be implemented on many different levels, e.g. in the OSI model they can be implemented from the application layer to the physical layer. They can depend on application-specific information or can be independent of the application. They can use the input data or additional information to perform decisions. In the following sections, we provide some examples of splitter and merger implementations in common architecture redundancy patterns, each of which can be modelled in our framework.

3.3.1 Splitter implementations

A splitter is an element that replicates its input data to all its output ports, which are connected to the redundant channels. It can be implemented in multiple ways, e.g.:

1. *Broadcast/Multicast message*: the redundant channels are connected to the same network component that has broadcasting or multicasting capability, e.g. broadcast message in a CAN bus in Figure 3.11.
2. *Individual connections*: the input node is mapped on a resource that is connected to the redundant channels via separate output ports and separate network components. The message is sent on each of the output ports in sequence or parallel, depending on the input generation node implementation. An example is shown in Figure 3.12.
3. *Subsequent transmissions in a network*: the input node sends the same data to the redundant channels in separate transmissions, using the same network, e.g. a switched Ethernet network as in Figure 3.13.

For example, in the M-out-of-N (MooN) pattern shown in Figure 3.14, which is an example of homogeneous redundancy, the splitter functionality is identified in the part of Figure 3.14a highlighted by the dashed ellipse. The splitter is used to send the same data to the three systems. In this example, the merger corresponds to a *voter*.

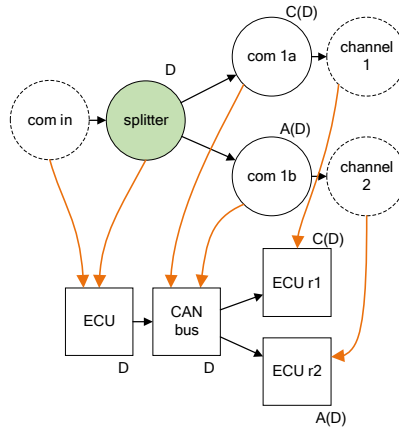


Figure 3.11: The splitter sends the messages to the redundant channels via a broadcast message. Note that the CAN bus is a possible source of a CCF, thus its ASIL D specification.

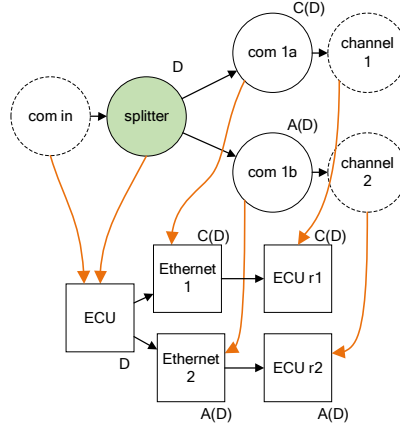


Figure 3.12: The splitter sends the messages to the redundant channels by using separate communication resources.

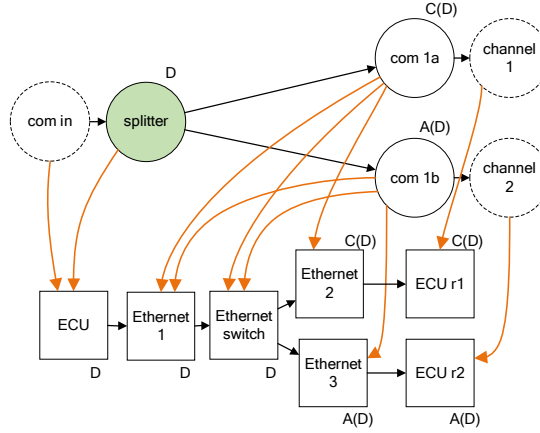


Figure 3.13: The splitter sends the messages to the redundant channels in an Ethernet switched network. Note that the resources Ethernet 1 and Ethernet switch are possible sources of CCFs, thus their ASIL D specifications.

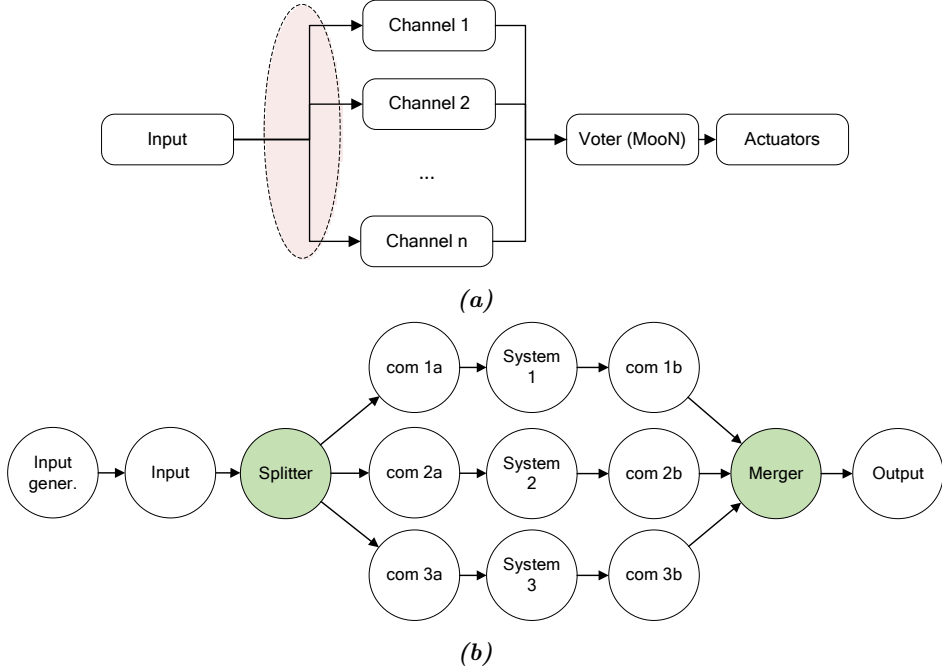


Figure 3.14: Identification of the splitter functionality in the MooN architecture pattern (a) and the application layer in the modelled version of a MooN pattern (b).

Figure 3.14b shows the application layer of a modelled *MooN* architecture pattern.

3.3.2 Merger implementations

While the splitter functionality is simple, even if it can be implemented in a variety of ways, the merger functionality is not straightforward. The decision that the merger must take about which output data to generate is based on multiple aspects, which can depend for example on the application, input types, synchronization, or external information. Its implementation can be either in software or hardware. We identify five types of mergers based on the data that they use to make the decision:

1. *Data oriented*: the merger directly compares the input values.
2. *In-band oriented*: the merger uses additional information contained in the transmitted packets to generate its outputs.

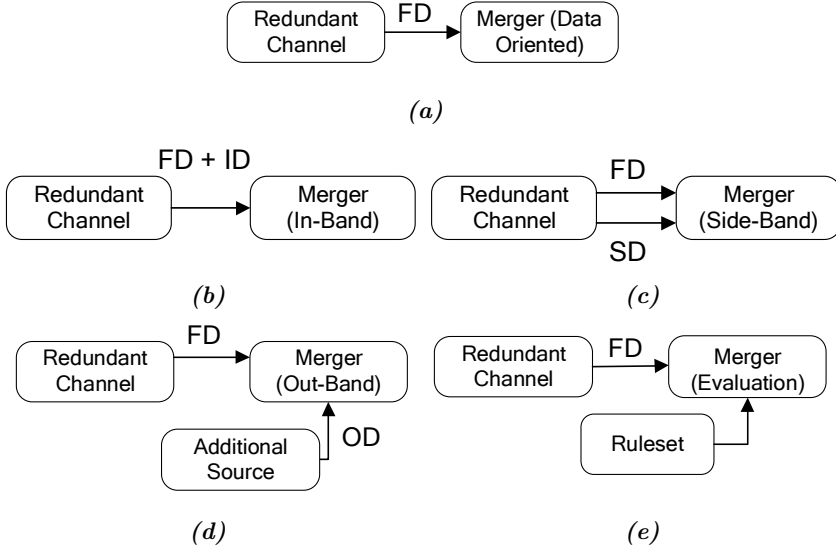


Figure 3.15: Merger types and data that is transmitted by each redundant channel. (a) Functional Data (FD), (b) In-band Data (ID), (c) Side-band Data (SD), (d) Out-band Data (OD), and (e) Data Evaluation (EV).

3. *Side-band oriented*: the merger uses additional information sent separately from the transmitted data to decide how to generate its output.
4. *Out-band oriented*: additional external information is sent to the merger to generate its output.
5. *Data evaluation*: the merger analyses the transmitted data and decides if it is valid.

Figure 3.15 summarizes the possible merger implementation based on the data they use.

The voter of the *MooN* architecture pattern falls into the first category, as it compares the values of the input data to vote which should be forwarded as correct. In a more general implementation, the (redundant) data may arrive at different times, and even out of order, or not arrive at all. The merger functionality must then include storing, resynchronization mechanisms, reordering, etc. before it can compare the data.

In a different scenario, a merger can use in-band or side-band information to make a decision. With the term *in-band* we refer to data that is contained in the transmitted packets, e.g. Error Correction Codes (ECCs). With *side-band* we refer to additional information that can be related to the received data, e.g. the

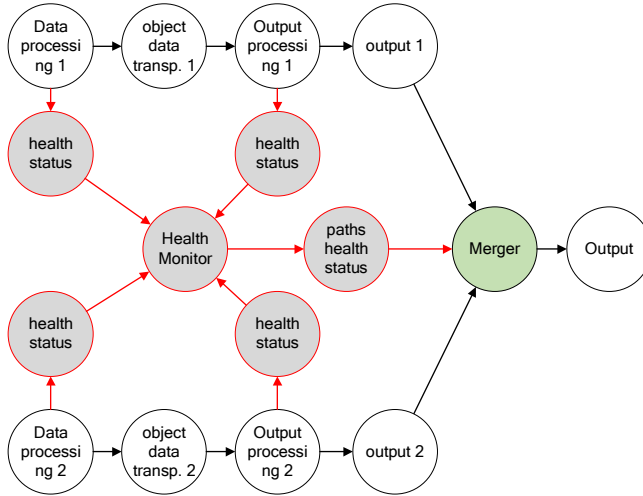


Figure 3.16: Example of Out-band oriented checks: a health monitor node sends information to the merger to decide which output to forward.

number of objects that should correspond to the number of objects present in the actual data.

Out-band data is external information that is usually sent by a different node to the merger. For example, in Figure 3.16 the Health Monitor node collects status information from the redundant paths and sends additional information to the merger. In this case, the *paths health status* node is only used as additional information to make the correct decision, and not as an input to be forwarded. Note that the Health Monitor node is in common between the two redundant paths, and could violate the independence requirements for the safety analysis. As a single point of failure, it inherits the original FSR ASIL value in the same way that splitters and mergers do.

Data evaluation type instead evaluates the output data of each channel by using additional inputs. In contrast to data-oriented checks, in this case, the redundant channels are not compared with each other, but evaluated according to a specific set of rules, e.g. the Intel Responsibility Sensitive Safety ruleset [59].

3.3.3 Splitter and merger in the safety-executive architecture pattern

In Chapter 2 we have discussed different safety-oriented architecture patterns, in particular the protected single channel, the safety executive, and the homogeneous/heterogeneous redundancy patterns. The splitter and merger functionality are necessary only in the presence of redundancy, which means that they are absent

in the protected single-channel pattern. In the previous sections, we described the ACC system as an example of heterogeneous redundancy, in which a virtual splitter is used. In this section, we identify the splitter and merger in the other safety-oriented architecture pattern, the *safety executive*.

In the previous section we have already encountered a *safety executive* pattern when describing the out-band oriented merger, in Figure 3.16. The health monitor sends to the merger the information it requires to select the correct data.

Figure 3.17a shows a general *safety executive* architecture pattern and the highlighted parts correspond to the splitter and merger functionalities. Two paths, the nominal and the safety channels perform redundant data processing to obtain an output signal for the actuators. A safety coordinator monitors the channels and data integrity to establish which outputs should be used in the actuators. The splitter node is part of the input source block. It can either be a virtual splitter, as in the previous example of the ACC system, or a real splitter in which the same input source sends the data to the different channels in one of the three previously mentioned ways.

The merge operation is performed by the safety coordinator and the actuator block: based on the health status of the nodes in the nominal and the safety channels, the safety coordinator determines which output is forwarded to the actuator node.

The implementation of the *safety executive* architecture pattern can vary. For example, we assume that the health status that is collected by the safety coordinator in each processing step of the two channels. The merger functionality is a multiplexer one, controlled by the *Arbitration message* that the *Safety Executive* node sends. In Figure 3.17b we show the modelled application layer for this scenario.

The splitter and merger operations can be found also in other types of redundancy, for example in communication protocols. In the IEEE 802.1CB Frame Replication and Elimination for Reliability (FRER) in the TSN stack [76], we identify the splitter operation when the Ethernet frame is sent on multiple communication paths. The merger operation is performed at the receiver, where the first of the two frames that arrives is selected and the other will be discarded on arrival.

3.3.4 Splitter and merger and virtual resources

In Section 2.6 we discussed how virtualization helps when sharing a resource to isolate nodes that are not fail-silent. However, it can also be used for the isolation of redundant parts of an application. If only one non-virtualized resource with a higher ASIL level is used, as shown in Figure 3.18a, when the nodes are not fail-silent they will interfere with each other, and the resulting failure probability of the system may be higher than the required one. If we instead use VRs, as shown in Figure 3.18b, the two nodes do not interfere with each other despite being both executed by the same resource *ECU*, and the resulting ASIL level of the system is

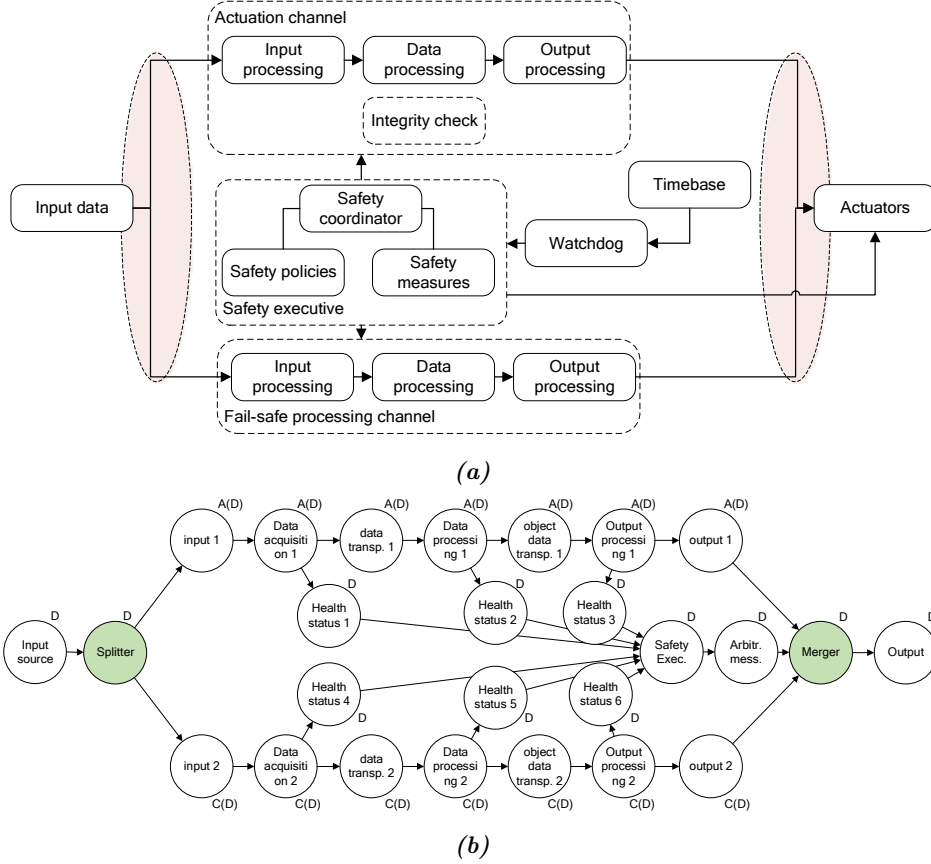


Figure 3.17: In the safety executive pattern the splitter operation is performed in the Input Source block and the merger is part of the Safety Coordinator and actuator blocks, receiving arbitration signals from the safety executive block (a), the model for a specific safety executive implementation is shown in (b).

the expected one. The resource *ECU*, being a single point of failure, inherits the ASIL of the original FSR.

When virtualization is used to map redundant parts of an application to VRs, we can identify three scenarios related to the position of the splitters and mergers:

1. Splitters and mergers are mapped to separate VRs but on the same hardware resource.
2. Splitters and mergers are part of the virtualization kernel.
3. Splitters and mergers are mapped to separate hardware resources.

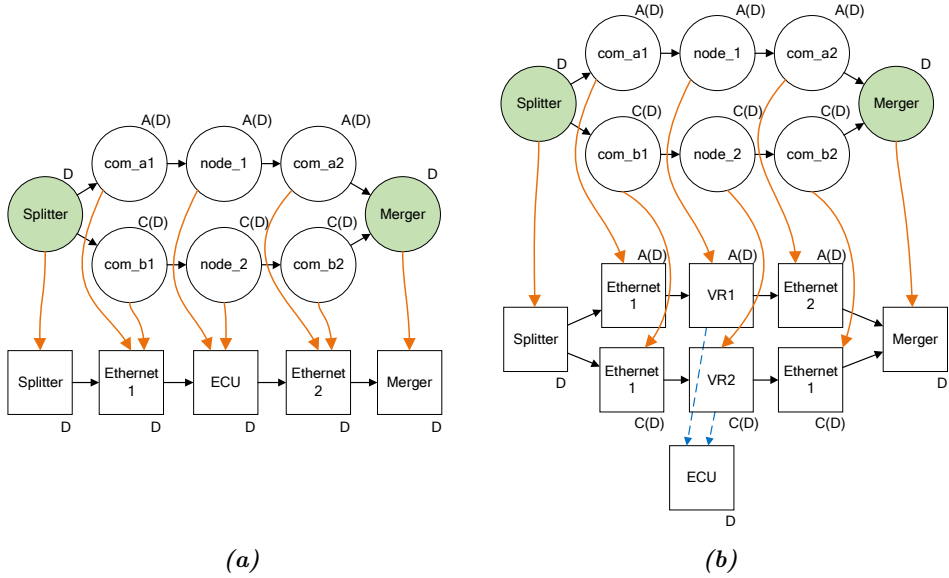


Figure 3.18: Two redundant nodes mapped on non-redundant resources (a) or to two VRs (b).

In the first scenario, direct communication must be allowed between different VRs. While theoretically possible, this could potentially break the isolation mechanism between the VRs. Moreover, the VRs on which the splitter and merger operations are performed must have an ASIL specification that matches the non-decomposed ASIL requirement, since these operations are performed at the pre-decomposition FSR ASIL. However, if the virtualization mechanism allows this, no additional resources are necessary, separate and isolated VRs are used for the safety-critical splitters and mergers nodes, and the overhead for the additional communication is low.

In the second scenario, the virtualization mechanism itself must provide the tools to perform the splitter and merger operations. These operations are often application-dependent, meaning that the user would require direct access to the virtualization kernel. A modification of this would invalidate any previous ASIL certificate that the virtualization mechanism had. Again, if the virtualization mechanism has this built-in functionality, the splitters and mergers have a minimum overhead and no additional resources or VRs are necessary. One can envision that several standard splitters and mergers could be offered at the highest ASIL by the virtualization mechanism, e.g. FRER [76] or MooN [36].

In the last scenario, no modification of the virtualization mechanism is required. Moreover, no inter-communication between VRs is necessary. Because of these reasons, we are using this splitter and merger implementation in the following

analysis, using the VRs only for mapping the redundant branches of the application layer. While this is the simplest solution, it is also the one with the most overhead, because it uses additional resources to run the splitter and merger nodes. If the first two options are available at a reasonable cost in the virtualization mechanism, they should always be preferred.

3.4 Quantitative analysis of the model

The analysis that we perform in our framework consists of the calculation of five metrics: failure probability of an application, cost of the system, total communication cable length, total functional load, and total communication load. Each metric is calculated with an analysis of the three-layers model and by using either input values or metrics related to the ASIL level of the elements.

3.4.1 Fault tree generation and failure probability calculation

The first step of our evaluation is to assess the failure probability. We use a fault tree, which has to be generated for each application graph. A fault tree is a diagram that represents the modelled system, in which a *top-level event* corresponds to the failure of the whole system. In static fault trees, *AND* and *OR* gates connect the *basic events* and the *events* to the top-level event. Basic events correspond to the failure of an element of the system, while events (or intermediate events) are intermediate points in the fault tree. Failure rates are associated with the basic events, allowing for the calculation of the failure probability of the top-level event. In particular, we assume an exponential failure probability $P(t)$ associated to the base events, that follows Equation 3.2. For a given mission time t , $P(t)$ represents the probability of the base event fails before t . The failure rate λ correspond to the associated failure rates described in Section 3.2. For simplicity, in the rest of the thesis we will consider $t = 1h$ in all the failure probability calculations.

$$P(t) = 1 - e^{-\lambda t} \quad (3.2)$$

Many techniques have been used for the evaluation of a fault tree [130]. We chose to use the open source tool SCRAM [122] as we describe later in Section 3.4.2.

Before we can evaluate the fault tree, we must generate it from the three-layer model. The very first step is to apply a graph transformation to enforce one-to-one mapping of the application and the resources. This makes sure that each communication node is mapped to one resource, and each resource is mapped to one location. We will describe the details of this transformation in Section 4.2.3 in the next chapter.

Our fault tree generation method is a recursive procedure based on [103] and extended by us in [55]. We assume that the violation of an FSR related to an application graph corresponds to the failure of at least one of the actuators. To

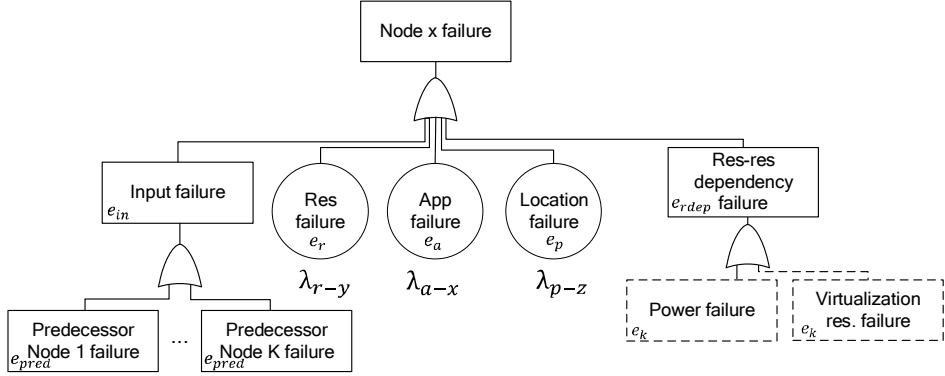


Figure 3.19: Sub-tree structure for the node x *with* the fail-silent assumption. The events are marked with the names they have in Algorithm 3.1 and the basic events are annotated with the corresponding failure rate.

compute the failure of an actuator in the fault tree, all nodes providing it with data are inserted in the tree. The application graph is traversed recursively starting from each actuator node until the sensors are reached. With the assumption of the sense-think-act paradigm and fully connected applications, all the nodes in the application graph are visited. Whenever a node is visited for the first time, a sub-tree structure is created. Cycles are often present in automotive applications and can exist in our model, e.g. feedback loops in control applications. When a node is visited for a second time, its sub-tree, already present in the fault tree, is connected to the parent event, and the recursion is interrupted.

In Section 2.6, we referred to the assumption of *fail-silent* elements. The assumption has an impact on how the fault tree is generated. We focus on the behaviour of the application nodes, and we assume that they are either fail-silent or not. The generated fault tree patterns are different in the two scenarios.

Figure 3.19 shows the generated sub-tree for node N when we assume that the nodes are fail-silent. Because of this assumption, the failure of the application, marked as e_a , depends only on the failure rate of the application node N . In the figure, rectangles correspond to fault tree events, while circles represent fault tree basic events.

In the case nodes are not fail-silent, the basic event e_a turns into an event, connected via an *OR* gate to the failure of all the application nodes that share the same resource, as shown in Figure 3.20. If nodes with the same ASIL requirement share a resource, their failure rates will be similar. If a relevant number of same-ASIL nodes is sharing the resource it could invalidate the system-level requirements, as we will describe later in this section. When mixed-critical nodes (i.e. with different ASIL requirements) share a resource, the failure rates of the nodes with the lower ASIL requirements will differ from the higher ASILs by orders

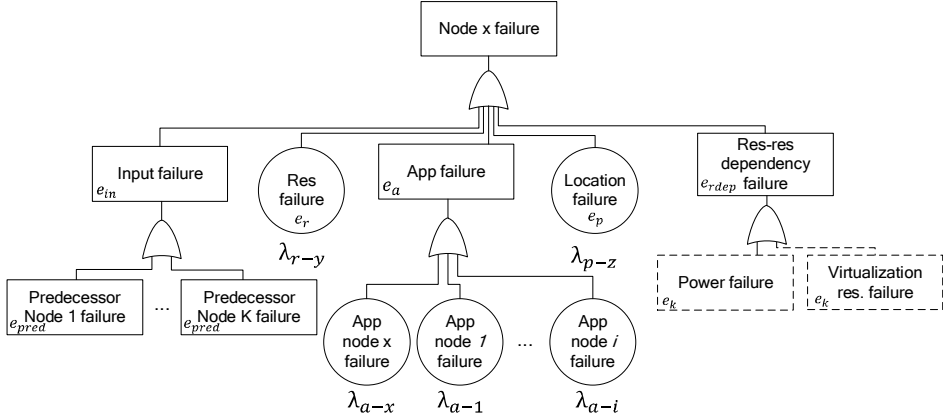


Figure 3.20: Sub-tree structure for the application node x **without** the fail-silent assumption. The event e_{a-x} depends on the failure rates of all nodes that shares the same resource as x .

of magnitude. The failure probabilities of the higher ASIL applications are in this case negatively impacted by that. Note that this can happen even with a single application after the ASIL decomposition process has been applied.

In the experiments of Chapter 4 we assume that the nodes are fail-silent. In Chapter 5 we will first analyse different architecture topologies with the fail-silent assumption and then see what differences not using it makes. Virtualization or physical separation techniques are used to avoid resource-sharing problems.

In this work we assume that the other elements of the system are *fail-silent*: for example, a failing VR does not affect the physical resource that runs the virtualization mechanism, or a failing resource does not affect its power supply. The *fail-silent* discussion can be extended to these other elements as well. However, these scenarios are less likely to arise compared to application-level failures, and we leave this topic open for future investigation.

The failure rates described in Section 3.2 are assigned to each basic event:

- λ_{a-x} is assigned to the failure of an application node x (basic event e_{a-x});
- λ_{r-y} is assigned to the failure of a resource y (basic event e_{r-y});
- λ_{p-z} is assigned to the failure of a location z (basic event e_{p-z});

The power line and power source are resource-resource dependencies. No application nodes are mapped on these types of resources. They appear in the fault tree in relation to the resources to which they supply power. Each power line or power supply has an associated λ_r , similar to the other resources. In Figure 3.19 the power line connection is dashed because only physical resources are connected

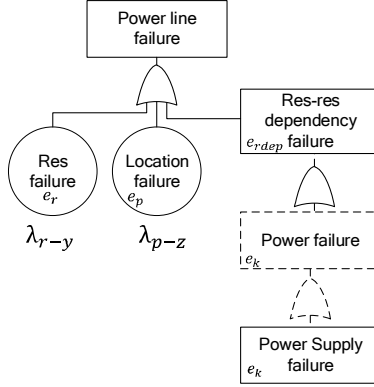


Figure 3.21: Sub-tree structure for the power supply resource-resource dependency. Each power line resource has a failure corresponding to the resource failure and one corresponding to the location failure. The dependency chain continues until a power source resource, which is a resource without dependencies on another resource, is found.

to a power supply. One or more power lines can connect the power supply to a physical resource. Figure 3.21 shows the expansion of the event in the power supply resource-resource dependency. Similarly, a VR has a resource-resource dependency with the physical resource that runs the virtualization mechanism, and the sub-tree contains the failure of the resource and the failure of the location as shown in Figure 3.22.

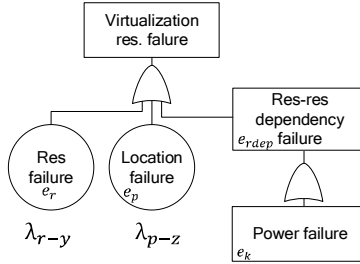


Figure 3.22: Sub-tree structure for the virtualization resource-resource dependency. The physical resource itself has a power supply dependency.

From each input node, a new sub-tree is developed in the same way, until the sensors are reached. The sensor nodes do not have additional predecessor nodes other than virtual splitters. Figure 3.23 shows an example of the results of the fault tree generation on a simple sensor-communication-actuator graph.

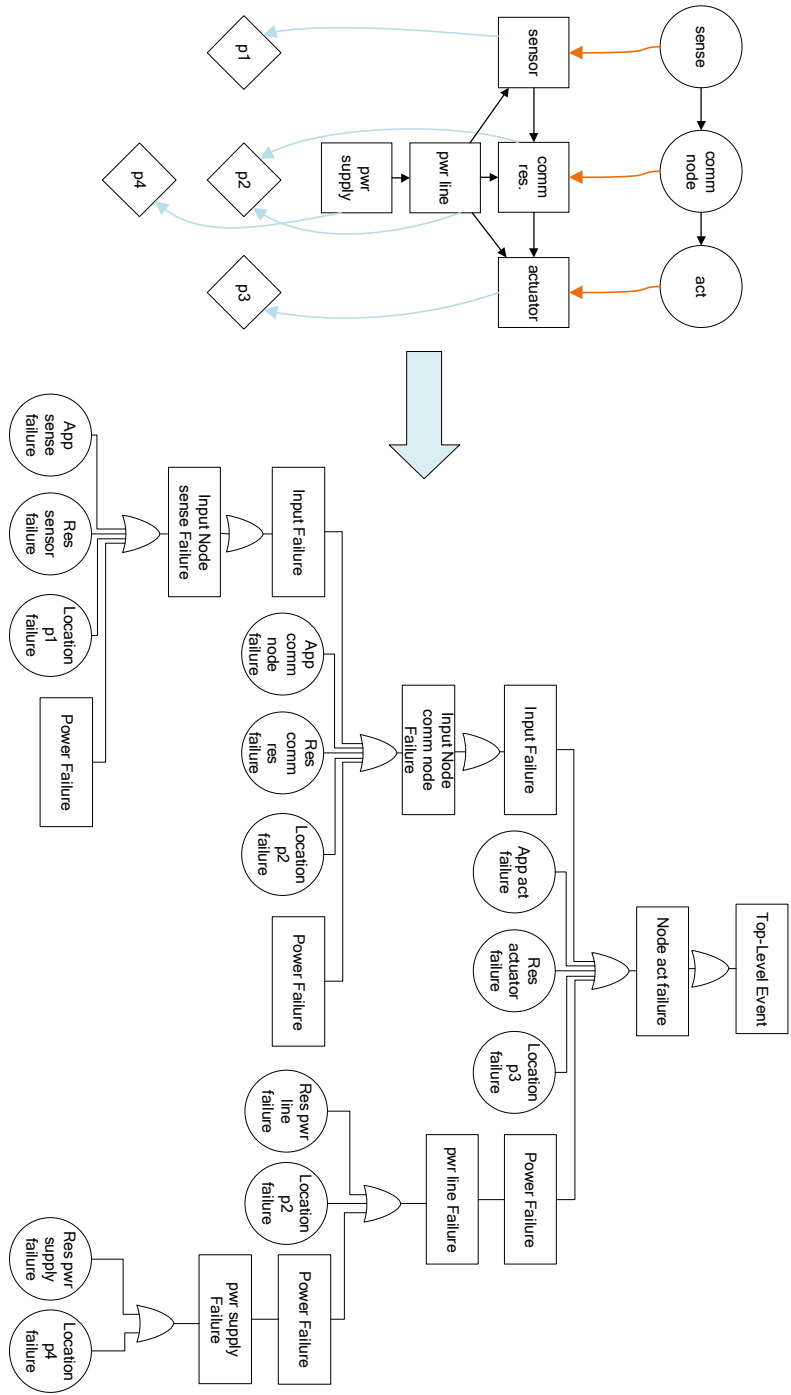


Figure 3.23: Generated fault tree from a simple application (fail-silent application nodes).

Algorithm 3.1 describes the fault tree generation procedure. The inputs of the fault tree generation procedure are an application graph G_a , the mappings between the three layers M_{ar} and M_{rp} , and the set of resource dependencies edges E_{dep} . The output of the algorithm is the fault tree representing the application described by G_a . A new mapping function is constructed while traversing the tree application graph: M_{ae} maps the nodes in G_a to the corresponding event in the fault tree. The set of starter nodes N_{start} is populated on lines 5 to 7 by selecting either the actuators or the corresponding virtual mergers present in the application graph. On line 14, from each starting node, the sub-tree patterns are created by travelling the application graph with the *DevelopSubTree* function. The functions *Successor*(n, G_a) and *Predecessor*(n, G_a) return the set of successor or predecessor nodes of n respectively. The events connected to the actuators (or virtual mergers) are then connected with an OR gate to the top-level event of the fault tree (line 17) with the *CreateGateOR*(e, E) function. *CreateGateOR*(e, E) creates an OR gate with its output connected to the event e and its inputs connected to the events in the set E . The function *CreateGateAND*(e, E) behaves in a similar way, but creates an AND gate instead.

Algorithm 3.1 Fault Tree Generation

Inputs: Application graph G_a , Applications mapping M_{ar} , Resources mapping M_{rp} , Resource dependencies E_{dep}
Output: Fault Tree Graph

```

1: procedure GENERATEFT( $G_a, M_{ar}, M_{rp}, E_{dep}$ )
2:    $E_{ft} = \emptyset$ 
3:    $M_{ae} : G_a \rightarrow E_{ft}$ 
4:    $N_{start} = \emptyset$ 
5:   for  $n_i \in G_a$  s.t.  $\text{NodeType}(n_i) == \text{actuator}$  do
6:     if  $\text{Successor}(n_i, G_a) == \emptyset$  then
7:        $N_{start} = N_{start} \cup n_i$ 
8:     else
9:       for  $n_{succ}$  in  $\text{Successor}(n_i, G_a)$  s.t.  $\text{NodeType}(n_{succ}) ==$ 
10:         virtual merger do
11:            $N_{start} = N_{start} \cup \text{Successor}(n_i, G_a)$ 
12:    $E_{act} = \emptyset$ 
13:   for  $n \in N_{start}$  do
14:      $e_{act} = \text{DevelopSubTree}(n, G_a, M_{ar}, M_{rp}, E_{dep}, M_{ae}, E_{ft})$ 
15:      $E_{act} = E_{act} \cup e_{act}$ 
16:    $e_T = \text{CreateTopLevelEvent}()$ 
17:    $\text{CreateGateOR}(e_T, E_{act})$ 
18:   return  $e_T$ 

```

Algorithm 3.2 shows the procedure that travels through the application graph

to create and connect the sub-tree patterns. An event is created and the application node is mapped to it with the function M_{ae} on line 3. If a node has already been visited, it is connected to the parent event and the recursion is interrupted, breaking eventual cycles in the application graph. The resource on which the node is mapped and the location on which the resource is mapped are found with the function $Map(x, M_{xy})$ on lines 4 and 5. For non-sensor nodes, the inputs failure event is created (line 7).

The base events are created with the $CreateBaseEvent()$ functions. These functions create the base event object and add the respective failure rate to each of them. If the application nodes are *fail-silent* (line 8), the base event e_a is created, with the failure rate associated to the node n . If the application nodes are not fail-silent, an event e_a is created. The base events associated with the failure of all the application nodes mapped to the resource r are connected with an OR gate to the event e_a (line 16).

The resource-resource dependency set of events is created and populated respectively on lines 20 and 23. Each resource dependency is developed with the $DevelopResourceSubTree()$ function described in Algorithm 3.3, and then connected to the e_{rdep} event with an OR gate (line 22).

On line 25 Algorithm 3.2 finds the predecessors of the node n . If they have already been visited (line 26), their sub-tree pattern is added to the set of input events (line 27). If not, for each predecessor a new sub-tree pattern is created recursively (lines 28 to 31). If the predecessor node is a virtual splitter, no additional sub-tree patterns are created, as we assume that a virtual splitter does not have base failure events.

When the current application node is a merger, a distinction must be made in the fault tree depending on the merger implementation. The algorithm shows the resulting fault tree for a *data oriented* with two channels, a *in-band*, or a *data evaluation* merger. The only inputs come from the predecessor nodes, and belong to the redundant channels. In this case, the input failure event happens only if all of the merger's predecessor nodes fail, hence the use of an AND gate in place of the OR gate after the input failure event e_{in} (line 33).

Finally, on line 36, the OR gate that connects the top-level event of the sub-tree pattern is created and connected to the events e_a , e_r , e_p , e_{rdep} , and e_{in} .

Algorithm 3.2 Fault Tree Generation, sub-tree pattern

```

1: procedure DEVELOPSUBTREE( $n, G_a, M_{ar}, M_{rp}, E_{dep}, M_{ae}, E_{ft}$ )
2:    $F_{sub} = CreateNodeEvent(n, E_{ft})$ 
3:    $M_{ae}(n) = F_{sub}$ 
4:    $r = Map(n, M_{ar})$ 
5:    $p = Map(r, M_{rp})$ 
6:   if NodeType( $n$ )  $\neq$  sensor then
7:      $e_{in} = CreateInputFaultEvent(n)$ 
8:   if Fail-silent application nodes then

```

```

9:      $e_a = \text{CreateAppBaseEvent}(n)$ 
10:  else
11:      $e_a = \text{CreateAppEvent}(n)$ 
12:      $E_a = \emptyset$ 
13:     for  $w \in \text{AppNodesMappedToResource}(r, M_{ar})$  do
14:          $e_{a-w} = \text{CreateAppBaseEvent}(w)$ 
15:          $E_a = E_a \cup e_{a-w}$ 
16:      $\text{CreateGateOR}(e_a, E_a)$ 
17:      $e_r = \text{CreateResourceBaseEvent}(r)$ 
18:      $e_p = \text{CreateLocationBaseEvent}(p)$ 
19:      $e_{rdep} = \text{CreateResDepEvent}(r)$ 
20:      $E_k = \emptyset$ 
21:     for  $r_k \in \text{ResDep}(r, E_{dep})$  do
22:          $e_k = \text{DevelopResourceSubTree}(r_k, M_{rp}, E_{dep})$ 
23:          $E_k = E_k \cup e_k$ 
24:      $\text{CreateGateOR}(e_{rdep}, E_k)$ 
25:     for  $n_j \in \text{Predecessor}(n, G_a)$  do
26:         if  $\text{Map}(n_j, M_{ae})$  then
27:              $E_{pred} = E_{pred} \cup \text{Map}(n_j, M_{ae})$ 
28:         else
29:             if  $\text{NodeType}(n_j) \neq \text{virtual splitter}$  then
30:                  $e_{pred_j} = \text{DevelopSubTree}(n_j, G_a, M_{ar}, M_{rp}, E_{dep}, M_{ae}, E_{ft})$ 
31:                  $E_{pred} = E_{pred} \cup e_{pred_j}$ 
32:             if  $\text{NodeType}(n) == \text{merger} \parallel \text{NodeType}(n) == \text{virtual merger}$  then
33:                  $\text{CreateGateAND}(e_{in}, E_{pred})$ 
34:             else
35:                  $\text{CreateGateOR}(e_{in}, E_{pred})$ 
36:      $\text{CreateGateOR}(F_{sub}, \{e_a, e_r, e_p, e_{rdep}, e_{in}\})$ 
37:     return  $F_{sub}$ 

```

Algorithm 3.3 describes the last procedure that generates the resource-resource dependency structure in the fault tree. The procedure is similar to the *DevelopSubTree* one, but no application node is present. The resource dependency contains the failure of the resource and the location on which it is mapped on. The recursive function will continue until no resource-resource dependencies are present.

Algorithm 3.3 Fault Tree Generation, resource-resource dependency

```

1: procedure DEVELOPRESOURCESUBTREE( $r, M_{rp}, E_{dep}$ )
2:    $p = \text{Map}(r, M_{rp})$ 
3:    $e_{rdep} = \text{CreateResDepEvent}(r)$ 
4:    $E_k = \emptyset$ 

```

```

5:   for  $r_k \in \text{ResDep}(r, E_{dep})$  do
6:      $e_k = \text{DevelopResourceSubTree}(r_k, M_{rp}, E_{dep})$ 
7:      $E_k = E_k \cup e_k$ 
8:   CreateGateOR( $e_{rdep}, E_k$ )
9:    $e_r = \text{CreateResourceBaseEvent}(r)$ 
10:   $e_p = \text{CreateLocationBaseEvent}(p)$ 
11:   $F_{resSub} = \text{CreateResourceEvent}(r)$ 
12:  CreateGateOR( $F_{resSub}, \{e_r, e_p, e_{rdep}\}$ )
13:  return  $F_{resSub}$ 

```

In case of *data oriented* mergers with more than two redundant channels, *side-band*, and *out-band* mergers, additional predecessor nodes are sending information to the merger to take its decision. In this case, if the additional signals are not redundant, they can cause a direct failure of the merger. For example, in Figure 3.24 the fault tree relative to a *2oo3* merger is shown: only when any of the two predecessor nodes fail does a failure of the merger occur. Figure 3.25 shows the fault tree structure for an out-band merger: in this case, the input failure of the merger is caused either by the failure of both the redundant channels or by the failure of the out-band data. In the fault tree generation algorithm and in the rest of the experiments in this work, we use only mergers belonging to the first group (data oriented with two channels, in-band, or data evaluation mergers). To support the other merger types, an addition to the fault tree generation is required, in which the type of the merger is identified and the fault tree is generated accordingly.

Once the fault tree is obtained, we can calculate the failure probability of the application. Note that the failure rate is given as *failures/hour*, and in our calculations we use $t = 1h$.

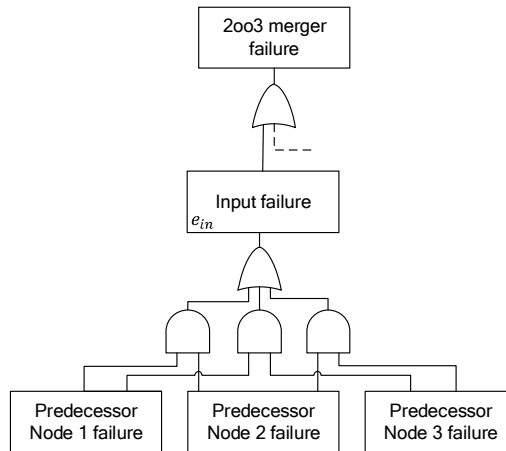


Figure 3.24: Fault tree structure for a *2oo3* merger.

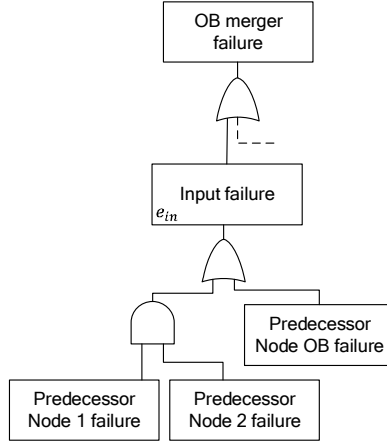


Figure 3.25: Fault tree structure for a out-band oriented merger.

3.4.2 Static fault tree analysis

Once the fault tree is generated, we perform static fault tree analysis with the open source tool SCRAM [122]. SCRAM is a command line tool for probabilistic analysis. It capable of performing static fault tree analysis by using state of the art techniques, and can calculate single-time failure probabilities for our model in matters of milliseconds. We export our the fault tree model into the Open-PSA Model Exchange Format [49], a XML-based format used to exchange fault trees and event trees between various tools. For each application graph G_a a separate fault tree file is generated, and analysed with the tool by setting the mission time to 1. The format does not allow for OR and AND gates with a single input. In our model, it is possible to obtain OR gates with a single input in some parts of the fault tree. For example, when only one input node is connected to the current node, only one e_{pred} is connected via an OR gate to e_{in} . In this case, we add a *dummy event* with $P(t) = 0$ ($\lambda_{dummy\ event} = 0$).

Listing 3.1 shows the definition of the OR gate that defines the event *failure_app_n*. In this example, the node n is mapped to the resource r , which in turn is mapped to the location p . Listing 3.2 shows an example of a base event definition in the input format of the tool. In this case, the failure rate $\lambda_{base_event_name} = 1 * e^{-9}$.

```

1 <define-gate name="failure_app_n">
2   <or>
3     <basic-event name="event_0_sw_n"/>
4     <basic-event name="event_1_res_r"/>
5     <basic-event name="event_4_loc_p"/>
6     <gate name="failure_app_n_inputs"/>
7     <gate name="failure_res_r_dependencies"/>

```

```

8   </or>
9 </define-gate>

```

Listing 3.1: Gate definition in Open-PSA Model Exchange Format

```

1 <define-basic-event name="base_event_name">
2 <sub>
3   <float value="1.0"/>
4   <exp>
5     <mul>
6       <neg>
7         <parameter name="lambda-base_event_name"/>
8       </neg>
9       <system-mission-time/>
10    </mul>
11  </exp>
12 </sub>
13 </define-basic-event>
14 <define-parameter name="lambda-base_event_name">
15   <float value="1e-09"/>
16 </define-parameter>

```

Listing 3.2: Base event definition with exponential failure probability in Open-PSA Model Exchange Format

3.4.3 Approximation of the fault tree

When analysing the redundant parts of the application with the fault trees analysis, the failure probability associated to the redundant block is dependent on both the splitter and merger and a combination of the failure rates of the nodes in the redundant branches. In the fault tree, the merger has an AND gate that combines its inputs. If the nodes in separate redundant branches are independent, their combined contribution to the redundant block failure probability is proportional to a multiplication of the individual failure probability of each branch. When the failure rates are in the order of 10^{-5} to 10^{-10} , the contribution of the redundant branches is up to 10^5 to 10^{10} times smaller compared to that of the splitter and merger.

By approximating the fault tree as shown in Figure 3.26, the difference in the calculated failure probability can be negligible, as we will demonstrate in Section 4.5.1. In the experiments in Chapter 4, we will use failure rates proportional to the ASIL values: a decomposition that does $\text{ASIL D} \rightarrow \text{ASIL A(D)} + \text{ASIL C(D)}$ and they have failure rates of 10^x , 10^{x-3} , and 10^{x-1} respectively (x is -9 in the experiments). With these values, with N being the number of nodes in each redundant branch, and with two redundant branches, the approximation holds

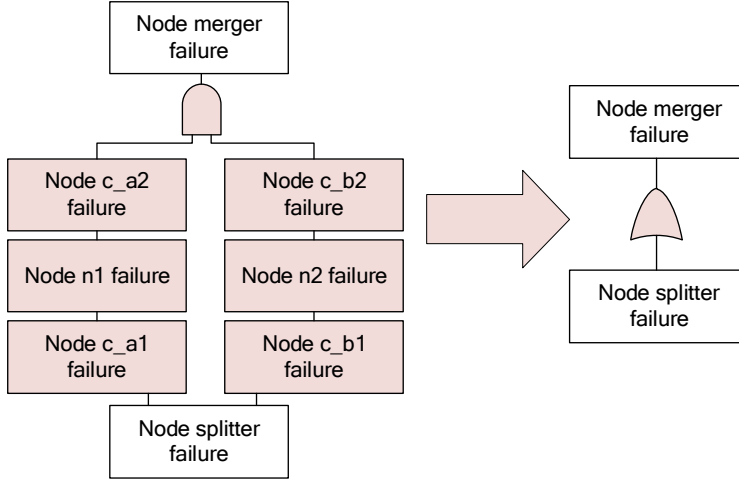


Figure 3.26: Fault tree approximation that considers only single-point of failure elements.

when:

$$\begin{aligned}
 2N * 10^{x-3} * 10^{x-1} &\ll 2 * 10^x \\
 N * 10^{2x-4} &\ll 10^x \\
 N &\ll 10^{-x+4} \\
 N &\ll 10^{13}
 \end{aligned}
 \tag{3.3}$$

Note that on the first line of Equation 3.3, the $2 * 10^x$ refers to the combination of the failure rates of the splitter and of the merger. In this case both have the same value 10^x .

Moreover, the approximation holds when the nodes in the separate branches are independent, as required by the standard ISO26262 for redundant elements of the system. We assume that nodes are independent in the modelled system if they do not share any basic event in the fault tree. We perform a CCF analysis on the application by checking that redundant branches do not share basic events to validate the approximation. If a basic event is found in both branches, a warning for a potential CCF is issued. The warning informs the user that the ASIL decomposition is not valid in this case, as well as the results obtained by using the approximation in the failure probability calculation. The approximation is not necessary for the static fault tree analysis, as the complexity of the generated fault trees is not a concern for the used tools. However, this result is meaningful in terms of design of the system: the elements that are part of the redundant branches will not influence the system failure probability, as long as the redundant branches are independent and Equation 3.3 is valid. The splitters and mergers instead, together

Table 3.2: Measures for the avoidance of systematic failures recommended by the ISO26262 standard. "++" means that the method is highly recommended for this ASIL, "+" that it is recommended, "o" that the method has no recommendation for or against its usage for this ASIL.

Methods	ASIL			
	A	B	C	D
Deductive analysis (e.g. FTA, reliability block diagrams)	o	+	++	++
Inductive analysis (e.g. FMEA, ETA, Markov modeling)	++	++	++	++

with the other single-point of failure elements of the system, contribute directly to the system failure probability, and should be optimized for safety.

3.4.4 Cost calculation

We assign a cost parameter to the resources in the resource layer. This parameter is related to the bill of material cost of the resource. Depending on the type of the analysis and at the stage in which it is performed, it can be a representative number or the actual cost of using the resource. In our case, we use an ASIL-based metric, in which the cost of a specific type of resource is proportional to its ASIL. We can find other works in literature that associate the cost of a resource to its ASIL specification, for example [13, 14, 106]. Each ASIL requires different procedures for validation and certification, for example the avoidance of latent faults is recommended for ASIL A and B in accordance to sound engineering practice, while the latent failure metric is an evaluation criterion for ASIL C and D [81]. Another example of different methods for validation applied to the system at different ASILs is shown in Table 3.2 [81]. Most of the methods that are suggested in the standard are highly recommended for ASIL C and D, while low or no recommendation is given for ASIL A and B. Because of this, we suggest a cost metric that has identical numbers for the pairs ASIL A and B, and ASIL C and D, as shown in Table 3.3. Compared to our criteria, in the previously mentioned works, a different cost was assigned to each ASIL value.

Table 3.3: Resources cost metric (arbitrary unit).

Resource Type	QM	A	B	C	D
Functional	5	500	500	50000	50000
Communication	4	400	400	40000	40000
Sensor / Actuator	8	800	800	80000	80000
Splitter / Merger	1	100	100	10000	10000

In the case of VRs, we assign an individual cost of 0, and we consider only the cost of the resource on which they are virtualized. The cost of the virtualization

mechanism is part of the cost of the physical resource that runs it. This cost value depends on the ASIL specification of the virtualization mechanism itself and the ASIL specification of the VRs that it can virtualize, according to Table 3.4.

Table 3.4: Hypervisor cost metric (arbitrary unit).

		VRs				
		QM	A	B	C	D
	QM	1	-	-	-	-
Hypervisor	A	10	100	-	-	-
	B	10	100	100	-	-
	C	100	100	100	1000	-
	D	100	100	100	1000	10000

In our tools, we give the cost metric as an input in the form of an editable Microsoft Excel table. For the total cost calculation, all the resources on which at least one application node is mapped are considered and their cost is added to the cost parameter. In case a resource has multiple types, the cost associated with each type is added up.

3.4.5 Communication cable length calculation

In our framework, the total cable length is calculated by using the properties of the resource and the physical layers. We assume that from each location in the physical layer we can reach any other location. This is a simplification of a real system, in which some points in the car are accessible only through specific paths, for example, the cables connecting sensors and ECUs cannot be in the passenger compartment for safety reasons. With an extension of the framework, routing of the communication cables and power lines is possible for a more accurate calculation of the total harness.

To calculate the total communication cable length of a system, we identify all the purely communication resources present in the resource layer. A cable (either Ethernet, CAN bus, or any other type of wired communication) is represented in the resource layer as a purely communication resource. Note that resources such as a gateway are not purely communication resources.

Once the list of purely communication resources is selected the calculation of this parameter can start. The two resources that are connected by each resource in the list are identified, and the distance between the locations in which they are mapped is calculated. In the case of a bus or a power line that connects more than two resources, only the longest distance is considered, which is an approximation of the actual routing of the communication resource. Given $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ the locations on which the resources are mapped, the

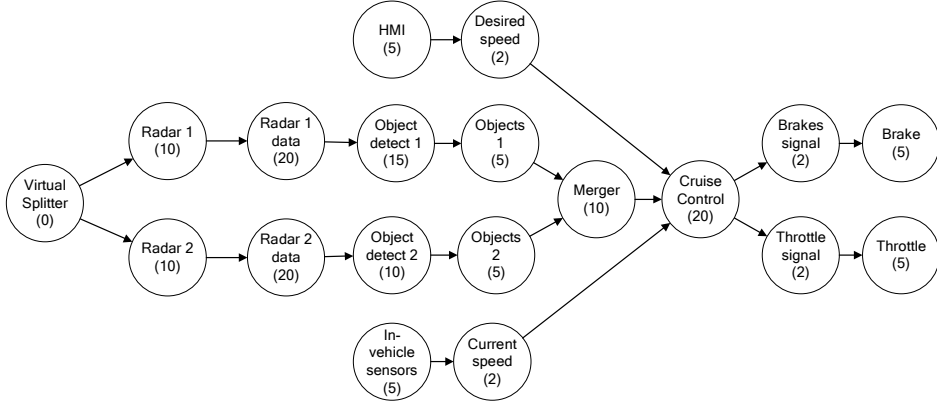


Figure 3.27: Functional and communication loads for the application nodes of the ACC example.

distance is calculated in two ways: Euclidean distance, according to Equation 3.4, and Manhattan distance, according to Equation 3.5.

$$\|p_1 - p_2\| = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (3.4)$$

$$\|p_1 - p_2\| = |x_1 - x_2| + |y_1 - y_2| \quad (3.5)$$

Because of the restriction with the placement of the wires in the vehicle, the Manhattan formula may provide more realistic results. However, we approximate the physical space by using two coordinates. The result of the total cable length calculation performed in this way is intended to have a relative comparison between different implementations.

3.4.6 Functional and communication loads calculation

In Section 3.2.1 and 3.2.2 we introduced the functional load and communication load for application nodes, and the maximum functional load and maximum communication load for the resources.

In Figure 3.27 we show the functional and communication loads used on the example application used in the previous chapter. The number inside the parenthesis is either the functional or the communication load depending on the type of the node. In the case of resources, the maximum functional load and the maximum communication load are two separate parameters.

For each application G_a the total functional load and total communication load is calculated, as well as the distribution of the loads on the different resources. By using the information contained in the three layers, we can create a load

distribution map, as shown in Figure 3.28, to identify how the different parts of an application are distributed. In the figure, the circles correspond to the functional loads, and the lines correspond to the communication load. The size of a circle corresponds to the number of resources mapped to that location, while the colour is proportional to the sum of the functional load on all the resources at that location, scaling from green to red. Similarly, the width of the edges scales with the number of communication resources that connects two locations, while its colour scales with the total communication load between the two locations.

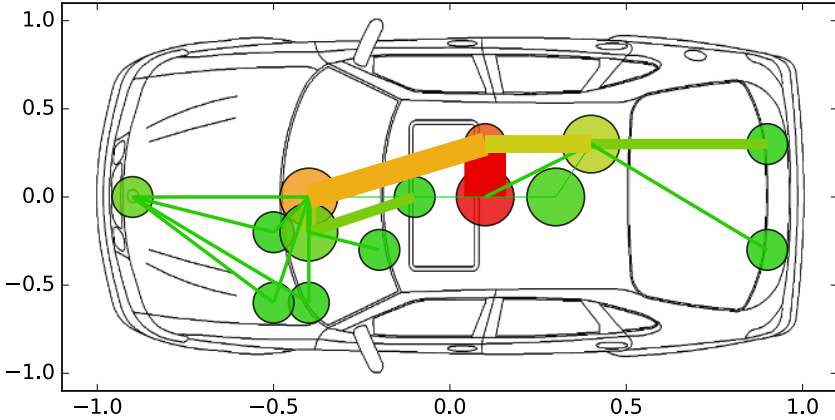


Figure 3.28: Example load distribution map in the physical space.

Functional and communication loads are the last two metrics that are evaluated, however, our framework allows the addition of new metrics.

3.5 Conclusions

In this chapter, we have introduced the three-layer model that we use to describe an automotive application and its relationship with the hardware resources and the physical space of the car. An overview of the symbols and the properties of the elements used in the model is shown in Appendix A. We focussed on describing redundancy with the use of special nodes, the splitter and the merger, that allow us to identify redundancy patterns in the application and resource graphs. We discussed possible implementations of these safety-oriented functionalities and their characteristics related to the ASIL decomposition technique of the ISO 26262 standard.

We then described our analysis framework, which allows us to evaluate a modelled system. The fault tree generation algorithm has been explained, and the resulting fault trees have been analysed with static fault tree analysis with the SCRAM tool. The fault-tree generation algorithms that are used to calculate the

failure probability of an application are explained. The cost metric is described. The communication cable length is calculated in the physical space with either Euclidean or Manhattan distance calculations. The total communication and total functional load are evaluated from the load values of the input model and the mapping of the application. The metrics used to evaluate these parameters are input to the framework and can be tuned to a specific project or system. Similarly, additional metrics such as weight, volume, etc. can be added to the model.

In the next chapter, we describe the model transformations that we use to introduce redundancy based on the pattern described by the splitter and merger nodes. We analyse then two use-cases in both redundant and non-redundant implementations.

4

Model transformations to introduce redundancy

4.1 Introduction

In this chapter we present the Automotive Safety Integrity Level (ASIL)-decomposition-based model transformations that we use to automatically introduce redundancy in the system. We identify five main transformations in Section 4.2: *Substitute*, *Connect*, *Separate*, *Reduce*, and *Collapse*.

The *substitute* and the *connect* transformations generate non-fully-mapped graphs as they modify a single layer in the model. Additional steps to complete the transformation process, such as remapping the newly introduced application nodes, are required to obtain analysable graphs. In turn, the additional steps may require further modification of the application or resource layers. The *separate*, *reduce*, and *collapse* transformations modify instead the mapping of either the application nodes to the resources or the resources to the physical locations. The output of these transformations are fully mapped graphs and are analysable.

In Section 4.3 we describe the full transformation process to introduce redundancy in a selected application node and generate an analysable graph that can be evaluated in our framework. In Section 4.4 we describe a different transformation process in which the user selects a resource instead of an application node.

In Section 4.5 we discuss experiments and examples of our transformation process and analysis framework. This chapter has been partially published in [55, 56].

4.2 Bottom-up ASIL decomposition

In this section, we discuss how we introduce redundancy in the system model. Our goal is safety by design, in which we develop the system with redundancy and ASIL decomposition technique in mind. As introduced in Chapter 2, the ASIL decomposition technique goal is to decompose an Functional-Safety Requirement (FSR) into multiple lower-level requirements. An FSR can be associated with redundant parts of a system with this technique. This is a top-down process, in which the FSRs are decomposed and assigned to existing parts of the system implementation, with independence requirements.

In [55] we introduced a bottom-up approach to the ASIL decomposition technique. We modify the system so that it can accommodate decomposed FSRs: we transform the application and the resource layers so that the redundant elements are independent of each other. We add the redundancy management parts to the system with the use of splitters and mergers, which have non-decomposed ASIL values according to the system level analysis for the FSR.

In the following subsections, we present the model transformations that we use to introduce redundancy. First, we describe five main transformations. Second, we describe the full transformation process, then apply it to an illustrative example first and to a real use-case second.

4.2.1 Transformation 1: substitution

The main transformation that we apply to a node or a resource in the graph is *substitution*. The user selects a communication or a functional node or resource and this transformation is applied to it. The four different types of substitution (application node, communication node, functional resource, and pure communication resource) are similar to each other. We describe the *application node substitution rules* and the *communication node substitution rules*, that are identical to the *functional resource substitution rules* and *pure communication resource transformation rules* respectively.

With respect to the ASIL decomposition, a substitution can be of two types. If we assign a value of 0 to QM, 1 to ASIL A, 2 to ASIL B, 3 to ASIL C, and 4 to ASIL D, we can describe the two types of transformation as follows:

$$\text{Type 1} \quad \begin{cases} ASIL_1 = \text{ceil}(ASIL_{\text{orig}}/2) \\ ASIL_2 = \text{floor}(ASIL_{\text{orig}}/2) \end{cases} \quad (4.1)$$

$$\text{Type 2} \quad \begin{cases} ASIL_1 = \text{floor}(ASIL_{\text{orig}}/2) + 1 \\ ASIL_2 = \text{ceil}(ASIL_{\text{orig}}/2) - 1 \end{cases} \quad (4.2)$$

Table 4.1 shows which ASIL decomposition possibilities (part of Figure 2.7) are covered by each transformation type. Note that we are not interested in the

Table 4.1: ASIL decomposition and transformation types.

Orig. req.	Decomp. req. 1	Decomp. req. 2	Transformation type
ASIL D	ASIL B (D)	ASIL B (D)	Type 1
ASIL D	ASIL C (D)	ASIL A (D)	Type 2
ASIL C	ASIL B (C)	ASIL A (C)	Type 1 and 2
ASIL B	ASIL A (B)	ASIL A (B)	Type 1

decomposition scenarios in which an ASIL is decomposed into the same level + a QM requirement since it does not lower the original ASIL value.

Figure 4.1 shows an example of a substitution applied to a node in the application layer. Its predecessor and successor nodes must be communication nodes. The new application graph is obtained by duplicating the node f after selecting one of the two transformation types. The *functional node substitution rules* are:

1. For each predecessor of the selected node f , a splitter node is added to the graph. The ASIL value of the new splitter node corresponds to the ASIL value of the selected node. A load parameter equal to the load of the predecessor communication node is assigned to the splitter, to represent the amount of data that the splitter has to replicate over the redundant communication channels.
2. Two communication nodes follow each splitter. Their ASIL values follow the Equations 4.1 or 4.2 depending on the type of the transformation. The load assigned to the two communication nodes is the same as the load of the predecessor communication node.
3. Two functional nodes ($n1_f$ and $n2_f$) are generated to represent the redundant functionality. Their ASIL values follow the Equations 4.1 or 4.2 depending on the type of the transformation. The load assigned to the two functional nodes is the same as the selected functional node.

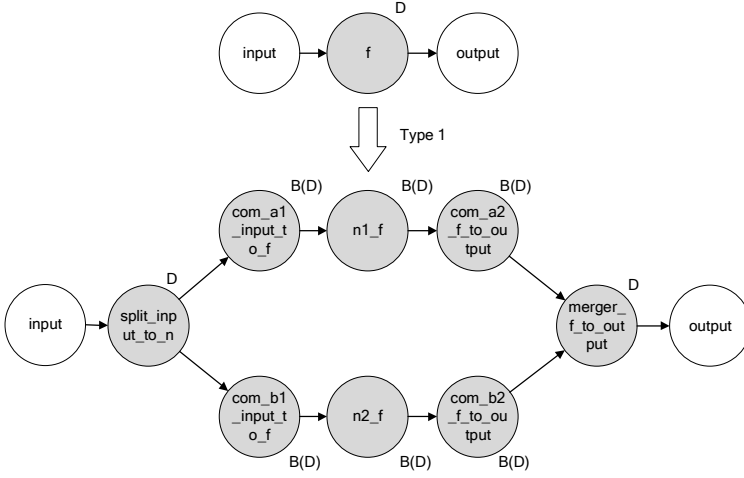


Figure 4.1: Type 1 node substitution applied to a functional node.

4. An successor communication node follows $n1_f$ and $n2_f$ for each of the original successors of the node f . Their ASIL values follow Equations 4.1 or 4.2 depending on the type of the transformation. The load assigned to these nodes is the same as the load of the successor communication node.
5. For each successor of the selected node f , a merger is added to the graph. Its ASIL value is the same as the ASIL of the selected node. The load assigned to the merger is the same as the load of the successor communication node. The merger is connected to the related successor communication node.

In case the selected node is a communication node, we use a different pattern as shown in Figure 4.2. The *communication node substitution rules* are:

1. For each predecessor of the selected node c , a communication node is added to the graph. The ASIL value of the new communication node corresponds to the ASIL value of the selected node. A load parameter equal to the load parameter of the selected communication node is assigned to the new node.
2. For each predecessor of the selected node c , a splitter node is added to the graph. The ASIL value of the new splitter node corresponds to the ASIL value of the selected node. A load parameter equal to that of the predecessor communication node is assigned to the splitter.
3. Two communication nodes are generated to represent the redundant communication. Their ASIL values follow Equations 4.1 or 4.2 depending on the type of the transformation. The load assigned to these nodes is the load assigned to the selected communication node.

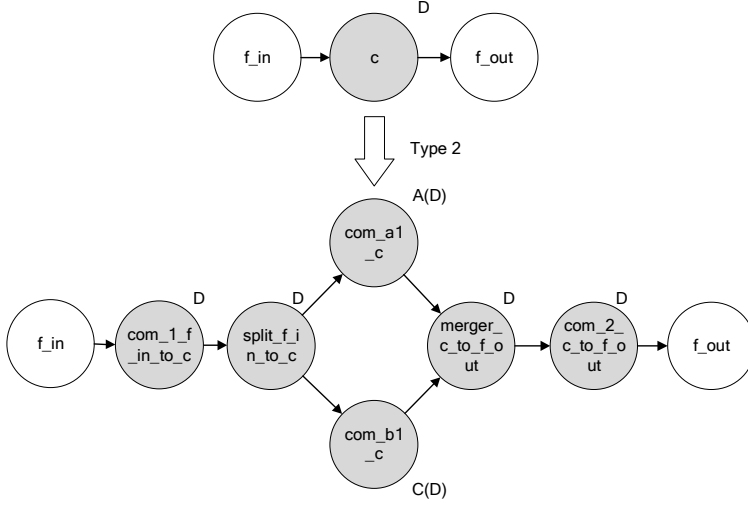


Figure 4.2: The node substitution applied to a communication node with type 2.

4. For each successor of the selected node c , a merger is added to the graph. Its ASIL value is the same as the ASIL of the selected node. The load assigned to the merger is the load assigned to the selected communication node. The communication nodes com_a1_c and com_b1_c are connected to the each merger.
5. For each merger, a successor communication node is added to the graph. The ASIL value of the new communication node corresponds to the ASIL value of the selected node. A load parameter equal to that of the selected communication node is assigned to the new node.

The communication node substitution is counter-intuitive: the new nodes $comm_1$ and $comm_2$ maintain the original ASIL of the selected node and, together with the splitters and mergers, they are the single point of failure nodes. In a functional node transformation instead only the safety-oriented splitters and mergers are the single-point of failure nodes. At first impression, the new graph is more complex and less reliable than the original, but the important difference will be in the successive mapping step. The new redundant graph allows for more advanced application mapping compared to the original graph. For example the nodes f_in , $comm_1_f_in_to_c$, and $split_f_in_to_c$ can be mapped to the same high-ASIL Electronic Control Unit (ECU) resource which sends two separate data transmissions with two separate communication resources, on which the nodes $comm_a1_c$ and $comm_b1_c$ are mapped.

The substitution transformation is applied to the resource layer in the same way. However, since resources can have multiple types, the functional resource

type is dominant when applying the substitution. The communication substitution pattern shown in Figure 4.2 is obtained only in the case of pure communication resources. Moreover, when a resource is substituted, all the nodes that were mapped to it are unmapped.

For both application and resource substitutions, the newly inserted element require mapping to the next layer. After substitution, the obtained graph is not analysable, and additional mapping steps are necessary, as we describe in Section 4.3.

4.2.2 Transformation 2: connect consecutive redundant patterns

The second transformation is used to connect consecutive redundant patterns either in the application or in the resource layer. The prerequisites for the *connect* transformation are:

1. Two consecutive redundant patterns are present.
2. The number of redundant branches must be equal in the two patterns.
3. The ASIL requirements of the nodes in the redundant branches must be compatible between the two patterns.
4. The ASIL of all the splitters and mergers must be equal.

Equation 4.3 describes the branch compatibility rules for redundancy patterns with two redundant paths. The term $branch_{xy}$ refers to the x-pattern and y-branch. The ASIL values of the two patterns must correspond to use the *connect* transformation:

$$\begin{aligned}
 ASIL(branch_{11}) = ASIL(branch_{21}) \& ASIL(branch_{12}) = ASIL(branch_{22}) \\
 OR \\
 ASIL(branch_{11}) = ASIL(branch_{22}) \& ASIL(branch_{12}) = ASIL(branch_{21})
 \end{aligned}
 \tag{4.3}$$

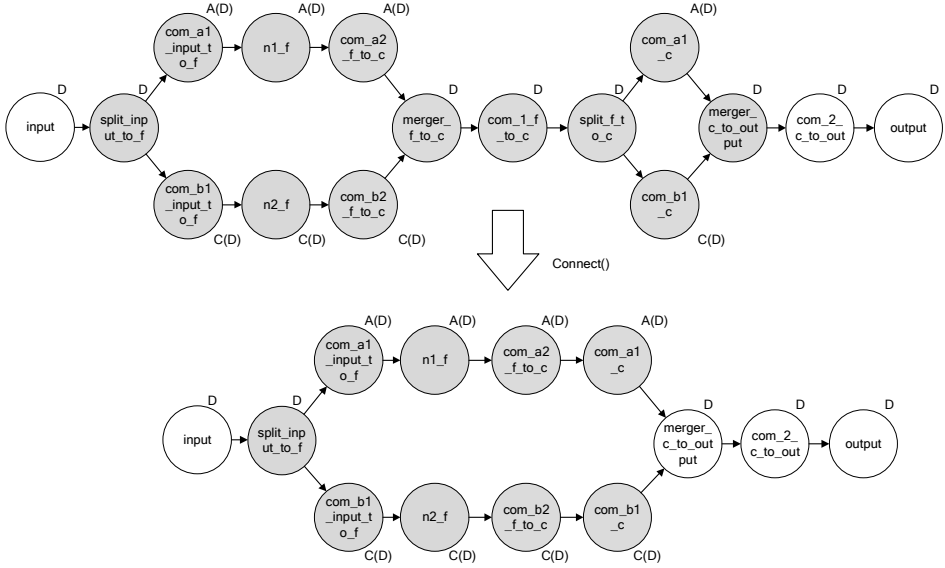


Figure 4.3: The connect transformation connects consecutive redundant patterns with functional-communication nodes.

The *connect* transformation consists in removing the middle nodes between two redundant patterns, the splitter-communication-merger single-point of failure part, and connect the compatible redundant branches, as shown in Figure 4.3. In the figure, we show the *connect* transformation applied to a functional redundant pattern followed by a communication redundant pattern. It is possible to apply the transformation also to communication-functional and communication-communication patterns, obtaining similar output graphs.

With fewer single-point of failure elements, the new graph has a different failure probability and cost. Each redundant block appears in the approximated fault tree as two single points of failure events, the failure of the merger and the failure of the splitter. Each of these events is formed by a combination of basic events, as shown in Chapter 3 in Section 3.4.1. The pre-transformation fault tree has five single-point of failure events, corresponding to the failure of the splitter and the merger for each of the two consecutive redundant patterns and the middle communication node. The post-transformation fault tree only has two single-point of failure events, as a single redundant pattern.



Figure 4.4: The connect transformation in the presence of multiple predecessors or successors nodes of the two redundant patterns.

The two redundant patterns can have additional predecessors or successors, each with a separate splitter or merger. In this case, those are not modified by the *connect* transformation, as shown in Figure 4.4.

Two-point failures and the connect transformation

From a single point of failure perspective, the transformed system benefits from the *connect* transformation since it has fewer events that contribute to the failure of the system. However, if we consider instead multiple-point faults, there is a significant difference between the pre-transformation and the post-transformation systems: in Figure 4.5a, the two faults can occur (at the same time), and the

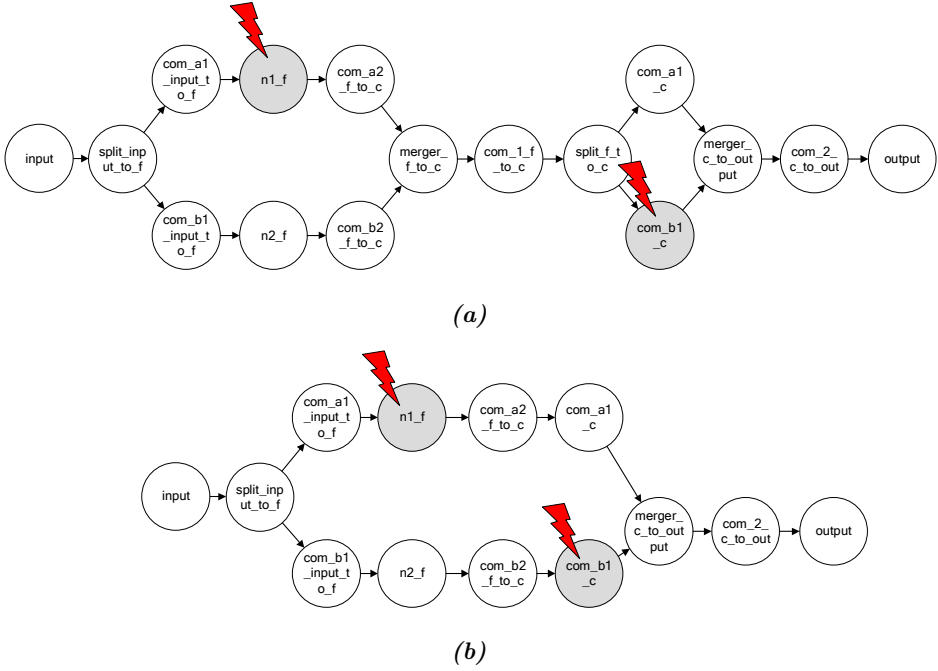


Figure 4.5: Two faults happening in a pre-transformed graph that do not lead to a system failure (a) and the same two faults in the post-transformed graph that lead to a system failure (b).

system will maintain its functionality because there is sufficient redundancy. After the transformation, as shown in Figure 4.5b, the occurrence of the same two faults causes the failure of the system, since both redundant branches fail. However, the probability of a simultaneous fault in both branches is negligible compared to single-point faults, as we have discussed in Section 3.4.3.

4.2.3 Transformation 3: separation of communication elements mapped to multiple resources or locations

The *separate* transformation is an auxiliary transformation used to separate the nodes or resources that are mapped to more than one resource or location into multiple communication elements. In an analysable graph, if a communication node is mapped to multiple resources, those are consecutive, meaning that they are either predecessors or successors of each other.

Communication nodes can be mapped over multiple consecutive resources to express a data transmission that travels over a complex network, and communication resources can be mapped over multiple locations, e.g. a communication

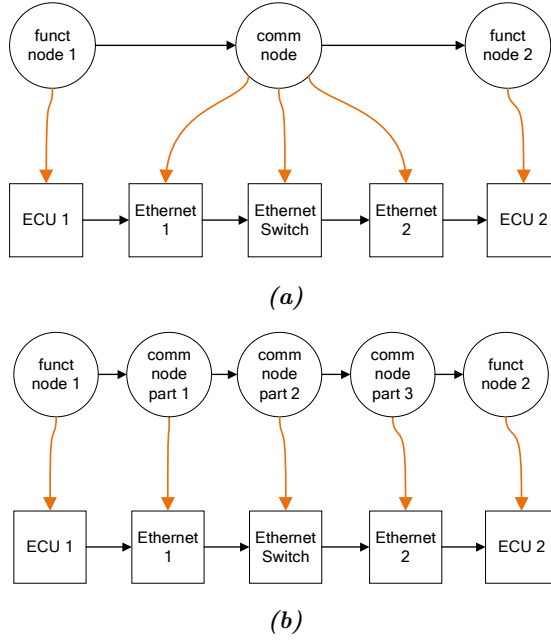


Figure 4.6: A communication node mapped to multiple resources (a) and the transformed application graph with the separate transformation (b).

bus. We generally use this transformation as an initial step before starting the transformation process described in the next section, to force a 1 to 1 mapping of the application layer to the resource layer and of the resource layer to the physical layer. This is beneficial because of two main reasons. First, we can select the individual part of a communication to apply substitution. Second, when applying a resource-oriented transformation process (Section 4.4), if the selected resource is part of the resources on which a communication node is mapped, only the correct part of the application node is consequentially substituted.

Figure 4.6a shows the pre-transformation graph of a communication node that is mapped to multiple resources. The communication node reaches the functional node mapped to ECU2 via an Ethernet switched network composed of the resources Ethernet 1, Ethernet Switch, and Ethernet 2. In Figure 4.6b we see the effect of the *separate* transformation, separating the node into multiple parts, one for each resource on which it is mapped. Part 1 is connected to the predecessor of the communication node, and part 3 is connected to the successor. The new communication nodes have the same properties as the original node.

4.2.4 Transformation 4: reduce the number of resources in the resource layer with optimized mapping

The *reduce* transformation reduces the number of resources necessary in the resource layer by collapsing the splitter resource with the predecessor communication and functional resources. The application nodes that were mapped to the three separate resources are now mapped to the single new resource. The *reduce* transformation can be similarly applied to the merger resource and the successive communication and functional resources. Figure 4.7b shows the results of the transformation applied to both sides of a redundancy pattern. Note that the new resources require multiple communication ports to interact with the redundant branches. In the pre-transformation system the splitter and the merger resources had multiple ports to interact with the redundant branches.

While the previous three transformations are necessary for a transformation process, the *reduce* transformation is an optimization one. We apply this transformation in our truck platooning lateral control application use-case of Section 4.5.2.

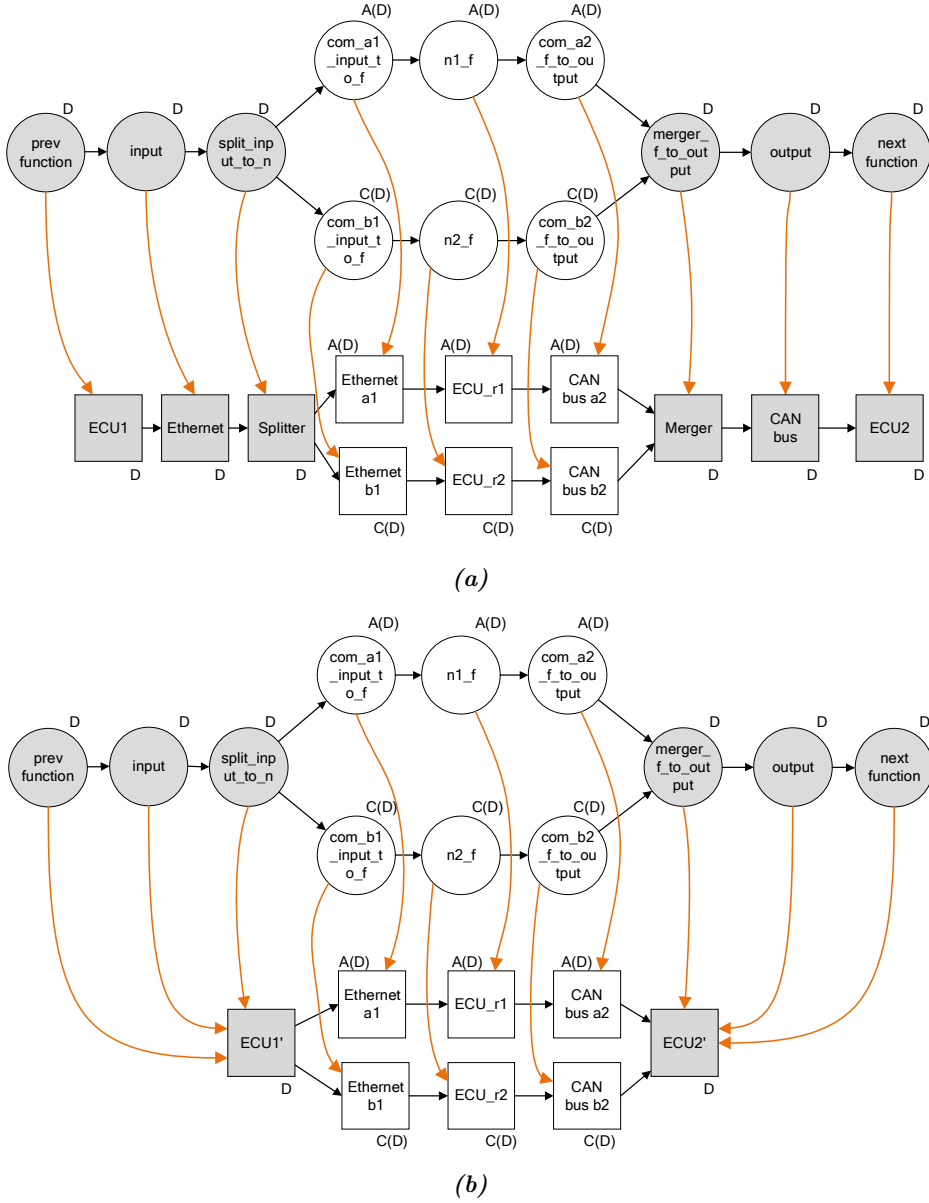


Figure 4.7: Part of an application graph and the mapping on the corresponding resources before the reduce transformation (a) and after the reduce transformation (b).

4.2.5 Transformation 5: collapse consecutive communication nodes

The *collapse* transformation is used when consecutive communication nodes are mapped to the same resource. In this case, there is no need for duplicate communication nodes, and the excess ones are removed.

This scenario can be an effect of the previous transformations. In particular, when *substituting* communication nodes, if the predecessor or the successor of the selected communication node is a communication node itself, the newly introduced communication nodes from point 2 and point 4 of the *communication node substitution rules* respectively are unnecessary. With the *collapse* transformation, they are removed from the graph, and the splitter or merger are connected directly to the predecessor or the successor of the node.

Another scenario in which we use the *collapse* transformation is after a *connect* transformation. The connect transformation generates a redundancy pattern which has consecutive communication nodes that correspond to the same data transport. With the *collapse* transformation we can remove these nodes to simplify the graph. Figure 4.8 shows the *collapse* transformation in this scenario.

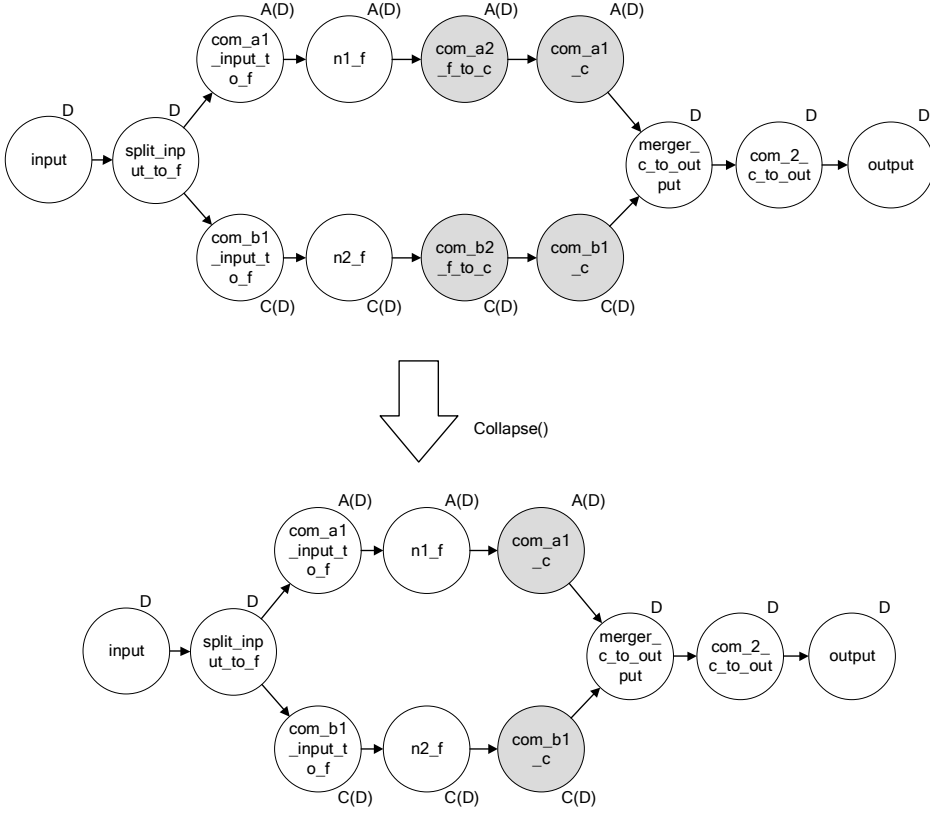


Figure 4.8: The application graph obtained after a connect transformation is transformed with the collapse transformation to remove the consecutive communication nodes.

4.3 Application-oriented transformation process

In this section, we describe the full transformation process that uses a series of the previously described transformations to introduce redundancy in a system, by selecting which part of the application shall become redundant. The process consists of selecting an application node to be *substituted* by redundant counterparts. The new application nodes must be mapped to the resource layer, and modifications to the resources may be necessary. With modifications of the resource layer, predecessors and successors of the node require substitution too. It is possible to modify the resource layer in various ways. We define three *strategies* to do so. Depending on the selected strategy, additional modifications to the other layers may be necessary. Figure 4.9 summarizes the transformation process, which

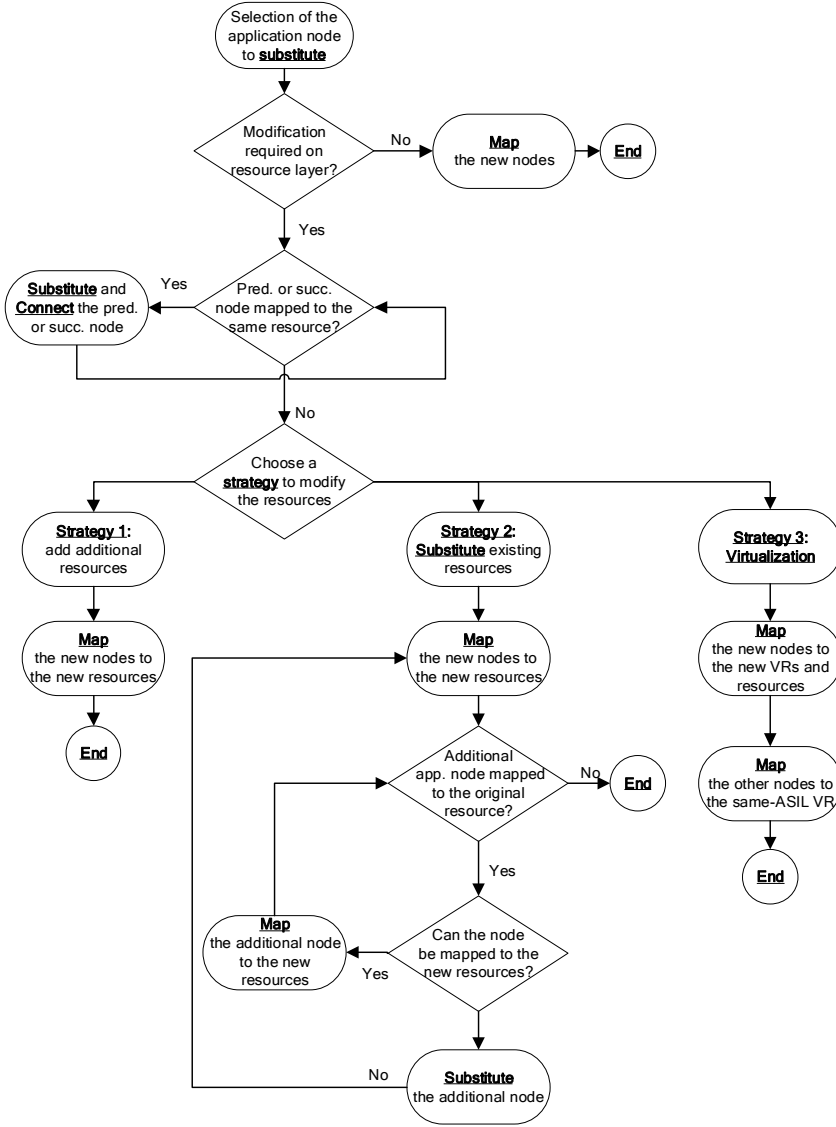


Figure 4.9: The transformation process to introduce redundancy from a selected application node.

generates from an analysable input graph an analysable output graph with redundancy in the selected application node and in the resources that concern it. Note that the individual application of the *substitute* or *connect* transformations does not generate an analysable output graph, as the nodes or resources require to be

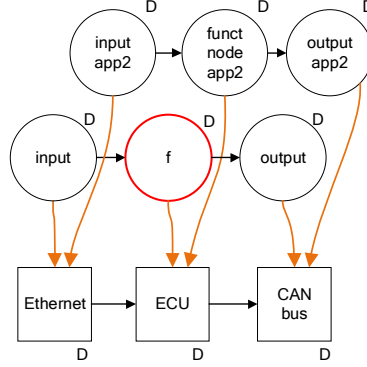


Figure 4.10: The node f is selected for substitution in this example.

mapped to the layer below and do not match yet with the graph characteristics described in Section 3.2.5.

4.3.1 Selection of the application node to substitute

The first step of the transformation process is the selection of the application node to *substitute*. We can select either a functional or a communication node, and the transformation described in Section 4.2.1 is applied to it. The resulting redundant nodes are not yet mapped, and the resource layer might require modification to accommodate the new nodes and to respect their ASIL and independence requirements. In the example shown in Figure 4.10 we select the functional node f for substitution. In this example, other nodes from a different application than the one node f is part of, are mapped on the same resource *ECU*.

If predecessors and successors of the selected node are mapped to the same resource, they are *substituted* and their redundant branches are *connected* to the ones of the selected node. These additional transformations allow the series of nodes to be remapped on the newly introduced resources without adding extra connectivity elements.

4.3.2 Modification of the resource layer and mapping of the new nodes

After the application layer has been modified with the application node substitution, the new application nodes must be mapped to the resource layer. We describe three alternative *strategies*. Depending on which strategy is used, the resource layer will be modified differently:

Strat. 1 (physical separation) For each new application node, a dedicated new resource is added in the resource layer. The new resources are connected

to the others so that the new application nodes can be reached by their predecessors and can reach their successors. A number of resources equal to the number of the application nodes introduced by the *substitution* are added to the system. As we discuss later in this section, this strategy requires the original resources to have additional communication ports to be connected to the newly introduced resources.

Strat. 2 (physical separation) The *substitute* transformation is applied to the resource the selected application node was mapped to. The new application nodes are mapped to the new resources. The other nodes that were mapped to the original resource require remapping as well since the original resource is not present anymore in the graph. Later in this section, we describe the additional details to remap the other application nodes.

Strat. 3 (virtualization) The resource on which the selected node is mapped is virtualized: the redundant parts of the application are then mapped to the Virtual Resources (VRs), while the splitters and mergers are mapped to external additional resources. The obtained pattern is similar to the one obtained with a *substitute* transformation applied to the resource (Strat. 2), but the main redundant part is formed by VRs. Other nodes are mapped to a third VR that keeps the original ASIL specification.

If the ASIL requirement of the selected node was lower than the ASIL specification of the resource it mapped to and the nodes are *fail-silent*, no modification of the resource layer is necessary, independently of the chosen strategy. All the new nodes are mapped to the original resource. The independence of the redundant nodes is justified by using a resource with higher ASIL specification than necessary in the original system, while the application nodes do not interfere with each other because they are *fail-silent*. Since the original resource already had an ASIL specification exceeding the application requirements, introducing redundancy does not provide any advantage in this case. In any other case, the resource layer is modified.

Strategy 1: additional resources

Figure 4.11 shows the results of the first strategy applied to the previous example (Figure 4.10). The node *f* is substituted, as in Figure 4.1. For each of the new nodes, a new resource is added to the resource layer. For example, for functional nodes, four communication resources, two functional resources, a splitter for each predecessor, and a merger for each successor node are added to the resource layer. The new splitters and mergers are then connected to the corresponding external resource. This adds a requirement of a free communication port on both the predecessor and successor resources, making the first strategy dependent on the system and resources used (e.g. if the predecessor resource is a point-to-point

communication cable, such as automotive Ethernet, it cannot be connected to multiple resources).

A direct mapping of the new nodes to the resource layer is possible, as the patterns in the application and the resource layers are identical. The original resource, *ECU*, still exists in the resource graph, and other nodes that were originally mapped to this resource, such as *funct node app2* are not affected by the transformation process.

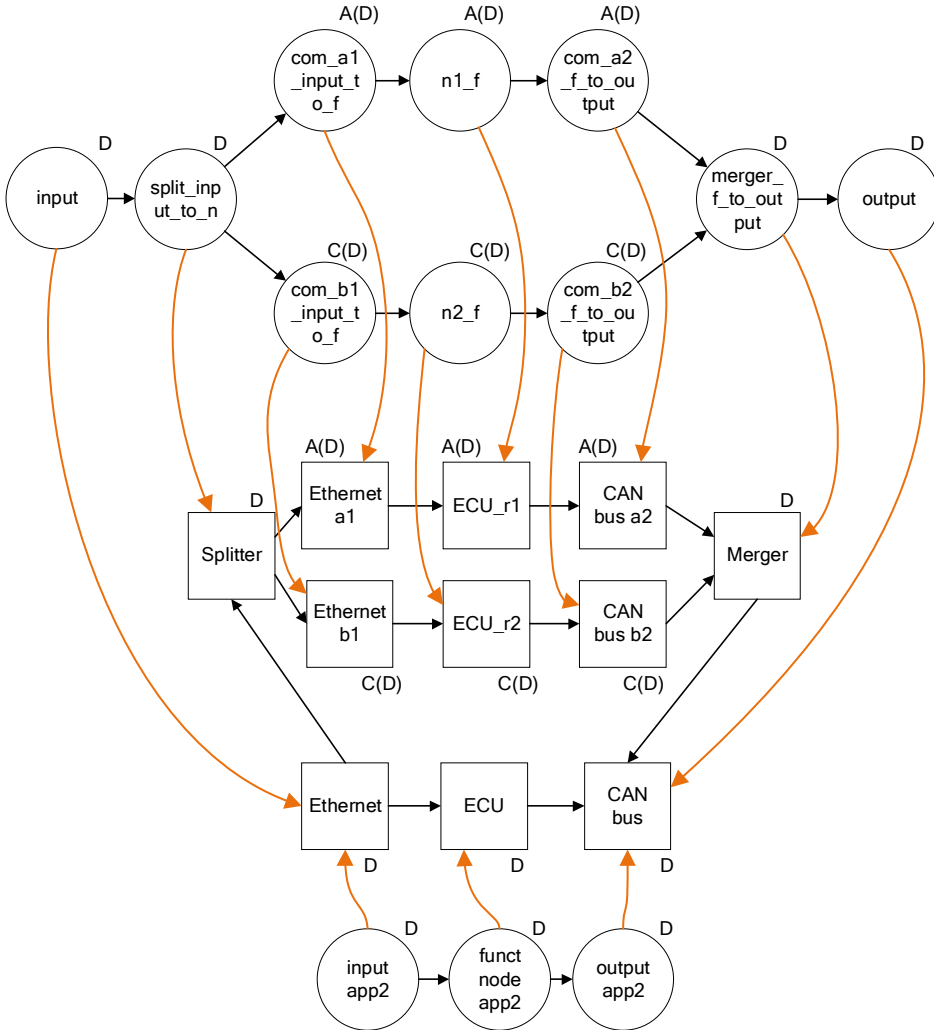


Figure 4.11: Strategy 1: Additional resources are added to the resource layer. For visual clarity, part of the application nodes are drawn below the resources.

Strategy 2: part 1 - Substitution of the resources

The second strategy uses fewer extra resources, at the expense of a more complex transformation and mapping process. For clarity, we divide this strategy into three parts: the first one is the *substitution* of the resource, the second and the third parts handle the additional nodes that were mapped to the original resource. Figure 4.12 shows an example of the second strategy applied to the previous example. The original resource *ECU* has in this case the same ASIL value as the selected node and must be substituted.

A direct mapping of the transformed nodes is possible. However, the original resource (in the example, *ECU*) is substituted by the two redundant resources, causing a problem for the other application nodes that were originally mapped to it. As indicated by the question mark in the figure, the other nodes such as *func node app2* need to be mapped to a new resource, but which one?

Figure 4.13 shows the decision process that is taken for each node that was originally mapped to the resource that has been substituted. We separate the additional nodes into two categories: the nodes that can be remapped without substitution (Case 1 in the figure, using one of the redundant resources) and the nodes that require substitution. (Case 2).

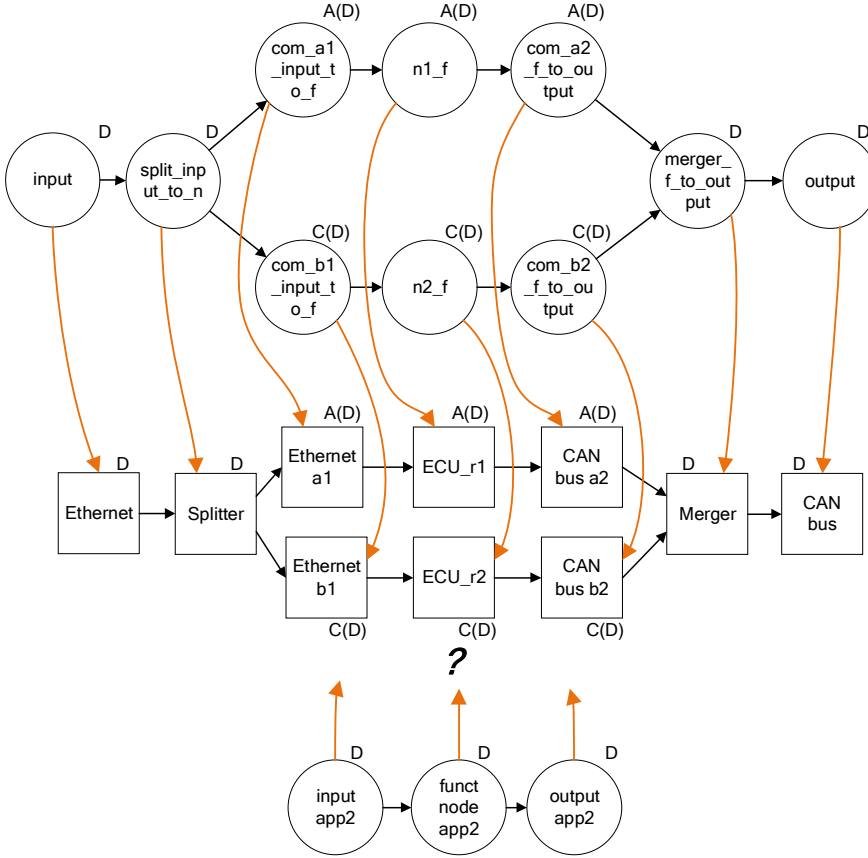


Figure 4.12: Strategy 2: Substitution of the original resource. Other nodes such as funct node app2 are not yet remapped.

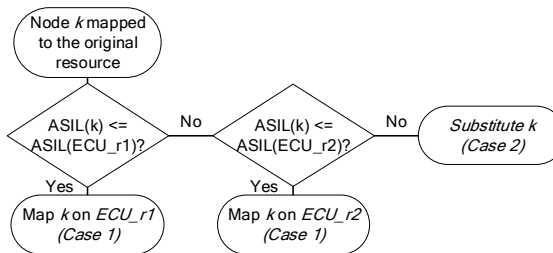


Figure 4.13: Flow chart to remap the additional nodes that were mapped to the original resource.

Strategy 2: part 2 - Remapping of additional nodes that do not require substitution

For each of the additional nodes that were mapped to the original resources, if their ASIL requirement is lower or equal than the one of the redundant resources, they do not require substitution. However, an additional remapping step shall be taken: the predecessors and successors of these nodes must be connected to them via the new splitter and merger resources. Two scenarios are possible:

1. The predecessor (or successor) of the node is mapped to a different resource.
2. The predecessor (or successor) of the node is mapped to the original resource.

In the first scenario: the predecessor and successor of the node, as in the example of Figure 4.14 with *input app2* and *output app2*, must be mapped to resources that can reach either *ECU_r1* or *ECU_r2*. In this scenario, we map the communication nodes to multiple resources: the *Splitter* resource becomes a *pass-through* for the communication node (in our model it requires the *communication* resource type other than the *splitter* type) and the communication resources internal to the redundant pattern are used to reach the redundant ECU. Symmetrically, the successor node uses the *Merger* as a *pass-through*. The final mapping configuration for this scenario is shown in Figure 4.14. The communication nodes mapped to multiple resources are then separated into parts with the *separation* transformation as discussed in Section 4.2.3.

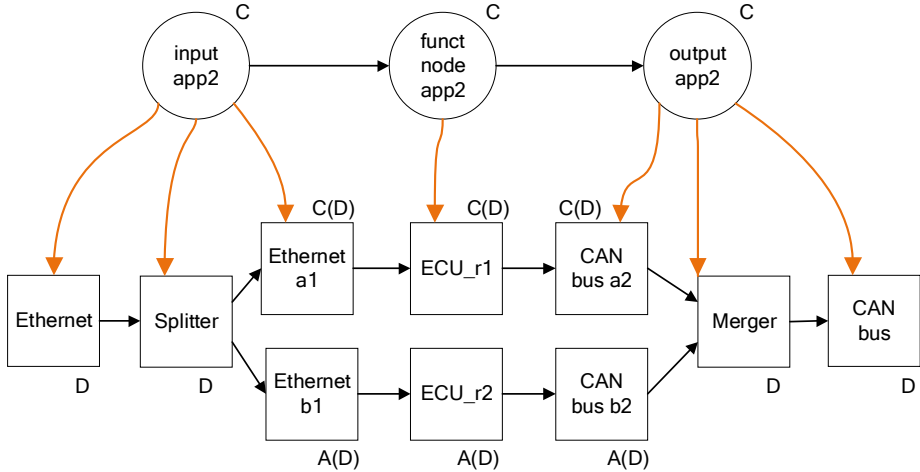


Figure 4.14: Remapping of the node *func_node_app2* and its predecessor and successor nodes to the new resources when the predecessors and successors are mapped to external resources (scenario 1).

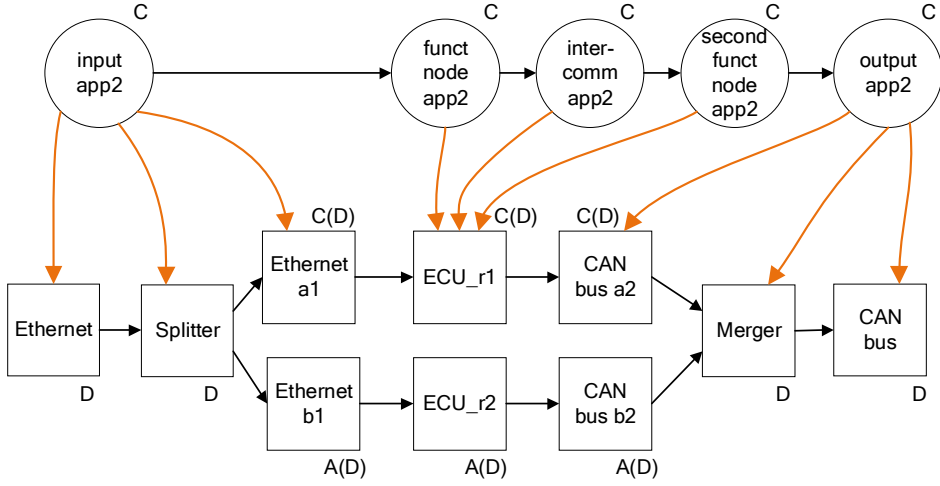


Figure 4.15: Remapping of the node *funct_node_app2* and its predecessor and successor nodes to the new resources (scenario 1 on the predecessor and scenario 2 on the successor).

In the second scenario: the predecessors and/or successors of the additional node were mapped to the same resource. They are mapped together with the current node (*funct node app2* in the example) to the same redundant resource. Figure 4.15 shows an example of this scenario, in which the nodes *inter-comm app2* and *second funct node app2* were mapped to the same original resource. The nodes *input app2* and *output app2* follow the same rules as the first case.

Strategy 2: part 3 - Remapping of additional nodes that require substitution

If the ASIL requirement of the additional nodes is higher than the ASIL specification of the new resources, they are *substituted*. Again, two scenarios are possible when remapping these nodes:

1. The predecessor and successor of the node are mapped to a different resource.
2. The predecessor and/or successor nodes of the current node are mapped to the original resource.

The first scenario is simple: the new redundant pattern is mapped directly to the new resources, as we have seen in the example in Figure 4.12 for the result of the substitution of the node *f*.

In the second scenario, the predecessor and/or successor nodes are *substituted* and then *connected* to the redundant pattern of the additional node. Finally, the

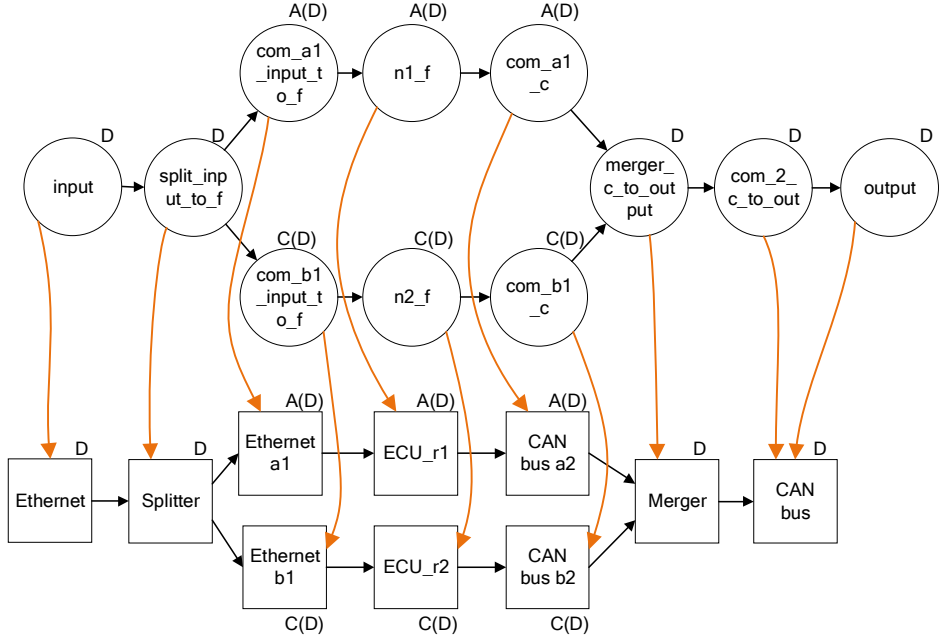


Figure 4.16: Two consecutive nodes are duplicated, connected, collapsed, and mapped to the new resources.

connected redundancy pattern can be mapped on the new resources. However, depending on the type of the nodes next to the splitter and the node next to the merger, additional steps may be required, as we have discussed in Figure 4.14.

When the resource has the functional and the communication types, we can find both communication or functional nodes next to the splitters and mergers. When a communication node is at one of the extremities of the redundancy pattern, the redundant communication nodes that are obtained must not be mapped to the functional resource, but to the communication resources that are connected to the splitter or merger, and then separated into parts with the *separate* transformation. In Figure 4.16 we see the result of the substitution of a functional node f and the consequent substitution of its neighbour node, the communication node c . The two redundant patterns are connected and mapped to the resulting resources.

Strategy 3: virtualization and remapping of application nodes

If virtualization is chosen over physical separation, the resource on which the node is mapped will, if not doing it already, run a virtualization mechanism such as a hypervisor. Two VRs are added to the resource layer, with a resource-resource dependency with the original resource. Their ASIL specifications correspond to

the ASIL values obtained with the *substitute* transformation in the application layer. The splitter and merger nodes are mapped to the new external splitter and merger resources, as described in Section 3.3.4, and connected to the VRs by new communication resources. Moreover, we assume that the resource has enough space for additional VRs, and we introduce another VR with the same ASIL specification as the original resource. In this way, fewer nodes will require substitution to be mapped to the new resource layer, similarly to the *Additional resources* strategy, where the original resource was kept.

Other nodes are mapped to the VR with the original ASIL specification and do not require modifications. Note that the original resource requires enough communication ports to connect to the new communication resources connected to the splitter and mergers. Figure 4.17 shows the results of the third strategy applied to the previous example.

Note that we virtualize only functional resources that can run a virtualization mechanism, while pure communication resources, such as Ethernet cables, must still be physically separated.

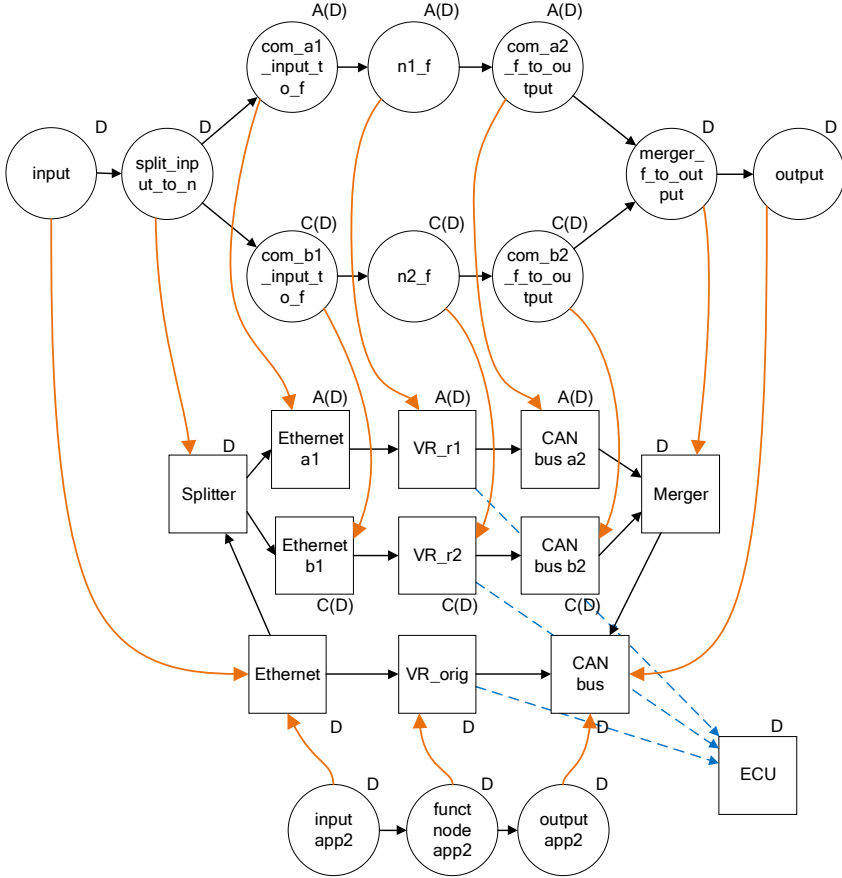


Figure 4.17: Strategy 3: VRs are added to the original resource, and the nodes are mapped accordingly.

4.3.3 Mapping of the new resources to the physical layer

When using physical separation strategies, the newly introduced resources must be mapped to the physical layer. We choose to minimize the distance between the resources by positioning them in the location in the physical space closest to the location of the connected resources. However, to maintain isolation and independence of the redundant resources, we introduce a constraint to map the resources belonging to each separate redundant branch in a different location, providing physical separation in the physical layer. For the virtualization strategy, this applies only to the external communication resources that connect to the splitters and mergers.

In our framework, the user only selects the node to substitute, the type of

the transformation, and the strategy to modify the resource layer. The tools will automatically generate the new application and resource graphs, and map the new application nodes and the new resources to the respective layer. By following the mapping steps, the redundant parts of the application have no Common-Cause Fault (CCF) sources with insufficient ASIL values, so they are considered independent for ASIL decomposition purposes.

Note that more remapping strategies can be implemented, for example, the tools can search the resource graph for suitable existing resources to map the new redundant nodes instead of inserting new ones. We use the three described strategies in the rest of this work to limit the mapping problem, as it is not our current focus to optimize it. The transformation process can be part of an automatic design-space exploration that optimizes the mapping of the new redundant application nodes, minimizing the cost and failure probability of the system. However, this is out of our current scope, and we leave a possible design-space exploration implementation for future work.

4.4 Resource-oriented transformation process

The system developer may want to introduce redundancy not on the application, but on the resource layer: for example, if a resource with an ASIL D specification is not available, the developer can choose to *substitute* it with ASIL A(D) and ASIL C(D) resources. With a variation of the transformation process, this is possible. If instead of selecting an application node, the user of our framework selects a resource, a different transformation process is used:

1. The *substitute* transformation is applied to the selected resource.
2. The application nodes mapped on the resource may require substitution as well, following the flow chart of Figure 4.13.
3. The new resources are mapped to physical locations, as described in Section 4.3.3.

This process follows the same steps as the previous ones, but consider all application nodes as *other nodes*. Moreover, variations of this process can be used to, for example, introduce VRs instead of physically separated resources. For the rest of this work, we use the application transformation process, while the proposed framework can be modified to use different processes depending on the needs of the developer.

4.5 Experiments

In this section, we assume that the application nodes are *fail-silent* and that the fault tree is generated with this assumption, as explained in Section 3.4.1.

We first provide a base scenario in which the application is analysed and then transformed and re-evaluated. Then we perform a complete analysis with multiple transformation processes on a complex application, the lateral control application of the EcoTwin truck platooning system, to discuss the results obtained in our automated framework compared to the manual design process performed by the designers of the original system.

Failure rates used in the experiments

In this section, we select a failure rate metric based on the ASIL value of each element. Table 4.2 shows the failure rates that are assigned to the basic events of the fault tree. The failure rates are related to the type of the basic event and the ASIL value associated with that event.

Table 4.2: Basic events failure rates (failures/hour).

Basic Event Type	QM	A	B	C	D
Location	10e-11				
Splitter/Merger/VR	10e-6	10e-7	10e-8	10e-9	10e-10
Other basic event	10e-5	10e-6	10e-7	10e-8	10e-9

In case of the failure of a location z (λ_{p-z}) we assign a value that is not related to ASIL requirements or specifications, and we assign to it a lower failure rate compared to the other event types. As mentioned in Chapter 3, this value is related to vibrations, temperature, and other environmental conditions. In this work, this type of failure condition is an important element for the CCF analysis to identify violations of the independence requirements of redundant nodes.

The failure rates λ_{a-x} and λ_{r-y} are instead associated with the ASIL value of the node x or resource y from which they are generated. In our experiments, we separate each ASIL level by an order of magnitude, to align with the ISO26262 standard (Part 5, Annex E) [81]. Moreover, we assume that splitter, merger, and virtualization are always aimed to improve the reliability and isolation of the system. Because of their safety-oriented purpose, we assign to these basic events a failure rate that is one order of magnitude lower compared to non-safety-oriented elements.

These values are used for the experiments of the following sections, but different metrics can be used in our model. The failure rates can be assigned individually to each basic event or a different metric can be used, for example, to differentiate different types of hardware resources.

4.5.1 Simple application example

The purpose of the first experiment is to describe the analysis and the full transformation process on a simple scenario, observing the changes in the fault tree and

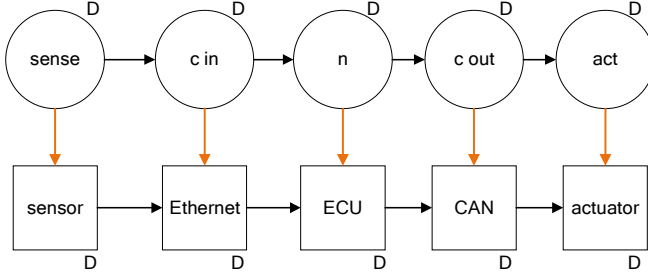


Figure 4.18: An example application mapped on a resource layer ($[D]$).

in the analysis parameters. For the experiments in this chapter, we assume that the nodes are fail-silent, meaning that a failing node will not interfere with others mapped to the same resource as discussed in Section 2.6.

The example system (application and resources) has a simple ASIL D application and is shown in Figure 4.18. The power supply and the physical layer are not shown for simplicity, but each resource is powered by the same ASIL D power supply and mapped to a different location. By using the above-mentioned failure rates, marked in this experiment as *Failure Probability (Reliable)*, the cost metric of Table 3.3, marked as *Cost (Cheap)*, and by assigning the coordinates to the locations (not shown in the figure), we obtain the results shown in Table 4.3 in the column $[D]$.

To see how the effects can vary depending on the failure rate and cost metrics that are selected, we modify our metrics by changing the values related to the splitter and merger elements. With the variation, we assign to the splitter and merger an *unreliable* failure rate value, and we obtain the parameter *Failure Probability (Unreliable)*. The unreliable failure rates metric gives the splitter and merger a failure rate 10 times higher than the original reliable metric, i.e. the same value as other ASIL D elements. In terms of cost, with the variation we assign to the splitter and merger resources a cost 10 times higher than in the previous metric, obtaining the parameter *Cost (Expensive)* as opposed to the previous value *Cost (Cheap)*. By combining the results, we can analyse the effect of using a lower cost but less reliable solution for splitter and merger resources or a more reliable but more expensive implementation. Note that in non-redundant scenarios, the different metrics have no effect since no splitters and mergers are present in the system.

Table 4.3: Analysis parameters calculated on the initial applications with different redundancy transformations applied.

	[D]	[B(D)+B(D)] type 1	[A(D)+C(D)] type 2
F. Probability (Rel.)	1.005e-08	8.440e-09	8.440e-09
Cost (Expensive)	290000	442600	453130
F. Probability (Unrel.)	1.005e-08	1.204e-08	1.204e-08
Cost (Cheap)	290000	262600	291190
Functional load	100	200	200
Communication load	70	210	210
Cable length	1.695	2.704	2.704

We then start the transformation process by selecting the node n for substitution. In the *substitute* transformation we use either the *type 1* or *type 2* transformations, obtaining the columns $[B(D) + B(D)]$ and $[A(D) + C(D)]$ in Table 4.3. The effects on the resources are obtained using the *Substitution of resources* strategy, described in Section 4.3.2, which substitutes the resources if necessary. After completing the transformation process, we observe that with non-reliable splitters and mergers (F. Probability Unreliable), the failure probability of the application increases with redundancy. This effect might be acceptable, e.g. the transformation reduces the cost of the system while failure probability is still below the maximum failure probability for the ASIL requirement of the system.

We also observe that the cost of the splitter and merger has a great significance in the final cost of the system by comparing the two cost parameters in the redundant scenarios. When using the expensive metric that increases their cost by 10 times, the cost of the system is almost doubled. Their impact on the cost is reduced when applying the *connect* transformation to reduce the number of consecutive redundant blocks. However, the *connect* transformation is not applicable in this small example since only one redundant block exists.

We then evaluate the example after adding two other applications. First, we add an ASIL B application, mapped on the same resources as the original one ($[D] + [B]$). The analysis of this scenario is shown in Table 4.4. Note that the shown failure probability in the tables is the one of the initial application, which, when assuming *fail-silent* nodes, is not influenced by additional applications. When evaluating this scenario before the transformation process, the only difference is found in the functional and communication loads, since the resource layer is not changed by the addition of other applications. When we apply the transformation process and select the node n , in both substitution types ($[B(D) + B(D)] + [B]$ and $[A(D) + C(D)] + [B]$) the nodes of the ASIL B application do not require a substitution, since they can always be mapped on part of the redundant resources, as discussed in Part 2 of the Strategy 2 description in Section 4.3.2. The resulting system is shown in Figure 4.19.

Table 4.4: Analysis parameters calculated on variations of the base example with redundancy in the main application and extra ASIL B application. Each application is indicated by its ASIL inside square brackets. The main application is transformed either with Type 1 ($[B(D)+B(D)]$), or with Type 2 transformations ($[A(D)+C(D)]$).

	[D]+[B]	[B(D)+B(D)] + [B] type 1	[A(D)+C(D)] + [B] type 2
F. Probability (Rel.)	1.005e-08	8.440e-09	8.440e-09
Cost (Expensive)	290000	442600	453130
F. Probability (Unrel.)	1.005e-08	1.204e-08	1.204e-08
Cost (Cheap)	290000	262600	291190
Functional load	140	240	240
Communication load	220	410	410
Cable length	1.695	2.704	2.704

Table 4.5: Analysis parameters calculated on variations of the base example with redundancy in the main application and extra ASIL C application. Each application is indicated by its ASIL inside square brackets. The main application is transformed either with Type 1 ($[B(D)+B(D)]$), or with Type 2 transformations ($[A(D)+C(D)]$).

	[D]+[C]	[B(D)+B(D)] + [C] type 1	[A(D)+C(D)] + [C] type 2
F. Probability (Rel.)	1.005e-08	8.440e-09	8.440e-09
Cost (Expensive)	1.005e-08	442600	453130
F. Probability (Unrel.)	290000	1.204e-08	1.204e-08
Cost (Cheap)	290000	262600	291190
Functional load	140	280	240
Communication load	220	660	410
Cable length	1.695	2.704	2.704

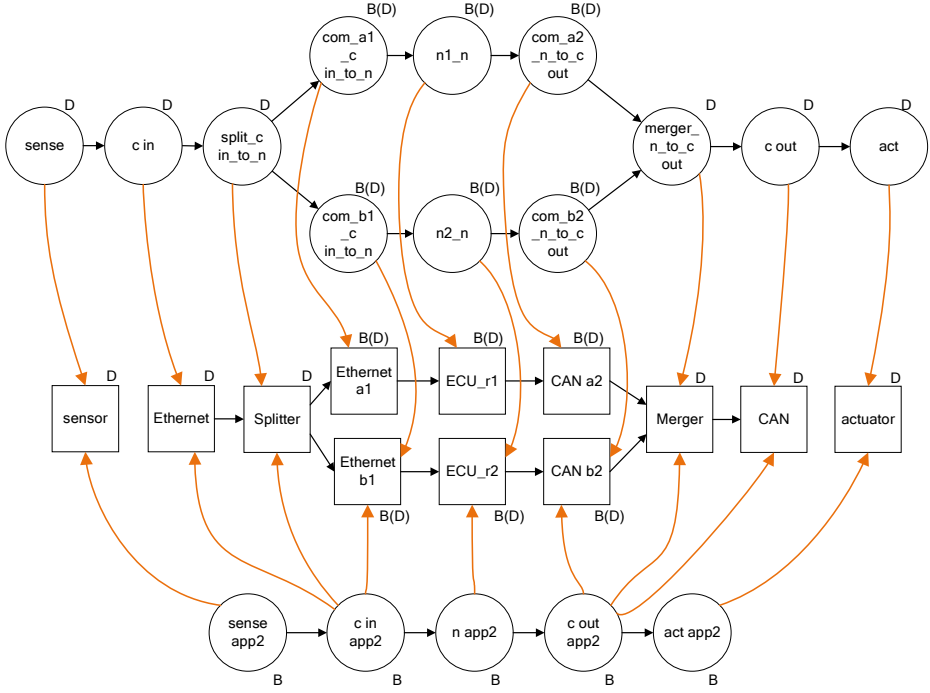


Figure 4.19: The transformed example application mapped to the substituted resource layer together with the extra ASIL B application after using type 1 substitution ($[B(D)+B(D)]+[C]$). The ASIL B application does not require substitutions.

In the last version, we add instead of the ASIL B application, an additional ASIL C one ($[D] + [C]$). The analysis of this scenario is shown in Table 4.5. As seen in Case 3 of Section 4.3.2, the ASIL C nodes mapped on the same resource as the node that is being substituted may require substitution to be remapped on the new resource layer: in *type 1* substitutions ($[B(D) + B(D)] + [C]$) no ASIL C or higher resource is available, while in *type 2* ($[A(D) + C(D)] + [C]$) the additional application is remapped on the ASIL C(D) part of the resource layer. The result of the transformation process when using *type 1* substitution is shown in Figure 4.20, where the additional application nodes are substituted as well. In the $[B(D) + B(D)] + [C]$ scenario we observe higher increment of the functional and communication loads compared to $[A(D) + C(D)] + [C]$, since every node that was originally mapped on *ECU* has been substituted. The conclusion to be drawn from these two cases is that the type of decomposition used influences the system parameters based on which applications are present on the modified resources. The choice of the substitution type is even more relevant in complex systems,

in which multiple applications share the same resources. The correct choice can minimize the increment of the functional and communication loads, and ultimately the cost of the system: higher functional and communication loads translate to higher requirements for the resources, or to requiring more resources to perform the communication or the computation of the nodes.

By adapting the metrics to the system under evaluation, with our framework, we can quantitatively express the trade-offs between redundant and non-redundant implementations, and identify the best choices to implement redundant parts for the ASIL decomposition rules.

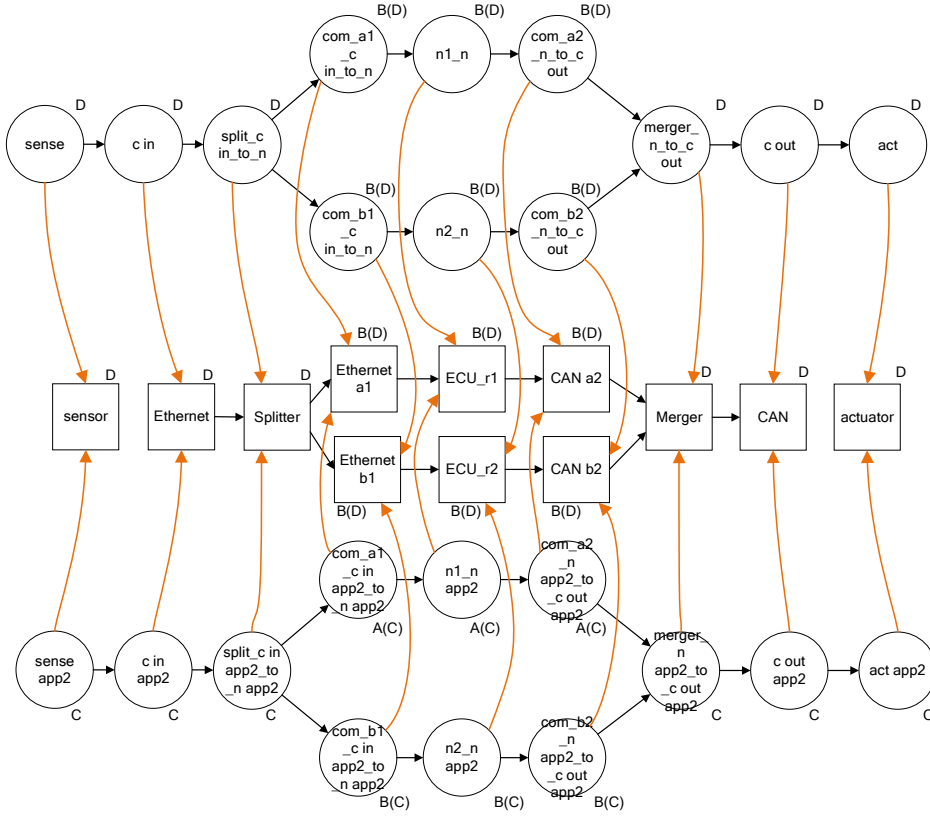


Figure 4.20: The transformed example application mapped to the substituted resource layer together with the extra ASIL C application, after using type 1 substitution ($[B(D)+B(D)]+[C]$). The ASIL C application requires substitution.

The effect of the transformation process on the fault tree

The fault tree for the base version of the initial example is shown in Figure 4.21. For visual clarity, the power supply part is hidden. The circled area represents the part that is modified in the transformation process, and in particular, in Figure 4.22, we observe the fault tree part related to the redundant nodes in the transformed version, while Figure 4.23 shows the approximated fault tree.

In this simple example, we can analyse the effect on the estimated failure probability due to the fault tree approximation described in Section 3.4.3. In the approximated fault tree, for the system to function correctly, no basic event can happen: the redundant parts are approximated, and the remaining events are all the single point of failure events of the system.

In the non-approximated fault tree, the events related to the splitter and the merger contribute directly to the system failure probability, while the failure probabilities of the redundant nodes are multiplied between the two redundant branches. With the failure rates that we are using, mentioned in Section 4.5, the failures related to the redundant part of the application are in the order of 10^{-14} (two times the ASIL B failure rate of each of the two redundant branches), while the events related to the splitter and merger have a failure rate of 10^{-10} . With these numbers, approximating the fault tree corresponds to an underestimation of the failure probability of 0.01%. The absolute value of the underestimation is related to all the redundancy patterns present in the application graph, to the number of nodes present in each redundant branch, and to the failure rates related to the base events connected to the failure of these nodes, as seen in Section 3.4.3.

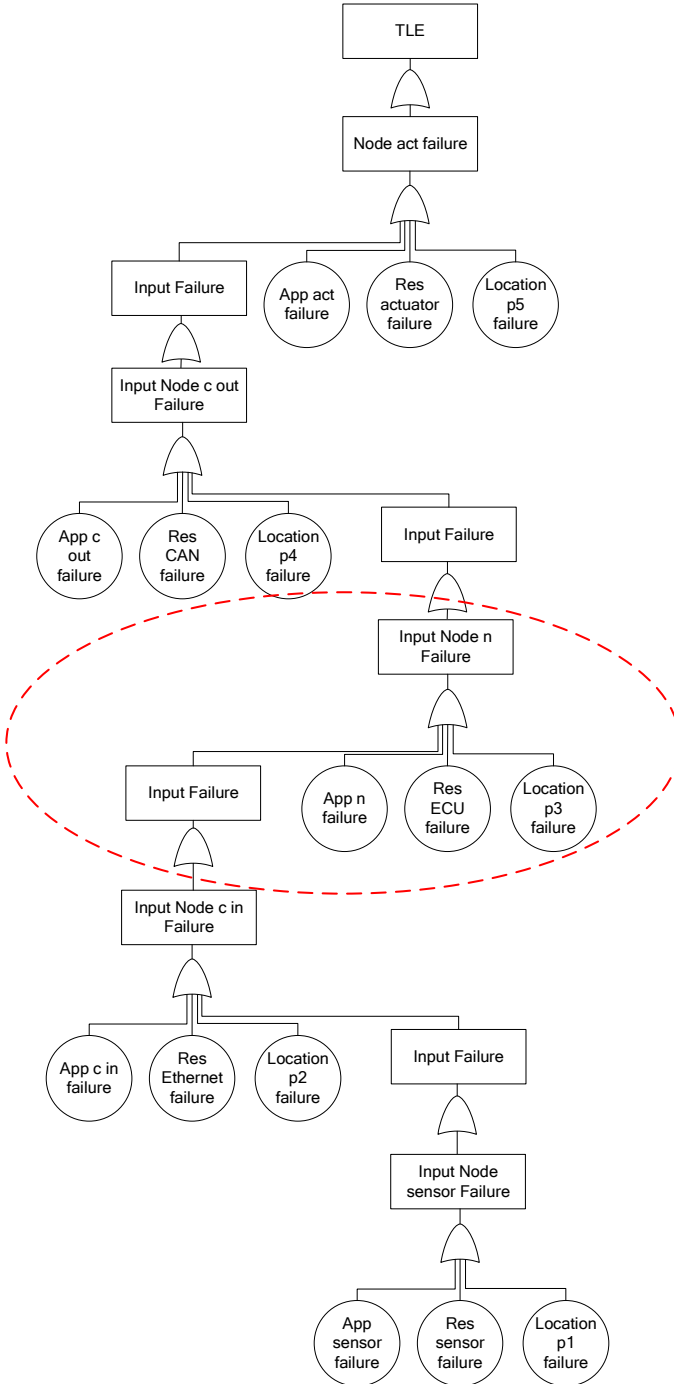


Figure 4.21: The fault tree generated from the simple example application of Figure 4.18. For visual clarity, the power supply failures are hidden.

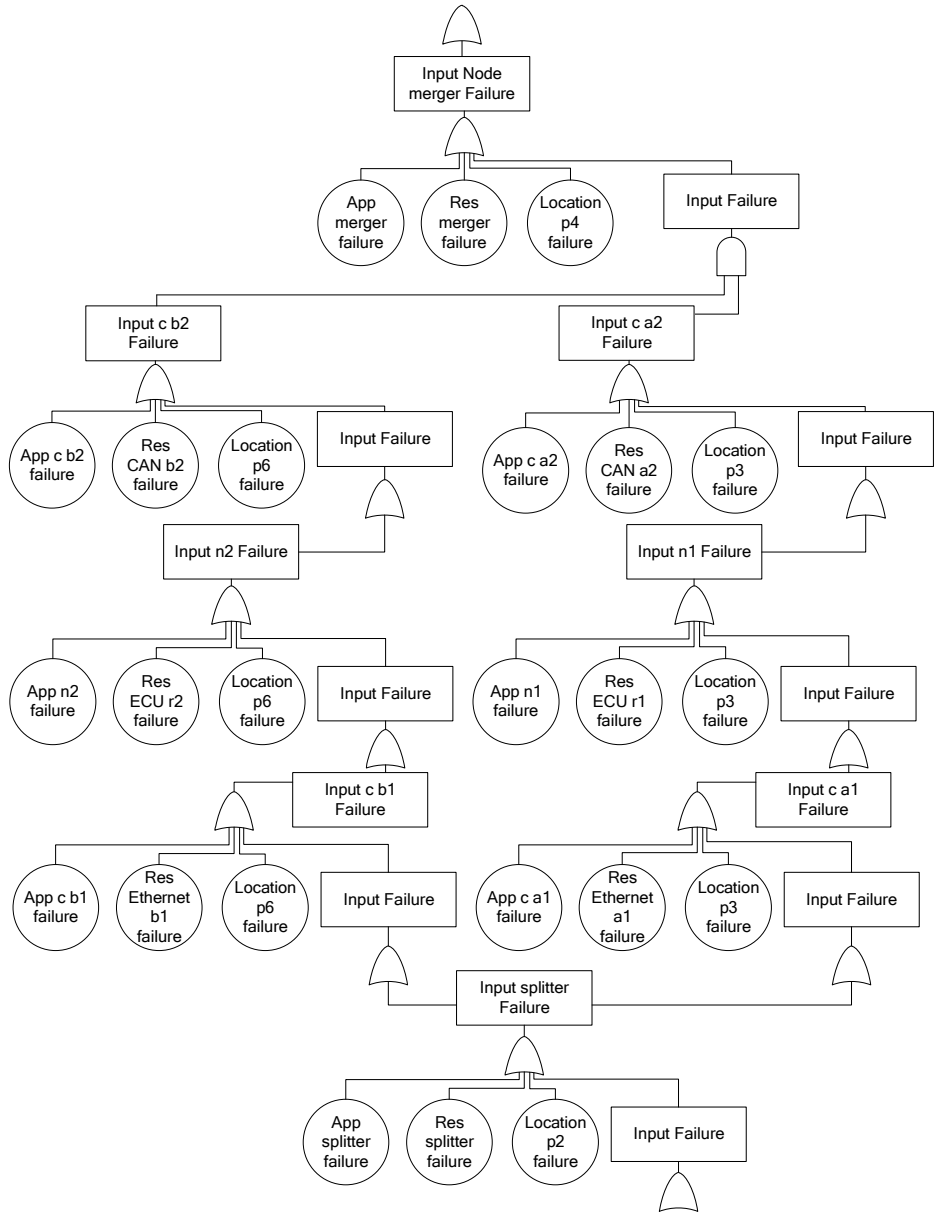


Figure 4.22: The redundant part of the duplicated example application in the fault tree. For visual clarity, the power supply failures are hidden.

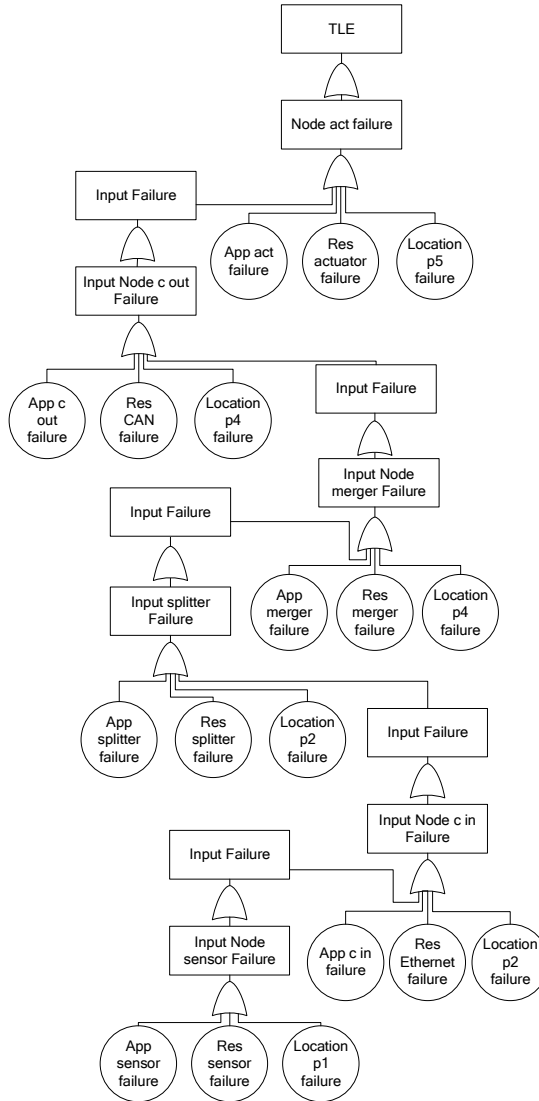


Figure 4.23: The approximated fault tree for the redundant example application. For visual clarity, the power supply failures are hidden.

4.5.2 Use case: ECOTWIN II platooning system

In this section, we apply the analysis and transformations to the lateral control of the truck platooning project of the EcoTwin consortium [23], a collaboration between NXP, TNO, DAF, and Ricardo. In a truck platoon trucks autonomously follow a primary vehicle with a short distance, allowing significant fuel savings. The EcoTwin project focused on the development of a SAE level 2 autonomous system architecture by using a safety executive pattern, in which two channels, a main and a safety backup, work in parallel and are monitored by a health management system.

The starting point of the truck platooning project consisted of a non-redundant system that did not satisfy the functional safety requirements because of the non-availability of ASIL-D-ready resources to execute the self-driving algorithms. From this starting point, a redundant system that uses the safety executive pattern was designed, meeting the ASIL D requirements via ASIL decomposition on the redundant implementation. We retrace the manual design process that creates the redundant architecture starting from the non-redundant system with our automated framework. We analyse the architecture providing cost and reliability information at each consecutive redundancy step. Our framework's input is an analysable graph, which models the ideal system where ASIL D resources are available. Our model transformations automate the previously manual steps and evaluate the cost and reliability of each step. This example is presented in our publication [56], and we report our results in the following sections.

System model

The truck platooning system is a combination of cooperative Adaptive Cruise Control (ACC) and a lane-keeping assist system. The first system controls the headway time between the following truck and the leading truck, while the second system keeps the following truck between the lane markings on the road. In addition, safety modules monitor the state of the system. In Figure 4.24 the functional architecture of the truck platooning system is shown.

We focus our analysis on the lateral control application and its safety monitor, which are used in the lane-keeping assist system. The analysis can be extended to the full truck platooning system. The application graph in Figure 4.25 represents the initial lateral control application. Partial redundancy is present in the sensing part of the application: virtual splitters are used when multiple sensors are acquiring information about the same objects, which are then used in data fusion nodes. However, a single non-redundant path analyses sensors data to create a world model and provides the steering control signal for the steering actuator. This part of the application is safety-critical and requires redundancy to satisfy the ASIL requirements. In our step-by-step analysis, we select each node that is part of the world modelling and vehicle control parts for substitution, highlighted

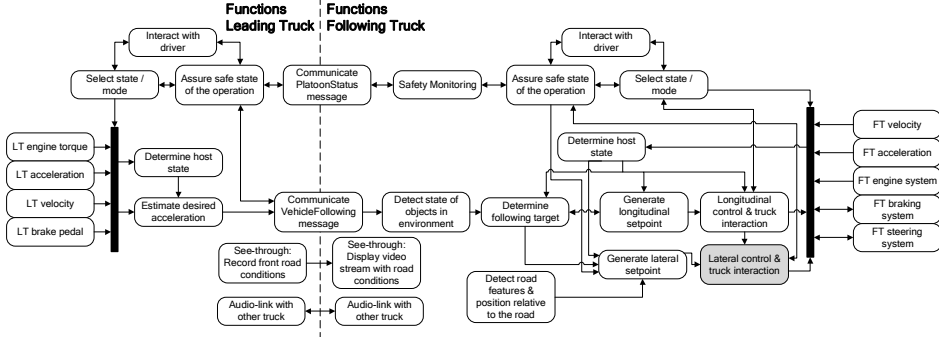


Figure 4.24: EcoTwin truck platooning system functional architecture, adapted from [24]. Highlighted, the Lateral control application modelled in our experiments.

in blue in Figure 4.25. We analyse the cost versus failure probability graph for each transformation step.

Transformation steps to introduce redundancy

The transformation process is applied sequentially to each of the highlighted nodes in the application graph of Figure 4.25. After each transformation step, a new system is generated that includes redundancy in the selected part. The new system is evaluated with the framework to obtain the new cost and failure probability. After the system is fully redundant, the *connect* and *collapse* transformations are applied to reduce the number of splitter, merger, and communication nodes.

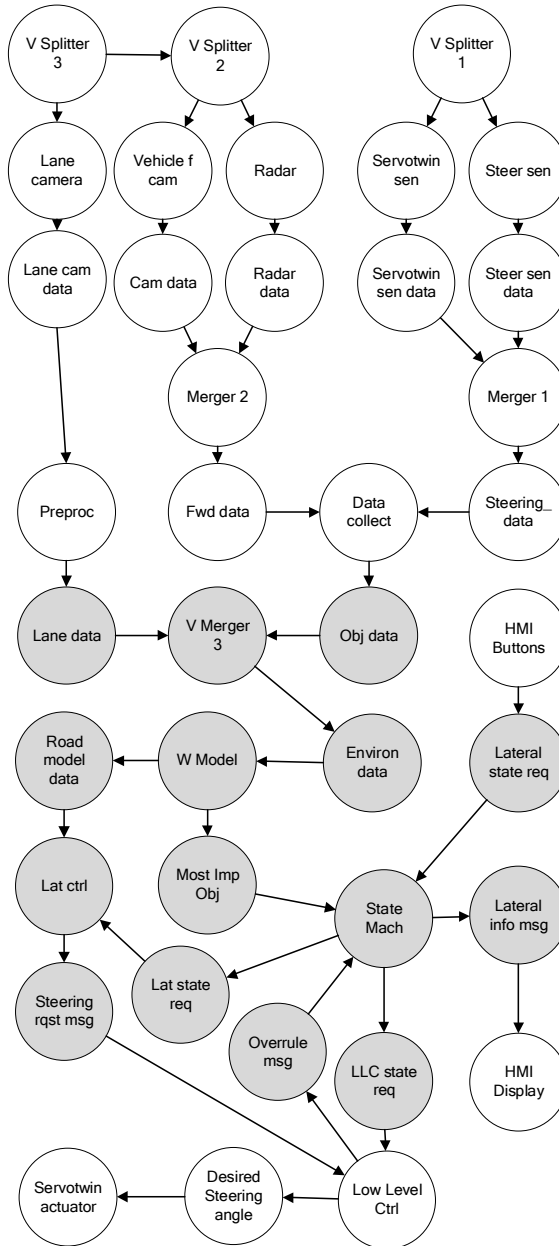


Figure 4.25: Initial EcoTwin lateral control application graph. The nodes that are substituted in the experiments are highlighted.

Analysis results

The transformation steps that introduce redundancy in the lateral control application of the truck platooning system transform the application of Figure 4.25 into the redundant one of Figure 4.26. First, all the highlighted nodes are substituted and remapped to new resources. After all these nodes are substituted, the *connect* transformation is used to connect consecutive redundant application nodes and resources. Finally, the *reduce* transformation is used to improve the final mapping of the application. In this experiment, we use the *Additional resources* strategy to remap the new nodes, which adds a new resource for each new application node. Moreover, we use *type 1* substitutions, which for example transform an ASIL D requirement into ASIL B(D) redundant ones.

After each transformation, the system is re-evaluated and the five analysis parameters are re-calculated. In this section, we focus on the cost and failure probability of the application.

Figure 4.27 shows the results of the analysis for each transformation step. Each substitution adds to the cost and the failure probability of the system, despite the splitter and mergers having lower failure rates and lower costs compared to other ASIL D resources. After applying the *connect* transformation to the application and resource layers, the cost and failure probability of the system are lowered. Finally the *reduce* transformation reaches the point *C* in the figure.

The obtained cost and failure probability are higher than the original values, however, the system now uses specific safety-oriented resources to perform the split and merge operation, while the main part of the application is executed redundantly in resources with at most ASIL B specification. Despite the slightly higher parameters, we have an advantage in terms of the availability of such resources. In particular, it is difficult to have general-purpose ASIL D resources that can provide high enough computing performance for self-driving applications. On the other hand, safety-oriented resources that deal with specific tasks are easier to develop and certify at a high ASIL specification. That is the case in the truck platooning prototype, in which redundancy patterns were necessary to achieve the ASIL requirements obtained from the Hazard Assessment and Risk Analysis (HARA) [23].

Figures 4.28a and 4.28b show the number of elements in the generated fault trees after each substitution without and with the approximation respectively. On average, each transformation adds 20 new basic events without the approximation and 7 with it. In the final system, point *C* in Figure 4.27, the non-approximated fault tree has 288 basic events and the approximated fault tree has 159 basic events. In Appendix C we show the graphs and the fault trees generated with our framework.

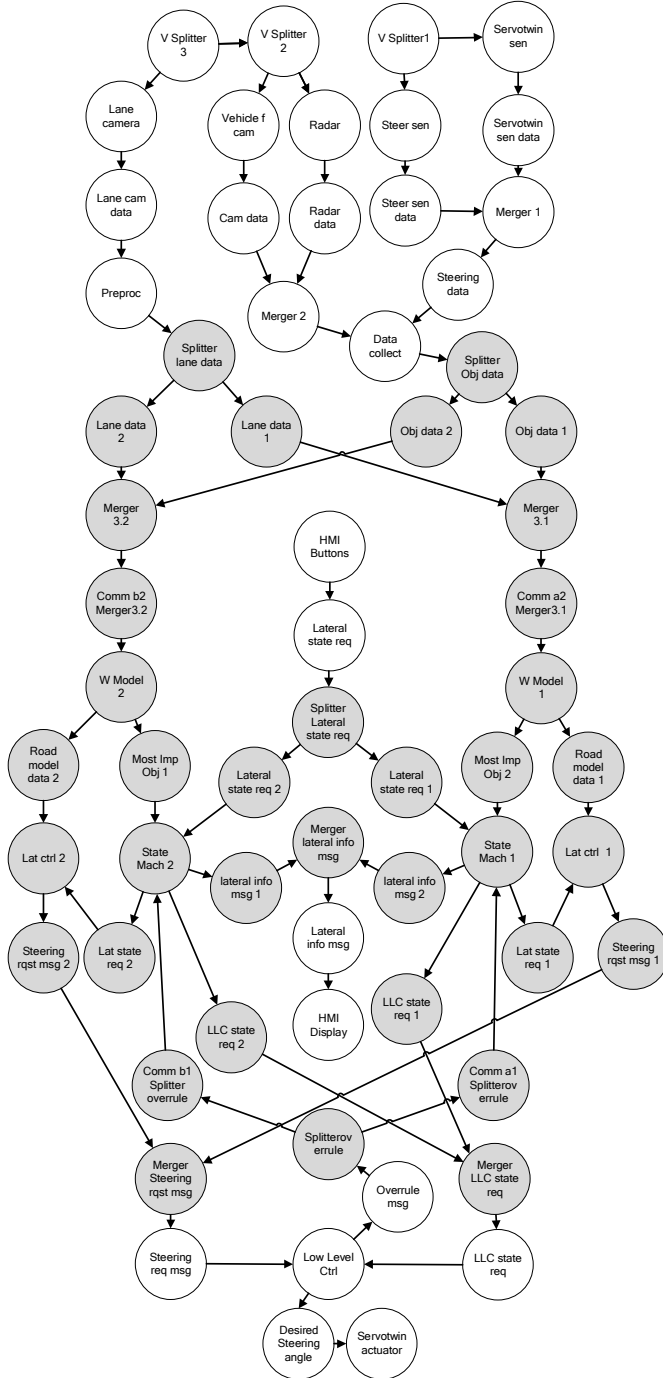


Figure 4.26: Redundant EcoTwin lateral control application graph. Highlighted, the transformed part of the application graph.

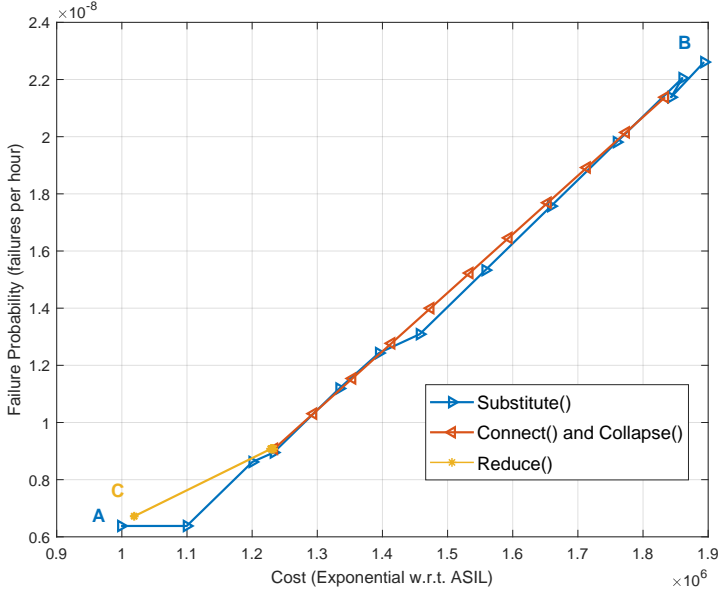


Figure 4.27: Step by step cost vs failure probability analysis of the transformations applied to the modelled lateral controller application.

The engineers that designed the original platooning system manually modified and evaluated the system to be redundant and ASIL D compliant [23]. With our framework instead, we introduce redundancy semi-automatically by manually selecting parts of the system and automatically transforming them, with step-by-step re-evaluations of the system. The only difference in the system model between the manually obtained system and the one generated with our transformations is that in the original project the merge operation is performed by a health monitoring subsystem, which controls an *arbiter* to select either the nominal or the safety channel. In our model, this operation is performed by a merger that uses Side-Band information, as described in Chapter 3. We can modify the *substitute* transformation to obtain the same result as the original project: by introducing an Out-Band merger instead of a Side-Band one, and by adding the health status of the nodes in the redundant branches as out-band data that the merger uses for its decision.

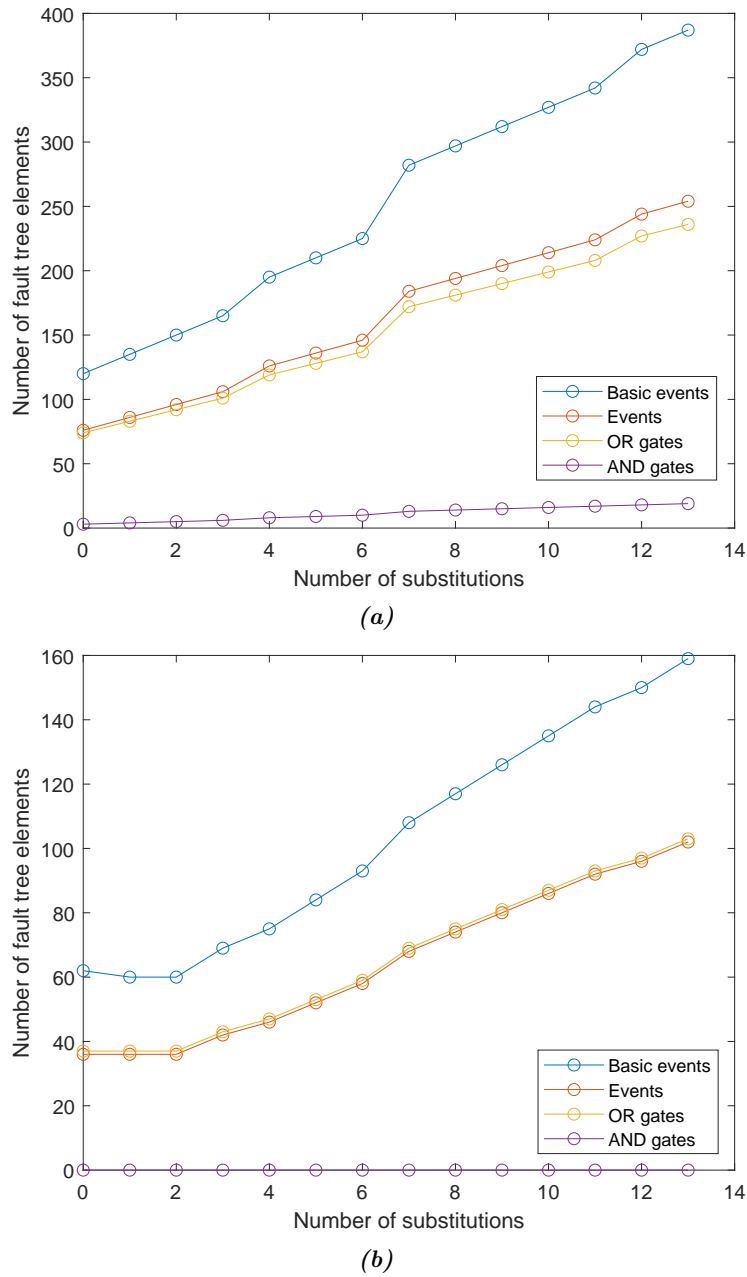


Figure 4.28: The number of elements in the fault tree after each substitution transformation without approximation (a) and with approximation of the tree (b).

Table 4.6: *Alternative cost metric 1: ASIL separation.*

Resource Type	QM	A	B	C	D
Functional	5	50	500	5000	50000
Communication	4	40	400	4000	40000
Sensor / Actuator	8	80	800	8000	80000
Splitter / Merger	1	10	100	1000	10000

Table 4.7: *Alternative cost metric 2: expensive functional resources.*

Resource Type	QM	A	B	C	D
Functional	8	80	800	8000	80000
Communication	2	20	200	2000	20000
Sensor / Actuator	1	10	100	1000	10000
Splitter / Merger	3	30	300	3000	30000

Variations of the substitution transformation types and cost metrics

In the previous experiment, we showed how our framework can introduce redundancy and analyse a real use-case scenario. However, we have only used a limited part of the framework functionality. In the next experiments, we vary cost metrics and use different types of substitution transformations. Tables 4.6, 4.7, and 4.8 show the cost metric alternatives that we use. In the first one, we separate the cost for ASIL A and B and ASIL C and D, but we leave the B and D values the same as the original cost metric (Table 3.3). In the second one, functional resources are more expensive compared to the first, while communication, splitter, merger, sensor, and actuator resources are cheaper. In the third alternative, functional, splitter, and merger resources are cheaper compared to the first, while communication resources are more expensive.

Moreover, we use the *type 2* substitution as well. While with *type 1* substitutions an ASIL D requirement is transformed in two ASIL B(D) ones, with *type 2* it is transformed into an ASIL C(D) and an ASIL A(D) requirements. We can apply the two types consistently throughout all the substitutions or mix the two types. In Figure 4.29, the lines marked as *BB* use *type 1* substitutions, *AC* use

Table 4.8: *Alternative cost metric 3: expensive communication resources.*

Resource Type	QM	A	B	C	D
Functional	3	30	300	3000	30000
Communication	8	80	800	8000	80000
Sensor / Actuator	8	80	800	8000	80000
Splitter / Merger	2	20	200	2000	20000

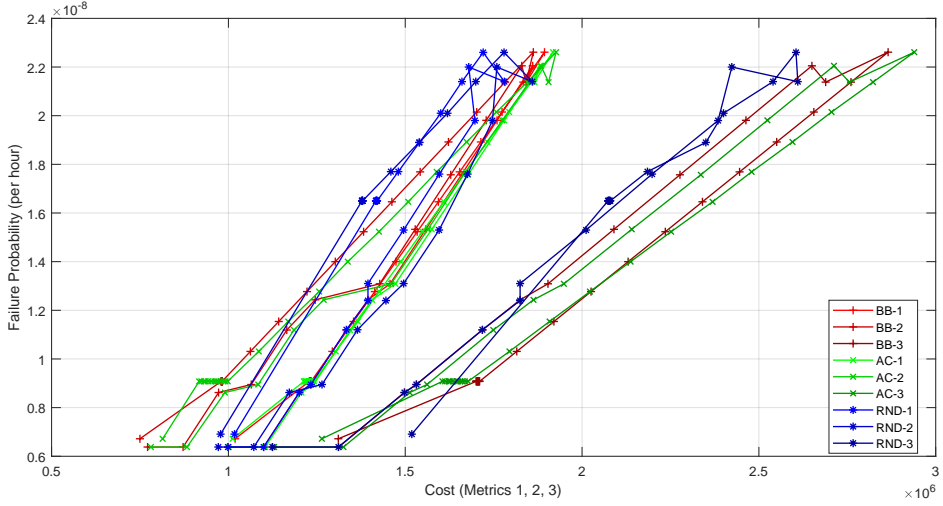


Figure 4.29: Step by step cost vs failure probability analysis of the transformations applied to the modelled lateral controller application.

type 2 substitutions, while *RND* randomly selects between the two types at each substitution step. The order in which the nodes are selected for the substitution is not important, and even if *connect* and *substitute* transformations are interleaved, the final result would be the same. This is because each selected node is mapped on a different resource in this use-case, and it is not modified by the previous transformations.

We observe how the cost metric 3, with expensive communication resources and sensors/actuators, increases the overall cost of the system the most. This result is expected, as the number of communication resources (8), sensors (15), and actuators (9) is higher than the number of functional resources (7). Moreover, as discussed in Section 4.2.1, the substitution of a communication element generates communication elements predecessor of the splitter and successor of the merger with the original ASIL value. This means that even in the transformed system, part of the communication still requires ASIL D resources. With the *reduce* transformation we limit this problem, using as ASIL D resource the already existing resource in which the predecessor (or successor) nodes are mapped. However, since this requires that the existing resource has *communication* and *splitter* or *merger* capabilities, the transformation is not always possible.

Varying the type of substitution instead has different effects: if *type 1* or *type 2* substitutions are applied consistently, the *connect* transformation is applied to all consecutive redundant patterns. If the type of substitution is chosen inconsistently, the ASIL values of the redundant branches in consecutive patterns do not always match. The *connect* transformation is then not always possible. In this use-

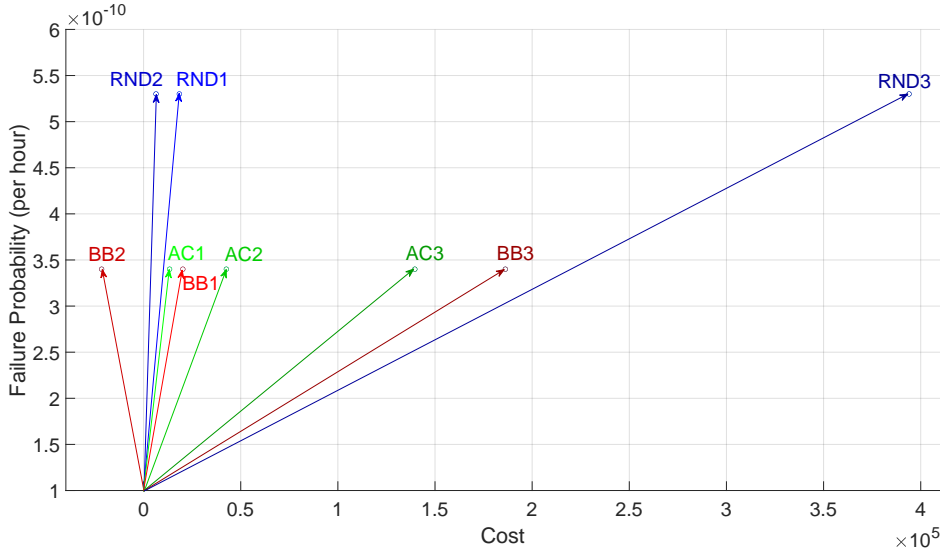


Figure 4.30: Variation in cost and failure probability after the full transformation process in each experiment.

case, we are considering a single application, and so transforming the system with inconsistent substitution types does not have any positive outcome (e.g. not requiring substitution of additional nodes mapped on the same resources).

Note that the *BB-1* curve corresponds to the one discussed in the previous section, as the cost values for ASIL B and D resources does not vary compared to the original cost metric.

Figure 4.30 shows the cost and failure probability variations of each setup from a normalized initial position to the final position at the end of the transformation process. We observe how the randomized substitution types lead to higher failure probabilities since not all redundant parts can be connected. Only the *BB2* combination, which has a high cost for functional resources and a low cost for communication, splitter, and merger resources, leads to a reduction in cost. The second cost metric works well when using *type 1* substitutions, but in case of *type 2* (*AC2*) the additional cost of ASIL C expensive functional resources still leads to a higher cost compared to the starting point.

4.6 Discussion of the transformation process

To conclude the chapter, we discuss some final aspects of our proposed transformation process:

- **Completeness:** as seen in the EcoTwin truck platooning use case, the proposed transformation process does not reach the same solution as the manual design. In particular, the manual design uses a safety-executive redundancy pattern, with a health-monitoring system that sends out-band information to the merger, as discussed previously in Section 3.3.3. In our case instead, we use a heterogeneous redundancy pattern, with an in-band data merger. However, with variations of the *substitute* transformation, we can introduce different nodes, e.g. the health-monitor or a third redundant branch for triple-modular redundancy, and implement the different redundancy patterns. With a different *substitute* transformation, the successive mapping steps require re-adaptation, but the process steps remain the same. Our open framework allows for the modification or the addition of different transformations and transformation processes to obtain different results, such as the safety-executive redundant pattern.
- **Order of transformations:** after each transformation process, a new node can be selected to introduce additional redundancy in the application, as we did in the EcoTwin truck platooning use-case. Switching the order in which the substitutions are executed does not change the final result. Moreover, the new transformation process can be applied inside an already redundant pattern by selecting one of the nodes in the redundant branches. Note that with the fault tree approximation, this transformation does not have an effect on the failure probability of the application, as it is already part of a redundant branch.
- **Optimal solutions:** in the proposed framework, the transformed graph is analysed and the values of the five parameters are re-calculated. To identify if the new implementation is optimal in terms of the parameters, an automated design-space exploration algorithm should be introduced. Our framework can be used to explore the possible implementations and redundancy levels to find optimal solutions in such an algorithm. To develop an automated design-space exploration, additional studies are required: first, meaningful optimization functions must be identified. Second, the system constraints that limit the exploration must be defined. In our tool, we use resource utilization as a system constraint, and additional parameters can be used, e.g. spatial limitations of the physical locations, cost limitations, etc. Finally, a scalable algorithm that chooses in which parts of the system redundancy shall be introduced must be developed. Mathematical models are needed to understand if minima in the optimization functions are local or absolute minima.
- **User manual input:** the only input required by the user is to select the application node (or a resource) in which redundancy is necessary. The result of the transformation process is a new analysable graph that can be evaluated or further modified.

4.7 Conclusions

In this chapter, we described the transformations that we use to introduce redundancy in the model. The transformations are used to substitute elements, connect consecutive redundant patterns, separate elements that are mapped to multiple resources or locations, and improve the mapping by reducing the number of utilized resources in the resource layer. Not all the main transformations generate analysable graphs, as an additional mapping step is necessary to map the newly introduced nodes or resources. The full transformation process applies a sequence of main transformations and remaps the new nodes and resources to the respective layer, generating an analysable output graph.

In the second part of the chapter, we apply the analysis and the transformation process to two examples: in the first one a simple application is evaluated and transformed to see the effects of varying the analysis metrics and to analyse the resulting fault tree; in the second one, a real-life use-case scenario, the lateral control application of the EcoTwin II truck platooning project, is analysed and multiple transformation steps are taken to obtain a fully redundant system. The results of our transformation process are compared to the manual design process that the designers of the original project did to identify the similarities and differences with our automated process. Overall, the results obtained with our process are positive and show the flexibility of our analysis framework, which can be adapted to different automotive systems and scenarios.

5

Automotive electrical and electronic architectures


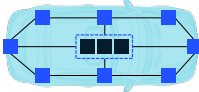


5.1 Introduction and related work

As the functionality provided by the vehicle software grows, the automotive architectures, often defined as automotive Electrical and Electronic (E/E) architectures, evolve with it. While in the past the focus was on electromechanical components, nowadays one of the most important aspects for the development of the new architectures is the computing part. Powerful computers are used to process the huge amount of data coming from the new sensors such as cameras, radars, and lidars. The networks that interconnect these components are evolving from being ad-hoc solutions into more structured and hierarchical versions.

The automotive companies that are investing in the research on autonomous vehicles are also looking at how the future architectures will look like, to develop scalable systems that can meet all the requirements related to level 3+ autonomous vehicles. In the technology trend analysis made by McKinsey [28], fourth and fifth generations of Autonomous Vehicles (AVs) will use first domain-based and then zone-based and centralized architecture topologies, as shown in Figure 5.1.

The view from other authors might slightly differ, for example Figure 5.2 illustrates what Siemens consider to become the main types of future E/E architecture topologies [150]. Siemens highlights the difficulties of an integrated development process in which multiple applications are designed to run at the same time on the vehicle by different development teams, flexibility and possibility for updates

■ ECU¹ ■ DCU² ■ Zone computer ■ Central computer

Generation	Archetype	Description	
5th generation: Vehicle centralized	Central brain	Powerful central general-purpose processing unit with mostly unprocessed inputs from electronic control units (ECUs) Optimized for camera and deep-learning-based approach to highly autonomous driving	
	Zone computing	Majority of domains controlled by central virtual domain units based on a general-purpose compute cluster Backed by distributed-zone computers across the vehicle Best for OEMs with limited number of platforms and variants	
4th generation: Domain centralized	Full-domain centralization	Domain-control-unit (DCU) architecture used for all domains Well suited for multiple platforms Suitable for level-3 autonomous driving	
	Selected-domain centralization	Selective domains using DCUs (typically starting with infotainment and ADAS ³) Good for cost minimization	

¹Electronic control systems.

²Domain control systems.

³Advanced driver-assistance systems.

Figure 5.1: Vision of future automotive architecture trends from McKinsey [28].

in the software are required, and safety-critical applications run in parallel to QoS oriented applications [149]. They also refer to the fact that the current automotive systems have a high number of ECUs and that in future vehicles they will be consolidated into a smaller number of powerful control units, but how to do the consolidation for each OEM to decide for themselves.

Elektrobit [64] discusses an architecture with a central powerful computing unit connected mostly via Ethernet, with the TSN protocols for safety-critical data, to the sensors and actuators. GuardKnox [67] presents a zone-oriented architecture in which the hardware consolidation process is based on the physical position of the elements inside the vehicle: it connects the sensors and actuators to the closest zone controller, which in turn are connected to centralized vehicle servers and computing units via a backbone Ethernet network. The resulting architecture is modular and optimizes the wiring costs. Multiple approaches in terms of network topologies are discussed: ring, tree, star, or hybrid backbone networks to understand which are the trade-offs in terms of network performances and reliability. The authors of [127] discuss a domain-oriented architecture in which the sensors, actuators, ECUs, and communication resources are grouped based on their functionality. Classic domains are *infotainment*, *chassis*, *powertrain*, and *body and comfort*. A special ECU, called *domain controller*, is established for each domain. The domain

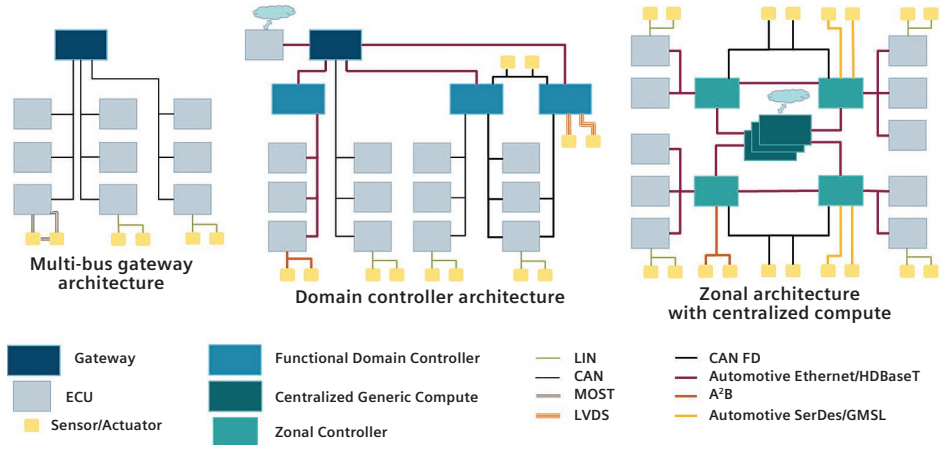


Figure 5.2: Vision of future automotive architecture trends from Siemens [150].

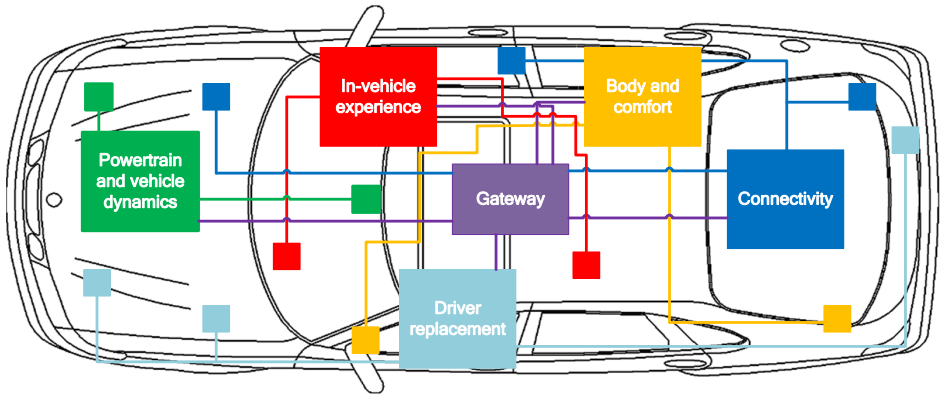


Figure 5.3: A domain-based architecture topology [125].

controller controls the domain network components and communicates with the backbone network. In [18] the authors propose a domain-oriented architecture in which a domain gateway is placed in parallel to the domain controller to access directly the internal domain networks. Different divisions in terms of domains can be found in the literature: for example, Figure 5.3 shows a domain-based architecture topology, in which the functionalities are divided into six domains as in [125]. For a more complete description of different domains and automotive functionalities, we refer to [15].

In Figure 5.4 the trends of future automotive vehicles architecture from NXP point of view are shown [151]. Starting from domain-based architectures, hybrid

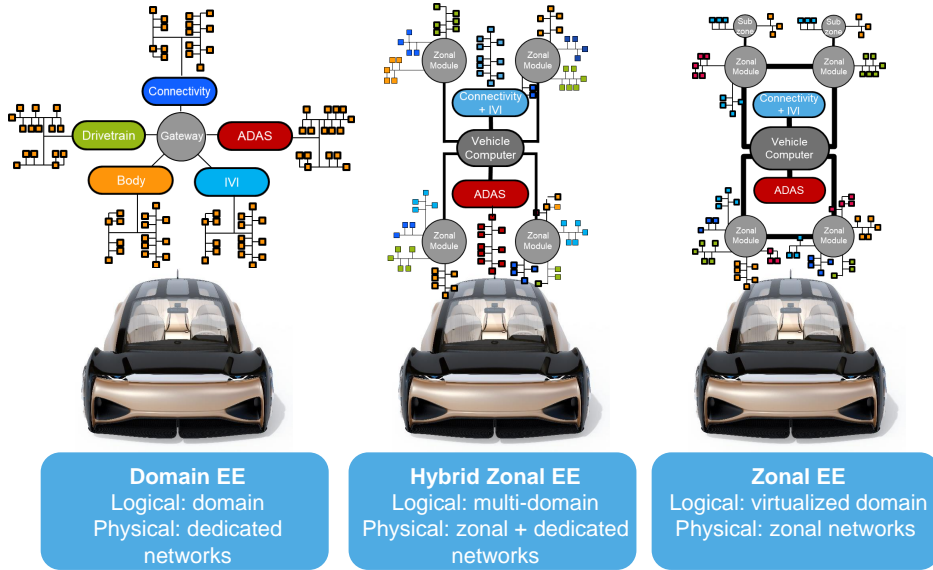


Figure 5.4: Vision of future automotive architecture trends from NXP [151].

and then full zone-based architectures with virtualized domains for isolation of critical applications are expected to become the main architecture trends.

As identified by the various architecture trends, domain-based and zone-based topologies are foreseen to be the main solutions for E/E architectures. In this chapter, we focus on the analysis of these architecture topologies, which are the opposite ends in the spectrum of future E/E architectures. Understanding their differences and their advantages and disadvantages with our quantitative method provides useful insights for future product development, in which the decision of the architecture topology helps in optimizing the system parameters. This chapter is based on our publication [57].

In Section 5.2 we provide definitions for the analysed architecture topologies. Each topology is based on the resource layout and how each application is mapped on them. In Section 5.3 we define an example case consisting of three applications that are implemented using the different architecture topologies that we defined. In Section 5.3.1 we apply the model transformations to introduce redundancy via the ASIL decomposition-based technique and analyse the impact of redundancy on the different architecture topologies with the five analysis parameters described in Chapter 3. We first assume that all the application nodes fail silently. We then discard this assumption in Section 5.4, in which we analyse the system with the use of either physical separation or virtualization. Finally we conclude this chapter in Section 5.5.

5.2 Architecture topologies and definitions

In this work we define an architecture topology as a combination of a *category* and a *mapping*. We consider two possible architecture categories: *Domain-Based (D)* and *Zone-Based (Z)*. In both categories, the sensors and actuators have a fixed position related to their functionality (e.g. front radar), while the rest of the hardware resources can be positioned freely in the vehicle. The two categories follow these rules to connect the sensors and actuators to the rest of the system:

- **Domain-Based (D)** groups the resources into domains. Resources that belong to the same domain will provide specific functionality, e.g. driver-replacement-related applications are mapped to the driver replacement domain resources. Each domain has a domain controller, a resource that bridges the domain to the backbone network, which is connected to a central resource and the other domain controllers. The sensors and actuators of a specific domain are connected to the corresponding domain controller either with a direct connection or by using a domain network.
- **Zone-Based (Z)** groups the resources according to their physical position in the vehicle. Several zones have to be selected and a zone controller is positioned in every zone. Each zone controller connects its zone to the backbone network. The backbone network is then connected to a central resource. Each sensor and actuator is connected to the nearest zone controller either with a direct connection or a zone network.

Separating the architectures into domain-based and zone-based is not enough to distinguish the different possibilities available when designing the E/E architecture. For example, the mapping of the application could be more or less centralized. For this reason, for each of the two categories, we consider two possible mapping rules of the application to the hardware resources:

- **Vehicle-Centralized (VC)**: all the functional nodes are mapped to the central resource. The sensors feed their data to the central unit (passing through the domain or zone network, the controller, and the backbone network), which then provides signals for the actuators. The domain or zone controllers perform only networking functions between the domain or zone network and the central unit.
- **Controller-Based (CB)**: when possible, the functional nodes are mapped to the controllers in the domain or zone, which in this case require computational capabilities. The central resource performs only the tasks that require data from or provide data to multiple domains or zones and thus are executed centrally.

We will abbreviate the architecture topologies with the four combination of the category and mapping: *D-VC*, *D-CB*, *Z-VC*, *Z-CB*.

As we discuss in Section 5.3.2, other variations over the four topologies that we analyse exist. With our framework, we can analyse the different scenarios to identify the system characteristics. In the following analysis, we use backbone networks with a star topology, and we do not allow inter-domain or inter-zone communication. This means that when inputs from or outputs to multiple domains or zones are necessary, a function must be executed in the central unit, which communicates with all the domains or zones via the backbone network.

5.3 Modeling and analysis of architecture topologies with fail-silent assumption

In this section, we show an example of the usage of our proposed framework to analyse the different architecture topologies in an illustrative system. First, we analyse three applications in the four topologies with our framework, then we introduce redundancy in two separate points in each topology, to evaluate how redundancy impacts the different implementations. We assume that the application nodes are *fail-silent*, thus the fault tree is generated with this assumption, as described in Section 3.4.1.

We define three illustrative applications that we use for our experiments. Each application is described by a separate graph in Figure 5.5, showing their functional and communication nodes and their logical connections. We divided the applications into a safety-critical one, in orange, and two non-safety-critical ones, in green. The safety-critical application contains typical operations that a self-driving vehicle performs, such as environment modelling and vehicle control, while the non-safety-critical applications provide additional information with a surround-view application that shows the back and front camera views to the vehicle passengers, and a comfort application that interacts with the heat, ventilation, and air conditioning system. We assume that only the central resource can meet high computational requirements in our experiments, as it is often true for a self-driving application that requires powerful computing resources, such as NXP Bluebox [111] or NVIDIA PX Drive [110]. The central part of the safety-critical application, which has high computational requirements, is therefore always executed in the central resource. In the experiments, this means that part of the data is always transmitted through the backbone network, and the safety-critical application cannot be fully isolated, e.g. in the driving replacement domain resources in Domain-based topologies. An alternative to this would be to utilize the high-performance computing resources as domain or zone controllers, which would distribute the application over multiple resources, but increasing the number of costly resources. By using a central resource to execute part of the application, we distribute it over multiple resources and obtain more meaningful analysis results, while if the application is contained in its safety-critical domain its analysis can be performed individually and separated from the rest of the system.

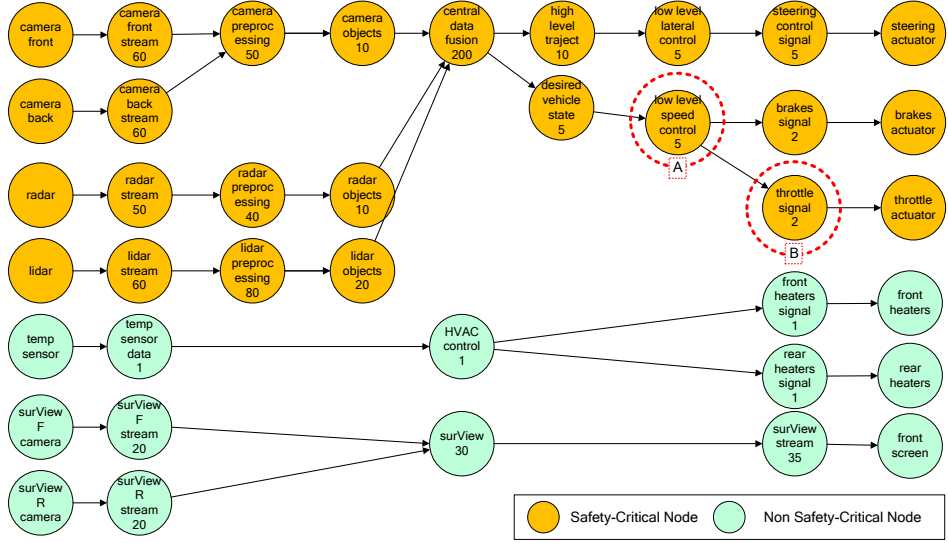


Figure 5.5: The set of applications used for the topologies evaluation.

For the simpler non-safety-critical application we use the common Sense-Think-Act paradigm [148]. For the safety-critical application, we extend it, dividing the Think block into Preprocessing, Data Fusion, and Postprocessing steps. This enables a realistic function mapping over multiple components. The applications have high bandwidth requirements on the sensing side (mostly in the safety-critical application), high computational requirements in the central data fusion node, and low bandwidth and computational requirements in the post-processing and actuation sides.

For quantitative analysis, we annotate the functional and communication loads on each node to reflect these requirements, as shown in Figure 5.5. In our example, we use realistic load proportions between the nodes for the sake of a final comparison of the load distributions over the different architecture topologies.

Nodes A and B are selected for redundancy for the experiments of Section 5.3.1. With the selection of these two nodes, we analyse the effects of introducing redundancy in either the post-processing part or in the communication between the post-processing and the actuators. These nodes have lower functional and communication loads compared to the sensing and pre-processing part (e.g. the Throttle Signal communication load is 30 times lower than that of the Camera Front Stream node). Nevertheless, the effect of introducing redundancy in these nodes is relevant in the results, as the modification of part of the resource layer leads to additional transformations in the application layer.

Moreover, we assume that the lidar and the radar are *smart sensors* that locally convert raw data into output objects. This means that the pre-processing part of

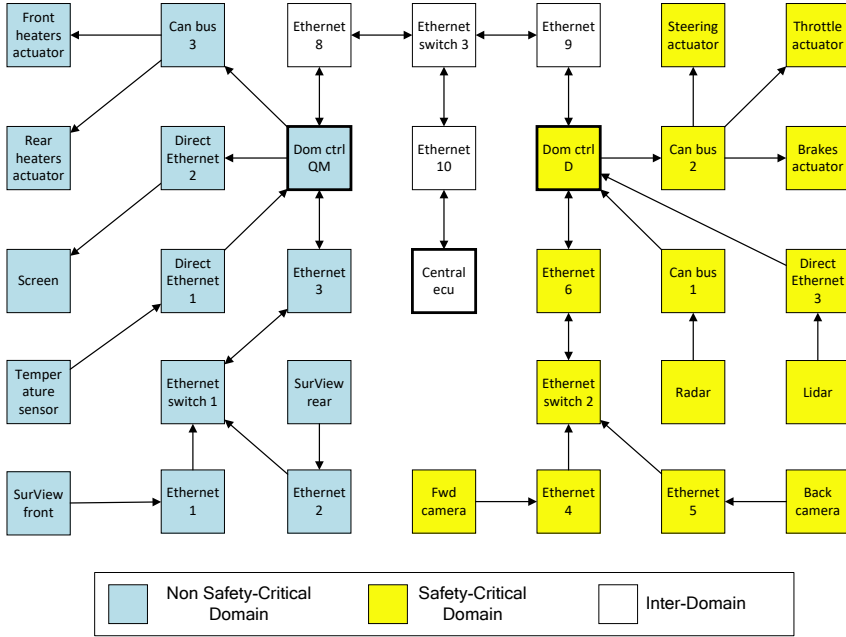


Figure 5.6: *Non-redundant Domain-based hardware resources.*

these sensors is mapped to the sensor resource itself. Traditional sensors' output is pre-processed in separate functional resources instead.

We use a representative non-redundant hardware architecture formed by two domains (or zones) and a central unit. In the domain-based architectures, one domain contains all the safety-critical resources, sensors, and actuators, while the other domain contains the non-safety-critical resources, sensors, and actuators. In the zone-based architectures, the two zones divide the vehicle into a front and a rear part, and sensors and actuators are connected to the zone controllers based on their position inside the vehicle. Figure 5.6 shows the chosen architecture for domain-based categories. The domains are connected to the central unit via a star switched-Ethernet network, while the domain networks are a combination of buses, switched-Ethernet networks, and direct connections. Figure 5.7 shows instead the non-redundant zone-based architecture and its networks configurations.

Each sensor and actuator has a fixed position in the physical space, while other resources can be placed with some freedom inside the vehicle. The central unit is placed in a central position with respect to the vehicle. The domain controllers are placed in an available location in a central position with respect to the domain's sensors and actuators, to minimize the total communication cable length between them. Figure 5.8 shows the positioning of these resources in the *D-VC* topology. The zone controllers are placed instead in a central position in the respective zone,

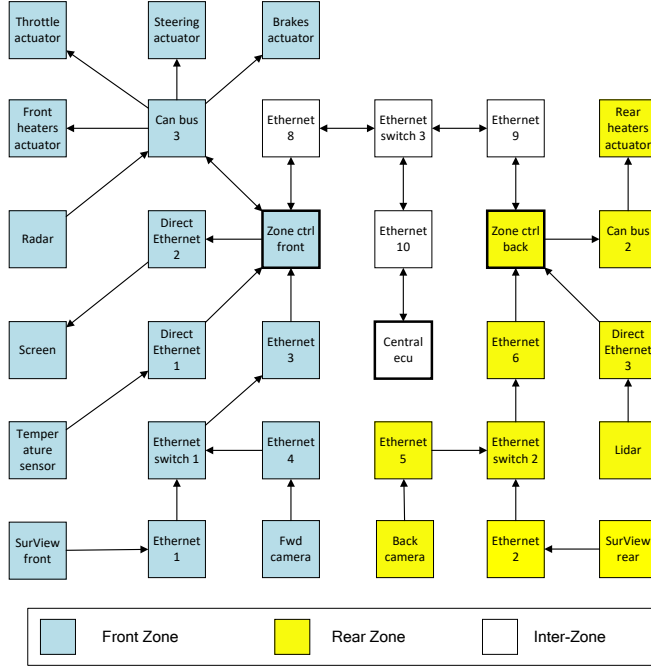


Figure 5.7: Non-redundant Zone-based hardware resources.

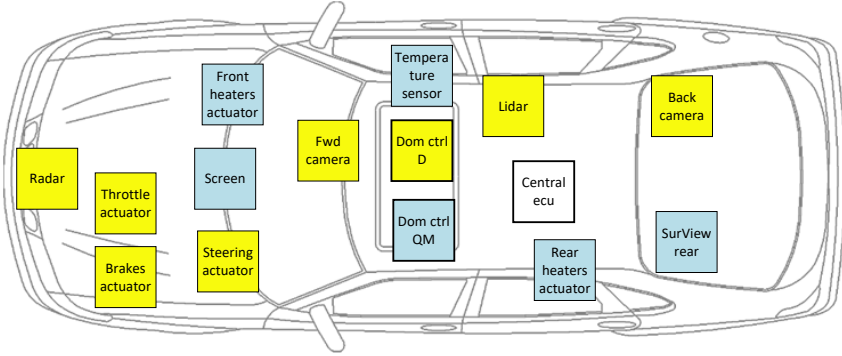


Figure 5.8: Positioning of the sensors, actuators, domain controllers, and central unit in the D-VC topology.

again for cable length optimization.

The initial resource layer for the topologies *D-VC* and *D-CB* is identical, but there will be differences in the redundant scenario due to the different mapping of the application. In the same way, the zone-based topologies resource layers will

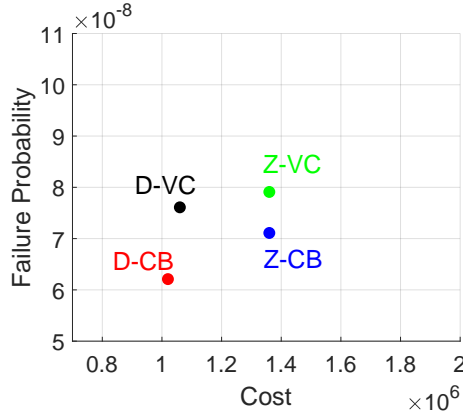


Figure 5.9: Failure probability vs Cost in the non-redundant scenario.

show differences only in the redundant scenarios.

Complete mappings of the application to the four architecture topologies are complex graphs and are shown for reference in Appendix D.

In the experiments, we analyse first the topologies without introducing redundancy in the system. The three applications of Figure 5.5 are mapped to the different architecture topologies based on VC or CB rules, and each solution is analysed.

We observe in Figure 5.9 that the cost of the domain-based topologies is lower than the cost of the zone-based ones, assuming the cost metric of Table 3.3. Since the zones have mixed-critical application nodes (both ASIL D and QM), the zone controllers and the dedicated units need to be more expensive ASIL-D ready resources to satisfy the safety-critical requirements. The difference in failure probabilities of the safety-critical application between domain-based and zone-based topologies is related to how the internal networks are configured: both categories use Automotive Safety Integrity Level (ASIL) D (or equivalent after ASIL decomposition) resources for the safety-critical application, which have the same failure rates, but in this case, the zone-based categories have one less communication resource. The VC and CB mappings differ in terms of communication paths and the number of utilized resources. For example, in the D-CB topology, the non-safety-critical part of the application does not reach the *Central Electronic Control Unit (ECU)* resource, and the *Ethernet 8* resource is not used.

Figure 5.10a shows the calculated total communication cable length, which is lower in the zone-based topologies, as expected since they connect sensors and actuators to the closest controller. The resource layers for VC and CB mappings are identical, however, the resource *Ethernet 8* is not used, and thus we do not add the length of this connection to the calculation.

Figure 5.10b shows the total functional and communication loads of the

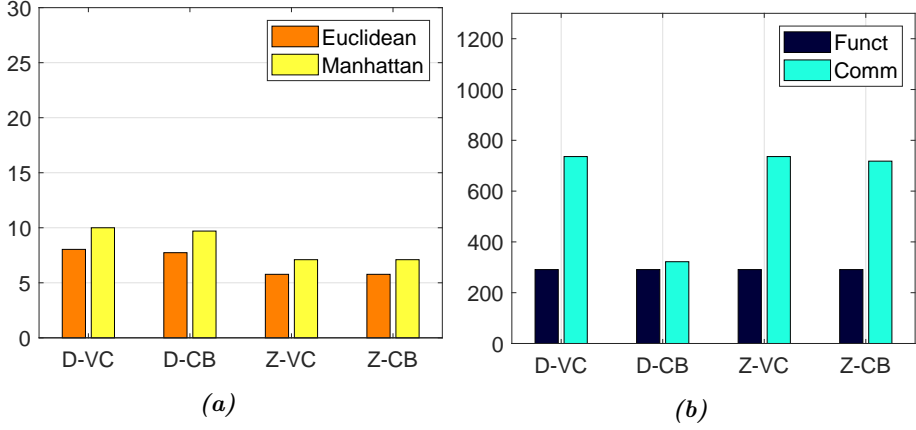


Figure 5.10: Total communication cable length (a) and total communication and functional loads (b) for the non-redundant scenario.

topologies. In the non-redundant scenarios, the total functional load does not vary as it is related only to the applications, but it is distributed differently over the architecture. The communication load varies instead based on the topology. The D-CB topology has the lowest communication load since all the processing steps are executed in the domain controller and the sensor or actuator data is sent only through the local domain network and not through the backbone network. The output data size of the post-processing step is highly reduced compared to the initial input raw sensor data. VC mappings instead require all the processing steps to be done in the central unit. This means that the raw sensor data has to be transmitted not only inside the domain or zone network but also through the backbone network. The Z-CB topology instead can perform only part of the functionality locally. Since we do not allow communication between the zones, the tasks which require inputs from or send outputs to multiple zones are executed in the central resource.

By combining the results of the communication load and total cable length calculations, we observe, as shown in Figure 5.11, that a) the D-VC topology has longer total communication cable length with higher bandwidth requirements, b) the D-CB topology has longer total communication cable length with lower bandwidth requirements, and c) zone-based topologies have shorter total communication cable length with higher bandwidth requirements.

5.3.1 Impact of redundancy on different topologies

In our experiments we introduce redundancy by using the transformation process explained in Section 4.3, following the ASIL decomposition rules to lower the ASIL requirements of the original application and the ASIL specifications of the

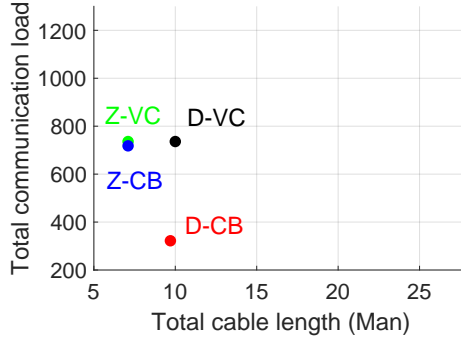


Figure 5.11: Communication load vs Total communication cable length (Manhattan) in the non-redundant scenario.

resource layer. While in the truck platooning use-case of Section 4.5.2 we applied the transformation process to multiple consecutive application nodes, here we transform a single application node in each experiment, and the results of this transformation are analysed in the different topologies. Each architecture topology is affected differently and our analysis framework can identify and highlight the differences. We observe that the key difference is made by the number of nodes that share the same resource as the node that is selected for the transformation process: when the resource is shared by many different nodes and different applications, additional substitutions are necessary and the analysis parameters such as total functional load and total communication load increase. The mapping of each node depends on the architecture topology that is chosen.

Transformation of the node Low-level Speed Control

For the first modification, we apply the node transformation to the Low-Level Speed Control (LLSC) node, marked with the dashed circle A in Figure 5.5. This allows us to observe how introducing redundancy in the low-level control part of the safety-critical application impacts the system. Low-level control nodes are the ones that interact directly with the vehicle dynamics, being able to provide signals to the actuators. They are a very critical part of the system, despite being less computationally intensive than higher-level functions [83, 160]. The redundancy transformation affects different parts of the system based on its topology since the LLSC node's mapping varies. Figure 5.12 shows the effects of the transformations on the application layer for the D-VC and the Z-VC topologies after following the transformation process described in Chapter 4. We use in this experiment the resource modification strategy that uses the *substitute* transformation on the resource layer, removing the original resource on which the selected application node is mapped. This strategy was described as *Strategy 2* in Section 4.3.2. The preprocessing, data fusion, and post-processing parts of the safety-critical

application are all mapped to the central unit, which is duplicated because of the redundancy in the LLSC node. All these parts become redundant as well, obtaining the redundant branches a and b in Figure 5.12. The non-safety-critical applications are not affected by the transformation since they can be mapped on one of the two redundant central ECUs because of their lower ASIL requirement. In the D-CB and Z-CB architecture topologies, the transformation modifies a smaller part of the application, as seen in Figures 5.13 and 5.14. This is because fewer nodes are mapped to the same resource as the LLSC node, as seen in Figure 5.15.

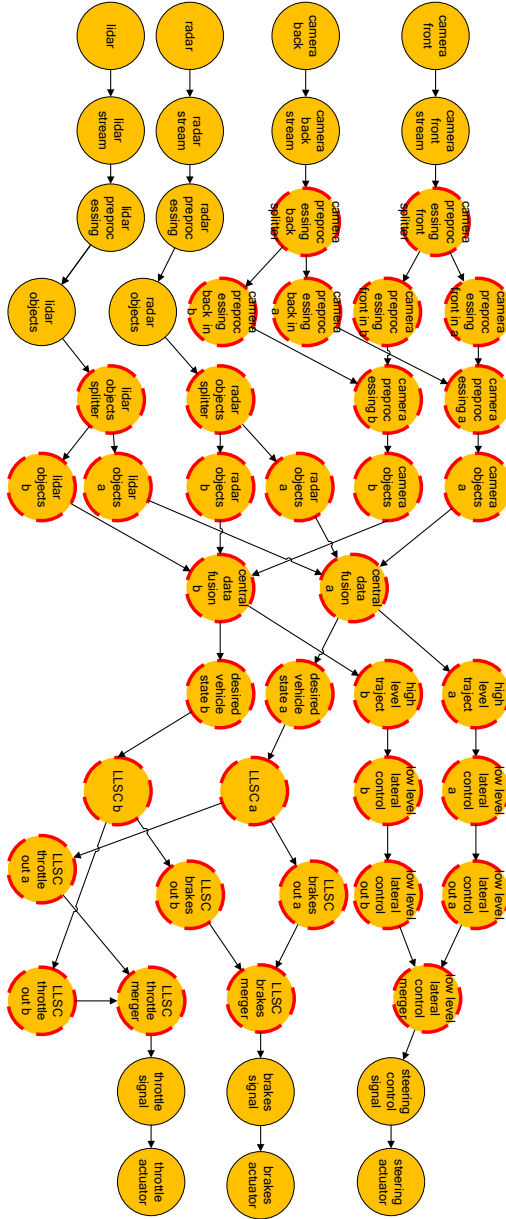


Figure 5.12: Safety-critical application in the D-VC and Z-VC topologies: application layer after the transformation of the LLSC node. The nodes with a dashed red line are changed compared to the non-redundant version.

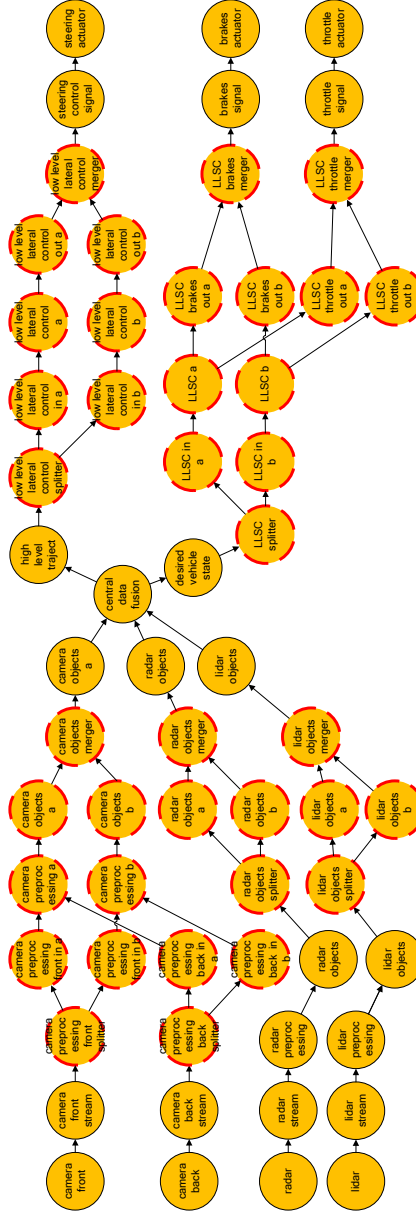


Figure 5.13: Safety-critical application in the D-CB topology: application layer after the transformation of the LLSC node. The nodes with a dashed red line are changed compared to the non-redundant version.

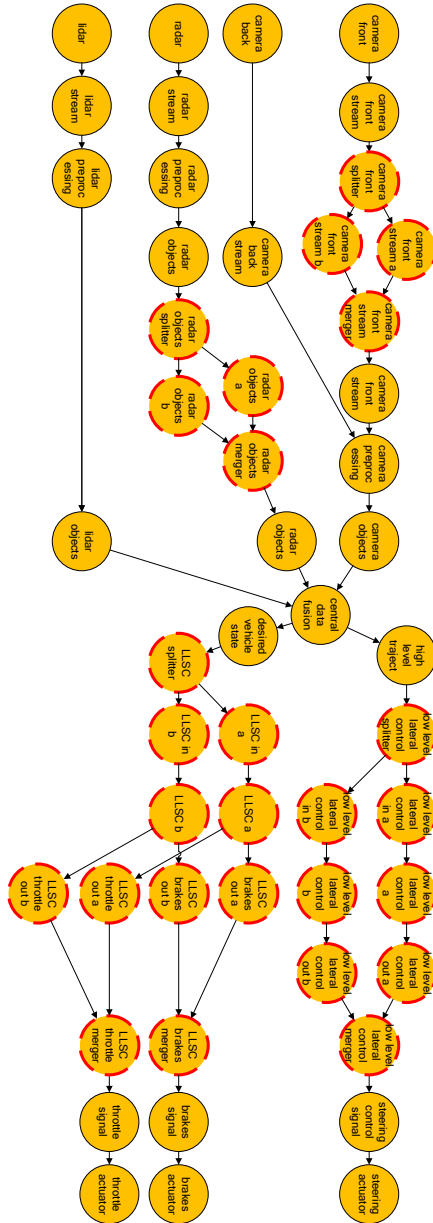


Figure 5.14: Safety-critical application in the Z-CB topology: application layer after the transformation of the LLSC node. The nodes with a dashed red line are changed compared to the non-redundant version.

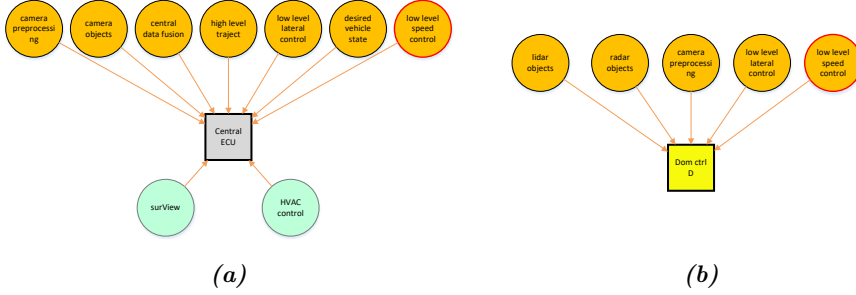


Figure 5.15: The nodes mapped on the Central ECU with the node LLSC in the D-VC topology (a) and the nodes mapped on the safety critical domain controller with the node LLSC in the D-CB topology.

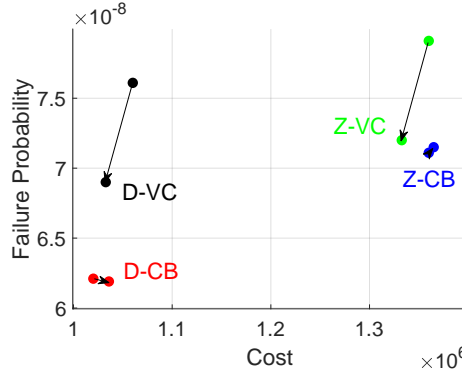


Figure 5.16: Failure probability vs Cost for redundant LLSC node compared to the non-redundant version.

We observe that the architectures with a redundant LLSC have similar cost values compared to non-redundant ones, e.g. the D-VC topology costs 0.4% less and the Z-CB topology is 2.6% more expensive than their non-redundant versions, as seen in Figure 5.16. While non-redundant architectures must use ASIL-D ready resources, redundant architectures can use ASIL-D ready splitters and mergers in combination with parallel lower-level resources, e.g. two ASIL-B ready ones. The cost of an ASIL D splitter or merger combined with ASIL B functional and communication resources is similar to the cost of the original ASIL D resource when following Table 3.3 (10000 for each splitter and merger, 500 for each of the two new redundant functional resources, 400 for each new redundant communication resource, compared to a single ASIL D resource with a cost of 50000), but can vary with the cost metric that is selected. In the redundant scenario, the main contribution to the final cost is due to the ASIL D splitter and merger resources.

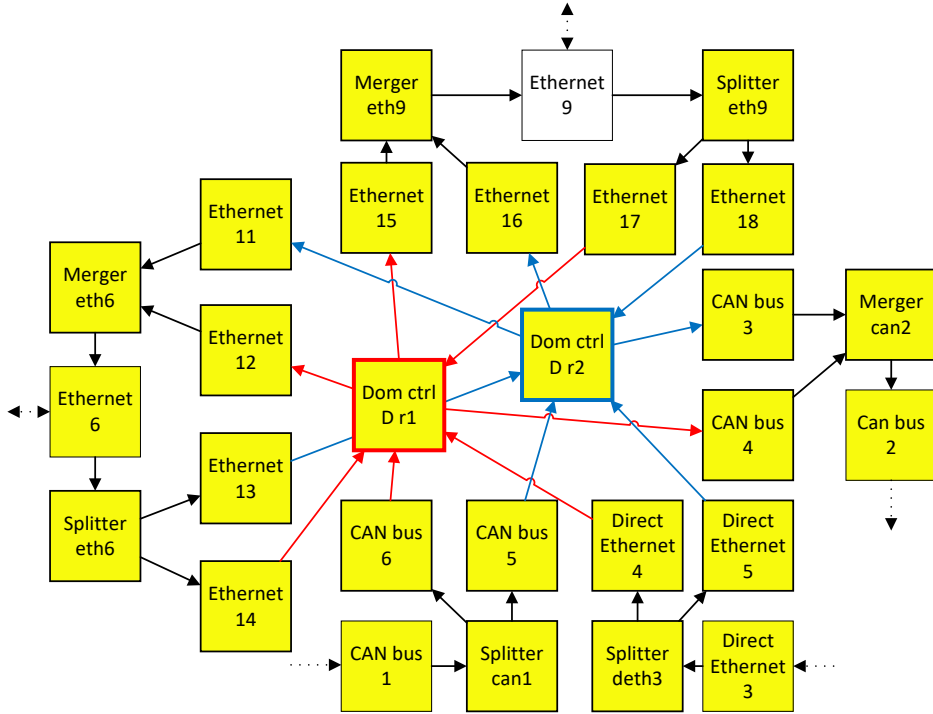


Figure 5.17: Redundant domain controllers of the redundant D-CB topology after the duplication of the LLSC node.

If lower-cost splitter and merger ASIL D resources would be available, redundancy would come also at a lower cost compared to non-redundant ASIL D solutions.

Compared to the non-redundant case, the failure probability is lower for VC mappings, e.g. D-VC topologies have a 9.4% lower failure probability, as shown in Figure 5.16. This effect is due to more nodes being mapped to the now redundant central ECU (as seen in Figure 5.15). The system can benefit from the lower failure rates of the safety-oriented splitter and merger resources. In CB mappings instead, this effect is hidden by the more complex communication interfaces that are connected to the now redundant controllers. The central ECU only has one Ethernet port, meaning that only one splitter and one merger resource will be required. The controllers are connected instead to multiple network components, and on each port, a splitter and/or a merger resource is required. For example, from Figure 5.6, the domain controller *Dom ctrl D* will have with the D-CB topology in LLSC-redundant scenarios four splitters (one for each input port) and three mergers (one for each output port), as shown in Figure 5.17.

When analysing the total communication cable length in this redundant sce-

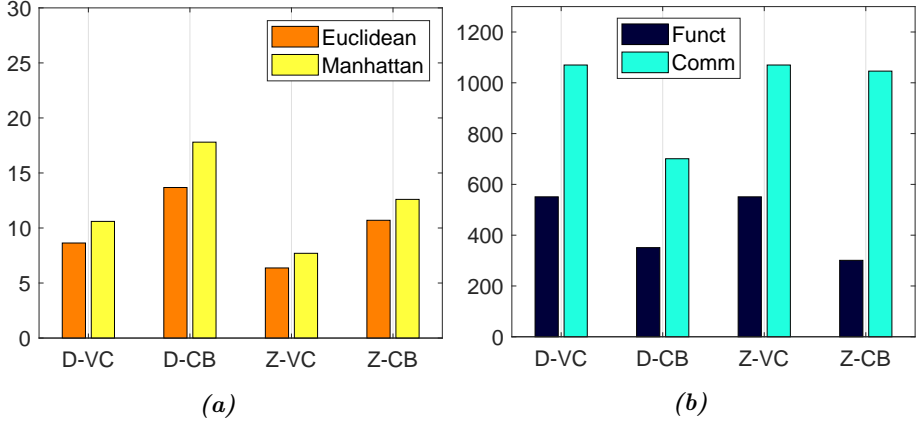


Figure 5.18: Total communication cable length (a) and total communication and functional loads (b) for redundant LLSC node scenario.

nario, we notice how the CB mappings have a greater impact on this parameter, shown in Figure 5.18a. Redundant controllers lead to an increase in the number of communication resources since the controllers are connected to both the local and the backbone networks.

Figure 5.18b shows the total functional and communication loads after introducing redundant hardware for the redundant LLSC nodes. The functional load varies between the topologies because of a different number of application nodes mapped to the original resource. The VC mappings lead to a high communication load when redundancy is introduced in the LLSC node (or in any other processing node, since they are all mapped to the central ECU). In the case of the Z-CB topologies, the expansion of the resources does not involve additional functional node transformations, since the LLSC is the only node mapped to the zone controller while the other nodes are mapped to the central unit. Figure 5.19 combines the results of the total communication load and the total communication cable length calculations.

The D-CB topology is still better than the D-VC topology in terms of failure probability, but has now slightly higher cost. We observe a similar situation for Z-CB and Z-VC, but for them the gap between the failure probabilities is closer. Overall, the zone-based topologies have lower total communication cable length, at the cost of increasing communication load.

To conclude, when a functional node of the safety-critical application is substituted, we observe greater differences between the different mappings compared to the non-redundant scenario. The D-CB topology performs better than the others, since only the domain controller of the safety-critical domain is substituted in our system, and a lower number of nodes related only to the safety-critical application

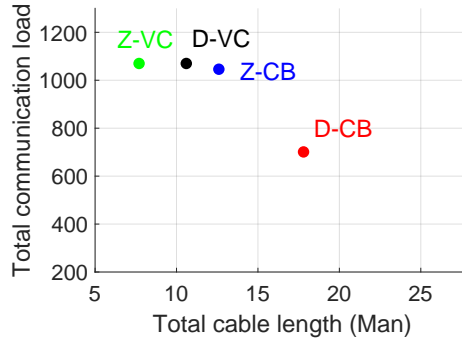


Figure 5.19: Communication load vs Total communication cable length (Manhattan) for redundant LLSC node scenario.

are mapped there. However, in case the goal of our design is to minimize the weight of the wire harness, Z-VC solutions provide better results in this parameter.

Transformation of the node Throttle Signal

Our third experiment and second modification of the system consists of applying the transformation process to the communication node Throttle Signal (TS), as marked with the dashed circle B in Figure 5.5. It is a low-level signal with a small communication load but with critical importance since it controls the throttle actuator. Depending on the topology it is mapped to different communication resources and more than one resource is affected by the transformation of the node. In CB mappings the signal comes from the domain or zone controller, while in VC mappings it comes from the central unit through the backbone and the domain or zone network.

Figure 5.20 shows the total cost and failure probabilities of the topologies for a redundant TS node. The VC mappings have a significantly higher cost and failure probability compared to the non-redundant scenario: the cost increases by 46.4% and 43.6% while the failure probability increases by 24.2% and 26.4% for the D-VC and the Z-VC topology respectively. This effect is due to the transformation of all the communication resources that carry the throttle signal, which in VC mappings are both parts of the backbone and of the domain or zone network. The CB mappings cost and failure probability also increase, despite the control signals being transmitted locally, but to a lower degree since fewer communication resources are involved.

As shown in Figure 5.21a, the zone-based topologies have a lower total communication cable length, and the CB mappings result in lower values compared to the other mappings. This happens because only part of a local network becomes redundant, which is reflected in Figure 5.21b in the form of a lower communication load. The results of these calculations are combined in Figure 5.22.

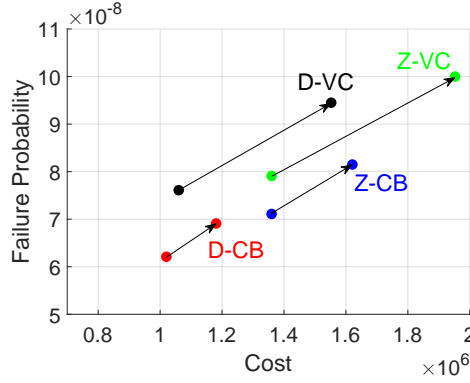


Figure 5.20: Failure probability vs Cost for redundant TS node compared to the non-redundant version.

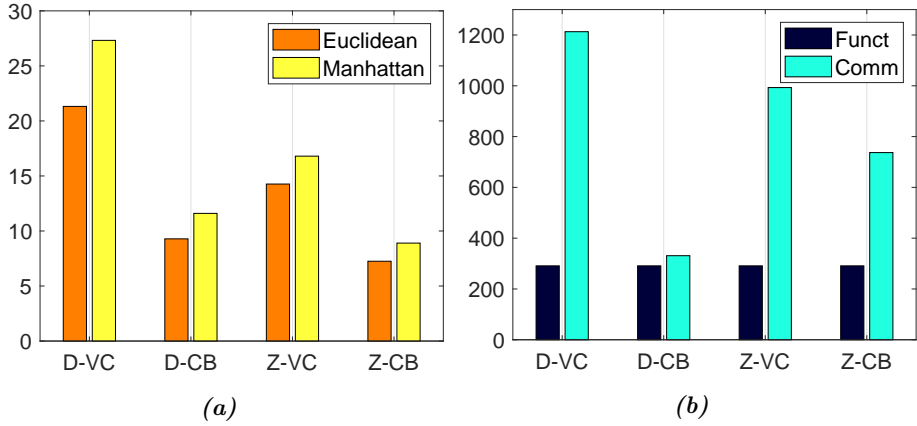


Figure 5.21: Total communication cable length (a) and total communication and functional loads (b) for redundant TS scenario.

The functional load is constant since only communication resources are expanded. The communication load of the TS node is low, but when making it redundant most of the communication nodes that are mapped on the same resources are consequently transformed following the rules of Chapter 4. In the case of the VC mappings and the zone-based topologies, higher-level communication data is mapped on these resources, such as raw camera streams or lidar and radar detected objects, with high communication loads. The transformation of a single low-level control signal leads to the modification of many parts of the system.

When making a low-level signal redundant, D-CB topologies are again recommended, since they lead to a smaller modification of the system since the low-

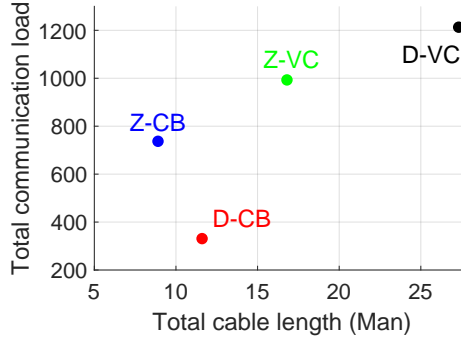


Figure 5.22: Communication load vs Total communication cable length (Manhattan) for redundant TS node scenario.

level signals are transmitted only inside the domain networks, as it happens in our use-case. Based on our experiments, only in case cable length is the target optimization, Z-CB topologies should be preferred to D-CB.

Final comparison

We observed in the previous experiments how redundancy applied in two different application nodes impacts the system properties. We observed how the D-CB topology has better load and cost results than the other topologies. However, in some scenarios, zone-based topologies are better in terms of failure probabilities and total communication cable length.

Figure 5.23 shows the distribution of the functional load over the computational resources. Note that the different topologies have different requirements: the VC mappings require only the central unit to process the data, while the CB mappings require the central unit to process the data fusion part of the application, but also require the controllers to have computational power for pre and post-processing. Figure 5.24 shows the communication load distributions in the topologies. The zone-based topologies show more balanced communication loads between the zones (*Network 1* and *Network 2*), as a result of having sensors from a specific domain, for example, the front and back cameras, distributed over the car. In this case, a significant part of the total communication load is placed in the backbone network, since most computation is performed centrally and a large part of the data generated inside a zone is sent via the backbone network to the central unit.

A final comparison between the topologies in the different redundancy scenarios is shown in Figure 5.25.

The parameters for each scenario are normalized over the topologies, where a 0.00 corresponds to the lowest parameter value across all topologies and a 1.00 corresponds to the highest (for each analysis parameter, the lower the better).

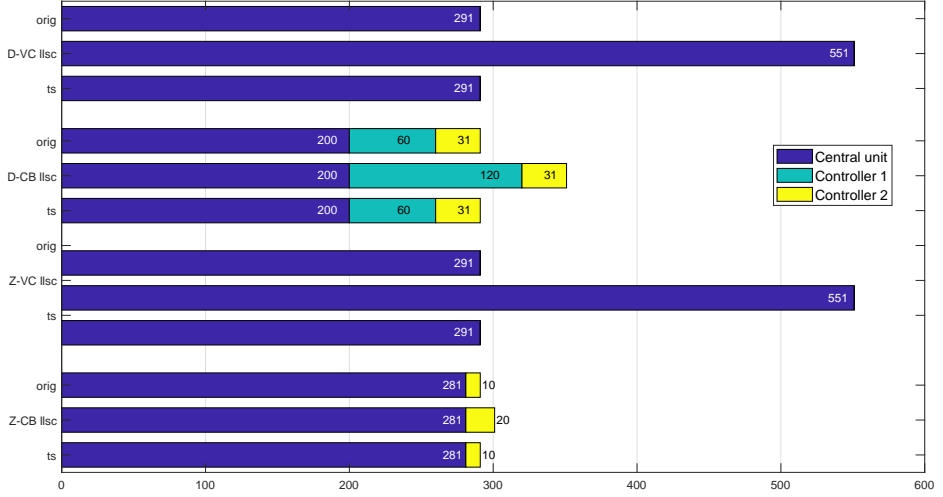


Figure 5.23: Total functional load distribution

The zone-based topologies show a lower cable length. The good results in terms of the functional load of the Z-CB topology for the LLSC-Redundant scenario are related to the isolation of the LLSC node to the front zone controller, which is a result of the specific configuration and not a general characteristic of these topologies. To conclude, when the application requires redundancy, in the presence of high communication loads, and the controllers provide enough computational power to execute the application nodes a CB mapping is highly recommended. Instead, sharing of centralized processing should be considered in the presence of reasonable communication load requirements and when no functional redundancy is necessary.

The redundant scenarios are obtained by selecting the nodes LLSC and TS, but what would have happened by selecting different parts of the application? First of all, selecting non-safety-critical nodes for redundancy is not a relevant choice, as they have QM requirements and there is no necessity for redundancy. As for the safety-critical functional nodes, selecting any functional node of the pre-processing or post-processing part would have lead to the same results, as they are all mapped on the same ECU (either the controller or the central ECU, based on the topology). The only exception is for Z-CB topologies, as the camera-preprocessing node is mapped to the central ECU since it receives data from both the front and the back zone, compared to the selected LLSC node which is mapped on the zone controller. As for the central data fusion node, by design decision it is always mapped to the central ECU, and making it redundant would illustrate fewer differences between the topologies. If another actuator signal was selected instead of the TS node, the same results would have been obtained, as these nodes

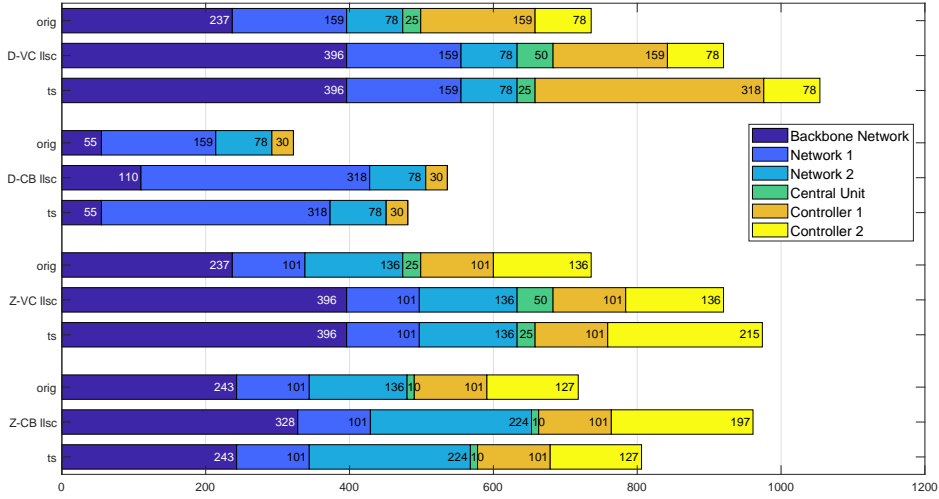


Figure 5.24: Total communication load distribution

	Non-Redundant				LLSC-Redundant				TS-Redundant			
	D-VC	D-CB	Z-VC	Z-CB	D-VC	D-CB	Z-VC	Z-CB	D-VC	D-CB	Z-VC	Z-CB
Fprob	0.82	0.00	1.00	0.53	0.70	0.00	1.00	0.95	0.82	0.00	1.00	0.40
Cost (EXP)	0.12	0.00	1.00	1.00	0.00	0.01	0.90	1.00	0.48	0.00	1.00	0.57
Cable (Man)	1.00	1.00	0.00	0.00	0.29	1.00	0.00	0.49	1.00	0.15	0.43	0.00
Func Load	0.00	0.00	0.00	0.00	1.00	0.20	1.00	0.00	0.00	0.00	0.00	0.00
Comm Load	1.00	0.00	1.00	0.90	1.00	0.00	1.00	0.93	1.00	0.00	0.81	0.60

Figure 5.25: Final comparison tables. For each parameter in each of the three scenarios, 1 corresponds to the maximum value and 0 to the minimum value between the architecture topologies.

are all mapped to the same resources. If sensor data or pre-processed data was selected, different communication resources would have been changed in the local domain or zone networks. However, the effect on the backbone network would be the same (except for the back and front camera streams), especially for the vehicle-centralized topology where all the communication passes through it.

5.3.2 Discussion of the architecture topologies variations

Hybrid topologies

In the previous section, we have identified the advantages and disadvantages of Domain-based and Zone-based solutions, with a focus on redundant systems. The optimal architecture topology depends on the application characteristics. Even in the same system, different applications benefit from different architecture topologies.

We expect that future AVs and Advanced Driving Assistance Systems (ADASs) will use hybrid solutions. We have discussed how safety-critical applications have better results for some metrics in D-CB topologies, in which their cost, failure probability, functional load, and communication load are minimized. This effect is mostly related to the isolation of the safety-critical application to its separate safety-critical domain, keeping the communication from the sensors and to the actuators inside the safety-critical domain networks. When introducing redundancy, only the safety-critical domain is affected, and no other application is modified. Non-safety-critical applications instead have different requirements: the cost related to the non-safety-critical resources is lower, and there is no strict requirement on their failure probability. However, it is important to minimize their total cable length, functional load, and communication load, both to reduce the weight of the cables and to reduce the requirements they have on the resources, especially when shared with safety-critical applications. Zone-based solutions are efficient in this case, as the many non-safety-critical applications can be mapped to the zones independently of their domain.

In our use case, a hybrid architecture would use a safety-critical domain controller for the safety-critical application, and all the sensors and actuators related to this application would be connected to it via the domain networks. The other two applications would be implemented with two zone controllers, one in the front and one in the back of the vehicle, connected to the central resource via the backbone network. This hybrid solution is shown in Figure 5.26 focuses on isolating safety-critical applications, while optimizes the analysis parameters for non-safety-critical ones.

In future work, a more complex system with many applications is needed to study a hybrid solution like this one, as its advantages are clearer with a higher

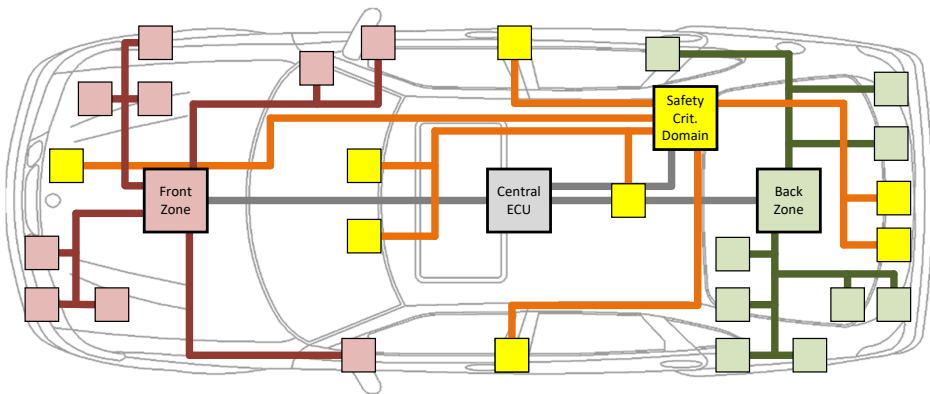


Figure 5.26: Hybrid architecture topology example with two zones and one safety-critical domain.

number of applications. In our example scenario, the three applications would be mapped to three controllers: two zone and one domain controllers. These controllers would be underutilized, as only small applications are executed on them. The cost and the failure probability of such an implementation are always higher than, for example, the Zone-based topologies, as it adds additional domain resources compared to it.

Configuration of the communication networks

In the previous examples we have used CAN buses, switched Ethernet, and direct Ethernet connections in the domains or zones, and a switched Ethernet connection in the backbone network. In particular, the backbone network had a star configuration. Variations of these configurations are possible: for example, the backbone network can be configured as a ring, as shown in Figure 5.27 for a zone-based topology. In this case, each resource in the backbone network is connected to the central unit and to two other resources in the network. This solution has intrinsic redundancy in the backbone network, which can be used for decomposing the ASIL requirements of the communication nodes that use it. The splitter and merger are in this case part of the backbone network itself, positioned in the entrance point of each controller or central ECU to the network.

The central ECU can be substituted by one of the controllers: for example, Figure 5.28 shows a zone-based architecture topology with no central unit and a ring backbone network, in which each zone controller is connected to two other zones. One of the controllers, for example, the Front Right Zone controller, acts as a central unit and receives data from multiple zones. Also in this scenario redundancy in the backbone network can be used to send redundant data via

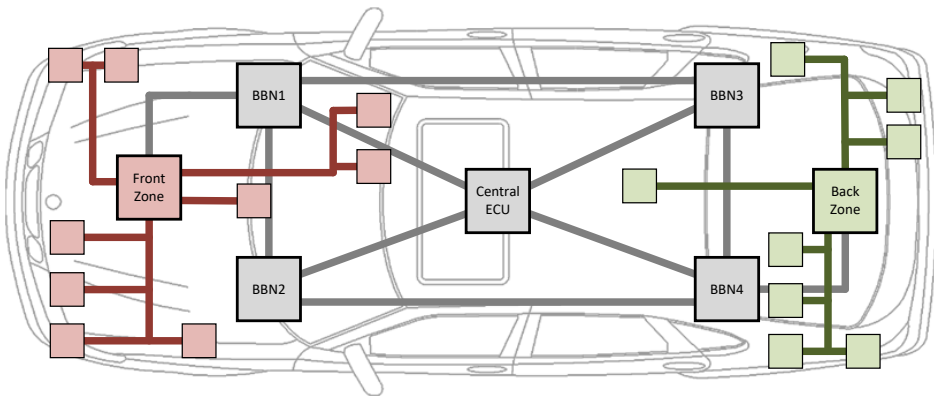


Figure 5.27: Backbone network with a ring configuration in a zone-based topology.

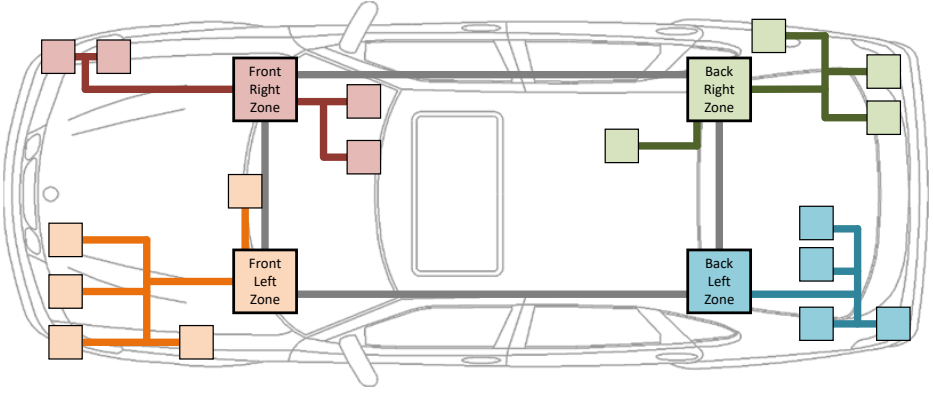


Figure 5.28: Zone-based architecture topology with ring backbone network and no central unit.

redundant paths.

As a future direction, our transformation process described in Section 4.3 can benefit from the intrinsic redundancy of the system by using mapping and task allocation algorithms such as [21] and [147] with additional constraints related to the independence of the redundant tasks.

5.4 Analysis of redundant architecture topologies without fail-silent assumption

In this section, we analyse the effects of non-fail-silent application nodes on the system. As described in Section 3.4.1, the application failure event of the fault tree is now a combination of the basic events related to all application nodes mapped to the resource, reproduced here in Figure 5.29.

In this scenario, sharing a resource between nodes with different ASIL requirements will drastically affect the failure probability of the more safety-critical applications. As discussed in Section 2.6, two techniques are used to isolate the non-fail-silent nodes: *physical separation* and *virtualization*. With physical separation, nodes are mapped to different resources to not interfere with each other. With virtualization, a virtualization mechanism provides isolated Virtual Resources (VRs) in a single physical resource, making the separate VRs *fail-silent* with respect to each other. We map the application nodes with different ASIL requirements to different VRs so that mixed-critical nodes do not interfere with each other. Even by using the two techniques, if different applications but with the same ASIL requirements are mapped to the same resource, they affect each other's failure probability and the system requirements might not be met. In this

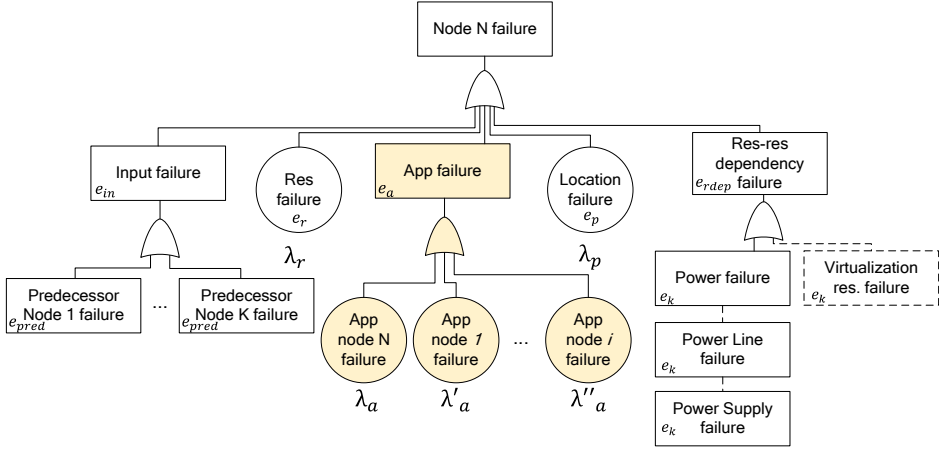


Figure 5.29: Sub-tree structure for the node N *without* the fail-silent assumption. Highlighted, the application failure event which now depends on all the application nodes mapped to the same resource as N .

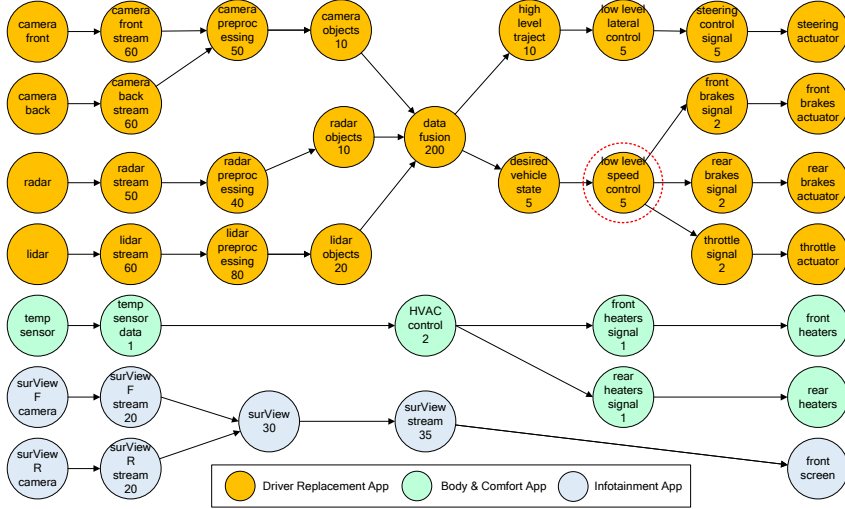


Figure 5.30: Illustrative applications before redundancy is introduced with the transformation process in the LLSC node, circled in red.

case, an analysis is required to calculate the failure probability of each application and compare it to its requirements.

5.4.1 Illustrative system implementation - application and resource layers

As in the previous section, we use three illustrative applications with different ASIL requirements, shown in Figure 5.30. The only variation with the previous experiments is the separation of the brakes actuator into two parts: a front brake actuator and a rear brake actuator. This variation modifies the mapping of the Low-Level Speed Control (LLSC) node in zone-based architectures since it must now deliver actuator signals to both the front and the rear zone. As seen in Figure 5.29, if a node fails, also the other nodes sharing the same resource fail. Without the use of a proper isolation technique and the *fail-silent* assumption for application nodes, lower ASIL applications could interfere with safety-critical ones, effectively leading to a high failure probability of the safety-critical application due to mixed-criticality on a shared resource.

In this experiment, we analyse the difference between the two isolation techniques in the presence of both sharing resources and redundancy. Redundancy is used in the LLSC application node, which is part of the ASIL D application. The transformation process is applied to the LLSC node. Note that the standard transformation process described in Section 4.3 uses physical separation since it introduces new separate redundant resources. When using physical separation, the transformation process uses *strategy 2* when deciding how to modify the resource layer, which substitutes the original resource with new redundant ones. When we use virtualization, we use instead *strategy 3*, which adds VRs with the decomposed ASIL specification, and adds one VR with the original one to reduce the number of application nodes that are modified. In this case, we assume that keeping the original virtual resource is not an expensive solution, as long as the original resource has enough space to host the VRs.

In terms of resource layer, we choose two architecture topologies: *domain-based* and *zone-based* architectures. In both cases, we consider a controller-based approach, which means that the controllers have some computational power and the application nodes are processed by them when possible. The backbone network is implemented again with a star topology.

In our domain-based architecture scenarios, nodes with different ASIL requirements do not share the same resources: the non-safety critical applications are confined to their domain, while only part of the safety-critical application, the data fusion node, is executed in the central ECU. The isolation is only required by the redundant nodes of the application (the redundant LLSC nodes). The resource layer for domain-based architectures before the transformation process is the same as the previous experiments, with the modification of the front and rear brakes actuator. Physical separation or virtualization is only used in the safety-critical domain controller after the transformation process. Figure 5.31 shows the part of the domain-based resource layer that is modified by the transformation process when using virtualization.

Figure 5.32 shows our zone-based hardware resources implementation with

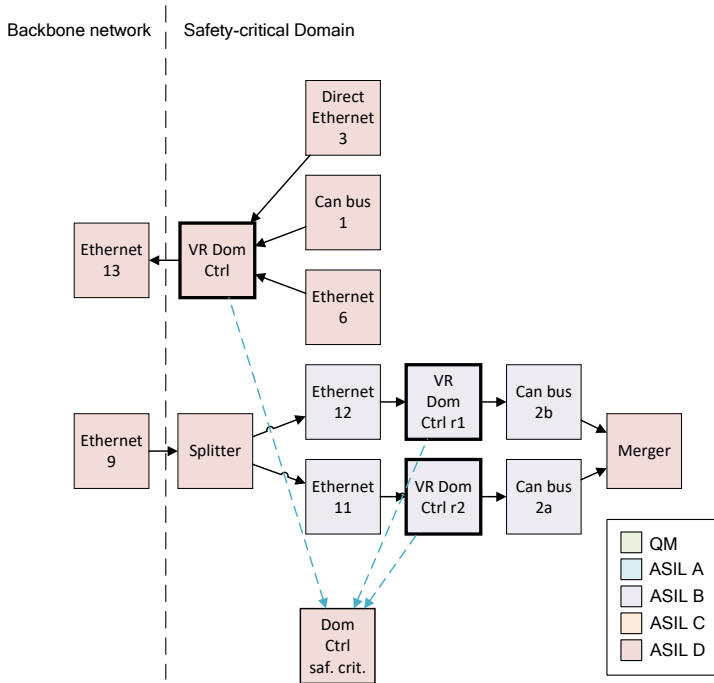


Figure 5.31: Two virtualized domain controllers are used to execute the redundant LLSC node, while a third virtualized domain controller executes the non-redundant safety-critical nodes.

physical separation. The system is divided into two zones (front and back) plus a star backbone network and physically separated central units. Each zone contains parts of both safety-critical and non-safety critical applications, compared to the domain-based topology in which the domain controllers only have application nodes belonging to the same domain and so with the same ASIL requirement. For this reason, the zone controllers, the Ethernet switch 1 and 2, the Ethernet 3 and 6, the backbone network, the central ECU, and the CAN buses 1 and 2 are physically separated.

Figure 5.33 shows instead the resource layer of the zone-based architecture when using virtualization.

Figure 5.34 shows the part of the zone-based architectures with virtualization that is transformed with the transformation process that substitutes the LLSC node with redundant ones. Four different virtual resources are used: one for the non-safety-critical applications, one for the safety-critical non-redundant application nodes, and two for the safety-critical redundant application nodes.

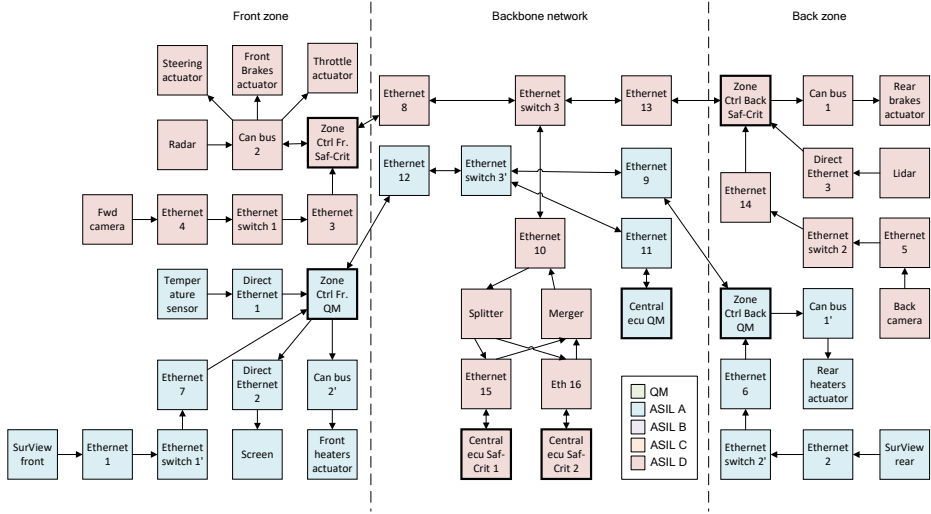


Figure 5.32: Resource layer in the zone-based architecture with physical separation after applying the transformation process to the LLSC node.

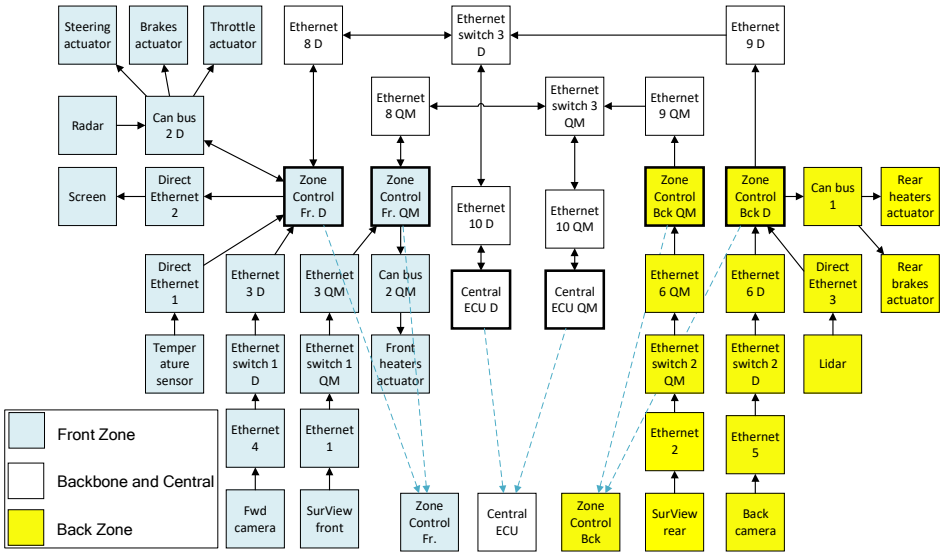


Figure 5.33: Resource layer in the zone-based architecture with virtualization before applying the transformation process to the LLSC node.

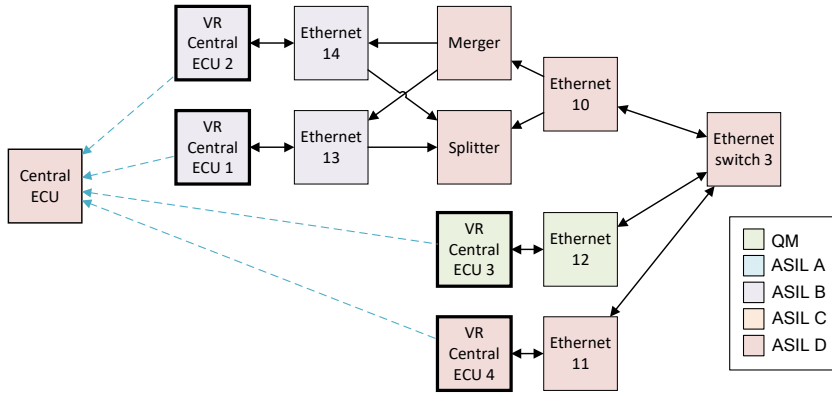


Figure 5.34: Four virtualized ECUs are used to execute the application nodes in zone-based virtualized architectures.

Table 5.1 summarizes the effects of the transformation to the applications in the different scenarios.

5.4.2 Evaluation of the four implementations

We evaluate the four different implementations by using the analysis described in Chapter 4. The first parameters that we analyse are the failure probability of the safety-critical application and the cost of the system. Their values are shown in Figure 5.35. The zone-based topology with physical separation is the most expensive since all the controllers and the central unit must be duplicated for both redundancy and mixed-criticality, which in the virtualized solution become Virtual Resources on the same resource.

In terms of failure probability, the best architecture topology is in this case the Zone Phys, which executes redundantly on the central ECUs most of the safety-critical application (pre-processing, processing, and post-processing parts). Since in this scenario the splitter and the merger failure events are dominant in the fault tree, the failure probability of the Zone Phys is lower than the non-redundant solution. This is because of the failure rate metric chosen in Chapter 4: splitters and mergers have a failure rate 10 times lower than other resources. In case of virtualization instead, part of the safety-critical application is executed non-redundantly in the ASIL D virtual central ECU, which has a higher failure rate compared to the splitter and merger resources, and thus the failure probability is in Zone Virt 3.7% higher than Zone Phys, while being 14.5% cheaper.

For domain-based topologies, part of the application (the central data fusion node) is always executed in the central ECU, in a non-redundant and non-virtualized way. The failure rates of these topologies are always higher than the zone-based ones since this part of the application does not benefit from using

Table 5.1: Summary of the system modifications in physical separation and virtualization scenarios in the presence of redundancy in the LLSC application node.

Architecture	Physical separation	Virtualization
Domain-based	(Dom Phys) The nodes that require isolation are mapped to the physically separate redundant safety-critical controllers, which have lower ASIL specifications due to the ASIL decomposition technique. The other nodes executed in the controller are redundant as well and require substitutions during the transformation process.	(Dom Virt) The redundant LLSC nodes are mapped to separate virtual domain controllers, which are generated by a safety-critical domain controller that runs a virtualization mechanism and have lower ASIL specification thanks to the ASIL decomposition. The other nodes executed by the original domain controller are mapped to a third virtual resource with ASIL D (original) specifications.
Zone-based	(Zone Phys) Multiple zone controllers and central ECUs are present in the resource layer because of mixed-criticality. The safety-critical central ECU is substituted during the transformation process and the nodes that were mapped on it require substitution during the transformation process.	(Zone Virt) Virtual resources are used for both mixed-criticality and isolation of redundant nodes. Four virtual central ECUs are used: one for the non-safety-critical applications, one for the safety-critical non-redundant nodes, and two redundant ones for the redundant safety-critical nodes. Note: if more ASILs were present, even more VRs would be required.

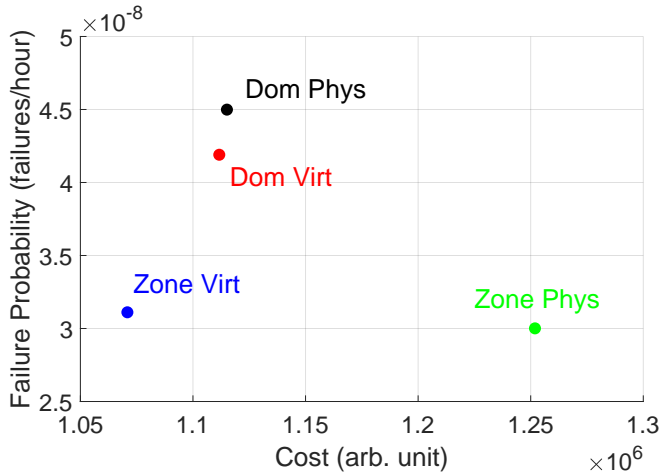


Figure 5.35: Failure probability (failures/hour) versus Cost of the four implementations.

safety-oriented resources with a lower failure probability. Compared to the zone-based topologies, when using physical separation the failure probability is higher than when using virtualization: this effect appears because when the domain-controller is substituted with redundant ones, an additional splitter or merger is necessary for each predecessor or successor of the resource, and the domain controller is connected to many different communication resources. While for zone-based topologies the central ECU is connected only to the backbone network and a single splitter and a single merger are needed, in our domain-based topologies four splitters and two mergers are needed. In the case of virtualization, again only one splitter and one merger are used since we introduce a third virtual resource with ASIL D to map the non-redundant safety-critical application nodes. In general, when the resource that is being substituted has a lot of connectivity, virtualization helps in reducing the complexity and the number of splitter and mergers necessary in the system (even when they are mapped to separate resources as described in Section 3.3.4), and those are directly contributing to the failure probability of the application.

Figure 5.36 shows the communication and functional loads of the system based on the values given to each application node. The zone-based topologies, despite their lower failure probabilities, have higher total communication and functional loads: more raw data (with high bandwidth) from the sensors is transmitted towards the redundant central units, which are also performing the most computational heavy tasks redundantly. Virtualization does reduce these two parameters in zone-based topologies since the preprocessing and the central data fusion parts are executed non-redundantly in the virtual ASIL D central ECU. The communication

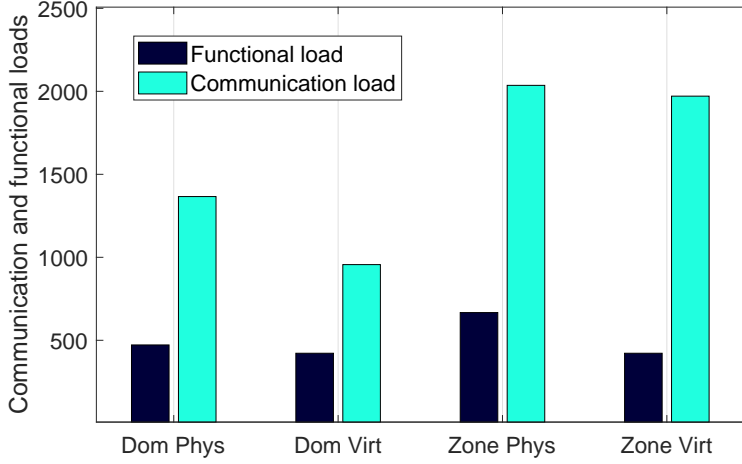


Figure 5.36: Total communication load and total functional load of the four implementations.

load is still higher than in the respective domain-based topology. The functional load is instead equal to that of the respective domain-based topology since the same number of functional nodes is redundant.

In the domain-based scenario, the sensor raw data stops at the domain controllers to be processed, resulting in lower utilization of the backbone network and thus lower total communication load. Moreover, virtualization allows keeping part of the application that was mapped on the original safety-critical domain controller in a non-redundant way on a virtual ASIL D resource, lowering both the functional and the communication load.

Figure 5.37 shows the total cable length based on the normalized 2D coordinates associated with the physical layer. Another advantage of zone-based topologies is a lower total cable length. Moreover, virtualization removes the need to duplicate communication links, thereby requiring fewer resources, and hence a lower total cable length.

With these experiments, we show that depending on the characteristics of the system, one of the two isolation techniques is preferable to the other. In our scenario, virtualization has clear advantages in terms of the cost of the final system, since the cost of the virtualization mechanism is lower compared to the use of multiple separate resources. Moreover, by using virtual resources to isolate the part of the application that requires redundancy from the rest of the nodes mapped on the same resource, the modifications that are required to be made to the application are contained and the total functional and communication loads are lower compared to using physically separated resources. With fewer resources, also the total cable length is lower when using virtualization. However, depending

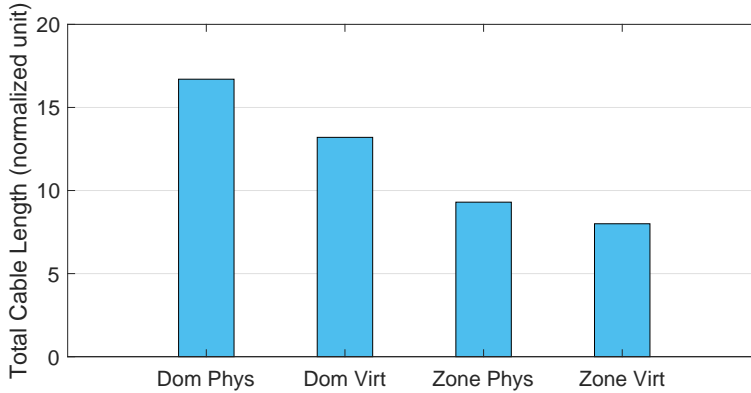


Figure 5.37: Total communication cable length of the four implementations.

on the system configuration, the failure probability of virtualized systems can be higher than physically separated ones. For example, this happens in our zone-based scenario: the combined failure probability of the additional virtualization mechanism and the splitter and merger resources is higher than the failure probability of the physically separated system, which only required one splitter and one merger.

5.5 Conclusions

In this chapter, we presented an analysis of domain and zone-based automotive architecture topologies, with vehicle-centralized or controller-based mappings. The automotive system is modelled and analysed with the three-layer model presented in Chapter 3. The developed framework allows a system designer to introduce redundancy in selected nodes of the system, with an automated procedure that follows the ISO26262 ASIL decomposition guidelines, as described in Chapter 4.

We first assume that the application nodes are *fail-silent* so that sharing a resource does not impact the failure probability of an application. Our results show how introducing redundancy impacts the chosen architecture topologies. In our experiments, when redundancy is required, the domain-based controller-based (D-CB) topology offers the best balance between the analysed parameters in the case of fail-silent nodes. The zone-based topologies excel instead in terms of total communication cable lengths. We expect hybrid solutions to appear in the future with zones for some applications (non-safety-critical, e.g. body and comfort functions) and separate domains with their isolated domain controller for others (safety-critical, e.g. driver replacement ADAS).

We then discard the *fail-silent* assumption, and use two isolation techniques, physical separation and virtualization, to separate non-fail-silent application nodes

so that they do not interfere with each other. Moreover, we use virtualization to isolate the redundant nodes with decomposed ASIL requirements, since VRs provide independence that is necessary for the redundancy. Two architecture topologies with redundant applications are analysed when using these two techniques. We identify trends in the different topologies when introducing redundancy or applying isolation techniques. For example, virtualization results in a reduction of the total communication cable length, the functional load, and the communication load. In our experiments, the Zone Virt topology is the best in terms of total communication cable length and cost, while if the optimization parameter is different, another topology is recommended, e.g. Dom Virt for the minimum communication load.

Based on the architecture topology and the system configuration, the effects of physical separation and virtualization can vary. A complete analysis must be performed, as we did in this chapter for our systems, before choosing which type of isolation technique and architecture topology is better for a different system since many factors will impact the final results.

Note that the results of our experiments are obtained with the cost and failure rates metrics described in Chapter 4. For example, the failure probability of the system in domain-based topologies when virtualization is applied is reduced, despite introducing additional Common-Cause Fault (CCF) possibilities due to the use of virtualization. For example, with a higher failure rate of the application nodes mapped to the VRs (i.e. due to an increased sharing of the VRs), the results of the analysis would change and could, for example, exceed the maximum failure probability that an ASIL D application can have. However, our method is general and by analysing each system we can evaluate and quantify the different solutions.

6

Conclusion and future directions

6.1 Conclusions

With this work, we discussed multiple topics related to emerging automotive systems. We defined a model to describe an automotive system and created a tool to analyse it with a quantitative analysis. The evaluation consists of the calculation of five parameters: failure probability of an application, cost of the system, total communication cable length, total functional and communication loads. Moreover, a Common-Cause Fault (CCF) analysis is performed on the generated fault trees to validate the independence requirements of redundant elements (Chapter 3). By following the guidelines of the ISO26262 standard we can modify a modelled system and obtain redundant applications and hardware resources in compliance with the ASIL decomposition technique (Chapter 4). The insights gained with this work help us define the benefits and costs of upcoming automotive systems and give us intuitions for characteristics necessary in future automotive resources. The discussion about the *splitter* and *merger* elements emphasises the similarities between redundancy techniques, which are necessary for fail-operational safety-critical systems. We use our framework to evaluate different use-cases. An in-depth analysis of different architecture solutions is performed to identify which architecture topologies will be prevalent in the future. Last, we analyse how, without a fail-silent assumption, mixed-criticality nodes could be a problem for safety-critical systems, and we describe isolation techniques to satisfy the Functional-Safety Requirement (FSR) of safety-critical systems (Chapter 5).

While developing our framework we encountered many challenges. First of all,

the variety of automotive applications, with completely different requirements in terms of performance, safety, bandwidth, etc. makes it difficult to generalize them with a single model. The key challenge is choosing the correct abstraction level, in which enough details are available to perform meaningful analysis, but some of the more specific technical details are left out. For example, we encountered this when we generalized the failure rates of applications and resources, since the low-level parts that form the final failure rate are very specific technical details that are often dependent on the actual components that are used. Moreover many different types of resources are available: multi-core Electronic Control Units (ECUs), network components, communication cables, power supplies, sensors, actuators, etc. With our analysis, we abstract from the real component that is used, and for example, we do not differentiate the cores in a multi-core processor. The resource layer can be expanded with more technical details if the functional safety analysis requires it, however the lower the level of details, the higher the number of special cases in the analysis and the introduction of redundancy with the model transformations.

The model transformations that we use to introduce redundancy in the system are intuitive when applied to small-scale projects, but once the system has multiple applications and many resources with possibly multiple domains and controllers, manually tracking all the modifications that the system requires when making part of it redundant becomes a difficult task. Our automated framework allows the user to immediately see the effects of such transformations even in a more complex system, and the analysis provides insightful numbers to understand if the transformation improves the current system or not. While the proposed transformations do not cover the vast range of redundancy patterns that one can imagine, the transformation process describes all the steps that are necessary to generate a new and redundant valid graph. With adaptations of the base transformations and the transformation process, even different redundancy patterns can be obtained, as described in Chapter 4.

Discussion over the high-level research question

At the beginning of this thesis, our initial research question was: *How can we model fail-operational and mixed-critical automotive systems so that 1) the Electrical and Electronic (E/E) architecture is evaluated together with the functional safety of the system, 2) quantitative metrics can be used to compare different solutions, and 3) fail-operational capabilities can be introduced in the system at need?*

The answers to these questions are found in the different chapters that describe our model and analysis framework. First of all, we propose a model that allows for the evaluation of different parameters of an automotive system, while at the same time performing a functional safety evaluation related to the Automotive Safety Integrity Level (ASIL) requirements of the applications and the resources specifications. The parameters that we choose to calculate allow us to provide system-level details related to different system implementations. With our discussion, we highlight the necessary details that the model requires to provide a meaningful

analysis. While we focus on functional safety and system-level properties of the system, other analyses can be performed, e.g. performance analysis. Our open framework is adaptable and additional properties can be added to the different layers to perform different types of analyses.

In terms of fail-operational capabilities, our transformations model heterogeneous redundancy in the system and introduce splitters and mergers with safety-critical characteristics to manage the redundancy. As discussed in Chapter 2, there are many redundancy patterns and different ways to implement fail-operational systems. With the proposed transformation process, we provide the details for one of them, and with minor adaptations to the process even different patterns can be introduced in the system.

We have analysed mixed-critical applications with no fail-silent application nodes separately from the rest of the experiments. The effects of sharing the same resource between applications with different ASIL requirements are critical when the *fail-silent* assumption is not valid. The techniques that are used to isolate the safety-critical applications affect the E/E architecture by either increasing the number of resources or by requiring virtualization. The five analysis parameters vary depending on the topology and isolation technique that is used.

While abstracting from the low-level technical details of the implementations, the system model must remain realistic and usable. The decisions that we made in its description are made to generalize as much as possible and describe all types of automotive systems, while still generating interesting and non-trivial results.

We have discussed how most of the analysis parameters are highly dependent on the system configuration, both in terms of applications and resources that are used, their mapping, the architecture topology that is chosen, and the input metrics. Even generic and intuitive assumptions are not always correct: for example, we have seen how zone-based architecture topologies, which are built to minimize the total cable length, do not always achieve that objective in the presence of redundancy. For these reasons, we strongly believe that a complete analysis framework is necessary to analyse the different implementations of an Advanced Driving Assistance System (ADAS) or Autonomous Vehicle (AV) system, to provide quantitative and correct results.

6.2 Future directions and possible extensions of the framework

While in this work we created a stand-alone framework, its extension or integration with other tools can be done to improve and extend the analysis that is currently performed:

- **Fail-silent system elements:** in this work we have analysed applications with and without *fail-silent* nodes. However, a similar behaviour can be expected by resources sharing locations or other common dependencies such

as the power supply. Moreover, in the current model, the failure of a dependency propagates to the successor elements, but we are not considering the scenario in which the failure of an element affects its predecessors. In essence, each edge present in the three-layer model creates a dependency that allows faults to propagate to the rest of the system, and each edge can have a fail-silent or non-fail-silent behaviour. While it is possible to capture failure rates of application nodes, it is more difficult to define failure rates for failure of the resources that would lead to structural problems affecting other elements (e.g. short-circuits or explosion of batteries). For a complete functional-safety analysis of a safety-critical system, all these dependencies shall be taken into account.

- **Performance analysis of an application:** while the current analysis evaluates five system-level parameters, the model allows for the addition of other parameters, such as latency or throughput of an application. The model uses directed graphs, which are used in other models of computations such as Dataflow. Another option would be to integrate the safety analysis and the transformation process developed in this work with a tool that performs timing analysis, e.g. the network simulator OMNeT++.
- **Generation of fault tree pattern for different types of mergers:** as mentioned in Section 3.4.1, the implemented fault tree generation algorithm assumes that all mergers are data oriented, in-band, or data evaluation mergers. To support the generation of the fault tree pattern of additional merger types, an extension of the framework is required. The absolute numbers in the results of the experiments would change, but the considerations related to the architecture topologies remain valid.
- **Automatic design space exploration:** in the described framework, we manually select the points in the applications in which we want to introduce redundancy, then the system is automatically transformed with the transformation process. However, by using the analysis parameters as optimization metrics, an automatic design space exploration process can be performed. Such a procedure could identify implementations with a specific number of redundant elements that correspond to minima in the optimization cost metric. While this procedure is trivial for very small applications, the lack of scalability of automated design space exploration techniques is one of the main limitations of such methods.
- **Optimized mapping of transformed elements:** many works in the literature focus on the mapping and assignment of tasks to the hardware resources. Automating this process is a complex problem in terms of scalability, but in our case, only a limited number of nodes or resources require remapping during the transformation process. Currently, we add or modify existing resources to accommodate the new redundant nodes, but this

process can be optimized to identify existing resources to which the nodes can be mapped, respecting the resource utilization limitations, the ASIL requirements of the nodes compared to the specification of the resource, and the ASIL decomposition independence requirements. By optimizing the mapping of the applications, no unnecessary resources are used and the cost of the system is lower than unoptimized implementations.

Bibliography

- [1] N. Adler. PREEvision. Designing advanced systems - Safely! Model-based E/E development conforming to ISO26262, 2018.
- [2] M. Aeberhard, T. Kuehbeck, and B. Seidl. Automated driving with ROS at BMW. In *ROSCon2015*, 2015.
- [3] AltaRica. Xfta <http://www.altarica-association.org/members/araury/Software/XFTA/XFTA2.html> Last access: 2022-02-06.
- [4] M. Althoff, D. Althoff, D. Wollherr, and M. Buss. Safety verification of autonomous vehicles for coordinated evasive maneuvers. In *2010 IEEE Intelligent Vehicles Symposium*, pages 1078–1083, 2010.
- [5] J. Andrews and L. Bartlett. Efficient basic event orderings for binary decision diagrams. In *Annual Reliability and Maintainability Symposium. 1998 Proceedings. International Symposium on Product Quality and Integrity*, pages 61–68, 1998.
- [6] J. Andrews and R. Remenyte. Fault tree conversion to binary decision diagrams. In *Proceedings of the 23rd ISSC*, 2005.
- [7] L. Apvrille and A. Becoulet. Prototyping an embedded automotive system from its UML/SysML models. In *Embedded Real Time Software and Systems (ERTS2012)*, 2012.
- [8] A. Armoush. Design patterns for safety-critical embedded systems. PhD Thesis, RWTH Aachen University. 2010.
- [9] A. Armoush, E. Beckschulze, and S. Kowalewski. Safety assessment of design patterns for safety-critical embedded systems. In *2009 35th Euromicro Conference on Software Engineering and Advanced Applications*, pages 523–527, 2009.
- [10] AUTOSAR. <https://www.autosar.org/>, 2021. Last access: 2021-06-08.
- [11] AUTOSAR UML profile. https://www.autosar.org/fileadmin/user_upload/standards/classic/3-1/AUTOSAR_TemplateModelingGuide.pdf, 2009. Last access: 2021-06-08.

- [12] F. Auzanneau. Wire troubleshooting and diagnosis: Review and perspectives. *Progress In Electromagnetics Research*, 49:253–279, 2013.
- [13] L. d. S. Azevedo, D. Parker, M. Walker, Y. Papadopoulos, and R. E. Araújo. Assisted assignment of automotive safety requirements. *IEEE Software*, 31(1):62–68, 2014.
- [14] L. S. Azevedo, D. Parker, M. Walker, Y. Papadopoulos, and R. Esteves Araújo. Automatic decomposition of safety integrity levels: Optimization by tabu search. In M. ROY, editor, *SAFECOMP 2013 - Workshop CARS (2nd Workshop on Critical Automotive applications : Robustness & Safety) of the 32nd International Conference on Computer Safety, Reliability and Security*, page NA, Toulouse, France, Sept. 2013.
- [15] J. Bach, S. Otten, and E. Sax. A taxonomy and systematic approach for automotive system architectures: from functional chains to functional networks. In *International Conference on Vehicle Technology and Intelligent Transport Systems*, volume 2, pages 90–101. SCITEPRESS, 2017.
- [16] Baidu. Apollo, <https://apollo.auto>, 2020. Last access: 2021-06-08.
- [17] M. Baleani, A. Ferrari, L. Mangeruca, A. Sangiovanni-Vincentelli, M. Peri, and S. Pezzini. Fault-tolerant platforms for automotive safety-critical applications. CASES '03, pages 170–177, New York, NY, USA, 2003. Association for Computing Machinery.
- [18] V. Bandur, V. Pantelic, T. Tomashevskiy, and M. Lawford. A safety architecture for centralized e/e architectures. In *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 67–70, 2021.
- [19] P. Bernardi, M. Grosso, E. Sanchez, and O. Ballan. Fault grading of software-based self-test procedures for dependable automotive applications. In *2011 Design, Automation Test in Europe*, pages 1–2, 2011.
- [20] S. Bernardi and J. Merseguer. A UML profile for dependability analysis of real-time embedded systems. In *Proceedings of the 6th International Workshop on Software and Performance*, WOSP '07, pages 115–124, New York, NY, USA, 2007. Association for Computing Machinery.
- [21] A. Bhat, S. Samii, and R. Rajkumar. Practical task allocation for software fault-tolerance and its implementation in embedded automotive systems. *Real-Time Systems*, 55(4):889–924, 2019.
- [22] Z. E. Bhatti, P. S. Roop, and R. Sinha. Unified functional safety assessment of industrial automation systems. *IEEE Transactions on Industrial Informatics*, 13(1):17–26, 2016.

- [23] T. Bijlsma and T. Hendriks. A fail-operational truck platooning architecture. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 1819–1826. IEEE, 2017.
- [24] T. Bijlsma, T. Hendriks, J. Vissers, L. Elshof, T. Jansen, and B. Krosse. In-vehicle architectures for truck platooning: The challenges to reach SAE automation level 3. In *Proc. of ITS World Congress*, 2016.
- [25] BlackBerry. QNX, <https://blackberry.qnx.com/> Last access: 2021-07-19.
- [26] Bosch Mobility Solutions. Redundancy: a critical enabler for fully automated driving. From Bosch press release, <https://www.bosch-mobility-solutions.us/us/highlights/automated-mobility/redundancy-for-automated-driving/> Last access: 2021-06-02.
- [27] I. Broster, A. Burns, and G. Fohler. The babbling idiot in event-triggered real-time systems. In *Proceedings of the Work-In-Progress Session, 22nd IEEE Real-Time Systems Symposium, YCS*, volume 337, pages 25–28. Citeseer, 2001.
- [28] O. Burkacky, M. Kellner, J. Deichmann, P. Keuntje, and J. Werra. Rewiring car electronics and software architecture for the ‘Roaring 2020s’. *McKinsey article*, 2021.
- [29] C. Campolo, A. Molinaro, A. Iera, and F. Menichella. 5G network slicing for vehicle-to-everything services. *IEEE Wireless Communications*, 24(6):38–45, 2017.
- [30] F. Caron, E. Duffos, D. Pomorski, and P. Vanheeghe. GPS/IMU data fusion using multisensor Kalman filtering: introduction of contextual aspects. *Information Fusion*, 7(2):221–230, 2006.
- [31] L. Caudet, V. Von Hammerstein-Gesmold, and M. Talko. Road safety: Commission welcomes agreement on new EU rules to help save lives. Press release, https://ec.europa.eu/commission/presscorner/api/files/document/print/en/ip_19_1793/IP_19_1793_EN.pdf Last access: 2021-06-02, 2019.
- [32] S. Chakraborty, M. Di Natale, H. Falk, M. Lukasiewicz, and F. Slomka. Timing and schedulability analysis for distributed automotive control applications. In *2011 Proceedings of the Ninth ACM International Conference on Embedded Software (EMSOFT)*, pages 349–350, 2011.
- [33] Y.-C. Chang, L.-R. Huang, H.-C. Liu, C.-J. Yang, and C.-T. Chiu. Assessing automotive functional safety microprocessor with ISO 26262 hardware requirements. In *Technical Papers of 2014 International Symposium on VLSI Design, Automation and Test*, pages 1–4, 2014.

- [34] X. Chen, J. Feng, M. Hiller, and V. Lauer. Application of software watchdog as a dependability software service for automotive safety relevant systems. In *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 618–624, 2007.
- [35] A. Cherfi, M. Leeman, F. Meurville, and A. Rauzy. Modeling automotive safety mechanisms: a Markovian approach. *Reliability Engineering & System Safety*, 130:42–49, 2014.
- [36] D. T. Chiang and S.-C. Niu. Reliability of consecutive-k-out-of-n:f system. *IEEE Transactions on Reliability*, R-30(1):87–89, 1981.
- [37] J. Choi, V. Va, N. Gonzalez-Prelcic, R. Daniels, C. R. Bhat, and R. W. Heath. Millimeter-wave vehicular communication to support massive automotive sensing. *IEEE Communications Magazine*, 54(12):160–167, 2016.
- [38] J. Cui, L. S. Liew, G. Sabaliauskaite, and F. Zhou. A review on safety failures, security attacks, and available countermeasures for autonomous vehicles. *Ad Hoc Networks*, 90:101823, 2019. Recent advances on security and privacy in Intelligent Transportation Systems.
- [39] S. D'Angelo, C. Metra, S. Pastore, A. Pogutz, and G. Sechi. Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems. In *Proceedings 1998 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (Cat. No.98EX223)*, pages 233–240, 1998.
- [40] N. Das and W. Taylor. Quantified fault tree techniques for calculating hardware fault metrics according to iso 26262. In *2016 IEEE Symposium on Product Compliance Engineering (ISPCe)*, pages 1–8. IEEE, 2016.
- [41] J. Davis. ISO 26262 Independence and Related Terms, from <https://www.exida.com/Blog/iso-26262-independence-and-related-terms>. Last access: 2021-06-08, 2020.
- [42] P. Derler, T. H. Feng, E. A. Lee, S. Matic, H. D. Patel, Y. Zheo, and J. Zou. PTIDES: A programming model for distributed real-time embedded systems. Technical report, California University Berkley, Department of Electrical Engineering and Computer Science, 2008.
- [43] M. S. Dhouibi, L. Saintis, M. Barreau, and J.-M. Perquis. Automatic decomposition and allocation of safety integrity level using system of linear equations. In *PESARO 2014, The Fourth International Conference on Performance, Safety and Robustness in Complex Systems and Applications*, 2014.
- [44] M. Di Paolo Emilio. Under the hood: the innovation-rich golf 8, from <https://www.eetimes.com/under-the-hood-the-innovation-rich-golf-8/>, 2021.

- [45] K. Driscoll, B. Hall, M. Paulitsch, P. Zumsteg, and H. Sivencrona. The real byzantine generals. In *The 23rd Digital Avionics Systems Conference*, volume 2, pages 6.D.4–61, 2004.
- [46] P. Dubrulle, C. Gaston, N. Kosmatov, A. Lapitre, and S. Louise. A data flow model with frequency arithmetic. In R. Hähnle and W. van der Aalst, editors, *Fundamental Approaches to Software Engineering*, pages 369–385, Cham, 2019. Springer International Publishing.
- [47] A. Dutta, S. Alampally, A. Kumar, and R. A. Parekhji. A BIST implementation framework for supporting field testability and configurability in an automotive SOC. In *Workshop on Dependable and Secure Nanocomputing*, 2007.
- [48] Eclipse Papyrus. Modeling environment, <https://www.eclipse.org/papyrus/> Last access: 2021-06-02.
- [49] S. Epstein and A. Rauzy. Open-PSA Model Exchange Format. <https://open-psa.github.io/mef/> Last access: 2022-02-05.
- [50] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.
- [51] European Committee for Electrotechnical Standardization. Railway applications - communication, signalling and processing systems - Software for railway control and protection systems. EN 50128:2011/A1:2020. Standard, CENELEC, 2020.
- [52] European Organisation for Civil Aviation Equipment. Software Considerations in Airborne Systems and Equipment Certification. DO 178C/ED-12C. Standard, EUROCAE, 2012.
- [53] J. Farkas, L. L. Bello, and C. Gunther. Time-Sensitive Networking standards. *IEEE Communications Standards Magazine*, 2(2):20–21, 2018.
- [54] P. Franco and E. McCluskey. On-line delay testing of digital circuits. In *Proceedings of IEEE VLSI Test Symposium*, pages 167–173, 1994.
- [55] A. Frigerio, B. Vermeulen, and K. Goossens. A generic method for a bottom-up ASIL decomposition. In *Computer Safety, Reliability, and Security (SAFECOMP)*, pages 12–26, 2018.
- [56] A. Frigerio, B. Vermeulen, and K. Goossens. Component-level ASIL decomposition for automotive architectures. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 62–69, 2019.

- [57] A. Frigerio, B. Vermeulen, and K. Goossens. Automotive architecture topologies: Analysis for safety-critical autonomous vehicle applications. *IEEE Access*, 9:62837–62846, 2021.
- [58] A. Frigerio, B. Vermeulen, and K. Goossens. Isolation of redundant and mixed-critical automotive applications: effects on the system architecture. In *Vehicle Technology Conference - Spring (VTC)*, 2021.
- [59] B. Gassmann, F. Oboril, C. Buerkle, S. Liu, S. Yan, M. S. Elli, I. Alvarez, N. Aerrabotu, S. Jaber, P. van Beek, D. Iyer, and J. Weast. Towards standardization of AV safety: C++ library for Responsibility Sensitive Safety. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 2265–2271, 2019.
- [60] General Motors. Self-driving safety report, 2018.
- [61] M. Ghadhab, S. Junges, J.-P. Katoen, M. Kuntz, and M. Volk. Model-based safety analysis for vehicle guidance systems. In *International Conference on Computer Safety, Reliability, and Security*, pages 3–19. Springer, 2017.
- [62] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [63] W. Granig, D. Hammerschmidt, and H. Zangl. Calculation of failure detection probability on safety mechanisms of correlated sensor signals according to ISO 26262. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 10(1):144–155, mar 2017.
- [64] R. Grave. The vehicle architecture of automated driving level 2/3. Whitepaper, Elektrobit, 2017.
- [65] M. Green and J. Senders. Human error in road accidents. *Visual Expert*, 2004.
- [66] GreenHills. INTEGRITY multivisor, https://www.ghs.com/products/rtos/integrity_virtualization.html Last access: 2021-07-19.
- [67] GuardKnox. Zonal architecture: the foundation for next-generation vehicles. Whitepaper, GuardKnox, 2020.
- [68] R. Hammett. Design by extrapolation: an evaluation of fault tolerant avionics. *IEEE Aerospace and Electronic Systems Magazine*, 17(4):17–25, 2002.

- [69] S. Hasirlioglu, A. Kamann, I. Doric, and T. Brandmeier. Test methodology for rain influence on automotive surround sensors. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 2242–2247, 2016.
- [70] R. Heinzler, P. Schindler, J. Seekircher, W. Ritter, and W. Stork. Weather influence and classification with automotive lidar sensors. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 1527–1534, 2019.
- [71] C. Hensel, S. Junges, J.-P. Katoen, T. Quatman, and M. Volk. Storm Checker <https://www.stormchecker.org/> Last access: 2022-02-06.
- [72] C. Herber, A. Richter, T. Wild, and A. Herkersdorf. A network virtualization approach for performance isolation in Controller Area Network (CAN). In *2014 IEEE 19th Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 215–224, 2014.
- [73] B. Hu, S. Xu, Z. Cao, and M. Zhou. Safety-guaranteed and development cost-minimized scheduling of DAG functionality in an automotive system. *IEEE Transactions on Intelligent Transportation Systems*, 2020.
- [74] IBM. Engineering Systems Design Rhapsody, <https://www.ibm.com/products/systems-design-rhapsody> Last access: 2021-06-02.
- [75] INET Framework. <https://inet.omnetpp.org/>, 2021. Last access: 2021-06-08.
- [76] Institute of Electrical and Electronics Engineers. Frame replication and elimination for reliability. P802.1CB. Standard, IEEE, 2017.
- [77] Intel newsroom. Baidu to integrate Mobileye’s Responsibility Sensitive Safety model into Apollo program, from <https://newsroom.intel.com/news/baidu-integrate-mobileyes-responsibility-sensitive-safety-model-apollo-program/>, 2018. Last access: 2021-06-08.
- [78] International Electronic for Commission. Medical device software - Software life cycle processes. IEC 62304:2006. Standard, IEC, 2006.
- [79] International Electronic for Commission. Functional safety of electrical / electronic / programmable electronic safety-related systems. IEC 61508:2010. Standard, IEC, 2010.
- [80] International Organization for Standardization. Systems and software engineering - architecture description. ISO/IEC/IEEE 42010:2011. Standard, ISO/IEC/IEEE, 2011.
- [81] International Organization for Standardization. Road vehicles - functional safety. ISO 26262:2018. Standard, ISO, 2018.

- [82] Isograph. Faulttree+ <https://www.isograph.com/software/reliability-workbench/fault-tree-analysis-software/fault-tree-analysis> Last access: 2022-02-06.
- [83] K. Jo, J. Kim, D. Kim, C. Jang, and M. 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.
- [84] S. Kabir. An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications*, 77:114–135, 2017.
- [85] M. Kamali, L. A. Dennis, O. McAree, M. Fisher, and S. M. Veres. Formal verification of autonomous vehicle platooning. *Science of computer programming*, 148:88–106, 2017.
- [86] L. M. Kinnan. Use of multicore processors in avionics systems and its potential impact on implementation and certification. In *2009 IEEE/AIAA 28th Digital Avionics Systems Conference*, pages 1.E.4–1–1.E.4–6, 2009.
- [87] T. Kogan, Y. Abotbol, G. Boschi, G. Harutyunyan, I. Kroul, H. Shaheen, and Y. Zorian. Advanced functional safety mechanisms for embedded memories and IPs in automotive SoCs. In *2017 IEEE International Test Conference (ITC)*, pages 1–6, 2017.
- [88] P. Koopman. The big picture for Self-Driving Car safety. In *Keynote at Safe Systems Summit: Redefining Transportation Safety*, 2019.
- [89] P. Koopman, U. Ferrell, F. Fratrik, and M. Wagner. A Safety Standard Approach for Fully Autonomous Vehicles. In *SAFECOMP 2019. Lecture Notes in Computer Science*, volume 11699, 2019.
- [90] P. K. Lala. Fault tolerant and fault testable hardware design. Prentice-Hall, Inc., USA, 1985.
- [91] P. K. Lala. Self-checking and fault-tolerant digital design. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2000.
- [92] H. E. Lambert. Use of fault tree analysis for automotive reliability and safety analysis. *SAE transactions*, pages 690–696, 2004.
- [93] C. Lauer, R. German, and J. Pollmer. Fault tree synthesis from UML models for reliability analysis at early design stages. 36(1):1–8, Jan. 2011.
- [94] E. A. Lee and D. G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235–1245, 1987.

- [95] C. Lidström, C. Bondesson, M. Nyberg, and J. Westman. Improved pattern for ISO 26262 ASIL decomposition with dependent requirements. In *2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 28–35, 2019.
- [96] C.-W. Lin, L. Rao, P. Giusto, J. D’Ambrosio, and A. L. Sangiovanni-Vincentelli. Efficient wire routing and wire sizing for weight minimization of automotive systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1730–1741, 2015.
- [97] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang, and J. Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 751–766, 2018.
- [98] X. Lingyun and G. Feng. A comprehensive review of the development of adaptive cruise control systems. *Vehicle System Dynamics*, 48(10):1167–1192, 2010.
- [99] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [100] J. Losq. A highly efficient redundancy scheme: Self-purging redundancy. *IEEE Transactions on Computers*, C-25(6):569–578, 1976.
- [101] Y. Luo, A. K. Saberi, T. Bijlsma, J. J. Lukkien, and M. van den Brand. An architecture pattern for safety critical automated driving applications: Design and analysis. In *2017 Annual IEEE International Systems Conference (SysCon)*, pages 1–7, 2017.
- [102] P. Mallozzi, M. Sciancalepore, and P. Pelliccione. Formal verification of the on-the-fly vehicle platooning protocol. In *International Workshop on Software Engineering for Resilient Systems*, pages 62–75. Springer, 2016.
- [103] M. L. McKelvin Jr, G. Eirea, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli. A formal approach to fault tree synthesis for the analysis of distributed fault tolerant systems. In *Proceedings of the 5th ACM international conference on Embedded software*, pages 237–246, 2005.
- [104] S. Mukherjee, M. Kontz, and S. Reinhardt. Detailed design and evaluation of redundant multi-threading alternatives. In *Proceedings 29th Annual International Symposium on Computer Architecture*, pages 99–110, 2002.
- [105] P. Münzing, A. OstertagBertsche, and O. Koller. Automated ASIL allocation and decomposition according to ISO 26262, using the example of vehicle

- electrical systems for automated driving. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 11(2):123–130, apr 2018.
- [106] A. Murashkin, L. Silva Azevedo, J. Guo, E. Zulkoski, J. H. Liang, K. Czarnecki, and D. Parker. Automated decomposition and allocation of automotive safety integrity levels using exact solvers. *SAE International Journal of Passenger Cars - Electronic and Electrical Systems*, 8(1):70–78, apr 2015.
 - [107] A. Nardi and A. Armato. Functional safety methodologies for automotive applications. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 970–975, 2017.
 - [108] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang. An introduction to the Safety Force Field. Whitepaper, NVIDIA, 2019.
 - [109] D. Nistér, H.-L. Lee, J. Ng, and Y. Wang. The Safety Force Field. Whitepaper, NVIDIA, 2019.
 - [110] NVIDIA. Drive PX, https://www.nvidia.com/content/nvidiaGDC/sg/en_SG/self-driving-cars/drive-px/ Last access: 2021-06-02.
 - [111] NXP. Bluebox automotive high performance compute development platform, <https://www.nxp.com/design/development-boards/automotive-development-platforms/nxp-bluebox-3-0-automotive-high-performance-compute-ahpc-development-platform:BlueBox> Last access: 2021-06-02.
 - [112] NXP. Investor Day 2018, <https://investors.nxp.com/static-files/9dc92dd5-6568-4e08-89bd-5a7708f289fa> Last access: 2021-07-19.
 - [113] N. Oh, S. Mitra, and E. McCluskey. ED⁴I: error detection by diverse data and duplicated instructions. *IEEE Transactions on Computers*, 51(2):180–199, 2002.
 - [114] OMNeT++. <https://omnetpp.org/>, 2021. Last access: 2021-06-08.
 - [115] F. Oszwald, P. Obergfell, M. Traub, and J. Becker. Reliable fail-operational automotive E/E-architectures by dynamic redundancy and reconfiguration. In *2019 32nd IEEE International System-on-Chip Conference (SOCC)*, pages 203–208, 2019.
 - [116] W. Pananurak, S. Thanok, and M. Parnichkun. Adaptive cruise control for an intelligent vehicle. In *2008 IEEE International Conference on Robotics and Biomimetics*, pages 1794–1799, 2009.
 - [117] Y. Papadopoulos. HiP-HOPS <https://hip-hops.co.uk> Last access: 2022-02-06.

- [118] D. Parker, M. Walker, L. S. Azevedo, Y. Papadopoulos, and R. E. Araújo. Automatic decomposition and allocation of safety integrity levels using a penalty-based genetic algorithm. In M. Ali, T. Bosse, K. V. Hindriks, M. Hoogendoorn, C. M. Jonker, and J. Treur, editors, *Recent Trends in Applied Artificial Intelligence*, pages 449–459, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
- [119] T. P. Peixoto. Graph-tool: An efficient python module for manipulation and statistical analysis of graphs. <https://graph-tool.skewed.de/> Last access: 2021-06-02.
- [120] K. Poland, M. P. McKay, D. Bruce, and E. Becic. Fatal crash between a car operating with automated control systems and a tractor-semitrailer truck. *Traffic Injury Prevention*, 19(sup2):S153–S156, 2018. PMID: 30841795.
- [121] J. Qian, X. Wang, Q. Yang, F. Zhuang, J. Jia, X. Li, Y. Zuo, J. Mekkoth, J. Liu, H.-J. Chao, S. Wu, H. Yang, L. Yu, F. Zhao, and L.-T. Wang. Logic BIST architecture for system-level test and diagnosis. In *2009 Asian Test Symposium*, pages 21–26, 2009.
- [122] O. Rakhimov. Scram: a Command-line Risk Analysis Multi-tool. <https://github.com/rakhimov/scram> Last access: 2022-02-05.
- [123] S. Rambo. Shedding pounds in automotive electronics, <https://semiengineering.com/shedding-pounds-in-automotive-electronics/> Last access: 2021-07-19.
- [124] R. H. Rasshofer and K. Gresser. Automotive radar and lidar systems for next generation driver assistance functions. *Advances in Radio Science*, 3:205–209, 2005.
- [125] L. Reger. The EE architecture for autonomous driving a domain-based approach. *ATZelektronik worldwide*, 12(6):16–21, 2017.
- [126] F. Reimann, M. Glaß, J. Teich, A. Cook, L. R. Gómez, D. Ull, H.-J. Wunderlich, U. Abelein, and P. Engelke. Advanced diagnosis: SBST and BIST integration in automotive E/E architectures. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.
- [127] D. Reinhardt and M. Kucera. Domain controlled architecture - a new approach for large scale software integrated automotive systems. In *Proc. Third International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2013)*, 2013.
- [128] R. RemenYTE-PreSCott and J. D. Andrews. An enhanced component connection method for conversion of fault trees to binary decision diagrams. *Reliability engineering & system safety*, 93(10):1543–1550, 2008.

- [129] E. Rotenberg. AR-SMT: a microarchitectural approach to fault tolerance in microprocessors. In *Digest of Papers. Twenty-Ninth Annual International Symposium on Fault-Tolerant Computing (Cat. No.99CB36352)*, pages 84–91, 1999.
- [130] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15-16:29–62, 2015.
- [131] K. Saadeddin, M. F. Abdel-Hafez, and M. A. Jarrah. Estimating vehicle state by GPS/IMU fusion with vehicle dynamics. *Journal of Intelligent & Robotic Systems*, 74(1):147–172, 2014.
- [132] A. K. Saberi. Functional safety: A new architectural perspective: Model-based safety engineering for automated driving systems. PhD Thesis, Eindhoven University of Technology. 2020.
- [133] SAE International. Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles. SAE J3016::2021. Standard, 2021.
- [134] M. Safar. ASIL decomposition using SMT. In *2017 Forum on Specification and Design Languages (FDL)*, pages 1–6, 2017.
- [135] B. Sari and H.-C. Reuss. A model-driven approach for dependent failure analysis in consideration of multicore processors using modified EAST-ADL. Technical report, SAE Technical Paper, 2017.
- [136] B. Sari and H.-C. Reuss. Fail-operational safety architecture for ADAS systems considering domain ECUs. In *WCX World Congress Experience*. SAE International, apr 2018.
- [137] N. Saxena and E. McCluskey. Dependable adaptive computing systems - the ROAR project. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, volume 3, pages 2172–2177 vol.3, 1998.
- [138] T. Schmid, S. Schraufstetter, S. Wagner, and D. Hellhake. A safety argumentation for fail-operational automotive systems in compliance with ISO 26262. In *2019 4th International Conference on System Reliability and Safety (ICSRs)*, pages 484–493, 2019.
- [139] W. Schneeweiss. Calculating the probability of boolean expression being 1. *IEEE Transactions on Reliability*, 26(1):16–22, 1977.
- [140] A. Schnellbach. Fail-operational automotive systems. 2018.

- [141] V. Schönemann, H. Winner, T. Glock, E. Sax, B. Boeddeker, S. vom Dorff, G. Verhaeg, F. Tronci, and G. G. Padilla. Fault tree-based derivation of safety requirements for automated driving on the example of cooperative valet parking. In *26th International Technical Conference on the Enhanced Safety of Vehicles (ESV) 2019*, 2019.
- [142] T. Schuster and D. Verma. Networking concepts comparison for avionics architecture. In *2008 IEEE/AIAA 27th Digital Avionics Systems Conference*, pages 1.D.1–1–1.D.1–11, 2008.
- [143] H. Shaheen, G. Boschi, G. Harutyunyan, and Y. Zorian. Advanced ECC solution for automotive SoCs. In *2017 IEEE 23rd International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 71–73, 2017.
- [144] S. Shalev-Shwartz, S. Shammah, and A. Shashua. On a formal model of safe and scalable self-driving cars. *CoRR*, abs/1708.06374, 2017.
- [145] D. Shapiro. Eyes on the road: how autonomous cars understand what they’re seeing. From NVIDIA blog, <https://blogs.nvidia.com/blog/2016/01/05/eyes-on-the-road-how-autonomous-cars-understand-what-theyre-seeing/>. Last access: 2021-06-02, 2016.
- [146] D. Sherwin and A. Bossche. Boolean algebra and probability laws for reliability evaluations. In *The Reliability, Availability and Productiveness of Systems*, pages 121–132. Springer, 1993.
- [147] Y. Shoukry, M. W. El-Kharashi, S. Hammad, A. Kumar, and G. Bahig. Software allocation in automotive networked embedded systems: A graph-based approach. In M.-M. Louërat and T. Maehne, editors, *Languages, Design Methods, and Tools for Electronic System Design: Selected Contributions from FDL 2013*. Springer International Publishing, 2015.
- [148] M. Siegel. The sense-think-act paradigm revisited. In *1st International Workshop on Robotic Sensing, 2003. ROSE’03.*, pages 5–pp. IEEE, 2003.
- [149] Siemens Digital Industries Software. The criticality of the automotive E/E architecture. Whitepaper, Siemens, 2020.
- [150] Siemens Digital Industries Software. Real-world considerations for vehicle E/E architecture design. Whitepaper, Siemens, 2021.
- [151] F. Sindaco. Identifying the right path forward for future vehicle EE architecture. NXP training presentation, 2021.
- [152] P. Sinha. Architectural design and reliability analysis of a fail-operational brake-by-wire system from ISO 26262 perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, 2011.

- [153] R. M. Sinnamon and J. Andrews. Improved efficiency in qualitative fault tree analysis. *Quality and Reliability Engineering International*, 13(5):293–298, 1997.
- [154] J. R. Sklaroff. Redundancy management technique for space shuttle computers. *IBM Journal of Research and Development*, 20(1):20–28, 1976.
- [155] T. Slegel, R. Averill, M. Check, B. Giamei, B. Krumm, C. Krygowski, W. Li, J. Liptay, J. MacDougall, T. McPherson, J. Navarro, E. Schwarz, K. Shum, and C. Webb. IBM’s S/390 G5 microprocessor design. *IEEE Micro*, 19(2):12–23, 1999.
- [156] I. Šljivo, G. J. Uriagereka, S. Puri, and B. Gallina. Guiding assurance of architectural design patterns for critical applications. *Journal of Systems Architecture*, 110:101765, 2020.
- [157] F. Smirnov, F. Reimann, J. Teich, and M. Glaß. Automatic optimization of the VLAN partitioning in automotive communication networks. *ACM Trans. Des. Autom. Electron. Syst.*, 24(1), Dec. 2018.
- [158] P. Sundaram and J. G. D’Ambrosio. Controller integrity in automotive failsafe system architectures. In *SAE 2006 World Congress & Exhibition*. SAE International, apr 2006.
- [159] SysML. <https://sysml.org/>, 2021. Last access: 2021-06-08.
- [160] H. Tabani, R. Pujol, M. Alcon, J. Moya, J. Abella, and F. J. Cazorla. ADBench: benchmarking autonomous driving systems. *Computing*, pages 1–22, 2021.
- [161] Y. Tang, Y. Yuan, and Y. Liu. Cost-aware reliability task scheduling of automotive cyber-physical systems. *Microprocessors and Microsystems*, page 103507, 2020.
- [162] TTTechAuto. How to build fail-operational systems for autonomous driving. Webinar, 2019.
- [163] P. Tzirakis, G. Trigeorgis, M. A. Nicolaou, B. W. Schuller, and S. Zafeiriou. End-to-end multimodal emotion recognition using deep neural networks. *IEEE Journal of Selected Topics in Signal Processing*, 11(8):1301–1309, 2017.
- [164] Underwriters Laboratories. Standard for evaluation of autonomous products. UL4600. Standard, UL/ANSI, 2020.
- [165] U.S. Department of Transportation. Automated vehicles for safety. From NHTSA press release, <https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety>. Last access: 2021-06-02, 2021.

- [166] M. Van den Brand and J. F. Groote. Software engineering: Redundancy is key. *Science of Computer programming*, 97:75–81, 2015.
- [167] P. van der Perk. A distributed safety mechanism for autonomous vehicle software using hypervisors. *MS thesis, Eindhoven University of Technology*, 2019.
- [168] A. V. Varadarajan, M. Romijn, B. Oosthoek, J. van de Mortel-Fronczak, and J. Beijer. Development and validation of functional model of a cruise control system. *arXiv preprint arXiv:1603.08635*, 2016.
- [169] VDA QMC. Automotive Software Improvement and Capability dEtermination. ASPICE. Standard, VDA QMC, 2017.
- [170] L.-T. Wang, C. E. Stroud, and N. A. Touba. System-on-chip test architectures: nanometer design for testability. Elsevier, 2010.
- [171] D. Ward and S. Crozier. The uses and abuses of ASIL decomposition in ISO 26262. In *7th IET International Conference on System Safety, incorporating the Cyber Security Conference 2012*, pages 1–6, 2012.
- [172] G. R. Widmann, M. K. Daniels, L. Hamilton, L. Humm, B. Riley, J. K. Schiffmann, D. E. Schnelker, and W. H. Wishon. Comparison of lidar-based and radar-based adaptive cruise control systems. In *SAE 2000 World Congress*. SAE International, Mar 2000.
- [173] Y. Wiseman and I. Grinberg. The trolley problem version of autonomous vehicles. *Open Transportation Journal*, 12:105–113, 2018.
- [174] H.-J. Wunderlich. BIST for systems-on-a-chip. *Integration*, 26(1):55–78, 1998.
- [175] G. Xie, Y. Chen, Y. Liu, R. Li, and K. Li. Minimizing development cost with reliability goal for automotive functional safety during design phase. *IEEE Transactions on Reliability*, 67(1):196–211, 2018.
- [176] G. Xie, H. Peng, J. Huang, R. Li, and K. Li. Energy-efficient functional safety design methodology using asil decomposition for automotive cyber-physical systems. *IEEE Transactions on Reliability*, pages 1–23, 2019.
- [177] Y. Yeh. Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, volume 1, pages 293–307 vol.1, 1996.
- [178] J. Zhou, X. S. Hu, Y. Ma, and T. Wei. Balancing lifetime and soft-error reliability to improve system availability. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 685–690, 2016.

List of abbreviations

ADAS	Advanced Driving Assistance System
ACC	Adaptive Cruise Control
ASIL	Automotive Safety Integrity Level
AV	Autonomous Vehicle
BDD	Binary Decision Diagram
CCF	Common-Cause Fault
CF	Cascading Fault
ECC	Error Correction Code
ECU	Electronic Control Unit
E/E	Electrical and Electronic
EMI	Electro-Magnetic Interference
FMEA	Failure Mode and Effect Analysis
FRER	Frame Replication and Elimination for Reliability
FSR	Functional-Safety Requirement
HARA	Hazard Assessment and Risk Analysis
IVN	In-Vehicle Network
ITE	If-Then-Else
LLSC	Low-Level Speed Control
MooN	M-out-of-N
OEM	Original Equipment Manufacturer
QM	Quality Management

QoS Quality of Service

RTE Runtime Environment

RTOS Real-Time Operating System

SG Safety Goal

SIL Safety Integrity Level

SoP Sum of Products

TLE Top-level Event

TS Throttle Signal

TSN Time-Sensitive Networking

TSR Technical Safety Requirement

UML Unified Modeling Language

VCAN Virtual Controller Area Network

VLAN Virtual Local Area Network

VR Virtual Resource

List of Terminology

The entries with a citation are reported directly from the source.

- **architecture:** representation of the structure of the system. Includes the functions to application and resource layers.
- **ASIL decomposition:** Apportioning of redundant safety requirements to elements, with sufficient independence, for the same safety goal. The objective being reducing the ASIL of the redundant safety requirements that are allocated to the corresponding elements [81].
- **basic event:** in a fault tree, an error or a failure in an element. A failure rate is associated to a basic event.
- **binary decision diagram:** a directed acyclic graph composed of terminal and non-terminal vertices (nodes) which are connected by edges. Terminal vertices correspond to the final state of the system, failure (1) or success (0), and non-terminal vertices correspond to the basic events of the fault tree. Each non-terminal vertex has a 1 branch, which represents basic event occurrence, and a 0 branch, which represents basic event non-occurrence [6].
- **event:** in a fault tree, an intermediate step that is expanded into lower levels via gate symbols.
- **element:** system or part of a system including components, hardware, software, hardware part, and software units [81].
- **fail-safe:** in the presence of a failure, the system reaches a safe condition.
- **fail-silent:** in the presence of a failure, the element does not cause harm or interferences with other elements of the system.
- **failure:** termination of the ability of an element or an item to perform a function as required [81].
- **fault:** abnormal condition that can cause an element or an item to fail [81].
- **fault-tolerant:** an element that continue operating properly in the event of one or multiple faults.

- **fault tree:** a directed acyclic graph that shows the logical connections between the failure of individual basic events and the failure of a top-level event. It uses a combination of type of events (basic, external, intermediate) and boolean gates (OR, AND, XOR). Advanced fault trees use additional elements, e.g. priority AND gates.
- **hazard:** potential source of harm [81].
- **independence:** absence of a dependent failure between two or more elements that could lead to the violation of a safety requirement [81].
- **merger:** redundancy-related element that ensures that only correct data out of its redundant input ports is forwarded to the next part of the system.
- **multi-point fault:** individual fault that, in combination with other independent faults, leads to a multiple point failure [81].
- **residual fault:** a (part of a) fault that is outside the diagnostic coverage of a safety mechanism.
- **safe fault:** a fault that does not lead to the violation of a safety goal.
- **safety case:** a document that provides proofs that a system is safe and follows the correct safety guidelines.
- **single-point fault:** a fault not covered by safety mechanisms that leads to the failure of an element.
- **splitter:** a redundancy element that replicates the data on its input ports to its output ports.

List of publications

Journals

- [J1] **A. Frigerio**, B. Vermeulen and K. Goossens. Automotive architecture topologies: analysis for safety-critical autonomous vehicle applications. *IEEE ACCESS*, 9:62837-62846, 2021.

Conference proceedings

- [C1] **A. Frigerio**, B. Vermeulen and K. Goossens. A generic method for a bottom-up ASIL decomposition. In *Computer Safety, Reliability, and Security (SAFECOMP)*, pages 12-26, 2018.
- [C2] **A. Frigerio**, B. Vermeulen and K. Goossens. Component-Level ASIL Decomposition for Automotive Architectures. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pages 62-69, 2019.
- [C3] **A. Frigerio**, B. Vermeulen and K. Goossens. Isolation of redundant and mixed-critical automotive applications: effects on the system architecture. In *Vehicle Technology Conference - Spring (VTC)*, 2021.

Additional publications

- [A1] T. Bijlsma, A. Buriachevsky, **A. Frigerio**, Y. Fu, K. Goossens, A.O. Örs, P.J. van der Perk, A. Terechko and B. Vermeulen. A Distributed Safety Mechanism using Middleware and Hypervisors for Autonomous Vehicles. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1175-1180, 2020.



Three-layer model recap

Table A.1: Overview of the three-layer model symbols

Symbol	Meaning
G_a	Application graph. A system contains one or more application graphs.
V_a	Set of application nodes, part of an application graph G_a .
E_a	Set of edges connecting the application nodes, part of an application graph G_a .
G_r	Resource graph. A system contains a single resource graph.
V_r	Set of resources, part of the resource graph G_r .
E_r	Set of edges connecting the resources, part of the resource graph G_r .
E_{dep}	Set of edges connecting the resource to the resources on which they depend.
G_p	Physical space graph. A system contains a single physical space graph.
V_p	Set of physical locations, part of the physical space graph G_p .
M_{ar}	Set of mapping edges from application nodes to resources.
M_{rp}	Set of mapping edges from resources to physical locations.

Table A.2: Overview of the layer properties

Property	Layer	Description
Type	Application	<i>Sensor, actuator, functional, communication, splitter, merger.</i> Value assigned to application nodes.
ASIL requirement	Application	ASIL requirement of an application node. The possible values are <i>QM, A, B, C, D</i> .
Original ASIL	Application	System-level ASIL requirement of an application node, before ASIL decomposition. The possible values are <i>QM, A, B, C, D</i> .
Functional load	Application	Positive integer value, representing the resource usage of a functional node.
Communication load	Application	Positive integer value, representing the resource usage of a communication node.
λ_{a-x}	Application	Failure rate of the node x expressed in failures per hour.
Type	Resource	<i>Sensor, actuator, functional, communication, splitter, merger, power source, power line.</i> Each resource has one or multiple types.
ASIL specification	Resource	ASIL specification of a resource. The possible values are <i>QM, A, B, C, D</i> .
Maximum load	Resource	Positive integer value, representing the maximum functional or communication load that a resource can handle.
Cost	Resource	The cost of a resource.
λ_{r-y}	Resource	The failure rate of the resource y expressed in failures per hour.
Virtualized	Resource	Boolean value. True if the resource is a Virtual Resource (VR), false otherwise.
2D Coordinate	Physical	2D coordinates of the physical location in the vehicle. Normalized between -1.00 and 1.00 .
λ_{p-z}	Physical	Failure rate of the location z expressed in failures per hour.

B

Input format

The structure of the input file that is passed to the analysis tools is shown in Listing B.1. Listing B.2 shows an example input file for a system with one application composed by three nodes, one multifunctional resource on which all the application nodes are mapped, and one physical location on which the resource is mapped. The input files can be manually written or generated from other sources: a python parser that uses the *xbrd* package is used to parse Microsoft Excel tables into the textual input format, while a macro function is used to extract the Microsoft Excel table from a visual graph designed in Microsoft Visio. Figure B.1 shows part of a Microsoft Excel table that can be parsed into the textual input format, taken from the table used for one of the experiments in Chapter 5. The Microsoft Excel file contains one tab for each layer of the model, plus an additional tab for each set of edges: E_a , E_r , E_{dep} , M_{ar} , and M_{rp} . Figure B.2 shows a part of the Microsoft Visio file that is used for generating the Microsoft Excel tables. The parameters are inserted as a string in the name of each figure. The file contains a separate tab for each layer, plus two additional tabs for the mapping sets M_{ar} and M_{rp} .

```

1 create
2 # --- APP
3 # addV add Vertex
4 # nodeLevel app
5 # appName
6 # nodeName
7 # OrigNode, "stop" when done
8 # nodeType - sensor, actuator, splitter, merger, functional,
9 #             communication
10 # ASIL
11 # original ASIL
12 # load
13 # --- APP_CONN
14 # addE
15 # source
16 # target
17
18 # --- RES
19 # addV add Vertex
20 # nodeLevel res
21 # nodeName
22 # Types - sensor, actuator, splitter, merger, functional,
23 #             communication, powerline, powersource - stop when you are done
24 # ASIL
25 # Virtualized
26 # maxLoad
27 # --- RES_CONN
28 # addE
29 # source
30 # target
31
32 # --- APP_MAP
33 # addE
34 # source
35 # target
36
37 # --- PHY
38 # addV Add Vertex
39 # nodeLevel phy
40 # nodeName
41 # x_pos
42 # y_pos
43
44 # --- RES_MAP
45 # addE
46 # source
47 # target
48
49 leave
50 save

```

Listing B.1: Structure of the input file in textual format.

```
1 create
2
3 addV
4 app
5 main
6 app_sensor
7 stop
8 sensor
9 D
10 D
11 100
12
13 addV
14 app
15 main
16 app_communication
17 stop
18 communication
19 D
20 D
21 100
22
23 addV
24 app
25 main
26 app_actuator
27 stop
28 actuator
29 D
30 D
31 100
32
33 addE
34 app_sensor
35 app_communication
36
37 addE
38 app_communication
39 app_actuator
40
41 addV
42 res
43 res_multifunctional
44 sensor
45 actuator
46 communication
47 functional
48 stop
49 D
50 0
51 100
52
53 addE
54 app_sensor
55 res_multifunctional
```

```

56
57 addE
58 app_communication
59 res_multifunctional
60
61 addE
62 app_actuator
63 res_multifunctional
64
65 addV
66 phy
67 phy_p1
68 -0.4
69 0
70
71 addE
72 res_multifunctional
73 phy_p1
74
75 leave
76 save

```

Listing B.2: Example input graph in textual format.

nodeName	nodeLevel	resType	resASIL	maxLoad
res_temperature_sensor	res	sensor	QM	100
res_central_ecu	res	functional&communication	D	100
res_dom_ctrl_comf	res	functional&communication	D	100
res_ethernet_switch_3	res	functional&communication	D	100
res_direct_ethernet_1	res	communication	D	100
res_ethernet_10	res	communication	D	100
res_ethernet_8	res	communication	D	100
res_surrView_f	res	sensor	QM	100
res_surrView_r	res	sensor	QM	100

Figure B.1: Part of a Microsoft Excel-based input file used for the experiments in Chapter 5.

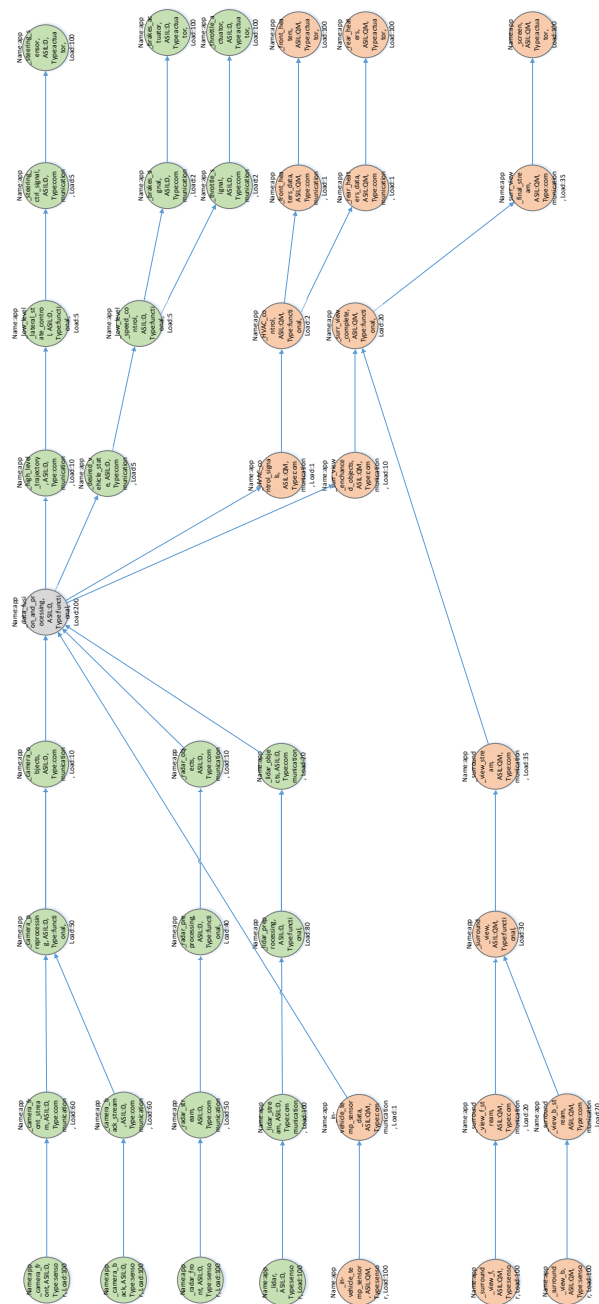


Figure B.2: Part of a Microsoft Visio-based input file used for the experiments in Chapter 5.



EcoTwin lateral control modelled graphs

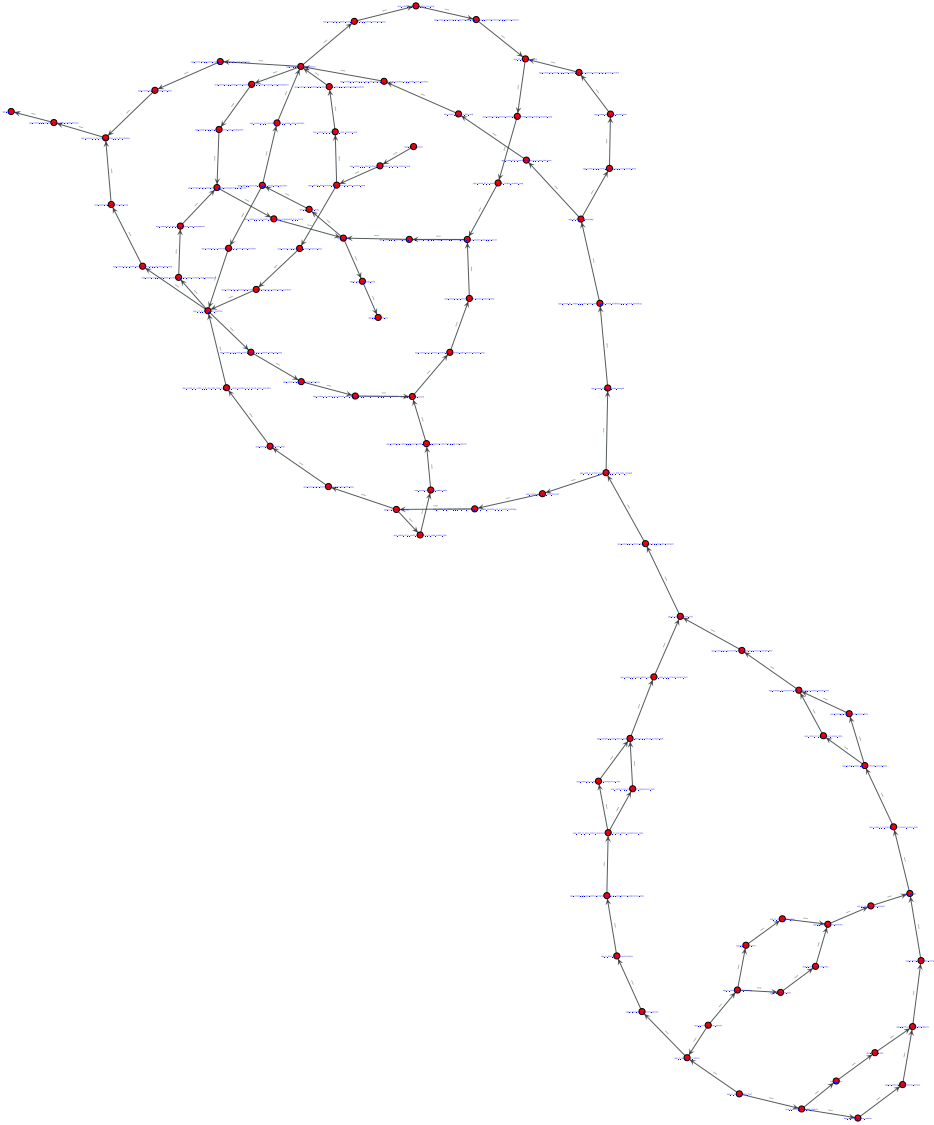


Figure C.1: The application graph of the EcoTwin lateral control example used in Chapter 4, after the full transformation process. Generated from the developed framework.

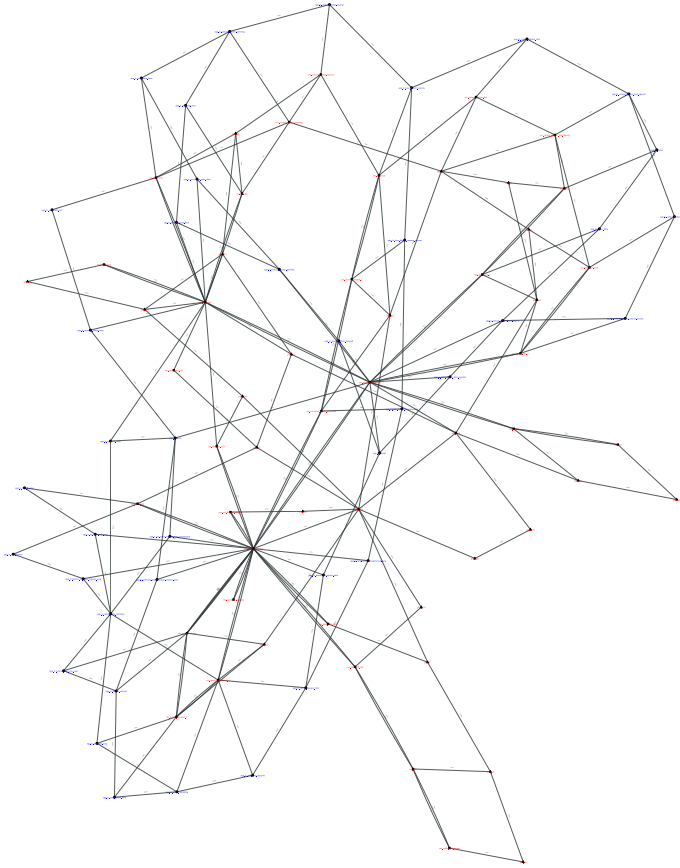


Figure C.3: The full graph of the EcoTwin lateral control example used in Chapter 4, after the full transformation process. Generated from the developed framework.

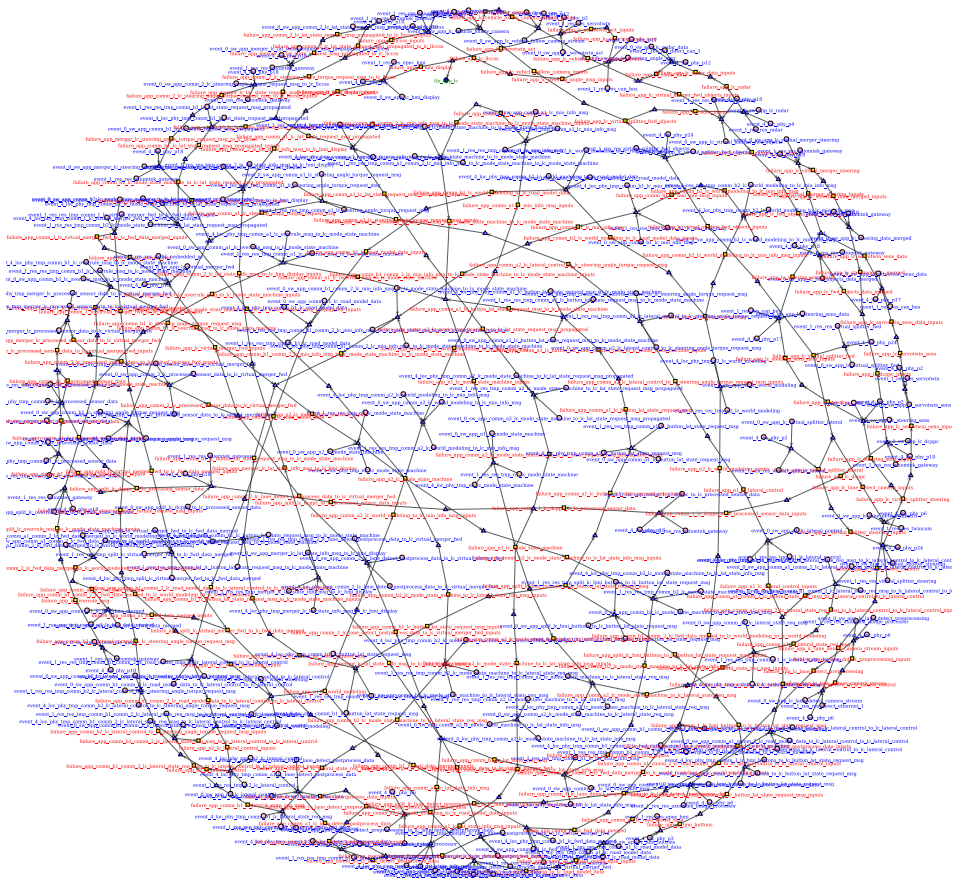


Figure C.4: The fault tree graph of the EcoTwin lateral control example used in Chapter 4, after the full transformation process. Generated from the developed framework.

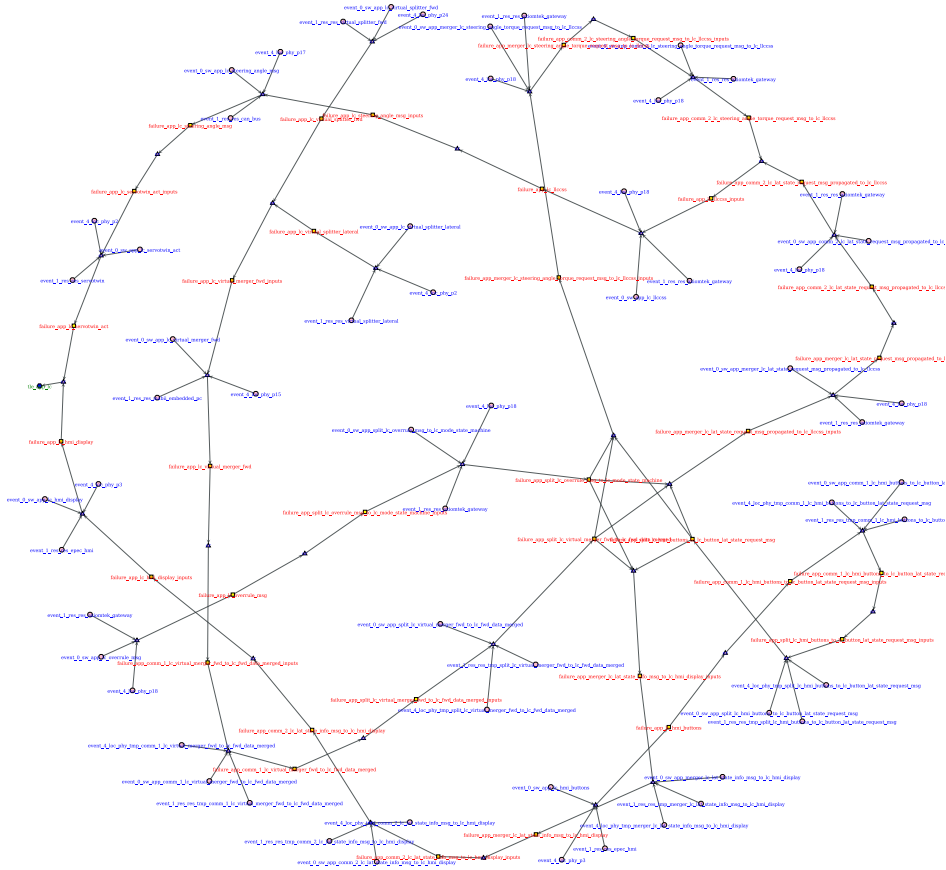


Figure C.5: The approximated fault tree graph of the EcoTwin lateral control example used in Chapter 4, after the full transformation process. Generated from the developed framework.

D

Mapping of applications to architecture
topologies

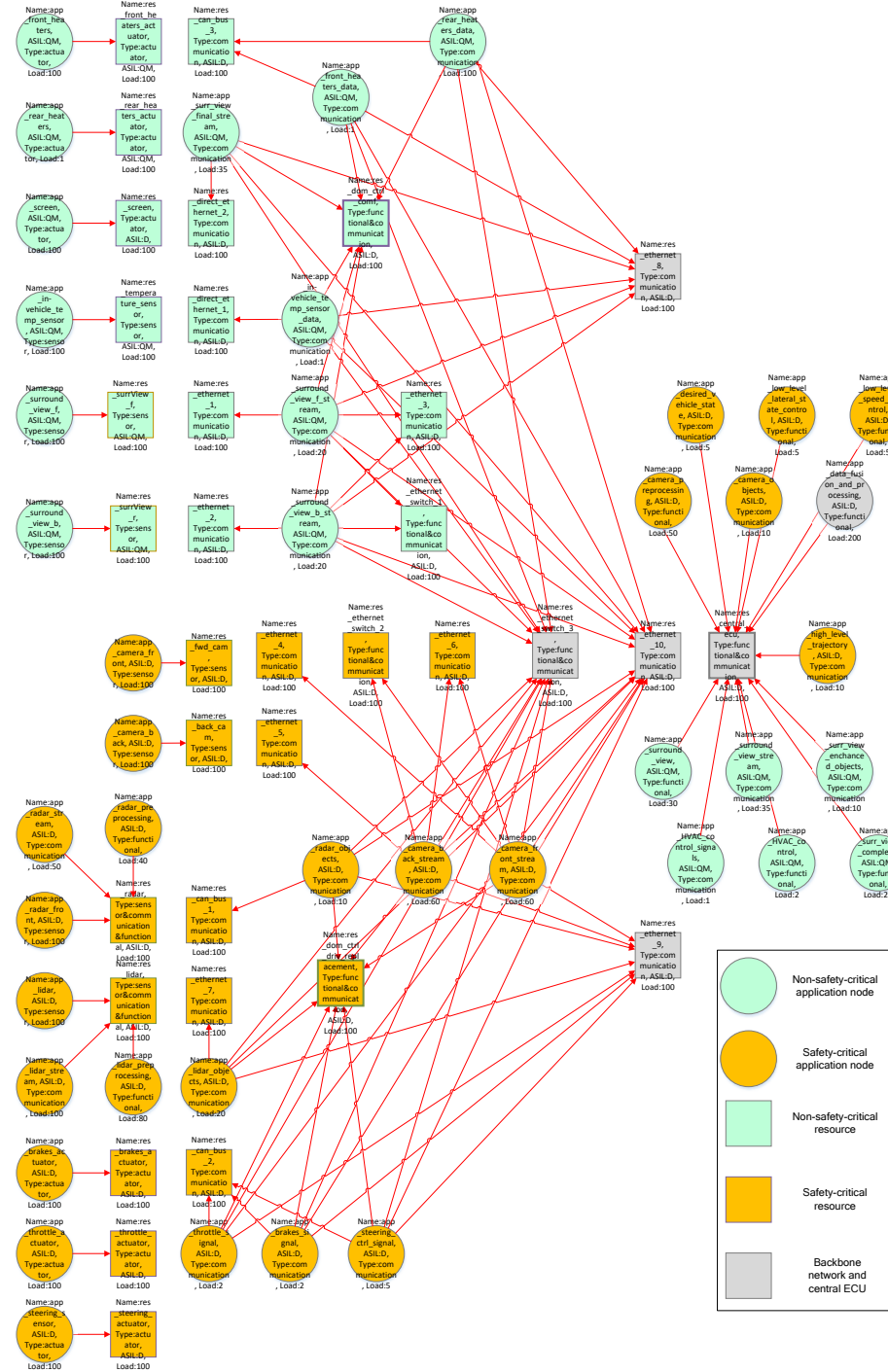


Figure D.1: Mapping of the applications in the Domain-based Vehicle-Centralized architecture (D-VC).

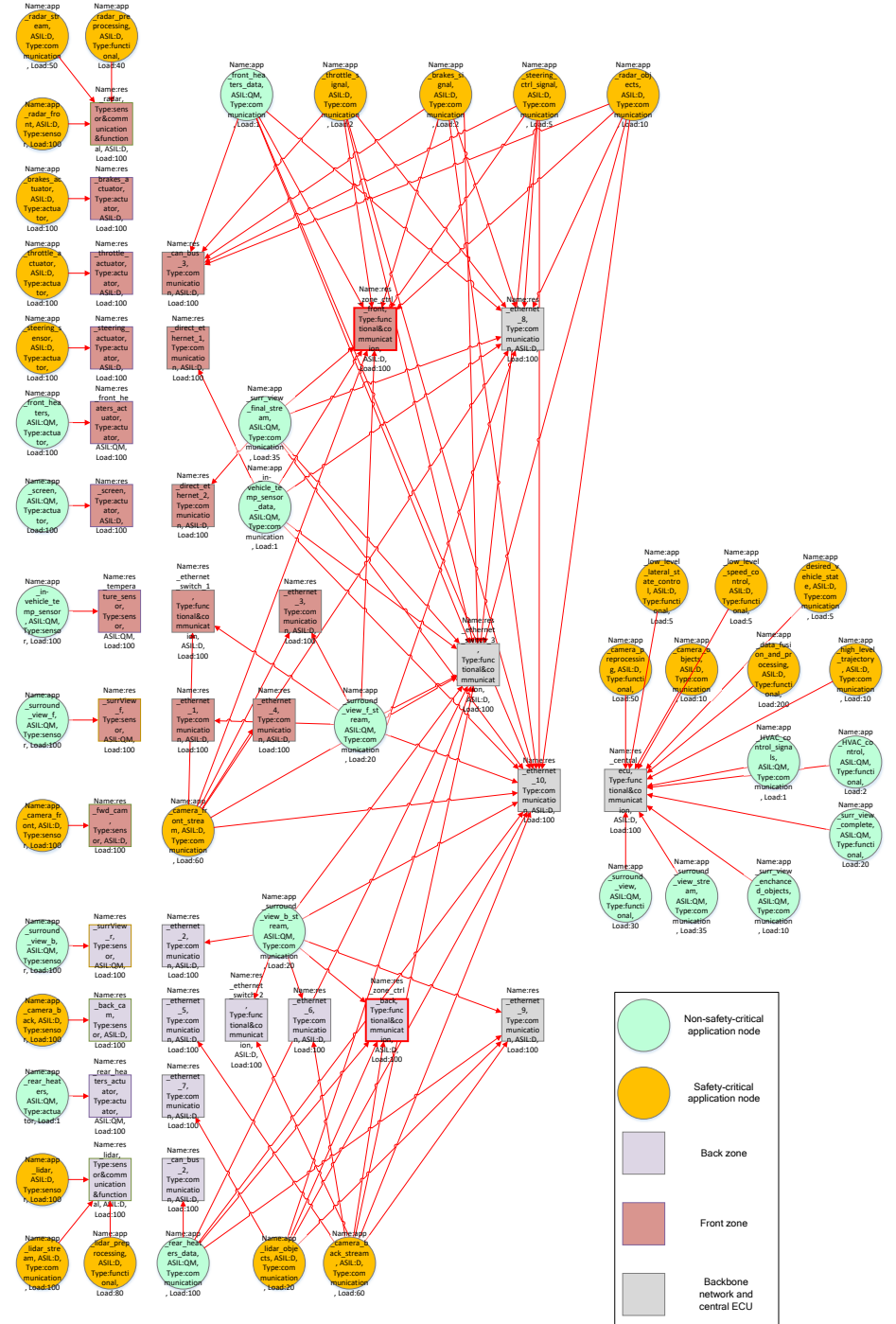


Figure D.2: Mapping of the applications in the Zone-based Vehicle-Centralized architecture (Z-VC).

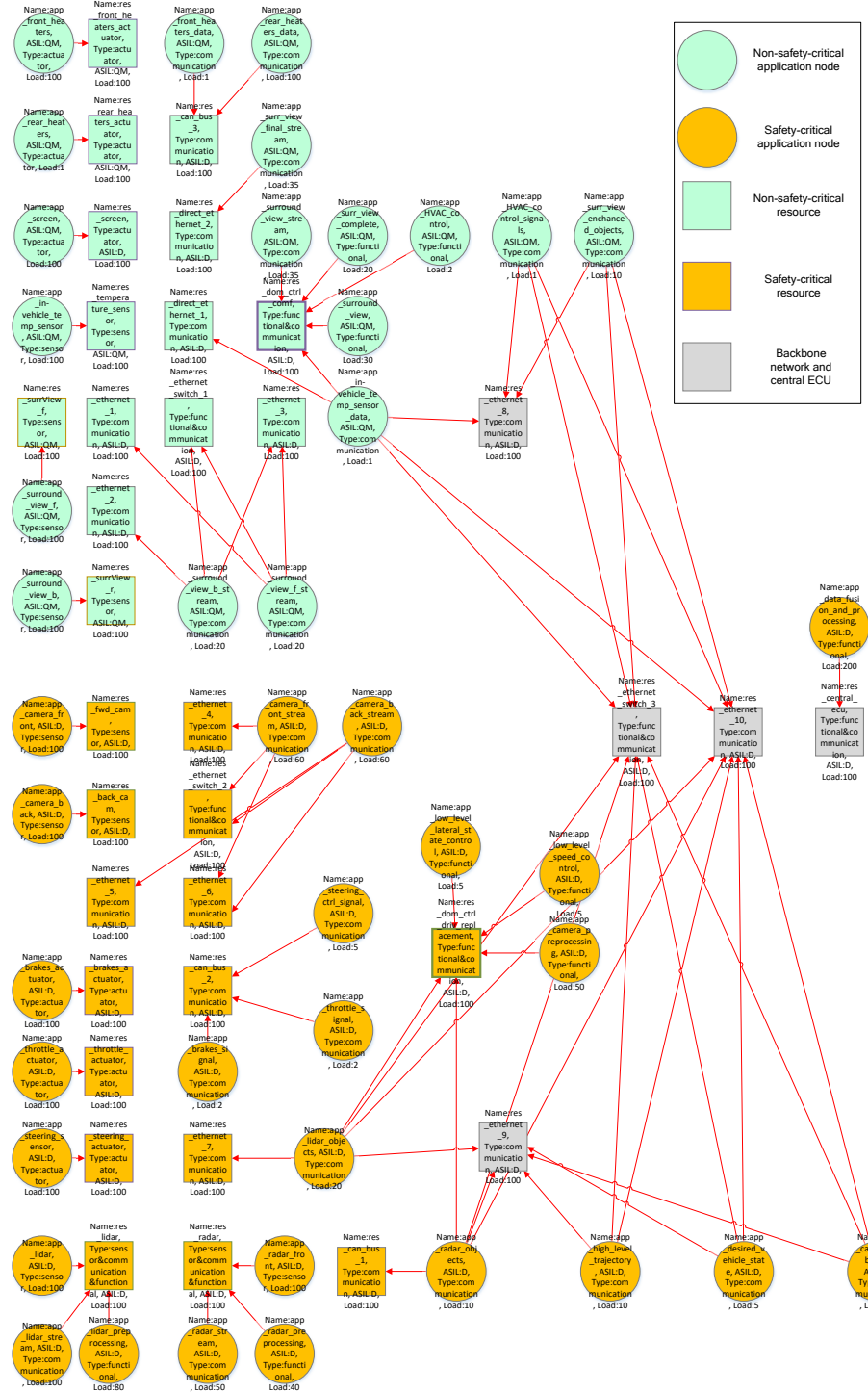


Figure D.3: Mapping of the applications in the Domain-based Controller-Based architecture (Z-VC).

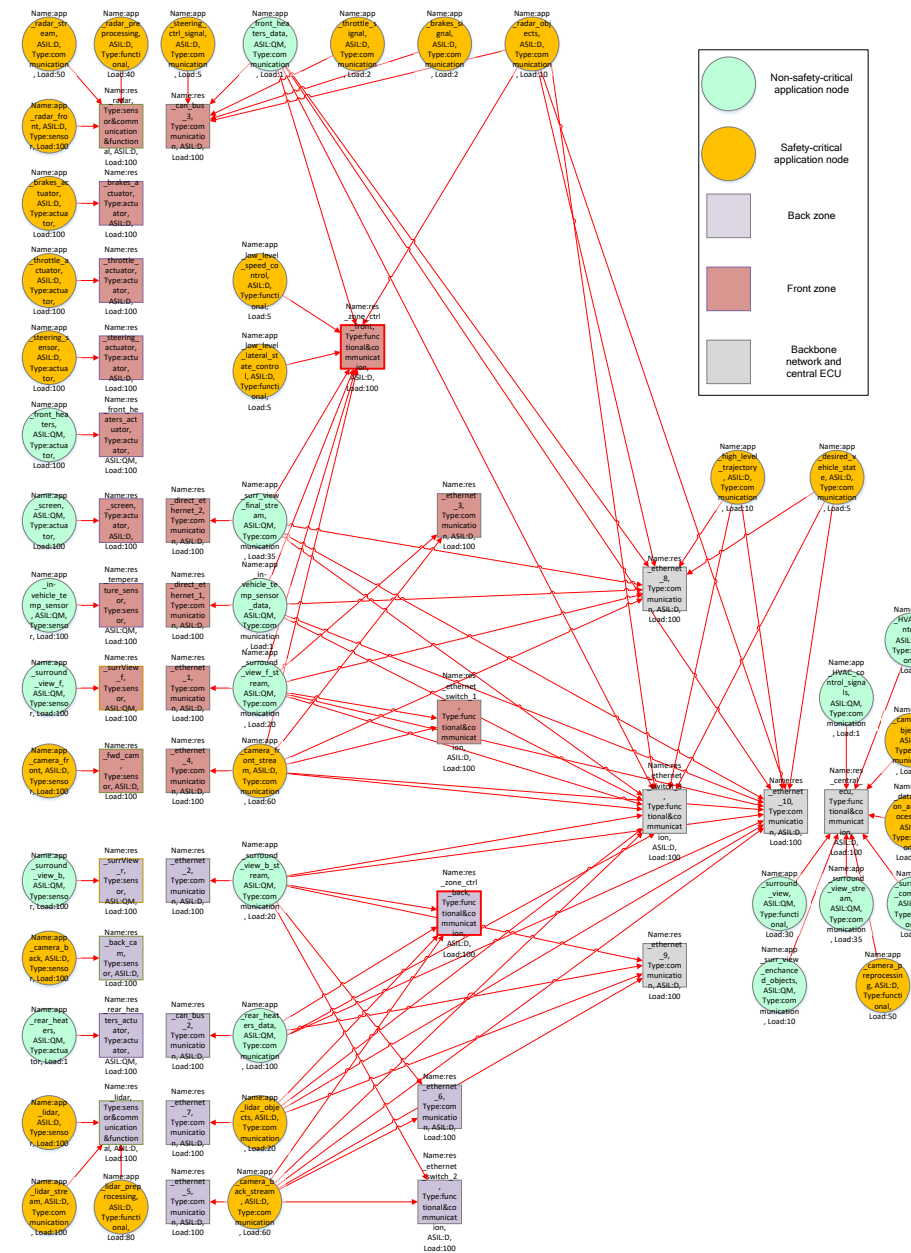


Figure D.4: Mapping of the applications in the Zone-based Controller-Based architecture (Z-VC).

Acknowledgements

I would like to thank my promotor, Kees Goossens. I am grateful for your constant support on my research. I appreciated discussing with you the technical parts, but, in particular, our conversations on our common hobby, food and cooking. The second person that I want to thank is my copromotor, Bart Vermeulen. Through you, I had a connection to the industry world at NXP Semiconductors. Your insights and feedback always helped me going through the difficult times. I am enthusiast to keep working with you at NXP and I am looking forward to keep collaborating with you. Furthermore, I am thankful to all the Doctorate Committee members for taking the time to review this manuscript and participate in the defense ceremony. Their constructive feedback helped improving the quality of both the thesis and the tools that I have developed in these years.

I have met some amazing people while working at the company. Amongst others, I would like to thank Andrei, Lulu, and Yuting who inspired me with their ideas and creativity.

During the time at the University I was able to connect with many colleagues, but most importantly, many new friends that shared the PhD experience with me. Thanks to Marja and Wendy for they support at the University, as well as the rest of the Electronic Systems Group staff. Thanks to the members of the CompSoC group for the collaborations we had and their help in getting accustomed to the University environment: thanks Andrew, Gabriela, Hamideh, Martijn, Rasool, Reinier, Sajid, Shayan, and Weijiang. In a world of overly enthusiastic people, Savvas' complaining kept me sane in these years. Thanks to Luc and Anouk for the fun time that we spent together in our now traditional Friday drinks. Thanks to Barry for defending the honour of the stampot. Thanks Berk, Emad, Kamlesh, Mojtaba, Paul, Sayandip, and Shima for all the time that we spent together. We enjoyed not only the time at the office, but also dinners, weekend futsal games, holidays and more. Some people that I have many good memories with moved to continue their careers in different countries, while others just arrived in Eindhoven. Thanks to Cumhur, Gagandeep, Joan, Martin, Ramon, Sajid, Sherif, and Zhan. I also thank all the other colleagues in the Electronic System Group that were part of this journey.

Thanks to my Italian friends of Gruppo 2 that with the constant discussion made me feel like I was still in Livorno with them, despite not being part of Gruppo 1.

Of course reaching this point was possible only thanks to the help of my family.

Thanks to my parents for always supporting me throughout my studies with all they could give me. Thanks to my brother for visiting me in the Netherlands many times. Thanks to all the words of encouragement from my grandparents that always made me want to reach the top.

Finally, I thank an incredible person that during these years was a colleague, a friend, family. Thank you Ilde for being by my side, for lending me part of your strength and willpower, and for making me happy (together with our cats Nigiri and Kiwi).

About the author

Alessandro Frigerio was born on 25-07-1991 in Biella, Italy. He studied Electronics Engineering at University of Pisa in Italy. He graduated in his bachelor's and master's degrees cum laude in 2013 and 2016 respectively. Since November 2016 he started a PhD project at Eindhoven University of Technology in the Electronic Systems group at the Department of Electrical Engineering, of which the results are presented in this dissertation. Since October 2021 he is employed at NXP Semiconductors, in which he continues the research activity started during the PhD project. His research interests include automotive E/E architectures, automotive cybersecurity, functional safety.