Optimization approach for the critical automotive embedded systems
Mohamed Slim Dhouibi

▶ To cite this version:

HAL Id: tel-01474888
https://tel.archives-ouvertes.fr/tel-01474888
Submitted on 15 May 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Optimization approach for the critical automotive embedded systems
Introduction

Cars today come equipped with a variety of embedded electronic systems ensuring complex and essential functionalities. The trend started many years ago allowing to replace the mechanical systems with electronic alternatives. This offered more flexibility, more user comfort and better cost. It also opened the door for more innovation as the car became more “intelligent” and capable of interacting with the user and the surrounding environment. The embedded systems control among other the breaking, steering and suspension.

With growing functionalities, the embedded systems needed to be designed and verified in a way to ensure that the safety of the car passengers are not in unreasonable risk. As these systems can be prone to failures for different reasons, such failures are analyzed, mitigated or controlled with state of the art techniques and concepts.

In order to achieve this, the determination of the hazards and the evaluation of the existing risks starts from the early design steps following a functional safety process. ISO 26262, the functional safety standard in the automotive industry, provide the guidelines for the safety activities, measures and acceptable risks for the compliance of the embedded systems design.

The embedded system design is a challenging task. The engineers are faced with technological, cost, complexity and safety constraints. More than ever before, the system engineering activities are in need to new approaches, techniques and tools to cope with the limits of the manual design process and to effectively take into account the design constraints. These constraints have a big impact on the system architecture and consequently on the final cost. Automotive industry actors, such as Valeo, understand the architecture design challenge and the need to conceive new system and safety engineering approaches to reach cost effective architectures.

Often, the architecture design process is manual where the architectural choices and decisions are based on the engineer expertise. These manual processes lead to compliant designs at an acceptable cost. But with the growing complexity and size
of the systems, they can lead to incurring unnecessary costs. It can be resulting from the development effort, the components choices or the redundancy introductions.

In order to propose the most effective solutions, an optimization approach taking into account the system engineering constraints as well as the safety constraints is needed. It should be able to respect the standards in vigor as well as the currently applied processes and tools. It should be able to implement the engineering choices and the available alternatives to conceive compliant solutions along with a cost estimation.

The work presented here focuses on the automotive systems architecture optimization. It focuses specifically on the architecture optimization regarding the safety from ISO 26262 point of view and cost attributes. It deals with the design challenges facing the engineers such as the functional safety concept and the physical architecture synthesis. Key tasks such as the safety levels allocation, the physical architecture components choice and the mapping of the functionalities to the system elements are addressed to help guiding the engineering decisions towards a cost effective solutions.

The initial task addressed in this work is the identification of the main parameters or attributes defining and impacting the system architectures and their cost. Secondly, we analyze these attributes and investigate the optimization possibilities and how they can be exploited. Third, we propose an approach with a potential implementation exploiting commonly used engineering tools and techniques.

Our study is based on industrial approach where we attempt to take into account Valeo state of the practice. We aim to propose a solution that can be applied in sync with the current processes in place. By reducing the impact of the approach on them, we facilitate its potential implementation and usage in real projects. Thus, some choices made here are motivated by the industrial context. The assessment of the approach is also based on a real life embedded system where we challenge the solution developed by Valeo. The study case proves the applicability of the approach, its limits and its advantages.

The developed approach is an optimization process that can assist the engineer during the design phase to reach a cost optimal solution. It consists of an architecture synthesis approach that adopts the design process in place. It automates the design space exploration allowing to check the set of possible architectures based on the functional architecture description. The candidate solutions are assessed to ensure the safety and engineering constraints are respected. While, multiple candidate solutions can be reached during this process, a cost estimation approach is used lastly to compare them. Consequently, the work presented here covers many of the system engineering activities. It is the crossing point between: System Engineering, Functional Safety Engineering and Cost estimation.

The thesis is structured as follows:

In Chapter two, we illustrate the basic concepts addressed during the embedded systems design process. We identify these systems properties and the challenges facing the designer. We focus first on the architecture concept, its viewpoints and the design choices addressed at each level. The chapter gives a summary of the design constraints and analyze in detail the safety constraints particularly from the ISO 26262 perspective.
The chapter 3 discuss the state-of-the-art processes, techniques, and algorithms of the architecture optimization approaches. A classification and a discussion of these approaches is proposed based on the optimization targets, the constraints taken into account, and the optimization logic. Since, we are focusing on the safety constraints, the second part of the chapter discuss the architecture safety evaluation. We review the safety process and the analysis techniques used along it. The third part discuss our optimization target, i.e the cost, by reviewing the cost estimation techniques and how they can be applied in our study scope.

Based on a real life example of embedded systems, we study the limits of the-state-of-the-art approaches in Chapter 4. We identify the axes of improvement and set our approach main ideas. The study case is a safety critical automotive system that address many of the design challenges. It implements multiple safety requirements with different safety levels. It interacts also with other systems in the vehicle which add more design constraints. It, thus, have been selected to help assessing the state-of-the-art approaches and identify the research gaps to study.

In chapter 5, the proposed approach is detailed. The main optimization process as well as the models and algorithms used for the design space exploration, architecture assessment and cost estimation are described. The hypotheses and arguments for each step of the process are discussed and illustrated using simple examples.

In chapter 6, an implementation of the approach is proposed based on the commonly used engineering in the system and safety process. The study case is used to evaluate the implementation and the algorithms performances. The obtained results and the reached solutions puts forward the advantages of applying our approach. In the same time, by applying it on such study case, we were able to review its limits and the amelioration perspectives.
Automotive Embedded Systems: Basic Concepts

2.1 Introduction

This chapter defines the basic concepts involved, used and discussed in our research subject perimeter. These definitions help to orient our research activities by defining the perimeter and focusing on specific engineering domains.

The concepts discussed in this chapter focus on the automotive embedded system design process. We discuss these systems main properties, its architecture and the constraints guiding its design. We introduce also the concepts of systems safety and particularly in the ISO 26262 context. The concepts impacting the system design proposed by this safety standard are key to understanding the research problem and the potential tools to tackle it.

In the last section, based on the concepts discussed through the chapter, we refine our research objective into research questions that will be the guiding lines for the rest of the thesis.

2.2 Embedded Systems

Embedded systems are single purpose electronic systems. They are present today extensively in transport systems such as cars, trains, planes etc . . . Generally, they are digital systems which typically perform specific functions, such as process control, digital signal processing, image processing or multimedia applications [55]. An embedded system is a special-purpose computer system designed to perform one or a few dedicated functions, sometimes with real-time computing constraints [10].

The application and the roles of embedded systems vary a lot making pinning down an exact definition for these systems complicated [85]. But across the various industries their characteristics are unchanged. They:
CHAPTER 2. AUTOMOTIVE EMBEDDED SYSTEMS: BASIC CONCEPTS

Figure 2.1 – Embedded System

— are generally single purpose computer systems within a larger system.
— satisfy one or multiple functionalities, well conceived beforehand.
— are designed, often, for controlling and monitoring purposes.
— are characterized by limited processing power, higher reliability, low power consumption and low per-unit cost, unlike general purpose systems.

These qualities justify their increasing presence in cars. The embedded systems, continuously replacing the classical mechanical solutions, have the advantage of being a low-cost alternative. They also open the door for new innovative functionalities. Embedded systems are present today in almost all the cars ensuring different functionalities. These vary from controlling braking, steering, air-conditioning to autonomous driving and automated parking etc...

The embedded system functionalities can be expressed as a simple block diagram. It computes outputs based on supplied inputs (Figure 2.1). Both consist often of connections with other elements of the larger system.

Many of these functionalities are complex with strong real-time and safety constraints that must be met. Developing these systems requires, thus, a rigorous system engineering approach allowing designing the system in a way it satisfies its missions and respects the different constraints. The V-model described in the standard ISO 15288 [46] is the most common and widely used across the industries (Figure 2.2).

The process defines the development steps and milestones allowing developing a system that meet the expectations expressed as stakeholder requirements. The first part of the process focus on the activities leading to the solution, i.e., the system architecture and implementation. While the second part aims to prove that the developed solutions corresponds and meets the functional or non-functional requirements. In our work, we focus on the architecture design phase. This step consists of synthesizing a solution that satisfies system requirements. “It identifies and explores one or more implementation strategies at a level of detail consistent with the system’s technical and commercial requirements and risks. From this, an architectural design solution is defined in terms of the requirements for the set of system elements from which the system is configured” [43]. The result of the design process is an architecture design that specifies the proposed solution for the system specifications.

The architecture, a crucial concept in the system engineering approach, is discussed in further details in the next paragraph.
2.3 System architecture

The goal of system architecture activities is to define a solution that relates to the basic representation in the figure above. The solution architecture, according to [98], is “an abstract, conceptualization-oriented, global, and focused to achieve the mission and life cycle concepts of the system. It also focuses on high-level structure in systems and system elements. It addresses the architectural principles, concepts, properties, and characteristics of the system-of-interest.”. In [85], “The architecture of an embedded system is an abstraction of the embedded device, meaning that it represents a generalization of the system that typically doesn’t show detailed implementation information such as source code or hardware circuit design”. Although the concept of system architecture is defined in different ways across the standards and the literature [46, 44], the majority of interpretations of system architecture are based on the concept of structure. A structure is one possible representation of the architecture, containing its own set of represented elements, properties and inter-relationship information [85]. A system can be described through a set of structures that are inherently related to each other. In [85, 71], an extensive discussion of the structures and the architecture viewpoints are discussed. In this thesis, we will focus on the logical and physical architectures because the main architecture and design activities consist of several iterations between the two until both are complete and consistent.

The Logical architecture aims to represent the functionalities and the behavior of the system. It includes, often, a functional architecture, a behavioral architecture and temporal architecture views. Whereas the Physical architecture aims to determine the main system elements that could perform the system functions and organize them into a physical architecture model. The selection of the technological solutions and design decision is guided by performance constraints or non-functional requirements such as safety and reliability in addition to design constraints like cost.
2.3.1 Logical Architecture

“The logical architecture of a system is composed of a set of related technical concepts and principles that support the logical operation of the system. It is described with views corresponding to viewpoints, and includes a functional architecture view, a behavioral architecture view, and a temporal architecture view. “[98]. Logical architecture aims to provide a description of the system functionalities and behavior. It is composed of a set of concepts that allow capturing such information. It includes often a functional, behavioral and temporal architecture views.

Functional architecture

It is defined as “a set of functions and their sub-functions that defines the transformations performed by the system to complete its mission” in [98]. The functional architecture describes the strategy and the functions integrated by the system to satisfy the system functionalities and stakeholders’ requirements. It can be detailed or abstract but remains technology independent. It focuses on the dependency between the functions and the input and outputs of each of these functions. In this logic, “in order to define the complete set of functions of the system, one must identify all the functions necessitated by the system and its derived requirements, as well as the corresponding inputs and outputs of those functions.” [98]. The functions can though be decomposed and described at different levels of granularity (Figure 2.4).

At the highest level, the system is described in a way similar to “black box”. At this level, the main functionalities of the system or the system itself are described as a single blocks. In order to understand how the system fulfills the functionality, it is decomposed into sub-functions. The decomposition can go deeper until reaching the desired refinement level. Often, the decomposition is carried out until reaching a level where the functions have known, feasible or imaginable physical
2.3. **SYSTEM ARCHITECTURE**

A behavioral architecture model is an arrangement of functions and their sub-functions as well as interfaces (inputs and outputs) that defines the execution sequencing, conditions for control or data-flow, and performance level necessary to satisfy the system requirements [14]. The behavioral architecture is described using inter-related scenarios of functions identified in the functional architecture. The scenarios describe the system behavior in its operational modes. It focuses on the trigger events to which the systems reacts. The system reaction depends on the operational mode. The scenarios set the sequence of functions that are performed by the systems. The sequence is described and driven by control flows that describe the transformation of the inputs into the expected outputs by the system.

**Temporal architecture**

A temporal architecture model is a classification of the functions of a system that is derived according to the frequency level of execution. Temporal architecture models include the definition of synchronous and asynchronous aspects of functions.

2.3.2 **Physical architecture**

A physical architecture is often defined as “an arrangement of physical elements (system elements and physical interfaces) which provides the design solution for a product, service, or enterprise, and is intended to satisfy logical architecture elements and system requirements. It is implementable through technologies”[47].

A physical architecture is thus a result of technological and allocation choices to select the system elements and the relationship between the physical and logical view of the system. The systems elements in the context of the embedded system
are hardware parts. This includes mechanical, electronic, electrical and software elements. In the architecture, these elements are bound together through physical interfaces which can consist of hardware parts and protocols. An example of the physical view of the architecture is illustrated in the Figure 2.5. In this architecture view, representing a part of an automotive embedded system architecture, the main elements of the system and the flows exchanged are represented at a high abstraction level. The system will be detailed and used as a study case in the next chapters.

As in the functional architecture view, the physical architecture can be described at different levels of abstraction depending on the context and the interacting systems at the considered refinement level.

The goal of the physical architecture development activities is to conceive the best possible solution. This relies on engineering skills to identify the suitable technological system elements and the physical interfaces and ensure that these choices answer the systems requirements and constraints. At the highest level the major elements that are considered are:
- ECU
- Input/Output devices
- Data Pathways / Busses
- Microcontrollers
- Memory

The developed architecture solution is often the result of a comparison with multiple alternatives to identify the best according to a set of criteria.

Based on the process presented in [98], building the candidate solutions is a process that can be summed in three main steps:

1. Partition and allocation of functional elements to system elements
2. Physical architecture candidates
3. Candidates assessment and identification of the most suitable solution

These steps are discussed in further details in chapter 5. They will be the key elements on which we will base our approach for the design space exploration and optimization.

From the system engineering approach point of view, the architectural views we discussed so far, are the key steps in the design of any systems, at least, most of
them. A special category, though, might require additional architectural views and a special treatment of some of the non-functional requirements. This is the case for safety critical systems as the safety requirements can have a deep impact on the system architecture. They require also additional attention to concepts such as the safety concept. In the next paragraph, we present the systems safety concept and how it impacts the automotive systems design.

2.4 System Safety

A safety critical system is a system whose malfunction, either directly or indirectly, has the potential to lead to safety being compromised [81]. Contrarily, non safety critical systems are free of such scenarios. The safety critical systems implement different strategies to avoid safety being compromised. The way these systems operate allows categorizing them into active safety and passive safety systems.

Passive safety systems allow reacting to an event (accident) and minimizing the effects and consequences of its occurrence such as airbags systems. Active safety systems aim instead on avoiding the accident though control and mitigation of the causes that might lead to it. We may cite for example driving assistance systems such as collision avoidance and adaptive cruise control etc...

System safety concept is built on a risk management strategy based on identification, analysis of hazards and application of remedial controls using a systems-based approach. From a system engineering point of view, system safety is the application of the adequate engineering and management principles to ensure the risks in the system are lowered to acceptable levels.

In the automotive context where embedded systems are electronics and software intensive, systems are reactive to external and internal stimulus. They implement functional safety strategies to detect, control and mitigate the failures. Functional safety is a part of the overall safety of a system that focuses generally on electronics and related software. It can be defined as the detection of a potentially dangerous condition resulting in the activation of a protective or corrective device or mechanism to prevent hazardous events arising.

In ISO 26262, “Functional safety is the absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems” [30]. ISO 26262 is the standard applied in the automotive industry for dealing with safety constraints during safety-critical systems design. Considering the scope of our subject, we will be focusing mainly on functional safety from this standard point of view. The concepts discussed in the next paragraphs will be mainly in the ISO 26262 perimeter.

2.4.1 ISO 26262

ISO 26262 is the new Functional safety Standard for automotive electric/electronic Systems. The recently introduced standard has a risk based approach that aims to address possible hazards caused by the malfunctioning behavior of electronic and electrical systems. It focuses primarily on risks induced by random hardware faults as well as systematic faults in the system design. The standard focus only on E/E systems. Though, in systems where mechanical and electronics parts interaction
exist, the mechanical parts faults may be included and contribute to safety requirements even though the safety process is not applicable to them.

The ISO 26262 is an extension to the IEC-61508 [42] standard. It brought uniformity by giving the industry its own standard. Adapting the general purpose IEC-61508 to the automotive industry came with multiple changes and new concepts. The new standard defined the safety activities to be performed during the project describing thus the safety lifecycle.

The safety activities follow the development lifecycle until the production and after. The figure Figure 2.6 represents the safety lifecycle as defined by the standard. We can identify three main phases: concept, development and production phases. Moving from a phase to the next is marked by important deliveries such as the functional safety concept and the release for production.

Since we are interested in the system design, we focus on the first phase by the end of which the system’s preliminary architecture and its functional safety concept are fixed. This step is of a special interest because it impacts the system architecture as explained in the next paragraph.

### 2.4.2 Functional safety concept

Functional safety concept is the “specification of the functional safety requirements, with associated information, their assignment to architectural elements, and their interaction necessary to achieve the safety goals” (ISO 26262). The functional safety concept defines the strategy implemented by the system to bring the risks to an acceptable level and achieve the safety of the system. It specifies how the top level safety requirements defined as Safety Goals are derived and allocated to elements in the system architecture. The functional safety requirements describe how
the safety goals will be realized without fixing a specific technological solution. They inherit from the safety goal their safety levels which are equally inherited afterwards by the system element implementing them. The safety levels allocated during this step have a very important role to play in the system design [61]. The functional safety concept is thus a design key step.

### 2.4.3 Automotive Safety Integrity Level (ASIL)

ISO 26262 extends the concept of safety integrity levels (SIL) proposed in the IEC-61508. It is adopted and redefined as Automotive Safety Integrity Level (ASIL). Five safety integrity levels are defined and ordered by criticality as follows: Qm (not safety critical), ASIL A, ASIL B, ASIL C, ASIL D (most stringent). In the Preliminary Hazard Analysis (PHA) phase, the identified undesired events are attributed a safety level based on three parameters: Severity, Exposure and Controllability. Severity illustrates the undesired event potential consequences. Exposure determines the rate of appearance of the event. Controllability highlights the possibility to control the undesired event. These parameters are attributed values from the list provided by the standard illustrated in Table 2.1, Table 2.2 and Table 2.3. The Table 2.4 illustrates how the combinations of these parameters lead to the safety level.

The obtained ASIL determines the qualitative and quantitative levels that the element, implementing the safety requirement, should meet. It identifies also the necessary safety activities to be conducted during the safety lifecycle to ensure that the risk is brought to an acceptable level.

<table>
<thead>
<tr>
<th>Severity Class</th>
<th>S0</th>
<th>S1</th>
<th>S2</th>
<th>S3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>No injuries</td>
<td>Light and moderate injuries</td>
<td>Severe and life-threatening injuries (survival probable)</td>
<td>Life-threatening injuries (survival uncertain), fatal injuries</td>
</tr>
</tbody>
</table>

Table 2.1 – ISO 26262 Severity Levels

<table>
<thead>
<tr>
<th>Exposure Class</th>
<th>E0</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
<th>E4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Incredible</td>
<td>very low probability</td>
<td>Low probability medium probability</td>
<td>High probability</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 – ISO 26262 Exposure Levels

<table>
<thead>
<tr>
<th>Controllability Class</th>
<th>C0</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Controllable in general</td>
<td>Simply controllable</td>
<td>Normally controllable</td>
<td>Difficult to control or uncontrollable</td>
</tr>
</tbody>
</table>

Table 2.3 – ISO 26262 Controllability Levels

The safety levels fix qualitative and quantitative targets for the system architecture for the corresponding safety goal. They address systematic failures and random
Table 2.4 – ISO 26262 ASIL determination matrix

<table>
<thead>
<tr>
<th>Hardware failures</th>
<th>ASIL B</th>
<th>ASIL C</th>
<th>ASIL D</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single point faults metric</td>
<td>&gt; 90 %</td>
<td>&gt; 97 %</td>
<td>&gt; 99 %</td>
</tr>
<tr>
<td>Latent faults metric</td>
<td>&gt; 60 %</td>
<td>&gt; 80 %</td>
<td>&gt; 90 %</td>
</tr>
</tbody>
</table>

Figure 2.7 – ASIL Fault Metrics

<table>
<thead>
<tr>
<th>ASIL Level</th>
<th>Random hardware failure target values</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>&lt; $10^8 h^{-1}$</td>
</tr>
<tr>
<td>C</td>
<td>&lt; $10^7 h^{-1}$</td>
</tr>
<tr>
<td>B</td>
<td>&lt; $10^6 h^{-1}$</td>
</tr>
<tr>
<td>A</td>
<td>&lt; $10^6 h^{-1}$</td>
</tr>
</tbody>
</table>

Figure 2.8 – ASIL Random Hardware Failure Targets

The ASIL allocated to the safety requirements implemented by the subsystems or components heavily impacts the architecture and the components choice as the targeted metrics increase with the ASIL level. It also has an impact on the development effort and cost since the ASIL is automatically inherited by all the subsystems and elements implementing the safety requirement. Though, it is possible to reduce the inherited levels allowing controlling the development effort and cost of the solutions. This technique is discussed in further details in the next paragraph.
2.4.4 ASIL Reduction

The safety integrity levels allocated to the hazards are inherited automatically by the resulting requirements and consequently by the elements to which the requirement are allocated. They are, in this way, inherited and implemented in the form of functional safety requirements. The standard leaves though an opportunity to reduce these inherited levels. Such opportunity can be very interesting for the system designer. It could allow reducing the requirements implementation complexity. Reducing the safety levels is though demanding and should be carefully applied and justified. Two approaches are proposed by the standard for this purpose. The first is using technological analysis to prove the non-likelihood of failures leading to the undesired events. The second is using the ASIL decomposition. It is not only a safety level reduction approach but a decomposition approach. In Part 9, the standard introduces an ASIL decomposition approach allowing reducing the safety levels by decomposing the safety requirements over redundant and sufficiently independent elements. The decomposition when applied results in safety requirements with lower ASIL allocated to the redundant elements. The decomposition follows the patterns presented by the Figure 2.9.

![ASIL Decomposition Patterns](image)

These patterns can be applied at different levels: system, component, hardware or software level and can be applied as many times as needed. It is important though that the decomposed requirements be allocated to sufficiently independent elements. Independence is defined by the standard as the “absence of dependent failure between two or more elements that could lead to the violation of a safety requirement; or organizational separation of the parties performing an action”. The dependent failures are, in the standard definition, “failures whose probability of simultaneous
or successive occurrence cannot be expressed as the simple product of the unconditional probabilities of each of them”. This can be translated to the existence of Common Cause Failures (CCF) or/and Cascading Failures (CF) (Figure 2.10).

As illustrated in Figure 2.10, a failure is considered as cascading if the “failure of an element of an item is causing other elements of the same item to fail”. Whereas, CCF corresponds to the situation where a root cause can lead to simultaneous failures of the elements.

The application of ASIL decomposition have been discussed in the literature, especially about its benefits and how it should be applied during the concept phase [108], [101], [21].

2.4.5 Mixed Criticality cohabitation

The elements implementing mixed ASIL safety requirements inherit the highest ASIL and is developed accordingly. This lead to implementing safety requirements at higher ASIL than what is allocated due to the cohabitation with higher ASIL levels requirements. But, this is not automatically applied. It is possible to avoid this and ensure the cohabitation without rising the safety level if the freedom from interference can be proved. As required in the standard, the cohabitation is allowed if the absence from CF can be proven. If the lower safety level elements can not lead to the failure of the higher safety levels, the cohabitation is tolerated and the elements can be developed at their respective ASILs.

2.5 Objective and Research Questions

Our research subject is focused on automotive embedded systems, their architecture and the functional safety from the ISO 26262 point of view. We illustrated in this chapter the main concepts that allow understanding the challenge of automotive safety related system design. Our work focusing on the architecture design phase, it allowed us to identify the key architecture design phases. The architectural choices are often made at early stages. The logical and physical architecture are the
result of engineering iteration at an abstraction level where the design constraints can be taken into account. Safety constraints, for example, need to be addressed since the early design decisions for effective and compliant solutions. We reviewed the functional safety concepts introduced by ISO 26262 that have an impact on the design decisions such as the ASIL and their quantitative targets as well as the ASIL decomposition solutions.

This chapter allows us to refine the research subjects into a set of questions which we will try to answer through the next chapters. These questions are the following:

— **Q1**: How can the embedded system architecture be optimized?

  The objective of our project is to optimize the automotive system architecture. A first step towards this goal is to identify the best approach to represent the architecture itself. It is trivial that in order to study, analyze or optimize the architecture, a sufficiently detailed and clear description (model) of this architecture should be presented in a form that can be exploited. We tackle the question of architecture modeling in the first section of the next chapter.

— **Q2**: How to take into account effectively the safety constraints and do not incur unnecessary costs?

  Safety constraints are the result of multiple design choices. The design choices allow reaching a feasible compliant solution but some lead to high cost solutions. Alternative combination of design choices can lead to a more cost effective design. In the chapter 5, we attempt an approach that can fulfill this objective.

— **Q3**: Which part of the Safety/System process can be partly automated for the optimization?

  Optimization may require the iteration of some design activities. We will need to identify the design activities to be iterated in order to explore the alternatives and investigate the ways they can be automated for the optimization purpose.
3.

Systems Architecture Evaluation and Optimization

3.1 Introduction

In this chapter, we discuss the system architecture optimization techniques and tools. Through discussing the state-of-the-art approaches, we identify the gaps and the applicability of some approaches in our subject scope. Different optimization techniques are often applied to reach the optimization targets. They are discussed in details in the first section of this chapter. The optimization approach is also based on evaluation techniques that allow analysing the candidate solutions and ensure that the design constraints are respected and the optimization target is attained. These activities are conducted around a model that allows describing the candidate solution and analysing it. Since multiple modelling languages are today used to represent different viewpoints of the architecture, we identify the models that can be applied for the embedded system architecture description and evaluation.

3.2 Architecture Optimization

The design activities are in the center of compromise making. The designers are faced with multiple constraints that impact each other mutually. During these activities, different architectural alternatives and technological choices can be available. Considering the design space size and the time it takes to investigate and evaluate all the alternatives, a design space exploration is needed to reach an optimal design. Optimization can be performed at different levels of the design: at logical architecture or at physical architecture. It can be focused on system architecture or just software. A design can be optimized using different techniques depending on the optimization targets. The optimization may consist of a single or multiple objectives approach and concerns one or multiple parameters such as cost, real time perfor-
The optimization is most effective when conducted early in the development process. At this level, the cost of modifications is low and more design alternatives can be considered (see Figure 3.1). Thus, we propose an optimization approach focusing on the concept phase. The advantages of applying it at the concept phase are:

— the ability to check alternative functional decomposition,
— the ability to explore alternative safety requirements decomposition and inheritance schemes,
— the ability to challenge the choices of the main architecture elements,
— the ability to control the impact of the design constraints on the solution cost at macro level,
— the effectiveness of the safety requirements decomposition when performed early in the concept phase.

The architecture optimization may differ though depending on the design target. In the next paragraph, we will review the main embedded systems design optimization targets.

### 3.2.1 Optimization targets and constraints

The optimization approaches aim, in general, to deal with one or multiple design constraints and reach solutions that respect or optimize some of the following parameters.

— Cost:

Cost is logically the most trivial parameter to consider during the optimization. It is a key decision in all industries. A design cost can though be approached in different aspects like the components costs, the development effort, the testing and validation activities and the time to market. The cost is often estimated based on engineering expertise or cost models such as COCOMO, COSYSMO, COSMIC. The approaches proposed in [48, 55, 93, 102, 1, 103, 113] are examples of taking into account the cost factor in the optimization process.
3.2. ARCHITECTURE OPTIMIZATION

— Performance:
The performance attributes of embedded system can have different aspects. It can be focused on response time, resource usage, or energy consumption. The performance targets can be reached using techniques based on hardware and software parameters selection. Optimization approaches for the embedded system target often these attributes. Some aim to optimizing the architecture to reach the response time targets such as [3, 72, 78, 79, 93, 105]. In [50, 73, 79, 80, 107], the approaches aim to optimise the resource usage and reach the required CPU loads. While in [77, 104], the energy consumption is targeted.

— Reliability:
Reliability is the ability of the system to continue providing the functionalities correctly during a specified period of time in certain conditions [86]. It is a key concept in the design activity. The architectural and technological choices can limit or ensure the capacity of the design to reach the desired reliability targets. The evaluation of the system reliability is based on probabilistic models and an analysis of the system failures and their causes as well as the usage profile of the system. The approach in, [75, 76, 91], proposes a reliability driven optimization approach.

— Safety:
A design optimization approach applied to critical system aims often at reaching the safety objectives. Often, the optimization focus on the scenarios leading to the violation of the safety goals and aims at eliminating these scenarios. If the hazard is latent then the optimization objective is the quantitative and qualitative targets that need to be met. Another optimization strategy is to focus on the safety levels. The safety levels and the allocation of the safety requirements to system elements have a deep impact on the implementation effort and the components choices. It impacts the effort and the cost of the solution as the qualitative and quantitative targets of the safety levels vary considerably. The cohabitation of mixed criticality elements and the use of multi-core to ensure the cohabitation is another of the main safety optimization concerns [38].

In our subject we are interested in safety critical systems. While all the aforementioned optimization targets are applicable, we will focus on the safety attributes and the cost of the system architecture. Safety will be approached from the ISO 26262 point of view. As for the cost it will be approached as early estimation of the architecture cost. The evaluation of the architecture regarding these parameters will be discussed in further details in the next sections.

3.2.2 Optimization Techniques

Optimizing system architecture is often conducted through two main mindsets. The first consists of starting from an initial design. It is often simple and does not meet the target criteria. The optimization process tend thus to modify/transform this solution using different techniques until reaching all the constraints. The second mindset consists of building/synthesis of a solution based on a more abstract description of the design (functional architecture for example). The synthesis process takes into account the constraints defined beforehand or that can be the result of the synthesis process itself. Both of the two approaches allow reaching multiple can-
didate solutions, if they exist. The optimal design is then selected based on criteria such as cost.

**Architecture transformation**

Starting from an initial design, the optimization approaches exploit engineering concepts to transform the design. These concepts are applied depending on the design constraints and the objective metrics. We detail briefly the main concepts used for architecture transformation and optimization:

— **Redundancy introduction**
Redundancy allocation is a technique widely used to deal with safety or reliability constraints. It consists of making an element or many redundant in the architecture by duplicating them and implementing a voting technique. This allows, if wisely introduced, to enhance the system reliability attributes or reduce the risks of the system failures. The redundancy can be introduced at different of the design: system or component level, functional or physical level. Redundancy can also be homogenous (same element) or heterogeneous (different elements). We can see that this allows a big number of choices and lead to high number of possible combinations that result from: choice of the element to make redundant, the redundancy strategy, the voting pattern, the elements choice. This problem is often entitled as Redundancy Allocation Problem. A large amount of research effort has been focused on this problem [76, 90].

— **Mapping & Design parameters (Scheduling, hardware parameters)**
During the design, the functional architecture is mapped to the physical architecture. This step allows allocating the functionalities to the architecture elements and the available resources. At this level, in order to meet the performance constraints and respect the resources consumption, approaches focused on the mapping problem in relation with attributes such performance, energy, reliability and safety. The problem is especially dealt with at the software components allocation to hardware. The approaches in [52, 114] are dedicated to the allocation problem modeling and optimization.

— **Elements choices**
While the mapping problem focus on the allocation of the functions to the elements of the architecture. Other approaches focus on the elements choice that the designers need to make. The system can be built using different elements capable of ensuring the same functionalities. The elements choice can have an impact on cost and non functional attributes, thus, approaches such as [83] propose optimization approach for an efficient elements choice regarding the design constraints.

— **Safety level allocation/decomposition**
Safety levels allocation are a part of the functional design concept and have an impact on the architecture, elements choices, development effort and cost. Often, multiple safety levels allocation scenarios can be applied. Approaches in [8, 12] allows first to identify the scenarios, non trivial to find, and identify the optimal combination considering the architecture and the safety requirements.

In the context of embedded systems we are interested in, these techniques can be applied and readapted to the context of the ISO 26262 as it has been proven in previous works. But, since we are considering the optimization at early design
phase, some detailed concept parameters may not available. Consequently, optimization techniques based on a detailed estimation of the resources usage or the energy consumption can not be applied. They require a software architecture which is supposed to be developed in later design activities in the design workflow we are considering. Considering the optimization is at a high abstraction level, an approach based on the safety levels allocation, elements choice and redundancy introduction seems better suited in case of an architecture based optimization.

Architecture Synthesis

The idea behind Architecture synthesis is start from a functional description that is implementation details free. The implementation step, i.e., physical architecture design is left to an optimization process. The process is in this case capable of generating candidate solutions that corresponds to the functional description and takes into account the considered design constraints. The candidate solutions can be at a detailed level, the solution is a feasible solution, fully/ enough detailed to allow moving to the next steps of development. It can also consist of just guideline that allows orienting the design decisions.

The latter can consist of design pattern with design decision trees. A design pattern is “description of communicating objects and classes that are customized to solve a general design problem in a particular context” [33]. It allows reaching a suitable solution for a common problem based on previous successful solutions. Design patterns and their application for architecture design are discussed in [5].

Architecture or design Synthesis is a translation process from a behavioral description into a structural description [31]. It is also referred to as High level synthesis, Electronic system-level synthesis. It consists of an automated design process that lead to a physical implementation of the system based on a logical description that is implementation decoupled. We can identify different types of design synthesis [gajski] illustrated depending on the abstraction level of the design.

The architecture synthesis is mainly used in System On Chip (SOC) design and covers multiple aspects of the SOC design. The approaches use a behavior architecture model to build the physical architecture. The models are referred to as Models Of Computation (MOC) and include data flow graphs (DFG), Process Networks (PN). The synthesis process may cover some of these activities to reach the desired design [15]:

- Lexical processing
- Algorithm optimization
- Control/Dataflow analysis
- Library processing
- Resource allocation
- Scheduling
- Functional unit binding
- Register binding
- Output processing
- Input Rebinding

In our study scope, system level synthesis and mainly on the resource allocation and control/dataflow analysis activities can be applied. These activities can be
CHAPTER 3. SYSTEMS ARCHITECTURE EVALUATION AND OPTIMIZATION

processed at high abstract level and particularly at system level. But in the current state of the art, no architecture synthesis approach takes into account the context of automotive embedded system into account.

3.2.3 Optimization algorithms

Optimization techniques may adopt different approaches for problem modeling, design constraint and optimization objectives but they use often common optimization algorithms. Depending on the problem in study, these algorithms are adapted in order to reach the optimal solution. The techniques and algorithms used in the state of the art can be put into three main categories: Exhaustive, Exact, and metaheuristics.

Exhaustive algorithms attempt to iterate through all the possible combinations in order to reach the optimal design. Of course such algorithms have scalability difficulties. Thus they are enhanced to reduce the number of combination checked and focus on the combinations that can lead to a constraint fitted solution. An approach with an enhanced exhaustive algorithm was used in [69] to optimize the safety levels allocations. To deal with the same problem, linear programming was used in [68] and [12]. Linear programming was used also for optimizing the deployment in [36, 39, 58, 78] and for architecture synthesis in [9]. Linear programming can be applied to the problems that can be translated into linear equations with an objective function to be minimized or optimized. If the problem is defined through binary expressions as in [12]. It can also be described as a set of logical properties as in [105] where SMT solver is used to solve and optimize the problem. These algorithms have the advantage of providing exact solutions to the problem if any exists. But, as in exhaustive algorithms, the problem size impacts these algorithms performance.

Metaheuristics, on the other hand, are algorithms that allows reaching the best solution considering a predefined target level or the time allowed. The design space exploration is guided by the problem size and nature-inspired algorithms. They have been widely applied in architecture optimization because of their efficiency in dealing with large scale combinatory problems. These heuristics can also be adapted and applied to a variety of architecture optimization problems such as redundancy, safety levels allocation, Elements choice. The Genetic algorithms [2, 4, 20, 54, 64, 72, 77, 79, 82, 84, 89, 92, 1, 109, 114], Simulated Annealing [52, 113], Ant Colony [29, 76] and Tabu search [8, 111] are the most used algorithms in literature.

3.3 Architecture Evaluation

Models can be used to support integration of the hardware and software components into a system, as well as to support verification that the system satisfies its requirements. This often involves integrating lower level hardware and software design models with system-level design models which verify that system requirements are satisfied. System integration and verification may also include replacing selected hardware and design models with actual hardware and software products in order
3.3. ARCHITECTURE EVALUATION

Models can also be used to define the test cases and other aspects of the test program to assist in test planning and execution. Models can be used to explore a trade-space by modeling alternative system designs and assessing the impact of critical system parameters such as weight, speed, accuracy, reliability, and cost on the overall measures of merit. In addition to bounding the system design parameters, models can also be used to validate that the system requirements meet stakeholder needs before proceeding with later life cycle activities such as synthesizing the detailed system design. Architecture evaluation can be put in two categories. The first concerns the performance and functional requirements. Architectures are evaluated to check if the design meets the system specification. This includes system behavior, functionality, and performance. Techniques such as simulation, model-checking, and fault-injection are used to ensure that the design meets the targets.

The second category concerns the non-functional requirements. This includes Reliability, availability, maintainability, and safety requirements. Different analysis techniques are used to ensure the requirements are met. We will limit here the scope to the safety requirements because it is our main research subject. The safety analysis techniques allow analyzing the system failures and the scenarios leading to the violation of the safety requirements.

3.3.1 Safety Analysis

The safety activities are conducted all along the project lifecycle. Its main role is to assure the compliance of the developed product as well as the safety of its use. The safety engineers aim through these activities to mitigate the risks and bring them to an acceptable level. Qualitative and quantitative analysis is, thus, needed. Different tools and approaches are used for that matter.

Safety Process

The safety activities target different stages of the development process and are conducted at different levels: system/component, functional/physical, see Figure 2.6.

The Figure 3.2 illustrates the safety activities conducted at system level. Based on the system requirements the PHA allows identifying the undesired events, the safety goals, and the required ASIL. These elements combined with the preliminary system architecture allows defining a safety concept. The safety analysis allows later to check the correct allocation and implementation of the derived safety requirements. It is based on analysis techniques such as FTA and FMEA. At system level, an additional effort is required also for the dependent failures analysis to ensure the independency or the cohabitation requirements are verified.

Safety Analysis techniques

The safety analysis aims to prove that the risk is reduced to an acceptable level. An analysis is thus needed to identify the possible causes and effects of components
Figure 3.2 – Safety Activities at System level

failures on the system behavior that may lead to a hazardous event. The causality relation could be studied qualitatively where only the failure logic is addressed. Fault tree and FMEA are often used for this matter. When the probability of the failures and the hazard are addressed, the system is studied quantitatively. The same tools used for qualitative analysis may be used here with a little variant: quantitative FT and quantitative FMEA.

Failures trees are among the first tools developed for risk analysis. Developed in the 60’s, it is the most common and by default tool used in many fields [81]. The analysis tool is a Top-down deductive approach. Starting from the Top-events, the different combinations of causes are hierarchically identified until reaching the Basic-events. The depth of the analysis and its perimeter can vary. The engineers can expand the analysis perimeters and the FT depth if needed [66]. The Fault trees are mainly composed of events and gates. Different tools support the Failure Tree analysis.

FMEA is an inductive approach. The analysis starts from the known causes and attempt to analyze their impact at higher level (Top-level). The exhaustive analysis of the failures modes ensure that no failure mode was left out. The FMEA bottom-up approach is sometimes redundant with the FTA analysis. Yet, both of these tools are still complementary.

These classical tools which appeared half a century ago are still the most used and the by default techniques in almost all the domains. FMEA and FTA are widely used and recommended by the different standards. When manually constructed, they rely on the safety engineer skills and expertise to identify the possible causes of a hazard and how the failure of the components may affect the behavior of the system. These tools have also limits when it comes to capturing and addressing new complex systems behavior properties resulting from the dynamic aspect. Extensions have been developed to cope with it. We may cite for example DFT, SEFT, CFP.
3.3. ARCHITECTURE EVALUATION

These techniques are described in more details in the Chapter A.

Model Based Safety Analysis

The growing size and complexity of the automotive systems led consequently to a growth in the safety analysis complexity and effort. Models were, as in the system engineering discipline, a solution to cope with these challenges. Introducing safety properties/behavior in the system engineering models or build dedicated models proved the benefits of the model based safety analysis. The models allowed, with different approaches, to provide the needed proofs required for the safety assessment of the architecture. Reuse, traceability and automation are the key advantages of these approaches.

The model based approaches used to capture the failure behavior of the system can be put into two categories: failure behavior coupled with the functional model or separate failure model.

Coupling the dysfunctional model with the functional model allows the synchronization between the two aspects of the system. By annotating the functional model, like in FPTN, FPTC, AADL ERROR ANNEX, EAST-ADL, LUSTRE FM, the dysfunctional model adopt the structure of the functional model and the dependencies defining its elements. Any changes affecting the functional model can affect consequently the dysfunctional model. This allows to keep a synchronization between the two models. Despite the link between both of them, during the analysis, the two models can be analyzed separately. In some approaches, the two models are tightly coupled and the failure behavior is integrated with the functional behavior. The used modeling languages do not support the safety concepts natively which leads to modifying the functional model. Such approach is proposed for example in [51] and is know to be hard to modify and maintain.

On the other hand, model based approaches such as Altarica, SAML aim to building a dysfunctional behavior model that is totally decoupled from the functional model.

For both cases, the models are exploited to ensure the system meets the safety requirements qualitatively or quantitatively. This is proven through the use of simulation [altarica, FPTC], model checking [aadl, east adl, lustre, altarica] or both techniques or using the FTA/FMEA [FPTN].

3.3.2 Cost Estimation

The need for a cost estimation techniques have been confirmed through the years particularly for software intensive systems such the automotive systems. During the early concept phases of an electronic system, the engineer has to decide on issues like the electronic hardware to be used, the mapping of the functionalities to these elements and the communication paths and technologies to be used. These choices are often made based on engineer expertise without an automatic examination of the alternative solutions. As shown in the previous chapter, different approach were proposed to find the alternative solutions and reach the optimal architecture. The common point between these approaches that are driven by cost is the use of a cost
estimation models. These models allows to have a cost indicator to help making trade-offs between the design alternatives.

The cost estimation models are used often during the first steps of the project life cycle with vague and little details about the system and the design. That in mind different approaches have been proposed to cope with the scarcity of the details and improve the precision of the estimation. The estimation methods developed in the state of the art could be classified in four main categories: Intuitive, Analogical, Parametric and Analytical:

— The intuitive method is based on the experience of the estimator. The result is always dependent on the estimator’s knowledge.
— The analogical method attempts to evaluate the cost of a set or a system from similar sets or systems.
— The parametric method seeks to evaluate the costs of a product from parameters characterizing the product but without describing it completely.
— The analytical method allows evaluation of the cost of a product from a decomposition of the work required into elementary tasks.

These methods cannot be used during the whole life cycle. Some methods are better adapted than others depending on the context. Fig shows the applicability of each model during the project life cycle steps. It is clear that we would be logically interested in the Analogical and Parametric methods since our works focus on the concept phase.

![Diagram of Cost Models Applicability in a Project Lifecycle](image)

Figure 3.3 – Cost models applicability in a project lifecycle

Analogical Cost Estimation Techniques employ similarity criteria based on historical cost data for products with known cost, such as regression analysis models or back propagation methods. On the other hand, Parametric models are derived by applying the statistical methodologies and by expressing cost as a function of its constituent variables. These techniques could be effective in those situations where the parameters, sometimes known as cost drivers, could be easily identified. Parametric models are generally used to quantify the unit cost of a given product. A wide range of parametric models can be found in the literature. The reader can refer to [28] for a detailed study of state of the art on the parametric models. These approaches remain very metier or product dependent as the parameters to be taken into account depends on the considered cost drivers. A widely used example of parametric model
is the COCOMO model. Mainly for software project, the model is commonly used and has been adapted to different industries. It remains based, amongst others, on estimations about the size of the code: a parameter that often cannot be estimated precisely during concept phase. A cost model such as COSMIC [62, 63, 99] has thus been proposed. For this model, it is not the size parameters but the functional points that are used to describe the size and the complexity of the functionalities of the system. We may cite for example the works of [6, 7]. This model remains though metier oriented and can’t be used for product level estimations. In order to take into account the software and the hardware architecture, some works proposed models to take into account efficiently more parameters. The works of [37, 56] brought the parametric approach to a product level allowing to take into account different cost drivers. These models treat the size and the complexity of the system differently using respectively functional points estimation and the requirement number and type. But while COSYSMO focus on estimating the development cost as an effort, the cost model proposed by [37] include also the architecture costs. It also to be noted that COSYSMO is a generic model that can be used for different industries. On the other hand, the model proposed by [37] is developed for automotive industry and not as generic as COSYSMO. The parametric method is very useful because of its rapidity of execution. It can be, though, criticized for working like a “black box”: that is to say that from the specifications the only results we obtain are different costs. We do not know the origin of these costs which can discourage users. It needs also a calibrating phase based on previous projects to improve the model precision.

3.4 Model Based System Engineering

The complexity of the systems and the high constraints that drive the design show the importance of the architecture design activities. Capturing the complexity and functionalities of the systems to support the development or to ease the communication are a challenge for the engineers. For these reasons, models are used during the design activities. There are many definitions of the model concept. In [87] a model is defined as "a selective representation of some system whose form and content are chosen based on a specific set of concerns; the model is related to the system by an explicit or implicit mapping."

In a model based systems engineering context, models help defining, analyzing and communicating a variety of concepts that are often used during the design. "Modeling serves to make concepts concrete and formal, enhance quality, productivity, documentation, and innovation, as well as to reduce the cost and risk of systems development "[98]. It facilitates also the cost estimation, the trade-offs evaluation and the design improvement tasks.

A system model represents aspects of a system and its environment. There are many different types of models, as there are varieties of purposes for which they are built. There are also multiple modeling techniques that allow the descriptions of the system architectural views. The logical architecture can be described, for example, using modeling techniques such as SADT [11, 41] (Structured Analysis Design Technique), SA-RT [45] (System Analysis & Real Time), Logical DFD [49], FFBD (Functional Flow Block Diagram) and eFFBD (enhanced Functional Flow
Block Diagrams) [67]. Physical Block diagrams and Physical DFD [71] are often used for the physical architecture modeling.

Model based engineering approaches are widely used in industry including automotive [17, 18, 19, 27, 57, 95, 94], avionics [60], Nuclear etc ... While the modeling techniques are commonly used between the different approaches, different modeling languages are used to implement the modeling techniques and ensure the efficiency and the compatibility between the models and the industry needs. Modeling languages such as SysML [87], Modelica [97, 23, 24], MARTE [34, 88, 112] or SIMULINK can be used for different systems modeling especially real-time and software intensive systems. Languages such as EAST-ADL [19, 18, 22, 27, 100, 110] and AADL [25, 26, 40] targets specific industries, automotive and avionic respectively.

Each of the modeling languages supports a set of concepts allowing reaching the purpose from system modeling. Models are developed not only for design specification. They are also built to support the integration, verification and validation activities. System modeling is thus a key step in design evaluation and optimization. The model is the support of design activities and optimization, eventually. In the next chapter, we discuss the modeling languages. We apply the fittest on a study case and investigate how the models can be exploited for the optimization.

### 3.5 Conclusion

In this chapter, we reviewed the state of the art approaches and techniques related to the main research questions Q1, 2 and 3 that we fixed in the previous chapter.

We studied the architecture optimization problem from the literature point of view: optimization targets, constraints and objectives. This allowed us to position our subject scope among these works. It proved that our research objective have common points and constraints with previous projects focused on the automotive or similar industries. As a first analysis, some of these approaches can answer a part of the research questions we set. But in order to assess the applicability and the eventual limits, we decided to select the most fitter among them and apply them on a study case. The results should allow us to assess the gap, readapt some of these approaches and propose a coherent and effective alternative.
Motivating example and Research Methodology

4.1 Introduction

In this chapter, we will discuss the approaches and the applicability of the tools discussed in the previous chapter. We will try to apply some of these elements in attempt to reach our objective and answer the identified research questions. Based on the results, we refine the research axes and the solution to be developed. It is here that we start taking into account the Valeo constraints in the choices made.

4.2 Study case

System overview In order to illustrate the constraints and the scope of design activities, we selected a real-life critical automotive system to use as a study case. This will allows a better understanding of the challenges and the needs of the systems and safety engineers. The study case consists of Passive Entry Passive Start System (PEPS). PEPS manages the access and the use of the car by authorized users. The passive entry means that the user when holding an authorized ID can have access to the vehicle without the need to manipulate it. Approaching the vehicle and manipulating the door handle should be enough to unlock the car and gain access. The passive start is the feature allowing after the access to start vehicle and use it without the need to manipulating the ID.

The system main functionalities consists of:
- Access management:
  - Lock: allow to lock vehicle and block the access to non authorized users
  - Unlock: grant the access to the car for the users with the wright ID
  - Power States management
CHAPTER 4. MOTIVATING EXAMPLE AND RESEARCH METHODOLOGY

— Power states: allows to manage and command the operational modes of the car and the electronic systems
— Engine start and stop: manage the engine start and stop upon the user commands.

Derived from the main functionalities, the system manages the steering lock by locking or unlocking it as required by the operational scenario. It also control the power relays for the power states management functionality as well as the engine state. The system description and the scope of its functionalities puts the system in the safety critical category. The failure in engine management or the steering lock control could lead to hazardous scenarios requiring a hazards analysis and the application of the ISO 26262 if required.

The preliminary hazards analysis identified six safety goals in relating with the system functionalities and its use scenarios. They are allocated ASILs ranging from A to D. This illustrates the interest of the study case. It has multiple safety goals with mixed criticality from the most stringent, i.e. ASIL D, to the less stringent, ASIL A. The safety goals and their allocated ASIL, illustrated in Table 4.1, are inherited later on by the system elements based on the preliminary system architecture.

<table>
<thead>
<tr>
<th>Safety Goals</th>
<th>ASIL</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG1: The system shall be able to crank the engine in emergency situation</td>
<td>A</td>
</tr>
<tr>
<td>SG2: The system shall not crank the engine when not required</td>
<td>B</td>
</tr>
<tr>
<td>SG3: The system shall not crank the engine when the steering column is locked</td>
<td>A</td>
</tr>
<tr>
<td>SG4: The system shall not lock the steering column when the vehicle is moving</td>
<td>D</td>
</tr>
<tr>
<td>SG5: The system shall not shut down the ABS function</td>
<td>B</td>
</tr>
<tr>
<td>SG6: The system shall not stall the engine</td>
<td>B</td>
</tr>
</tbody>
</table>

Table 4.1 – PEPS safety goals

The functionalities of the system are diverse and require an interaction with different systems and networks on the vehicle level.

**Proposed architecture** The proposed designed by the engineers is illustrated in the Figure 4.1. It consists of distributed architecture implementing the functional architecture over a set of ECUs interconnected through LIN and CAN networks as well as hardware lines. The basic functions identified in the functional architecture are allocated to these ECUs. Each play a role in the system mission as follows:

— **PEPS ECU** is mandatory for both passive entry and passive start. PEPS_ECU is hosting all functional strategies required for all PEPS system functionalities.
— **ABS ECU** allows the system to have an input information about the movement of the vehicle.
— **BCM ECU** is an ECU dedicated to vehicle body management. BCM has under its control the doors, trunk and backdoor latches.
— **ESCL ECU** is dedicated to the lock and unlock of the steering wheel
— **TCM** provide the needed information about the engine state
— **HSU/PSU**
4.2. STUDY CASE

PSU allow the detection of the user and allows the PEPS to prepare for the user commands.

HSU is the element allowing the interaction between the system and the user.

Figure 4.1 – PEPS Proposed Solution

Safety Concept In the proposed architecture, the ECUS inherit the safety requirement from the safety goals depending on their role in the mitigation of undesired events. This results in the ASIL allocation illustrated by the Table 4.2.

<table>
<thead>
<tr>
<th>PEPS ECU</th>
<th>ESCL ECU</th>
<th>ABS</th>
<th>ECM</th>
<th>Neutral Switch</th>
<th>Engine Switch</th>
<th>Start Relay</th>
<th>IG relay</th>
<th>Clutch pedal switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG2</td>
<td>A</td>
<td>-</td>
<td>A</td>
<td>A</td>
<td>QM</td>
<td>QM</td>
<td>QM</td>
<td>-</td>
</tr>
<tr>
<td>SG3</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>QM</td>
<td>B</td>
<td>B</td>
<td>QM</td>
</tr>
<tr>
<td>SG4</td>
<td>A</td>
<td>-</td>
<td>A</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SG5</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SG6</td>
<td>-</td>
<td>-</td>
<td>B</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SG7</td>
<td>B</td>
<td>-</td>
<td>B</td>
<td>QM</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 4.2 – ASIL Inherited by ECUs

In the proposed solution, a decomposition of the SG4 is used. The decomposition pattern ASIL D -> ASIL B & ASIL B was applied. The choice of decomposition was motivated by feasibility and cost constraints. It is based on a redundancy of the vehicle movement input.

By acquiring the data from independent and redundant sources, the decomposition can be applied over the acquisition requirements allocated to the redundant
elements. It led to independancy constraints resulting in design choices particularly in the PEPS ECU architecture and the redundancy of the movement data provided by the ABS ECU. The decomposition is illustrated by the fault tree Figure 4.3. Each of the redundant functional flows allows the respect of the safety goals independently and only the joint failures lead to the undesired event.

![Figure 4.3 – Safety goal decomposition](image)

The decomposition allows to reduce the ASIL levels inherited by the ECUs from the SG5 presented in Table 4.2. The ASIL allocated to these ECUs is brought down to ASIL B with independancy requirements. The independancy is ensured by adopting different physical implementations. This justifies the use of CAN and Hw flows for the vehicle speed exchange between the ABS and the PEPS ECU.

### 4.3 State of the art: Optimization approaches applicability

The approaches identified discussed in the previous chapter were selected based on their relevance to the scope and the objective of our research subject. But not
all of them are applicable to systems such as the study case. We decided to apply the fitted approaches and use the results to conclude their applicability and use it as a starting point for the next research activities.

First, we concluded that the system engineering process proposed in SEBOK and re-adapted by [94] as SysCars should be applied. The process is based on industry recommendation and previous experiences as well as the state of the art. Supporting SysCars process, an adapted SysML profile has been used. The profile has though a key limitation consisting of the lack of safety analysis support. This requires that the failure behavior needs to be developed and analyzed separately. EAST-ADL support, on the other hand, the safety analysis and has been developed with ISO 26262 in mind. The availability of a modeling framework, the support of safety model and the integration of safety analysis tools such as HIP-HOPS for the automatic FTA and FMEA analysis make EAST-ADL more adapted for the a safety critical system modeling.

For the failure analysis, HIP-HOPS have the advantages of intuitiveness and applicability at different levels (logical, physical architecture). The failure model is based on annotation of the functional model with failure behavior description in a tabular form. It allows the automatic generation of FTs and FMEAs. It provides also the analysis of the MCS and the optimization of the ASIL allocations.

Considering the system model abstraction level, the optimizations approaches based on software elements mapping and scheduling can not be applied. On the other hand, approaches based on redundancy introduction and elements choice as well as safety level allocation can be applied to the study case. HIP-HOPS integrates such optimization approach and we will use it to investigate the effectiveness of these approaches in optimizing the system architecture.

### 4.3.1 Optimization Model

We focus in the system modeling on three key viewpoints: Functional architecture, Physical architecture and safety concept. The aspects presented by these architectures allows to define the high level design and take into account effectively the main design constraints. It is true that with these viewpoints, the dynamical aspects of the system behavior are partly not taken into account, but these aspects have a greater impact rather on the detailed design at subsystem level. At the high level, the architecture is more affected by the design choices that are expressed by the functional, physical and safety architectures. Limiting our system model to these elements lead thus to a partial description of the system behavior but to a sufficiently enough detailed knowledge of the design to allow the analysis of the solution and the investigation of other candidates.

**Safety Levels Allocation Optimization**

For the optimization we considered at first only the SG5. The objective is to analyse the optimization approach based on ASIL allocations results and check if the proposed solution can be reached using this approach. The safety analysis is based on the architecture in Figure 4.1.
**Failure Model** The failure model is built as an annotation to the system physical architecture model as proposed in HIP-HOPS. The annotations are an expression of the failure propagation logic in the architecture. This logic is defined using “boolean” expression between the inputs and the outputs of the architecture blocks. With an approach similar to CFT, the set of these expressions once combined allows the synthesis of the failure logic at the system level.

The approach focuses on modeling the elements failures and the failure propagation and transformation between its inputs and outputs. The IFMEA proposed by the tool allows to describe the internal failures and how the combination with the failures observed on input flows could impact the output flows as illustrated in the Figure 4.4 for the ESCL for example.

![Figure 4.4 – IFMEA failure model for a sensor](image)

Based on the proposed architecture described in Figure 4.1, we added the failure logic of each of the elements. The safety analysis using the built in HIP-HOPS plugin allows to obtain the failure tree for the undesired events and the MCS. These results are exploited later to investigate the allocation scenarios. The next paragraph discusses how the allocation problem can be formalized and the results of the safety analysis used to obtain the allocation scenarios.

**ASIL Allocation problem definition** The ASIL allocation problem consists of finding the different allocation scenarios that can be applied considering the PHA results and the failure analysis results. There can be multiple solutions thanks to the decomposition patterns defined in the standard as discussed in section 2.4.4. A formalization approach of the problem proposed in [68, 69, 89] allows to describe and solve the problem using an ASIL algebra. The ASIL algebra transforms the allocation problem and the decomposition patterns into a system of linear equations. It attributes the ASILs integer values from 0 to 4 for QM to D levels. The ASIL...
decomposition patterns can consequently be described using the equation in 4.1. An allocation scenario is compliant if this equation is respected. The problem can, consequently, be described as a system of linear equations with the ASIL values as variables.

\[
\text{ASIL}_{\text{Initial}} = \sum \text{ASIL}_{\text{Decomposed}} \tag{4.1}
\]

As an example to illustrate the concept, let us consider an example where two redundant elements are considered for ASIL decomposition. We assume that the considered safety goal is an ASIL D. The allocation scenarios in this case the solutions to the equations \(X_1 + X_2 = 4\) where \(X_1\) and \(X_2\) are the integer values of the ASIL allocated to the redundant elements. There are in this case 5 allocation scenarios.

As can be seen in this example, in order to find the allocation scenarios, we need first to identify the redundant elements. We discussed earlier in section 2.4.4 the conditions for applying the decomposition. The elements can be considered redundant and a decomposition can be considered if only the joint failures of these elements lead to the safety goal violation. This condition can be verified by analysing the safety analysis results, particularly the minimal cut sets of the undesired events. The elements present in the same MCS can be considered redundant and the decomposition can be applied if the independancy can be proved later between these elements. The obtained system of equations obtained from the transformation of the mcs to linear equations, as illustrated before, is solved using different techniques that we reviewed in Section 3.2.

The system of equations have often multiple solutions. In order to choose among them, HIP-HOPS propose the use of a cost model allocating weights to the safety levels. These cost models can be linear, logarithmic or exponential.

**ASIL allocation results** The safety analysis of the SG5 led to the fault tree illustrated in the Figure 4.5, the result is in accordance with the FT manually constructed in the proposed solution in the Figure 4.3.

The allocation scenarios based on PEPS physical architecture are illustrated in the figure.

The safety levels allocation problem depends on the safety analysis level: functional or physical. It depends also on the used cost model. The cost model based on the safety levels only can not be accurate. The cost models supposes that all the elements have the same cost for a given ASIL. It is not the case, for example a sensor and a micro-controller allocated an ASIL B have not the same cost. A more accurate cost models that would allow an effective comparison is, thus, needed. An alternative could be also to approach the impact of the ASIL on cost indirectly.
Table 4.4 – Allocation Scenarios

<table>
<thead>
<tr>
<th>ASIL scenario</th>
<th>Element 1</th>
<th>Element 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0 [QM]</td>
<td>4 [D]</td>
</tr>
<tr>
<td>2</td>
<td>4 [D]</td>
<td>0 [QM]</td>
</tr>
<tr>
<td>3</td>
<td>2 [B]</td>
<td>2 [B]</td>
</tr>
<tr>
<td>4</td>
<td>3 [C]</td>
<td>1 [A]</td>
</tr>
<tr>
<td>5</td>
<td>1 [A]</td>
<td>3 [C]</td>
</tr>
</tbody>
</table>

The applied method here optimize the ASIL allocation based on the considered physical architecture. The redundant elements in the architecture are considered for ASIL decomposition. In this logic, if no redundant elements are present in the architecture, we can consider architecture transformation techniques based on redundancy introduction. The redundancies introduction can be guided by the safety levels allocation.

**Architecture Transformation Optimization**

We limited the model to the steering management functionalities linked to the safety levels allocation optimization model considered in the previous paragraph.

**Functional architecture** Following the architecture design process described in the second chapter, the system functionalities are decomposed from the abstract level to a refined level where the basic functions are identified. The refinement level is at a level where the functions can be implementable by known or conceivable physical solutions. The functional architecture model below allows to identify the functional strategies implemented by the system. The solution presented is generic and implementation independent. The functional decomposition of each of the sys-
4.3. STATE OF THE ART: OPTIMIZATION APPROACHES APPLICABILITY

System functionalities are modeled at the functional level as illustrated in the figure. We present here only the high level model of the steering lock management.

![Figure 4.6 – Abstract Functional architecture](image)

Physical architecture: We modeled the physical architecture that is going to be used for the optimization. The architecture include only the main ECUs concerned by the SG5. We limit for the moment the scope of the optimization to this safety goal as a first step to analyzing the benefits of the approach.

The redundancy introduced in the proposed solution is removed in order to check other alternatives. We will check if the proposed solution can be reached by the optimization process at some point. The resulting physical architecture to be optimized is illustrated in the Figure 4.7. The first step is to identify the elements that can be included in the optimization. We consider the ABS, PEPS and ESCL ECUs. These ECUs implement the main functionalities involved in the ESCL management and consequently implement the safety requirements resulting from SG5.

![Figure 4.7 – Physical Architecture Optimization Startpoint](image)

The optimization approach is based on the alternative implementation of the architecture elements. The implementation concept supposes that an element can be replaced in the architecture with a set of elements using one or multiple strategies. This can be considered as a design pattern implementation. For example, a single component functionality can be implemented by redundant components with a voting or standby strategies. During the optimization process, the elements included in the optimization can be replaced by a different implementation depending on the optimization objective: cost or safety levels. To illustrate the concept, we present in the Figure 4.8 the implementations considered for the PEPS ECU.

Optimization Results: The optimization through the redundancies introduction could be applied at different levels. The model abstraction level determines the
redundancy level. It is clear that applying the redundancies at Vehicle/ECU level is not interesting. Redunding an ECU is a very expensive solution and is rarely considered in the automotive systems. At a more detailed level, the redundancies can be applied effectively on the components of the ECU. But, this strategy does not allow reaching solutions implementing safety mechanisms at software level or complex safety strategies automatically.

The fact that the implementation requires that the output and input of the elements remain unchanged limits the solutions that can be reached by the automatic process. It requires that the exchanged flows in the architecture are fixed and can not be challenged. With such approach, the proposed solution by the engineers could not be reached starting from the initial architecture provided to the tool independently of the abstraction level we considered. The approach is interesting when it comes to design patterns application in the design process. But, in designs where the physical flows need to be considered in the optimization process it falls short. In the study case, the process didn’t reach the solution proposed by the engineers as it failed to consider the redundancy of the vehicle speed flows because of the implementations concept limits.

4.4 State of the art limits and gap analysis

The application of the selected state of the art approaches for the modeling and optimization of safety critical embedded systems allowed us to identify the limits of these approaches regarding our objective and the research questions we fixed previously.

— System Model
The reviewed system modeling frameworks and languages prove the continuous interest of such tools. The EAST-ADL language is of a special interest for the automotive system engineering field. The adaptation of the SysML concepts to the automotive systems and the support of the ISO 26262 concepts are the main benefits strong points of this language. But although, considering the variety of SysML modeling tools used and the current prototype status of the proposed framework, we judged more interesting to propose an approach that is modeling language independent. The objective is to facilitate afterwards its integration in different contexts. We discuss an example of the means of integration and its implementation in further details in the chapter 6.

— Failure Model
A failure model based on the annotation approach as proposed in HIP-HOPS proved
4.5. CONCLUSION

to be more adapted to the system level failure analysis. The failure propagation and transformation based approach allows an easy to implement and enoughly data rich process. The qualitative safety analysis results can be further exploited for an optimization approach taking into account safety constraints.

— Architecture Optimization

The optimization approaches based on the redundancy introduction are discussed and used widely in literature. The optimization processes based on this technique, especially if applied on the physical architecture level, can lead to missing a subset of potential solutions. The redundancy at physical level does not always correspond to the redundancies that can be implemented in the functional level. These redundancies can be implemented with multiple physical architectures depending on constraints with a considerable impact such as safety. A more adapted approach taking this parameter into account should be more adapted for embedded systems architecture optimization.

The safety levels allocation optimization based approaches are also very interesting and can improve the architecture design. Dealing with the problem independently from the architecture using only cost models make the results questionable. It is needed to adapt the technique and couple it with the architecture and the design process for more pertinent results.

EAST-ADL, the modeling language we considered the best fitted for the study case, does not support all the functionalities that are proposed by HIP-HOPS. For example, the optimization feature could not be used in the framework. We were forced to remodel the system in Simulink in order to evaluate this feature. This was an additional reason for considering a modeling language independent approach.

— Cost Models

The current cost models used in the discussed optimization approaches are useful to give an empiric values facilitating the comparison of the solutions. In the same time, they can hardly be used to make a reliable estimation of the cost of the solution for the design choices to be based on.

4.5 Conclusion

In this chapter, we applied two techniques often used in the optimization of the system architecture on a study case. The ASIL allocation optimization and the redundancy introduction can be applied on proposed physical architectures to reduce the impact of the safety constraints. This leads to optimizing this solution to meet the safety constraints and reduce the cost. Its results remain though dependent of the initial proposed solution. These techniques does not allow to challenge this solution by proposing different alternative architectures. This limits the scope of the assistance it can provide the designer to reach an optimal design.

We, thus, propose in the next chapter an approach that aims to overcome these limits. It helps reaching optimal design using an architecture synthesis process based on the functional architecture. It does not, thus, require an initial solution and can investigate different architecture concepts. While the architecture synthesis approaches exist and are widely used in the system-on-chip design, our work will differ by focusing on readapting the architecture synthesis approaches to the auto-
motive system design particularities. The ISO 26262 concepts are a key part of the synthesis approach.
Safety Driven Automotive Systems Optimization Approach

5.1 Introduction

The main objective of our research is to help the designer during the design phase and to guide the design process to reach the best fitted solution. In this chapter, we present our approach for optimizing the system cost by taking into account key engineering constraints. We start by presenting an overview of the approach and its process. We identify the main steps of the process along with the respective assumptions, inputs and the required input data for each. For each process step, we specify the posed problem and the algorithm allowing to solve it based on the available input data.

5.2 Approach Overview

As discussed through the previous chapter, we assume that the best approach to address the architecture optimization in the scope of our study is an architecture synthesis approach. By adapting the architecture synthesis approaches to the automotive context and adding the safety constraints from ISO 26262 point of view, an architecture synthesis approach can effectively address the architecture design optimization problem.

Based on [32], an architecture synthesis approach “must clearly specify:
— (a) the syntax and semantics of the input and output descriptions,
— (b) the set of algorithms for translating input into output descriptions,
— (c) the set of components to be used in the design implementation,
— (d) the definition and ranges of design constraints,
— (e) the mechanism for selection of design styles, architectures, topologies and components, and
— (f) control strategies (usually called scenarios or scripts) that define synthesis
tasks and the order in which they are executed.”

The approach proposed here follows this point of view. Considering the embedded
systems properties, we start by discussing the architecture choices that will be op-
timized by the approach. We set also the constraints that guide these choices. The
constraints are the translation of the problem scope and perimeter as defined at the
beginning of this thesis. Based on these elements, we define the synthesis process
steps (tasks) and the processing flow allowing reaching candidate physical solutions
starting from the functional architecture level. We specify also the algorithms al-
lowing the architecture synthesis, the constraints verification, the cost estimation
and ensuring the exploitation of the input models data through the process.

5.2.1 Key architecture definition choices and constraints

Optimization is “an act, process, or methodology of making something (as a de-
sign, system, or decision) as fully perfect, functional, or effective as possible.” With
different alternatives at hand, it is the process of selecting the most fitted solution,
with regard to an objective function and considering a number of constraints, from
the set of possible alternatives. The raw definition of the term optimization en-
courage us to think first about the elements that could define an alternative and to
which we can consider alternatives. Second, we need to formalize the constraints
that limits the set of possible alternatives. Third, it is essential to define the criteria
allowing to choose between the alternatives.

Key architecture choices

As we have discussed in chapter 2, the architecture is conceived at different levels.
At each level, the designer make motivated architectural choices that defines further
the final design and orient the following level design activities.

At the functional architecture level, the system scope and the main functions are
defined. The functional architecture is key step. It sets the main scope of the system
and propose a solution, reused or conceived, that is independent of the physical
architecture. Considering its importance in the design, this step can be included in
the optimization process. But, considering the need for a complex logical reflexion
it can not be automated. It can only be guided by design patterns to help adopt the
best architecture structure (pattern). In our approach, we consider the functional
architecture as an input. It is an key input that will be based on a manual process
not included in our optimization process.

The preliminary physical design is also a key step in the design. At this level,
the physical architecture is conceived based on the functional architecture. The
two architectures are linked by a mapping logic. The functions are mapped to the
components of the architecture. The flows are allocated to the interfaces. The com-
ponents choices and the mapping of the functions to these components are tasks,
often, iterated until the functional and nonfunctional requirements are met. As-
suming that the input data are available, an algorithm can automate the iterations
and propose potential implementations for a given functional architecture. Conse-
quently, we decided to include it in our optimization process.
Based on the preliminary architecture and the functional architecture, the safety requirements and the ASIL levels can be inherited by the elements allowing to define the functional safety concept. This step has an important impact on the architecture cost and must be taken into account. Also, since the safety analysis at the functional and physical level can provide the required inputs for the automation of this process, we included it in the process.

To summarize, the key architecture choices that need to be taken into account in the architecture synthesis process are:

— Physical architecture components
— Functions mapping to the components
— Flows mapping to interfaces
— ASIL allocation
— ASIL decomposition choices

These architecture choices are, though, guided by multiple constraints. We discuss these constraints in the following paragraph.

Constraints

For an architecture synthesis approach, it important to identify the design constraints taken into account. The constraints are the properties allowing identifying the candidate solutions while exploring the possible combinations. We consider, in our approach, the following constraints. They can be arranged in two main categories:

— Safety

As explained in the first chapter, the safety constraints have a deep impact on the design and the cost. The main safety constraints are related to the ASILs. The way the ASILs are inherited from the safety goals puts constraints on the components choice. We can identify three main constraints related to the ASIL levels that should be considered during the functions mapping to the system elements: The element safety level capacity, the independence requirements and the mixed criticality cohabitation constraints.

Firstly, the component should be able to reach the ASIL level inherited. The failures modes, the reliability and the applicable safety mechanisms of the components can determine its capacity to reach a certain safety level and with which cost. This parameter need to be taken into account to avoid later changes to the architectures that can be critical and costly.

Secondly, if an ASIL decomposition is made, Independence requirements must be met. This adds constraints to the components choice to comply such as the absence of common cause failures and the absence of interference.

Thirdly, mixed criticality functionalities allocated to the same component should be efficiently separated to ensure the absence of interference. This may require, sometimes, special features in the component. For a more efficient and compliant allocations, this constraint must be taken into account.

— Hardware

The choice of components is the result of a combination of constraints. A component is generally added to the architecture for its capacity to fulfill a given function. With numerous possible components capable each to fulfill
the functions, additional constraints come to guide the choice. As mentioned earlier, the safety constraints can impact this choice. While the non-safety related systems do not have such constraints, other constraints apply for both safety and non-safety related systems. We can mention for example the components interfaces and their performance. To ensure the exchange of data between components or/and ECUs, the components need to have the required interfaces and capabilities. Concerning the performance, the main constraints are the performance limits such as CPU load and the memory usage. These performance metrics cannot be estimated precisely at high abstraction level. The constraints can, though, be translated and estimated in alternative quantitative forms. The choice of components should take both constraints into account. This ensures the feasibility of the solution and reduces the cost.

Optimization criteria

The objective of the optimization is to reach the system design that reduce the cost of the architecture and respects the constraints mentioned above. The objective is a compliant design, thus, a solution need to validate all the constraints. The cost of an architecture can be approached in different ways. Here, we are not dealing with the cost as a general concept in which all the product lifecycle processes (such as production, logistics, time to market etc ...) are taken into account. We focus mainly on the development cost and the product materials cost. Estimating the cost of a solution is a difficult task and has been approached in a variety of ways. The need to estimate early and with few details led to the use of predictive cost models. The models in state of the art are discussed in further details later. We discuss the pertinence of the usage of these cost models and how they can be applied or adapted in the proposed approach.

5.2.2 Architecture Synthesis Flow

The architecture synthesis flow in the proposed approach is based on the design process described in 2.3. The difference though is that when manually done, only few alternatives are checked. The retained solution is often based on the designer expertise and predictions. The approach, as shown in Figure, covers a part of the design process that is often manually done allowing the exploration of different alternatives and eventually reaching an optimal design. It consists of five main steps: Functional architecture, ASIL allocation and Safety constraints extraction, architecture synthesis and mapping and cost estimation.

The approach fill the gap identified during the state-of-the-art review and the results from the study case example. It makes a link between the functional and physical architectures. It also takes into account the safety constraints from the ISO 26262 point of view.

The first step is at functional level. Based on the functional architecture, we proceed to the safety analysis. The result of the analysis allows identifying the different alternatives in which safety levels can be allocated to the safety related functions. Next, we synthesize an architecture and allocate the functions to its components.
The synthesized architecture is saved as a potential solution if it does not violate any of the constraints. These steps are discussed in further details in the next sections.

5.3 Functional Design

The first step in the synthesis process is the functional design. It includes the decomposition of the system functionalities and the variability in the system scope. These design choices are described in the functional architecture as discussed in the chapter 1.

5.3.1 Functional architecture and variability

The functional architecture describe a solution implementing the strategies and basic functions allowing the system to interact with its environment and ensure the intended functionalities. This solution can be detailed but remains independent from the implementation. It consist of input for the implementation in the synthesis process.

The designed system is expected to respond to the needs of different customers. It needs to be customizable to each customer and meets the cost target. Thus, a one-size-fits-all functional design may not be the best strategy. The customers needs and the cost targets are often disparate. The design should take into account the variability in the functional architecture leading to different versions of the system adapted to different configurations.

The variability identifies the possible versions of the system and the potential customers it can target. Such strategic decision is made early and is better described in the functional architecture. At this level, the variability can be described as optional functions. The functions described in the functional architecture could be put into two categories:

— Basic: functions that defines the main functionalities of the system and that are implemented and present in all the system “versions”

— Optional: functions that defines additional functionalities that are intended for a specific customer. They are not present in all the “versions” but can be implemented if required.

The variability can have an impact on physical architecture. The optional functions can lead to introducing new components or integrating additional software modules. An efficient design should take into account the variability. It should offer the
possibility to answer the customer specific needs in a cost optimal approach. We take into account the variability in the system design to reach a “standard platform” that allows implementing the functional architecture with its variable features in the most efficient way.

To illustrate the concept, let us reconsider the study case in the previous chapter. The PEPS system, as described, ensure amongst others the ESCL management functionalities. In some versions implemented in automatic transmission vehicles, this functionality can be removed as a similar one is offered by the powertrain system. The simple removal, eliminating architecture elements or software components, can have considerable impact considering the corresponding ASIL level (ASIL D). If this variability in the design is not taken into account early at the functional level, the obtained solution may become unadaptable or less cost competitive when the ESCL management functionality is removed.

5.3.2 Assumptions

The functional architecture can be approached in different ways. The system engineering methodologies provides guidance to efficiently develop and model it. In order to be able carry out the next steps of the proposed approach, we expect the next assumptions to be verified by the functional architecture:

— **Refinement level**
  The functional architecture can be decomposed at different levels. The more complex is the system, the more level are needed until the basic functions are identified. Functions decomposition can go deep and become time consuming. The key to remain at a sufficiently detailed level is to fix the refinement level to be targeted. For our approach, we consider the granularity level where the functions can be allocated to one component (host) of the physical architecture. The same rule applies to the flows. This assumption allows avoiding the scenarios where functions are allocated to multiple components. Allocating to multiple components can have an impact \( \sum \text{ASIL}_i = \text{ASIL}_{SR} \) on the safety level allocation and add complications to the process of the architecture synthesis.

— **Functions Outputs**
  The functions are assumed to have one output flow. Decomposing the functions to the refinement level fixed in the previous paragraph may not be enough. The functions at this level can still have multiple output flows. Multiple output flows can lead to inheriting multiple safety levels and independence requirements which increases the complexity of the safety requirements verification. In order to ensure the correct ASIL allocation logic and the ability to deal with the safety requirements, we opted to separate the flows and require a refinement level where the functions have a single output flow type. The output flow can though be duplicated towards the inputs of multiple functions.

— **Variability**
  We assume that the functional architecture identifies the optional and basic functions at a sufficiently decomposition level. The basic and optional
functions, if available, are supposed identified prior to moving forward to the next steps. The refinement level should comply to the assumptions aforementioned.

### 5.3.3 Model

Considering the definition of the functional architecture as stated earlier, describing the functional architecture requires the ability to represent the functions, their inputs and outputs and the flows. These describe the functional decomposition of the system and the dependencies between the functions.

As discussed in the chapter 3, different modeling languages support the functional architecture modeling from this perspective. We can cite for example block diagrams, Simulink and SysML. Each of these modeling languages is designed for specific usage. Simulink are used for functional and behavior simulation. SysML is designed with system engineering activities in mind.

For an architecture synthesis approach, we are interested in the functional decomposition and the dependency between the functions. These informations can be modeled in the majority of the modeling languages reviewed in the state-of-the-art. Our objective to provide an approach adaptable and generic enough to be used in different settings lead us to choose a modeling formalism that is modeling language independent. Using a specific modeling language may limits its applicability since project teams often use different modeling tools and languages. Thus, we proposed to use Data Flow Graphs for the functional architecture modeling. Data Flow Graphs allows modeling the functional architecture structure and the dependencies between the functions. A key advantage, it can be extracted from a variety of the modeling languages structures. It has also been widely used in the literature in architecture synthesis approaches [16, 32, 35, 74, 96].

Data Flow Graph is a graph defined as $G (V, A)$ where $V = (v_1, v_2, \ldots, v_n)$ is the set of nodes and $A = (a_1, a_2, \ldots, a_n)$ is the set of directed arcs or edges $a_i = (v_j, v_k)$ where $v_j$ is the origin node and $v_k$ is the destination node [53]. The nodes represent the operations or functions and the arcs represent the communication channels i.e. the data which is exchanged between the nodes. The DFG can be used not only for representation but also for simulation. Three execution rules apply to the DFG:

- The nodes are executed only when the inputs data are available on its ports.
- The execution consumes the data in the function inputs.
- Multiple functions can be fired simultaneously.

Here, we will not be interested in the simulation and verification capabilities of the DFG. It is used as an input model for the architecture synthesis and an intermediate formalism allowing to extract the required information from modeling languages such as SIMULINK and SysML. The data extraction from the simulink models, for example, will be discussed further in the chapter 6.

### 5.4 Functional Safety Concept

As discussed in the Chapter 2 and 3, ASIL levels can be inherited in different ways by the system elements thanks to the ASIL decomposition patterns proposed
CHAPTER 5. SAFETY DRIVEN AUTOMOTIVE SYSTEMS OPTIMIZATION APPROACH

by the ISO 26262. We also identified the ASIL allocation and decomposition as an optimization technique for system architecture optimization. Based on the results of the Chapter 4, we concluded that the ASIL allocation problem need to be addressed at the functional level and integrated in an architecture synthesis process for more effective optimization results.

5.4.1 ASIL allocation challenge

The ASIL allocation problem consists of finding the compliant possible scenarios allowing the elements of the architecture to inherit a safety level in respect to the safety goals and the safety analysis results. Multiple solutions can be considered. We aim to find all the possible scenarios for the ASIL allocation at the functional level. We adopt the ASIL logic presented earlier allowing the formalization of the Allocation problem.

In the previous works on this subject, the ASIL allocation problem is dealt with at the physical architecture level. The architecture elements are allowed to inherit multiple ASIL levels from the safety goals. The element will afterwards be allocated the highest ASIL inherited. This leads often to solutions where some parts of the system are allocated higher ASIL than required. Although some proposed allocation optimization approaches help to reduce this. It requires a complete design that is, in this level, still unavailable.

We propose, thus, to solve the allocation problem at the functional level since it has been additionally discussed that the ASIL decomposition is more effective at the functional level [61]. The ASIL allocations are based on the safety analysis. We assume that the safety analysis results are available before this step. We assume that the undesired events considered are specified and are allocated ASILs. We assume that the safety analysis allowed to obtain the MCS of the safety goals.

5.4.2 ASIL allocation scenarios

Using the the ASIL algebra allowing to formalize the ASIL decomposition and patterns discussed in 4.1 and based on the approach proposed in [68], the allocation problem can be described using linear equations.

For every MCS leading to violation of a safety requirement, the functions $F_i$ in the architecture verifies the following equation where the coefficient $a_i$ is null if the corresponding function failure is not in the MCS and equal to 1 otherwise:

$$\sum a_i \times ASIL_{F_i} = ASIL_{SR}$$  \hspace{1cm} (5.1)

Applied to all the MCS for all the safety requirements ($SR_i$), the allocation problem could be interpreted as a system of linear equations. In a matrix form, a possible allocation should be solution to the equation 5.2:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \times \begin{bmatrix} ASIL_{F1} \\ \vdots \\ ASIL_{Fn} \end{bmatrix} = \begin{bmatrix} ASIL_{SR1} \\ \vdots \\ ASIL_{SRm} \end{bmatrix}$$  \hspace{1cm} (5.2)
The obtained system of equations often have multiple solutions. The solutions are the ASIL allocation scenarios that could be applied to the functional architecture.

Transforming the ASIL allocation problem into a matrix equation allows to simplify solving it. The obtained equation 5.2 can be solved using different techniques and algorithms. It allows also to take into account ASIL preferences for a certain function or to avoid decomposition over some functions. These preferences are easily taken into account without adding complexity to the problem by adapting the equation 5.2. To fix a variable at the preferred ASIL, we withdraw its corresponding column from the system after extracting its value from the left part of the equation. In order to avoid decomposing over two functions, we merge their corresponding columns using logic 'OR’ operation.

The solutions to the equation 5.2 are often multiple since many ASIL allocation scenarios can be compliant. We aim to retrieve all the possible scenarios, thus, we used a classical approach using the Row Reduced Echelon Form (RREF) of the equation. The RREF is generally computed using Gauss-Jordan elimination. It allows to identify the basic and the free variables which corresponds to the columns with no leading entry. In order to find the solutions, we proceed into allocating to these variables a value in the the range of ASILs numerical values, {0,...,4}, and deduce the rest of the variables accordingly. The echelon form could also be used to test the solvability of the system. If equations of the form 0 = Cst, where Cst is non null, exist, we may deduce that no possible allocation can be found.

The algorithm used to solve the equation and find the allocation scenarios is described in Algorithm 5.2. The RREF and solve functions allows respectively, to calculate the row reduced echelon form and to solve the system. The Fix-value function, on the other hand, allows to fix the value of the variables in the systems. It consists of extracting a new system from the original one by eliminating the fixed variables. The merge-dependent function allows to merge the columns corresponding to dependent variables.
Illustrative example We consider a system with two safety requirements (SR1 and SR2) rated ASIL D and ASIL C respectively. The functional elements F1 ... F5 implement these safety requirements. The Fault Tree in Figure 5.3 describes how the loss of these functions could lead to the violation of the safety requirements. SR1 and SR2 can be decomposed over the element whose failure lead to the violation of the requirement. For example, SR1 can be decomposed over F2, F3 and F4. We apply the Algorithm 5.2 to find all the possible scenarios.
From this FT, three MCS lead to the violation of the SRs.
— (Loss of F1): ASIL D
— (Loss of F2, Loss of F3, Loss of F4): ASIL D
— (Loss of F3, Loss of F4, Loss of F5): ASIL C

We suppose that the functions F1 to F5 are sufficiently independent, as required to apply the decomposition. The ASILs allocated to these functions should verify then

\[
\begin{align*}
\text{ASIL}(F_1) &= 4 \\
\text{ASIL}(F_2) + \text{ASIL}(F_3) + \text{ASIL}(F_4) &= 4 \\
\text{ASIL}(F_2) + \text{ASIL}(F_3) + \text{ASIL}(F_4) &= 4(5) \text{ASIL}(F_3) + \text{ASIL}(F_4) + \text{ASIL}(F_5) &= 3
\end{align*}
\]

The possible allocations are thus solutions to the following equation:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1
\end{bmatrix} \times \begin{bmatrix}
\text{ASIL}(F_1) \\
\vdots \\
\text{ASIL}(F_5)
\end{bmatrix} = \begin{bmatrix} 4 \\ 4 \\ 3 \end{bmatrix}
\]

The next step is to reduce the matrix to its row echelon form. It shows that the variables \(x_4\) and \(x_5\), corresponding to the fourth and fifth column in the matrix, are the only free variables.

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 4 \\
0 & 1 & 0 & 0 & -1 \\
0 & 0 & 1 & 1 & 3
\end{bmatrix}
\]
In this case, we will have 25 iterations as these variables take the values from 0 to 4. We will limit here to the two first iteration where \((x_4 = 0, x_5 = 0)\) and \((x_4 = 0, x_5 = 1)\).

— 1st iteration : The system to solve becomes

\[
\begin{bmatrix}
1 & 0 & 0 & 4 \\
0 & 1 & 0 & 1 \\
0 & 0 & 1 & 3 \\
\end{bmatrix}
\]  

A first allocation could be than deduced where: \(\text{ASIL}(F_1)=4; \text{ASIL}(F_2)=1; \text{ASIL}(F_3)=3; \text{ASIL}(F_4)=0; \text{ASIL}(F_5)=0;\)

— 2nd iteration : The system to solve becomes

\[
\begin{bmatrix}
1 & 0 & 0 & 4 \\
0 & 1 & 0 & 2 \\
0 & 0 & 1 & 2 \\
\end{bmatrix}
\]  

A second allocation could be than deduced where: \(\text{ASIL}(F_1)=4; \text{ASIL}(F_2)=2; \text{ASIL}(F_3)=2; \text{ASIL}(F_4)=0; \text{ASIL}(F_5)=1;\)

Continuing the next iterations will allow us to find the rest of the possible solutions.

### 5.4.3 Independence Constraints

The decomposition choice leads to an independence requirement as required by the part 9 of the standard. As mentioned earlier, the ASIL decomposition is a special ASIL reduction approach. The decomposition results in reducing the ASIL allocated to elements and parallely adds new requirements. A correct ASIL decomposition requires the independence between the elements. The physical architecture needs to take into account these additional requirements. In the last paragraph, we presented how the allocation scenarios are calculated. In this paragraph, we focus on the resulting independence requirements of these scenarios and how to extract them automatically.

The ASIL allocation scenarios identified earlier can result from a decomposition of ASIL or not. In case ASIL decomposition is made, it is needed to detect it in order to correctly implement it. As required by the part 9 of the standard, the implementation of the functions with decomposed ASILs need to be implemented by sufficiently independent elements.

ISO 26262: “5.4.8 When using the decomposition schemes given in 5. This results in allocating 4.7, with the exception of the scheme in 5.4.7 a) 2), the following methods and processes shall be applied: a) confirmation measures (see ISO 26262-2:—, 6.4.6) shall be applied in compliance with the ASIL of the safety goal; b) independence of the elements after decomposition shall be justified;”

In order to provide all the necessary inputs for the synthesis step, we need to identify the functions to be implemented independently, the impacted flows and the failure modes. The first step in extracting the independence constraints is to detect the ASIL decomposition application. ASIL decomposition can be detected by comparing the ASIL allocated to the functions with the ASIL of the safety goal they are involved in. This can be based on the MCS used for ASIL allocations. For each mcs, if the asil of the functions is lower than the safety goal ASIL, we conclude...
that an ASIL decomposition is made and thus an independence requirement should be respected during the architecture synthesis. This allows to identify the functions to be concerned by an independence constraint. Based on the identified functions, the independence constraints can be defined.

An independence constraint is defined through:
- the concerned functions
- the impacted output flows
- the failure modes

**Illustrative example of ASIL decomposition and constraints extraction**

Let’s consider the second mcs of the illustrative example (Loss of F2, Loss of F3, Loss of F4): ASIL D. If the the allocation scenario is: ASIL(F1)=4; ASIL(F2)=2; ASIL(F3)=2; ASIL(F4)=0; ASIL(F5)=1; we can see that F2, F3 and F4 have safety levels that lower than the safety goal. This allows us to conclude that an ASIL decomposition approach is made. Consequently, the allocation of these functions must respect the independence constraint. The obtained constraint is described in 5.1.

<table>
<thead>
<tr>
<th>Functions</th>
<th>F2</th>
<th>F3</th>
<th>F4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows</td>
<td>Flow 2</td>
<td>Flow 3</td>
<td>Flow 4</td>
</tr>
<tr>
<td>Failure modes</td>
<td>Omission</td>
<td>Omission</td>
<td>Omission</td>
</tr>
</tbody>
</table>

Table 5.1 – Independence Constraint example

### 5.5 Architecture Synthesis

The functional architecture and the allocation scenarios identified in until now are technology independent. A solution needs though to propose a physical architecture with implementation choices. A physical architecture is an arrangement of physical elements. The architecture synthesis step described here allows reaching this arrangement based on the results of the previous steps. It consists of applying the ASIL allocations scenario, allocating the functions to system elements, allocating the flows to the buses, verifying the respect of the constraints and finally estimate the cost of the solution to compare it to the rest of the alternatives.

#### 5.5.1 Overview

Synthesizing a system architecture consists of finding a suitable physical architecture and the mapping of the functional architecture on its elements. The goal is to reach functions and flows mapping that respects the available resources and the safety constraints and, in the same time, optimizes the design cost. The mapping includes the spatial allocation of the functions and the flows in form of their deployment to hardware elements. It also includes the application of the implementation patterns such as the bus implementation pattern.
The problem can be described as follows: The joint HW resources choice and mapping problem consists of determining a suitable choice of the subsystem configuration and HW resources as well as the functions allocation. It consists of finding an architecture containing a set of subsystems $\{SS_1, SS_2, \ldots SS_n\}$ connected using a set of Buses $\{B_1 \ldots B_k\}$ where each subsystem $SS_i$ contains a set of HW resources (hosts) $\{H_1, H_2 \ldots H_m\}$ implementing a set of functions from the functional architecture. In the end, it comes to finding a mapping where:

- For each function $F$, the tuple $(ss, h)$ specifying to which subsystem and host the function is allocated, is determined.
- For each flow $FL$, the bus $B$ that will carry the flow is determined.
- The safety constraints due to the safety levels values are respected.

### 5.5.2 Assumptions

#### Physical Architecture

A physical architecture is often defined as “an arrangement of physical elements (system elements and physical interfaces) which provides the design solution for a product, service, or enterprise, and is intended to satisfy logical architecture elements and system requirements. It is implementable through technologies”. In the automotive system context, a physical architecture of an embedded system is a set of Subsystems or ECUs (Electronic Control Units) interconnected by physical interfaces.

It can thus be defined as: $PA = (SS, B)$ where $SS$ is the set of subsystems in the architecture and $B$ is the set of buses ensuring the flows exchange between the subsystems.

Each ECU contains also a set of interconnected elements that we can call components or hosts. It can be defined as : $SS = (H, HF)$ where $H$ is the set of hosts in the subsystem, $HF$ is the set of flows implemented by the subsystem to connect the host. The physical architecture is not approached here at the electronic design level of detail. It is rather at an abstract enough level allowing to define a potential solution that can be communicated and discussed by a multidisciplinary team. It is also detailed enough to verify the correctness, the feasibility and the expected cost of the solution.

#### Components/Hosts

The hosts can be Microcontrollers, sensors, transceivers etc . . . These components implement the basic functions needed to fulfill the system main functionalities. They provide also the interfaces implementing the flows. It allows them to communicate and exchange data with components or over a network.

In order to take into account the safety constraints, we assume that the failures modes are identified beforehand. The impact of these failure modes on the interfaces and the flows are also identified with the potential root causes. The max ASIL that can be reached is specified to ensure the asil allocation scenario can be reached.

In case of cohabitation of mixed criticality functions, special features allowing to
meet the requirements for cohabitation need to be present in the host. These features, if existing, are assumed known and can be used later for safety constraints checks. For the performance constraints, we assume that memory size and the functional size capacity of the hosts are known based on the component specifications. For the cost estimation, the cost of the components are known as well as the surface of these components for the PCB cost estimation.

A host is defined as tuple $H = (A, C, SF, CI, FM)$ where:
- $A$ is the highest ASIL that can be reached by the component,
- $C$ is the cost of the component,
- $SF$ is the separation features that could allow hosting mixed safety levels functions,
- $CI$ is the communication interfaces that could be implemented by the component,
- $FM$ is the set of failure modes of the host.

**Data Bus**

A bus is a communication system that transfers data between components inside a computer, or between computers. This expression covers all related hardware components (wire, optical fiber, etc.) and software, including communication protocols. The flows exchanged between the functions, defined in the functional architecture, are implemented between their hosts. These can vary depending on the interfaces present on the hosts as well as the ECUs to which they are allocated. We assume that the Data bus is implemented following the pattern defined by Figure 5.5 and inspired from the definition in ISO 11898 standard.
In each node of the network, the components illustrated by the figure are needed to be able to communicate. The illustrated pattern applies to CAN bus and LIN busses. It requires the existence of these elements but it does not impose them to be separate. Today, some micro-controllers have an integrated CAN controller and transceivers. But even in those cases, the patterns still applies.

We assume, in our synthesis approach, that the bus elements are implemented following the pattern presented above. In this case, we assume:

- For each node of a network at least a component shall fulfill each role.
- A micro controller is considered integrating the bus controller if it is has the corresponding interface.
- Multiple flows can be allocated to the same bus.
- A transceiver by node by network.

**Mapping**

Mapping is the task of allocating the functions and the flows identified during functional decomposition to the hardware resources. It consists, in our case, in the choice of the subsystem and the host for each function and the network or implementation for each flow. These choices allows to recognize the main elements of the architecture and the data network to be used or implemented.

The functions allocation to hardware resources assumes that:
— All the functions are allocated
— For each function at least a mapping is possible
— An allocation is unique

5.5.3 Architecture Synthesis process

5.5.4 Synthesis process

The synthesis process allows combining all the information about the functional architecture, the ASIL allocation scenarios, the design patterns and potential hosts to reach a candidate solution. It allows to browse the combinations and transform them into a preliminary architecture. The obtained architecture can, if the constraints are respected, be considered as a potential solution.

The process consist of the following main tasks:

— ASIL allocation scenario application: The process start by applying an ASIL allocation scenario. It allocates ASIL to each function and allows taking into account the corresponding independence constraints.

— Design space exploration: Based on the set of hosts for the functions and flows, mapping combination are browsed allowing to identify the solution main elements.

— Mapping combinations implementation: A mapping combinations identifies the elements of the architecture and the connections between them. This step takes into account the design patterns, such as for the buses, to bring together all the element and make a coherent architecture with them.

— Constraints verification: The obtained solution from the previous tasks are analyzed to ensure that the safety constraints are respected. If the solution satisfies all the constraints, it can be considered as a potential solution.

Design space exploration

The design space exploration consists of checking multiple combinations in an exhaustive or more efficient approach and analyze them. In our approach, this exploration is made at different levels. The first consists of the ASIL allocation scenario as multiple scenarios can be obtained from the ASIL allocation step. The second is due to the alternative hosts for each function. The third is the flow allocation possibilities that are correlated also with the functions allocation. The fourth is the implementation of the flows which consists of choosing the controllers and the routing of the flows for the bus choice. The last level is variability identified during the functional architecture decomposition and will be treated last in the optimization process.

Exhaustive Approach The exhaustive approach allows to check all the possible combinations. Checking these combinations can be interrupted at different levels and only few are analyzed thoroughly. The combination are discarded due to constraint violation or high cost. This allows to reduce the computation time and focus on the most efficient solutions. It is composed of two main algorithms.

The first algorithm is described in the figure 5.6.
Figure 5.6 – Exhaustive design space exploration
As can be seen the algorithm uses the assumptions we made on the inputs to explore the possible solutions. It starts by applying an ASIL scenario. This allows to allocate the ASIL levels to the functions and extract the constraints that should be respected for this allocations scenario. Based on the ASIL levels and the input about the possible hosts, some of these hosts are filtered if they do not meet the ASIL level. Once this done, it possible to navigate through the various possible combinations. These combinations are defined first by the hosts choices. Secondly, the allocations choices for the flows add extra possible solutions for a given host choice.

Each combinations is checked along the process for safety and cost constraint. If at some point the constraint are violated, the combinations is discarded. If a solution is maintained until the end of the process, this requires that the solution respects the safety constraints and has lower cost compared to similar solutions.

The checking of these constraints occurs at different levels in the algorithms. The first is after the host choice combination. It aims verifying the functions are allocated in respect with their ASIL level and the allocation scenarios constraints. It allows to detect the cases where mixed criticality elements cohabitation or independence requirements are violated. The second check is conducted after the flow allocation. At this level only the choice of the bus and the choice of the implementation (Controller choice) are checked. It allows to ensure that the ASIL level and the independence constraints are respected. The constraints verification algorithms are discussed in further details in the next section.

The second algorithm, presented by the figure 5.7, allows to deal with the variability discussed earlier in the functional architecture paragraph. This algorithm is executed after the first algorithms if variability has been introduced in the functional architecture. It consists of finding the best suited approaches to implement the extra functions. This implementation should respect the architecture established for the basic architecture. It could consist of adding additional component to the architecture or additional subsystems. The lower cost solution could be to use the existing elements of the basic architecture. In order to reach this, we propose an algorithm that minimizes the impact of the functions implementation on the basic architecture. It takes in addition as input, compared to the first algorithm, the candidate solutions for basic architecture retained by the first algorithm and the applied safety scenario.

In order to avoid an impact on the ASIL level of the elements of the basic architecture, we solve the allocation problem described earlier again. This time, we fix the values of the elements of the basic functions to avoid any changes. Once the scenarios identified, we use the same approach as for the synthesis of the basic architecture to allocate the flows and the functions. The safety constraints will be re-checked as the allocations scenarios for the optional functions can have an impact on the existing elements. Executing both algorithms leads to the cost optimal basic architectures and the best implementations of the optional features to get the full features architectures.
Figure 5.7 – Variability implementation algorithm
5.5. ARCHITECTURE SYNTHESIS

Genetic algorithm based approach  In the exhaustive approach presented earlier, we added branch and cut techniques to reduce the number of combination processed during the design space exploration. But, we know that the architecture synthesis problem is np-complete [32]. So, for large systems where the number of possible combinations can be very important, the exhaustive algorithm can become unpractically time consuming for iterative design process.

We have in reviewed in chapter 2, the heuristics algorithms that are often used in similar optimization problems. For our approach, we choose to use genetic algorithms. A genetic algorithm based design space exploration strategy can help to cope with the problem size. It is also adapted for a synthesis approach since no initial solution is required and the optimization targets can be brought to a single objective function.

Implementing the algorithm for an optimization problem requires modeling the solution as chromosome (encoding). The chromosome allows the expression of the different parameters that helps defining a potential solution. In our case, a candidate solution (combination) can be described with 2 sets of information. The first concerns the applied ASIL allocation scenario. The second concerns the functions mappings. A function mapping is defined using the subsystem and the element implementing the function.

To compare the combinations, a fitness function is needed to help rating the solutions. They can be compared based on their cost. The cost is thus implemented in the fitness function. In order to ensure that the optimization process lead to compliant solutions, we integrated in the fitness function a penalty correlated to the constraints. If the candidate solution violates a constraints, its fitness receives a penalty reducing its fitness score. By penalizing the candidate solution violating the constraints, we favor the individuals with compliant design.

1. Chromosomes:
The chromosomes define a solution as a set of choices that defines an individual of the population. The chromosome is composed by a set of genes. Each describes a part of the solution. In our case, the chromosomes are coded as described by figure 5.8. It allows describing the needed information to identify the allocated ASIL and the mapping of the elements of the functional architecture. The number of genes in the chromosome is, thus, defined by the number of functions to be allocated and mapped.

2. Genes:
As can be seen in 5.8, the chromosome has two types of genes. The first allows specifying the ASIL allocation through the choice of the scenario to be applied. The second type of genes allows determines the mapping of the corresponding functions. It defines the mapping with a couple of values determining the host and the subsystems to which it is allocated. The genes are coded as integers with values ranging in [0..Number of scenarios] for the first type and in [0.. size of the hosts set size] for the second.

3. Fitness Function:
The fitness function is an objective function that helps to identify how close a solution is to achieving the set goals. It is responsible for performing the evaluation of the solution. The fitness score reflects how optimal the solution
is. For our case, the key evaluation parameters are the cost and the safety constraints. The cost is considered as the score of the solution where it consists of the sum of the used elements costs. Lower costs correspond to higher fitness score. On the other hand, to take into account in this score the safety constraints, we add penalties representing the violation of these constraints. The penalties tend to lower the fitness score of the non-compliant solutions which favors the evolution towards respecting the constraints. We used for our implementation three penalties for ASIL allocation correctness, host choice and independence constraints respect.

![Composite gene](image)

**Figure 5.8 – Composite gene**

The genetic algorithm based approach reuses the implementation logic from the exhaustive approach. Unlike the exhaustive approach, if a combination implementation respecting all the constraints can not be reached it is not discarded. Instead, the best possible implementation that reduces the violated constraints is found. These constraints are taken into account in the fitness score.

In case of variability in the architecture, the same algorithm in 5.7 is used to implement the optional functions.

**Bus implementation algorithm**

In the assumptions section, we have fixed a pattern, based on the standard, for the implementation of the buses. We present here the algorithm used to ensure that the flows are implemented correctly. The implementation consists of identifying the needed elements to ensure that a flow
Figure 5.9 – Genetic algorithm based approach Fitness estimation
is correctly channeled from the origin element to the target. The origin element is the host of the origin function of the flow. Whereas, the target element is the host of the target function of the flow.

During the algorithm we make the distinction between three types of flows allocation. The three categories are resumed in the table 5.2.

<table>
<thead>
<tr>
<th>Type</th>
<th>criteria</th>
<th>allocation example</th>
<th>extra implementation constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>node to node (N2N)</td>
<td>the target and origin hosts are in different subsystems (nodes)</td>
<td>CAN, LIN, Flexray, etc ...</td>
<td>The pattern should be respected.</td>
</tr>
<tr>
<td>element to element (E2E)</td>
<td>the target and origin hosts are in the same subsystem but are different</td>
<td>SPI, UART, Hardline</td>
<td>The hosts should have the required interfaces.</td>
</tr>
<tr>
<td>function to function (F2F)</td>
<td>the target and origin functions are allocated to the same host</td>
<td>functional</td>
<td>none</td>
</tr>
</tbody>
</table>

Table 5.2 – three Flow implementation categories

Depending on their category, the flows are implemented differently and requires a different implementation process. Thus, the implementation algorithm starts by checking the flow allocation choice and organizing them to allow dealing with them each type apart.

For the F2F flows, no special implementation procedures need to be made at system level. The flows are exchanged through memory access and are implemented through hardware or software solution later in the development process.

The E2E flows allows the exchange of data between two elements of the architecture that happen to be in the same subsystem/ ECU. This consists often of hardline implementation or serial network such as SPI or UART. The condition is that the two elements have the interface in common. Since the flow possible hosts are defined based on the common interfaces between the original and target elements. The implementation does not, thus, require additional elements in between.

The N2N flows allows the exchange of data between elements from different subsystems. These subsystems are identified as nodes in the case of network such CAN or Flexray. For each node, the pattern identified earlier need to be respected. The missing elements to apply the pattern should be identified and integrated in the architecture. This must though be in respect with the safety constraints.

The algorithm to sort the flows in the categories is presented in the figure 5.10. Based on the functions allocations, it identifies the possible hosts for each flow. This allows to explore the possible combinations during the synthesis algorithm. Through the exploration algorithm described earlier, the F2F and E2E are implemented directly i.e no additional elements are to be added to the architecture as long as the required interfaces are present on the elements.

In case of N2N flows, a special process is followed to ensure that the required elements are integrated in the architecture. This consists mainly in the choice of
Figure 5.10 – Flow hosts identification algorithm
the controller for the bus in each node. We have made an assumption that at each node, one controller per bus type is allowed. Knowing that multiple bus could be implemented at a node and that an element can play the controller for multiple buses, the choice of the controller can have an impact on the architecture. The number of the controllers in the node impact the costs directly. In addition, safety constraints can impact the choice of the controllers. Independence constraints may require the implementation of the buses to be separate leading to have multiple controllers in the same node.

In order to take into account the assumptions made about the N2N flows implementations and the impact of the safety constraints, we implemented the following algorithm. It allows to identify the controllers for the buses in each node and ensure the compliance of the choices.

The algorithm starts by identifying the nodes and the networks in the architecture as well as the hosts of the flows origin functions. This helps to identify the first elements of the pattern. It also helps identifying potential controllers for the bus (origin components). These are filtered later based on the safety level and the independence constraints.

In order to ensure that the controller choice does not violate any safety constraints, we ensure that the asil level is met through the first step of filtering. In a second level, we check that controllers of the buses where independence is required are separate and meet the absence of common cause failures condition.
Figure 5.11 – Node controller choice algorithm
Constraints verification algorithm

In the previous paragraphs, we detailed the synthesis algorithms and the optimization steps. In these algorithms we often mentioned the safety constraints verification step. It consists of verifying that the mapping respects the ASIL level and the independence constraints if any.
The verification include the verification of the functions mapping and the flows mapping alike as both an lead to violation of the safety constraints.
During the mapping, the function are allocated to hosts. The main focus, at this level, is the ability of the host to implement the function and its capabilities to communicate with the rest of the elements in the architecture. But these are not the only mapping constraint to be taken into account. As specified from the start; the safety constraints are key constraints.
The safety constraint can be put in three categories:
— the capacity of the host to reach an ASIL level,
— the capacity of the host to ensure the cohabitation of mixed criticality level functions,
— the respect of the independence constraints.
As explained in the synthesis algorithm, the flows are mapped to architecture elements allowing the exchange of the data. This mapping can, in the same way as with the functions, lead to the violation of safety constraints. The checks performed in the synthesis algorithms allows to verify if the flows mapping respects these constraints. The constraints needed to be checked concern the same elements as the checks for the functions mapping. It consists of the respect of the ASIL level, the cohabitation requirements and the independence requirements.

Safety level constraints check  We assumed previously that an element is considered as a possible host for a function if it is capable to ensure the concerned function. We assumed also that each host have an attribute for the max safety level it can reach. It is essential to check that the functions allocated to the hosts do not exceed the host capacity. The same applies to the flows mapped to the bus. To ensure that the host choice is convenient we check if the ASIL levels of the functions mapped to a host verify: \( \text{ASIL}(\text{function}) < \text{maxASIL}(\text{host}) \)
The algorithms, below, allow to check if the condition above is respected in the candidate architecture for both functions and flows. If it violated, the combination is discarded since it is not compliant.

Mixed Safety level constraint check  The mapping can lead to an architecture where functions with different safety levels are allocated to the same element. If these functions should be kept at these safety levels, the standard require that freedom from interference between these functions must be proven. Otherwise, they should be brought to the highest level. Rising the safety level of the functions because of the mapping is of no interest as it is covered by others allocation scenarios. Thus, when it is needed, we try to prove the freedom from interference to keep the ASIL allocations at their levels.
The freedom from interference is the “absence of cascading failures between two or
more elements that could lead to the violation of a safety requirement”. A failure is considering as cascading if the “failure of an element of an item causing other elements of the same item to fail”.

The cascading failures between the elements of the architectures (components) are taken into account through the failure propagation analysis made at functional level. As for the functions allocated to the same element, we consider the integrated features that guarantee the non interference between the functions. This is may be the case for example for the software elements. The micro-controller in this case will need often special features allowing memory management to ensure the non interference between the modules. To ensure that the functions mapping, their safety levels and the host features, we added a checking algorithms for the mixed criticality functions cohabitation.

The check consists of checking if functions with different ASILs are allocated to the same host. The cohabitation between these functions is tolerated if the host is considered able to ensure the requirements for such a cohabitation. If the feature is not present, that would lead potentially to the violation of the non interference requirement. The algorithm, described in the figure below, detects this scenario and allows to avoid it during the design space exploration.

**Independence constraint Check** In the standard applying the ASIL decomposition requires the functions to be implemented by independent elements. Independence is defined by the standard as the “absence of dependent failure between two or more elements that could lead to the violation of a safety requirement; or
Figure 5.13 – Mixed Safety level constraint check algorithm
organizational separation of the parties performing an action".
The dependent failures are, also in the standard definition, “failures whose probability of simultaneous or successive occurrence cannot be expressed as the simple product of the unconditional probabilities of each of them”. This can be translated to the existence of common cause failures where a root cause can lead to simultaneous failures of the elements.

This means that the elements are considered independent if no common cause failures leading to the failure of the elements and the violation of the safety requirement exist.

As discussed in [59], for the independence, we consider only the external systematic causes. Potential causes to be considered are among others: mechanical aggression, temperature, humidity, vibration, electromagnetic field etc ...

In our approach, the independence constraints check aims at checking if the mapping respects the absence of common cause failures requirement. Based on the assumption that the failures modes and their impact on the interfaces are available for the hosts, we are able to identify the causes leading to the manifestation of the failures modes at functional level we are interested in. These are the ones described in the constraints. Crossing the information about the flows mapping to an interface of the host, the host’s failure modes and their causes allows to identify the causes that may lead to the violation of the independence requirement.

The check is conducted on the set of the scenario constraints. Detecting a violation of an independence constraint leads to discarding the combination as the solution is not compliant.

In order to illustrate the algorithm we propose an illustrative example. We use the example of constraint used in.

<table>
<thead>
<tr>
<th>Functions</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flows</td>
<td>Flow 2</td>
<td>Flow 3</td>
</tr>
<tr>
<td>Failure modes</td>
<td>Omission</td>
<td>Omission</td>
</tr>
</tbody>
</table>

Table 5.3 – Illustrative example : Safety constraints

For the example we suppose that $F_2$ and $F_3$ were mapped to the hosts $H_1$ and $H_2$ and that the output flow are allocated respectively to Interface 1. The table describes the failure modes of the hosts $H_1$ and $H_2$.

<table>
<thead>
<tr>
<th>$FM$</th>
<th>Impacted Interface</th>
<th>Impact on interface</th>
<th>Causes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FM_1$</td>
<td>Interface 1</td>
<td>Omission-output flow</td>
<td>$C_1, C_2$</td>
</tr>
<tr>
<td>$FM_2$</td>
<td>Interface 2</td>
<td>Commission-output flow</td>
<td>$C_3$</td>
</tr>
</tbody>
</table>

Table 5.4 – Illustrative example : Failure mode of host H1

As we can see from the tables, the interface to which the function output is allocated is impacted by a failure mode in each host. It is impacted by the omission of the output flow. The same failure at functional level we are interested in the constraint. This means that the causes leading to the failure mode $FM_1$ in $H_1$ and $H_2$
will lead to an omission of the function output flow mapped to these hosts. Since $C_1$ is a common cause, the independence constraint is violated. $C_1$ lead simultaneously to the failures $FM_1$ at $H_1$ and $H_2$ which are synonyms of the omission of the flows Flow 2 and Flow 3.

For the independence constraints check, the algorithm differs slightly. In the assumptions, the interfaces are considered for the bus. These are limited to the functions hosts only. The failures modes of the bus have a direct impact on the flows. Identifying the common causes consists of comparing the causes leading to the failure mode as the failure mode in the constraint.

<table>
<thead>
<tr>
<th>$FM$</th>
<th>Impacted Interface</th>
<th>Impact on interface</th>
<th>Causes</th>
</tr>
</thead>
<tbody>
<tr>
<td>$FM_1$</td>
<td>Interface 1</td>
<td>Omission-output flow</td>
<td>$C_1, C_4$</td>
</tr>
<tr>
<td>$FM_2$</td>
<td>Interface 2</td>
<td>Commission-output flow</td>
<td>$C_5$</td>
</tr>
</tbody>
</table>

Table 5.5 – Illustrative example: Failure mode of host H2

Figure 5.14 – Independence constraints check algorithm
5.6 Cost estimation

The synthesis process leads to multiple solutions. These are not all worth deep investigation by the engineers. It is needed to adopt an approach allowing the comparison of the solutions in order to identify the best fitted. The cost estimation is often used as a key parameter for decision making. It is, though, estimated and approached in different ways during the project lifecycle. We are interested here in the concept and design phase. We decided to focus on the architecture cost as an attribute for selection.

5.6.1 Cost Estimation Parameters

The cost of a design is the result of multiple factors. Some of these factors can not be calculated or estimated precisely. These factors are out of the scope of development and consequently out of the scope of our study. The cost of the elements used in the architecture, on the other hand, can be estimated based on the inputs and the synthesized architecture. The cost of the architecture is linked to the cost of the components of the architecture (sensors, actuators, micro-controllers ...) and the development effort.

The architecture cost

As we have seen through the approach until now, the architecture is considered as a set of ECUs, which is also a set of elements, and buses. The cost of the architecture can thus be approached as the cost these elements. Consequently

\[ Cost = \sum_{\text{elements in architecture}} \text{element cost} \]

We propose here a model for estimating the cost of these elements in order to estimate the cost of the architecture.

A component Cost

A component is an element of the architecture. It can consists of a sensor, a microcontroller, an actuator, a connector etc ... These elements are often acquired with a given price. The price of the element is not though its cost in the architecture. The elements can be implemented in different ways. The hardware engineering can lead to multiple hardware implementations with different costs. In addition, the same element can be implemented at different ASIL levels. Implementing the element to reach an ASIL requires the implementation of safety mechanisms to reach quantitative targets. This proves that the element price is different from its cost in the architecture.

The cost covers a wider scope than the price. In our estimation of the architecture cost, we judge more accurate for decision making to take into account the cost of an element not its price.

We assume that the cost of an element is the combination of the following factors:

- price
- surface
- targeted ASIL level
The price is a mean value for the acquisition cost of the element based on previous projects or the market estimations. To the price of the element we add the safety level impact. Reaching a certain ASIL requires adding additional elements. For example, a microcontroller implemented at an ASIL level can require the presence of a watchdog to ensure meeting the ASIL quantitative targets. This leads to different costs depending on the targeted safety level allocated to the element in the architecture context. As for the surface parameter, it allows to take into account the cost of the printed circuit board (PCB). The size of the PCB depends on the size of the component. Larger components, logically, participate more in the PCB cost. Its cost can be thus distributed over the components depending on the size of each.

An element cost is, thus, is partly fixed through its price and its surface. A second part is variable and depends on the ASIL level to be reached.

\[
\text{Element.cost} = \text{Element.price} + (\text{Element.surfacePCB.cost}) + \text{Element.implementation(ASIL)}
\]

A bus cost  We discussed in the architecture synthesis algorithm, the impact of a bus on the architecture and how it can be implemented. This implementation have logically an impact on the cost. While the impact consisting of the implementation pattern have been taken into account during the synthesis and thus its cost impact taken into account as components cost. A bus can impact the cost through the connectors. These elements that are present at each ECU have a considerable cost and should be taken into account.

Thanks to the information provided by the architecture, we can identify the interfaces implemented by each ECU of the architecture. These interfaces implemented by connectors can be associated a cost depending on the type of the interface and the number of pins.

The development effort Model

An architecture cost is not just the cost of its elements. A development effort need also to be considered. This effort depend on many parameters such as technology, safety levels, real-time constraints and reuse etc... The development effort estimation is a complicated task and has been often based on the expert judgment. Cost models such as COCOMO are widely used. The COCOMO model is a software cost estimation technique. It is based on estimation of the size of the software and other parameters. These are only available at advanced stages of development. It can be applied as soon as the software architecture is starting to be put in shape and the main software modules has been identified. Such details are not available at the level, we are comparing the solutions. We need a cost model adapted to the level of granularity and the inputs available.

Cost models such Cosmic allows the estimation of the software functional size and consequently an overview of the development effort. It is adapted to the estimation at the concept phase and can be extended to take into account specific needs.
Cosmic model and function points  COSMIC method is a functional size measurement method and an ISO standard (19761). It helps estimating the size of a software based on its functional specification. In COSMIC, the Functional User Requirements (FUR) are decomposed into elementary processes called “Functional Processes”. A Functional Process is “an elementary component of a set of FUR comprising a unique, cohesive and independently executable set of data movements”. The data movements are the key for the functional size estimation. It consists of the data exchanged with the functional users the software interact with. Once the data movements for each functional process is identified, the functional size can be measured by aggregating the results at functional process level. The measurement requires a level of functional decomposition where the functional process can be identified. These blocks allows to recognize the data movements: Entry, Exit, Read and Write movements.

— Entry: a data movement type that moves a data group from a functional user across the boundary into the functional process where it is required.
— Exit: a data movement type that moves a data group from a functional process across the boundary to the functional user that requires it.
— Read: a data movement type that moves a data group from persistent storage within reach of the functional process, which requires it.
— Write: a data movement type that moves a data group lying inside a functional process to persistent storage.

COSMIC is generic and for general use technique. It can be applied to business application and in real time systems. Previous works focused on the automation and the application of COSMIC in real time system development. The approach have proven its advantages in the estimation of functional size. It has also been used to enhance project planning and efficiency plans. Despite the positive reports on COSMIC application, the approach has been criticized for its limits. It lacks taking into account the complexity and the data types impact. It does not also take into account the safety into consideration. But these parameters have been proven to impact the software and its development cost. Taking into account the advantages and the need for an estimation approach similar to COSMIC, we decided to exploit the adaptation possibilities of the approach to propose an extension. The extension aims to tackle the limits identified above. We aim to adapt the approach for automotive systems and integrate the safety from ISO 26262 perspective.

Adapted cosmic approach  In cosmic, each data movement have a weight of 1 CFP. With this logic, the data movement’s interface type is not captured, neither is the functional complexity nor the safety level. An output on a hardware pin or on a network, CAN for example, are considered to participate equally to the software size independently of the functional complexity behind it and the safety level it was allocated. In reality, that’s not the case. These parameters have an impact on the development effort. For example, in order to communicate over a network like CAN, more development effort is needed for the communication layers compared to the hardware pin output. In order to extend the limits of the cosmic model, we adapted it, since allowed by the standard itself, to take into account the following parameters that are not supported by default:
— **Data interface types**
  The data movements have now different weights depending on the type of
  the interface.

— **Functional complexity**
  In addition to the data movement, the functional complexity of the functional
  processes are taken into account during the estimation.

— **Safety level**
  The impact of the safety level on the functional size is taken into account.

These parameters can be introduced during the functional points counting. As
no changes will be brought to the approach in its core, the estimation process is
conserved. Thus, the estimation goes through the functional processes identification
as defined in the standard. Identifying the functional processes is a key step. It
requires the identification of the triggering events and the functional users. In our
case, these steps are automated based on the functional architecture and the physical
architecture solution being investigated. For the automation we assume that:

— The input flows are the triggering events. The functional users are conse-
  quently, the hosts of the input flow identified as triggering events.

— The functional processes depend on the functionalities described by the data
  flow graph. For each functionality, the functions allocated to the software
  form a functional process.

— The mapping defines the inputs, outputs types of the functional process.

During the estimation, the parameters identified above are taken into account during
the functional points aggregation. Their impact on the effort development is trans-
slated in an impact on the functional size. The equations formalizing this impact are
defined as follows:

1. **Functional complexity**
   In the functional architecture, we assume that the functions complexity can
   be put in one of the classes: manageable(low complexity), Medium or Hard(high
   complexity). These levels are allocated weights allowing to capture the im-
   pact of the complexity level on the functional size. For example we propose
   a simple model where the complexity levels are allocated weights as follows:
   — complicated: 6
   — medium: 2
   — manageable: 1
   These values can be adapted and calibrated depending on the previous projects
   data. Once they are fixed they are used uniformly in the estimation to ensure
   the comparison is relevant. For each functional process, the functional com-
   plexity is the result of the sum of the functional complexities of the functions
   used by the functional process:

   \[
   \text{functionalprocessfunctionalcomplexity} = \sum_{f_i \in \text{functionalprocess}} \text{complexity}(f_i)
   \]

   The software functional complexity is than deduced and calculated as the
   sum of the functional complexities of the functional processes.

2. **ASIL weight**
   The ASIL level have an impact on the development effort. In order to have a
relatively precise estimation, we introduce it as a parameter in the functional size estimation. The ASIL impact is introduced as a weight that captures the ASIL level value and its impact on the functional size of the software. For the four safety levels, the values should be the result of a statistical study of the projects data. But for the illustration and the use of the model, we fix the values as follows:

— QM: 1
— A: 1.5
— B: 2
— C: 3
— D: 3.5

The ASIL weight is introduced in the functional size estimation through the equation:

\[ \text{functional process functional size} = \text{functional complexity} \times \text{weight}_{\text{ASIL}} + CFP \]

The weight allows to highlight the development and design effort that is needed to meet the target safety level.

3. Interface choice

Data can be exchanged in different technologies. The choice of the interface requires additional resources and additional software components. In order to take into account the software components or layers in the functional size estimation, two options are available. The first is to conduct the functional size estimation at the software levels. The second is to allocate prefixed functional size for these components.

The first option requires the knowledge about the software architecture, i.e. the layers and how they communicate. Such level of detail is not yet available at the studied abstraction level where the software is still approached as a black box. The second option is thus preferred assuming that such input information could be obtained. This translated into fixing functional size values for the software resulting from the implementing the communication components for the specified interface. For example, if the software output or input goes through CAN network, we add the functional size weight of the CAN layer to the software size estimation.

For example, we used the following values for the different interfaces types used in our tests and that can be adapted later depending on the projects and the considerations of the designer:

— CAN: 5
— LIN: 4
— Hardline: 1
— Flexray: 6
— SPI: 3
— UART: 3
The functional size resulting from the interfaces are added to the estimation from the functional complexity and data movements.

\[ \text{Functional size} = \sum \text{Interface size} + \sum \text{functional process functionalsize} \]

**Reuse and development effort**

COSMIC model do not only allow the estimation of the functional size of the software but also to estimate the development effort. The development effort is often approached through lines of code or man-hours. These are estimated roughly and based on expert judgment. The COSMIC model can approach the question in a more formulated and structured manner. It allows to compare the functional size of the software to develop based on previous solutions and the changes to be applied. Knowing that it is possible to conclude the number of lines of code from the functional size, it is also possible to use the previously used methods to estimate the cost.

### 5.7 Conclusion

In this chapter we described the proposed approach for the optimization of an embedded system architecture. It consists of an architecture synthesis approach taking into account the automotive systems context particularly the ISO 26262 standard. The process includes key architecture design tasks such as the functional safety concept, the physical architecture and the cost estimation.

At the functional safety concept, we proposed an approach allowing to solve the ASIL allocation and decomposition problem. It identifies the ASIL allocation scenarios applicable based on the safety analysis results at the functional architecture level. It also identifies the additional constraints to be taken into account during the physical architecture design to ensure the respect of the independence requirement for the ASIL decomposition.

The approach is based also on the architecture synthesis capable of reaching candidate solutions starting from the functional architecture and the ASIL allocation scenarios. The architecture synthesis consists of the architecture components choice and the implementation of the functional architecture on the obtained physical architecture. The implementation takes into account the design constraints, the safety constraints and the engineer preferences in the process. While it is inspired from the previous works on the architecture synthesis in the chips design industry, it differentiate by proposing an automotive specific architecture construction algorithms and an ISO 26262 perspective for the functional safety requirement implementation. It is an innovative approach which proposes new techniques allowing to jointly propose a physical architecture solution and ensure the compliancy with the ISO 26262.

The approach is developed based on the results and the observation made by applying state-of-the-art approaches. In the same research methodology, we apply the approach on the same study case to compare the results and analyze its limits in the next chapter.
Approach Implementation and Study Case

6.1 Introduction

We described in the last chapter the proposed approach for the architecture synthesis and optimization along with the algorithms and the assumptions guiding the process. The approach was developed to be generic and allow implementation in different context. We propose here a possible implementation using a set of system and safety engineering tools. At one hand, we aim to prove that the approach is implementable in the considered context. On the other hand, this allows us to apply the approach on a the study case studied in the chapter 4 and discuss compare the results and limits.

6.2 Tool Implementation

The approach discussed in the previous chapter is implemented using common engineering tools. Each tool allows to implement one or multiple tasks of the of the optimization process. Some of these are automated while other require the user input.

We propose an implementation for the input model using Simulink as functional and safety model support. A developed tool implementing the synthesis, constraints verification algorithms allows to import the data from the input model to reach candidate solutions.

6.2.1 Input Model

Functional Model

In section 5.3.3, we explained the choice of DFG for the functional architecture
modeling and the motivation behind it. It is mainly to allow the implementation of
the approach in a variety of modeling frameworks. Matlab Simulink is one of the
modeling frameworks that are widely used for the embedded systems modeling. We
developed the required interfaces in order to exploit the Simulink models for the
optimization process.

The Simulink models are based on a representation of the architecture elements
based on libraries addressing different viewpoints of the system architecture. At
functional level, the subsystem blocks and atomic blocks can be used to represent the
functional blocks of the architecture while the flows between these blocks represent
the data flows between the functional blocks.

These elements can clearly relate to the elements of the DFG, ie, nodes and
the flows. This allows us to extract the model data from the simulink model to
retrieve the DFG model. This transition requires the simulink model to respect the
assumptions discussed earlier for the system model 5.3.2. The lowest level blocks
should have one output for example.

The DFG contrarily to the Simulink model does not allow multiple level models.
This requires that the transitions between these models to be based on the atomic
level of the Simulink model.

Matlab Simulink models can be saved in different formats. They can easily be
exported to XML files. The structured format of these files allows the parsing and the
extraction of the data accessible and relatively simple. The proposed implementation
is based on a file based exchange between Simulink and the optimization framework
illustrated in the figure Figure 6.1.

![Figure 6.1 – Model import process](image)

The parsing of the Simulink model allows to identify the functional blocks and
the flows in the model. The Simulink model is described as a tree structure. The
functional blocks corresponds to the leafs of this structure. In order to flatten the
model and remove the abstraction layers concept in the model, we seek only to reach
the lowest level. In case the functional blocks are linked to a Simulink library, the
functional blocks can be missed. To avoid this we export the Simulink models in
XML with the option ensuring to break all the links. But breaking the links

The transition from a Simulink model to DFG follows the pattern presented in
figure Figure 6.2. The functional blocks identified in the model are added to the
functional architecture model based on the previous chapter assumptions.
6.2. TOOL IMPLEMENTATION

Failure Model
The failure model is based on two inputs. The first is the result of the safety analysis on the functional architecture. The second is the failure data of the hosts describing the failures modes, their causes and their impact on the interfaces as well as the attainable safety levels.

The failure analysis is conducted using the HIP-HOPS plugin in Simulink. The plugin allows obtaining the UE fault trees and the MCS at functional level. These elements are exported in XML files. The analysis results are extracted and processed to calculate the ASIL scenarios and the independence constraints that may result from them.

Host Library  In the optimization approach, the design space size is mainly defined by the set of functions hosts. These are identified prior to the optimization process. They are described through the parameters detailed in the previous chapter. The hosts data consists of safety, cost and interfaces features of the element. These elements can be gathered based on previous projects or engineering expertise. The data can be implemented as a database. But at the actual progress state, such solution is not implemented. The Hosts library are entered manually in the tool.

6.2.2 Optimization Engine

Once the first step focusing on the problem definition (system model, safety analysis) are finished, the inputs for the optimization process are available. The optimization engine imports the data from the XML files used as interfaces with the rest of the tools mainly Simulink and HIP-HOPS.

Overview
The optimization tool Figure 6.3 implements the approach through the integration of a set of software tools implementing the steps of the process. This requires an ASIL allocation scenarios solver, an independence constraints extractor, an architecture synthesis engine, a constraints verification engine and a cost estimator.
The ASIL allocation scenario solver implements a parsing algorithm allowing to parse the safety analysis results stored in XML files. After transforming the problem into the equation form, different solving algorithms can be used. We implemented the solving algorithm presented in the previous chapter as a primary solving algorithm. We implemented to also solve algorithm such the one proposed in Maenad. Another implementation alternative consisted of using generic mathematical solver. We implemented for this an interface to Jacob solver.

Implementing the interface to solvers such as Jacob is motivated by the evolutionary potential and the continuous maintenance contrary to the dedicated algorithms. Though we expect a better performance results from the dedicated algorithms. These will be confirmed through the results from the application in the study case and presented in the next section of this chapter.

Independence constraints extractor

The independence constraints, identify the independence constraints that results from the ASIL allocation scenarios. Based on the values in the scenarios, it provides the architecture synthesis engine the independence constraints corresponding to each scenario. It consists of identifying the flows concerned by an independence requirement resulting from the decomposition and the respective failure modes leading to the undesired event. The tool implements the algorithm presented in chapter
5. The results of the process are exploited by the constraints verification engine which ensures that the independence constraints are respected in the architecture.

**Architecture Synthesis**

Architecture synthesis engine implements the design space exploration step in the optimization process. It focuses on the architecture building and the technological choices implementation. It transforms the functions and flows hosts choices into a coherent architecture. It takes into account the ASIL allocation scenarios, the independence constraints, and the system model and provide candidate solutions. The design space exploration is implemented using the algorithms discussed earlier. The exhaustive algorithm is implemented in a dedicated engine, while, the genetic algorithm is implemented using the library JGAP.

JGAP is a configurable tool to implementing and applying genetic algorithms to optimization problems. Although it is capable of doing all the evolutionary simulation in a relatively generic manner, it cannot decide either how the solution is coded or how to define the fitness of the potential solution. These should be specified by the user depending on the application. In general in order to use the library, the problem needs to be formalized through a coding and a fitness and evaluation strategy. The coding determines which information are needed to describe a potential solution. These are coded as a chromosome. The fitness function, on the other hand, to evaluate the potential solutions and rate them to retain the most fitted. The aim is to reach a solution that reduces the cost without incurring penalties. In JGAP, the chromosomes are instantiated during the simulation based on a provided sample chromosome. As shown in figure 5.8, the sample chromosome is an array of composite genes. The size of the array corresponds to the number of functions. The genes are described as integers with limited range depending on the number of available alternatives for each value.

Both algorithms have different strategies to reach an optimal and compliant design, but they follow the same architecture construction logic and rules. The candidate solutions are assessed by the constraints verification engine and cost estimator. The system model, ASIL allocation scenarios are the inputs allowing to define the alternative designs and assess them. Two design space exploration algorithms are implemented to offer more flexibility to the process. Depending on the system size and complexity, a choice of the algorithm to be applied shall be made. This choice can have an impact on the execution as well as the results obtained. We use the application results on the study case to discuss these elements with more details in the following section of this chapter.

**Constraints verification engine**

The constraints verification engine is used to assess the candidate architecture during the design space exploration. Its main role is to check the constraints related to the design compliance.

The verifications are conducted based on the concepts introduced in chapter 4. The synthesis engine can request from the constraints verification engine to check the following properties:
6.3 Study Case: Results and Discussion

We have studied the optimization approaches in chapter 4 using a real world example consisting of the PEPS system. We use the same example here as a study case to assess the benefits of the proposed approach. Unlike the optimization process, we take into account all the safety requirements and the variability in the architecture into account.

6.3.1 Functional model/Failure model

The functional model describes the functional decomposition of the system main functionalities and the basic functions. It follows the assumptions fixed in the previous chapter. Simulink models, Figure 6.4, represents the functional model corresponding to the solution proposed by the engineers and developed manually. The functional architecture represents here the different operational logic implemented to ensure the system functionalities. But unlike the description provided in the chapter 4, the solution here is described independently of the physical architecture. It focuses only on the functional dependencies between the basic functions with less focus on the components and interfaces implementing them. These will be specified by the optimization process later on. The functional architecture model in Simulink, represented in the figure, is decomposed to an abstraction level that corresponds to the level meeting the inputs of the optimization model. The model is then exported to the XML format to be imported by the optimization framework.

A second difference from the applied state of the art approaches is the variability feature. At this level, we identify the functions that could be dropped in some versions of the system. The advantage is that the impact of the integration or the removal of those functions can be analyzed and taken into account effectively. The concerned functions are represented in the functional model and will be tagged as optional in the optimization model in the next step.

Safety analysis is needed at this level for the safety levels allocation problem. The failure model is annotated on the functional architecture using the Hip-HOPS plugin. We take into account all the undesired events specified in the chapter 4. The results
6.3. STUDY CASE: RESULTS AND DISCUSSION

(a) steering

(b) power-management

Figure 6.4 – Simulink model: Functional Architecture - Part 1
Figure 6.5 – Simulink model: Functional Architecture - Part 2
of the analysis, presented in the Figure 6.6, represent the fault tree and the MCS for the SG5. The MCS are included later in the optimization model for the ASIL allocation scenario calculus along with the other SGs.

![Fault Tree](image1)

(a) UE fault tree

![MCS](image2)

(b) UE MCS

Figure 6.6 – ESCL management: Safety Analysis results
6.3.2 Optimization model

Functional architecture
We extract the functions and the flows from the Simulink models through a parsing and a flattening processing. To illustrate the process and the results, the obtained elementary functions and flows for the ESCL management are summed in the table Table 6.1. The full listing of the functionalities are represented in the annex. At this level, the variability is taken into account. The functions concerned are tagged and taken into account as such in the optimization. We will assume here that the ESCL management functionality is optional and may be removed from some version of the system. The corresponding functions are tagged as optional in the optimization model. The impact of the variability on the optimization results will be ignored at first and reintroduced and discussed later by comparing the results with and without the variability.

<table>
<thead>
<tr>
<th>Functions</th>
<th>Detect_acc_change, Check_valid_ID, Check_steering_lock_status, Vehicle_speed, Vehicle_mvt_state, Check_IG_state, Unlock_ESCL, Warning_unable_unlock, Lock_ESCL, Warning_unable_lock, ESCL_unlock_conditions_check, ESCL_check_is_unlocked, ESCL_lock_conditions_check, Command_lock, Save_ESCL_state, ESCL_check_is_locked, Check_speed_condition_for_lock</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Flow</th>
<th>Origin function</th>
<th>Target function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc_change</td>
<td>Detect_acc_change</td>
<td>ESCL_unlock_conditions_check</td>
</tr>
<tr>
<td>Valid_ID</td>
<td>Check_valid_ID</td>
<td>ESCL_unlock_conditions_check</td>
</tr>
<tr>
<td>Steering_lock_state</td>
<td>Check_steering_lock_status</td>
<td>ESCL_check_is_unlocked</td>
</tr>
<tr>
<td>Steering_lock_state2</td>
<td>Check_steering_lock_status</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Steering_lock_state3</td>
<td>Check_steering_lock_status</td>
<td>ESCL_check_is_locked</td>
</tr>
<tr>
<td>Doors_state_change2</td>
<td>Detect_doors_states_changes</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Vehicle_speed_data</td>
<td>Vehicle_speed</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Vehicle_mvt_state_data</td>
<td>Vehicle_mvt_state</td>
<td>Check_speed_condition_for_lock</td>
</tr>
<tr>
<td>IG_state</td>
<td>Check_IG_state</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Unlock_ESCL_request</td>
<td>ESCL_unlock_conditions_check</td>
<td>Unlock_ESCL</td>
</tr>
<tr>
<td>Unlock_ESCL_request2</td>
<td>ESCL_unlock_conditions_check</td>
<td>ESCL_check_is_unlocked</td>
</tr>
<tr>
<td>Warning_unable_unlock_request</td>
<td>ESCL_check_is_unlocked</td>
<td>Warning_unable_unlock</td>
</tr>
<tr>
<td>ESCL_ok_for_lock</td>
<td>ESCL_lock_conditions_check</td>
<td>Command_lock</td>
</tr>
<tr>
<td>ESCL_ok_for_lock2</td>
<td>ESCL_lock_conditions_check</td>
<td>Save_ESCL_state</td>
</tr>
<tr>
<td>ESCL_ok_for_lock3</td>
<td>ESCL_lock_conditions_check</td>
<td>ESCL_check_is_locked</td>
</tr>
<tr>
<td>ESCL_lock_request</td>
<td>Command_lock</td>
<td>Lock_ESCL</td>
</tr>
<tr>
<td>Warning_unable_lock_request</td>
<td>ESCL_check_is_locked</td>
<td>Warning_unable_lock</td>
</tr>
<tr>
<td>ESCL_lock_request_confirmation</td>
<td>Check_speed_condition_for_lock</td>
<td>Lock_ESCL</td>
</tr>
</tbody>
</table>

Table 6.1 – ESCL management: DFG

Hosts The functions hosts are the elements that can implement a given function. Based on the functional model, we identify the set of elements for each function. The flows are also given a set of potential hosts consisting of the buses that can be used for the flow exchange. The Table 6.2 provides an example of the Hosts specified for the
functions and flows extracted from the functional architecture model. We provide here the considered Hosts for the ESCL_management functionality functions. In Annex B, the hosts for the rest of the functions are provided.

The set of hosts is put following these three properties:

— covering the possible ASIL for the host capabilities:
  In order to allow the optimization process to effectively check the ASIL scenario, we provide hosts capable of implementing the functions at any ASIL. For example, for the functions to be allocated to a microcontroller, we set four different hosts with ASIL level capabilities from ASIL A to ASIL D

— Providing hosts with different interfaces for more flows allocation flexibility
  The interfaces of the hosts have an impact on the flow allocation and can limit the solutions spectrum. We provided thus hosts with interfaces covering the different implementation possibilities such as CAN, LIN, FLEXRAY, SPI, UART ...

— Generic component for the out of perimeter subsystems limited to these subsystems properties
  Some functions will be allocated to subsystems out of the system perimeter. Since these subsystems can not be redesigned and can only implemented as they are, we set unique generic host for functions allocated to these subsystems. The generic host is a description of the possible interactions offered by the out-of-perimeter subsystems that need to be taken into account in the architecture design. For example, in the optimization model we consider that the BCM_ECU can interact with the PEPS only via CAN network. To take this into account, we set the host for the functions allocated to the BCM_ECU as a single “BCM_component” which have only a CAN interface (see B).

The elements identified in the table are the constituents of the architectures that will be proposed by the optimization process. The specified elements can be chosen in order to explore more possibilities or be limited to direct the optimization approach to a preferred solution. This allows to take into account the engineers preferences although the design process is automated. These preferences can consist of microcontrollers choices, buses choices, etc... Omitting citing an element from the hosts lists removes that design path. The list, thus, can be exhaustive if the objective is to rethink the design or limited for solutions in a pre-chosen direction.

For the architecture synthesis constraints, the Hosts attributes such as cost, interfaces, failure modes, safety level are specified following the model presented in the chapter 5. The Table 6.3 illustrate an example of a microcontroller host used in the optimization model and its attributes.

### 6.3.3 Optimization results

**Problem Size** The optimization model contains 59 functions and 85 flows. The hosts identified for these functions and flows leads to around $2 \times 10^{58}$ possible combinations. These combinations increases with the results of the ASIL allocation scenarios. For our case study, the ASIL allocation solver identified 72 scenarios,some are represented in the Table 6.4, rising the possible combinations even more.
The high number of combinations, considering we can also add the combinations resulting from the controllers choice for the buses, proves the fact that the process can not be manually done. The designer are faced with a high number of alternatives where the choice can be difficult to make. This proves the practical benefit that our automated design process can bring. In the same time, the high number of
6.3. STUDY CASE: RESULTS AND DISCUSSION

<table>
<thead>
<tr>
<th>Host</th>
<th>Mic1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interfaces</td>
<td></td>
</tr>
<tr>
<td>Internal C2C</td>
<td>Hardline, SPI</td>
</tr>
<tr>
<td>External E2E</td>
<td>CAN, LIN, Hardline</td>
</tr>
<tr>
<td>Cost</td>
<td>10</td>
</tr>
<tr>
<td>Max ASIL</td>
<td>B</td>
</tr>
<tr>
<td>Failures</td>
<td>Impacted Interface</td>
</tr>
<tr>
<td>Bad Output on CAN</td>
<td>CAN</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 6.3 – Host attributes example

<table>
<thead>
<tr>
<th>Function 1</th>
<th>Failure mode</th>
<th>Function 2</th>
<th>Failure mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vehicle Speed</td>
<td>Bad value</td>
<td>Vehicle Mvt state</td>
<td>Bad value</td>
</tr>
<tr>
<td>ESCL.Lock_conditions_check</td>
<td>Bad value</td>
<td>ESCL_speed_condition_check</td>
<td>Bad value</td>
</tr>
</tbody>
</table>

Table 6.5 – Independance Constraints in the ESCL management functionalities

combinations can put the optimization process in difficulty.

During the process, using a branch and cut approach, the number of combination processed is reduced using the ASIL filtering and the constraints avoiding high processing time. The comparison of the compliant candidates and avoiding the processing of very similar solutions allowed also to reduce the optimization time.

The allocations scenarios leads often independence requirements. Through the scenarios presented in Table 6.4, allows to identify 16 independence constraints. The functions from the same mcs and with ASIL lower than the safety goal prove that a decomposition is made and thus the functions should be independent. This lead us to 16 independence constraints to be respected when applying the ASIL allocation scenarios obtained, an example of the obtained constraints for the ESCL management functions is illustrated by the Table 6.5.

**Optimization results** Using the exhaustive algorithm, coded in Java, the optimization process took 20 hours running on a pc equipped with 8gb of RAM and I7 processor. The obtained architecture is similar to the solution proposed by the engineers as presented in Chapter 4. The obtained architecture, illustrated in the Figure 6.7, integrate the ECUs involved in the PEPS functionalities. Some are out of the system perimeter such as BCM_ECU, ICM_ECU and ECM_ECU, while the PEPS_ECU and ESCL_ECU are the main subsystems of the system in study. The out-of-perimeter ECUs internal architecture are not detailed and are represented as
an abstract component. But for the PEPS_ECU and ESCL_ECU, more details about the components and the interfaces are indicated.

The obtained architecture implements an ASIL decomposition scenario where the requirements from the safety goals are decomposed. Unlike, the solution proposed by the engineers, the decomposition is applied three times on each of the safety goals. This, though, lead to a similar solution since the ECUs and the elements introduced in the architecture depends on the optimization model, i.e. the hosts inputs. If the same design logic used by the engineers, is reintroduced in the optimization model it is normal that the results keeps the proposed solution main topology. Some differences are though present but they consist of the ASIL allocation scenario, the functions partitioning on the architecture elements and the components choices.

The time needed for the optimization process is though considerably important. For thus study case, the optimization process reached more than 24 hours. While, the process may need to be launched multiple times after each modification of the functional architecture, the exhaustive algorithm, although, efficient on small size examples, can not be used in this case.

Contrarily, the processing time is reduced significantly while using the genetic algorithm. The process takes only 9 minutes. The processing time difference can be easily explained by the number of combinations or candidate solutions that were checked regarding the constraints and evaluated for cost. Through the genetic algo-
6.3. STUDY CASE: RESULTS AND DISCUSSION

<table>
<thead>
<tr>
<th>Subsystems</th>
<th>Components</th>
<th>Allocated Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Door_handle</td>
<td>HSU</td>
<td>Detect_user_position, Detect_user_presence</td>
</tr>
<tr>
<td>ABS_ECU</td>
<td>ABS_component2</td>
<td>Vehicle_mvt_state</td>
</tr>
<tr>
<td></td>
<td>ABS_component1</td>
<td>Vehicle_speed</td>
</tr>
<tr>
<td>ESCL_ECU</td>
<td>MIC1</td>
<td>Check_steering_lock_status, Command_lock</td>
</tr>
<tr>
<td></td>
<td>MECA</td>
<td>Lock_ESCL, Unlock_ESCL</td>
</tr>
<tr>
<td>PEPS_ECU</td>
<td>MIC1</td>
<td>Detect_acc_change, Check_IG_state, Save_ESCL_state,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disable_peps, ESCL_check_is_locked, Check_access_conditions,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check_lock_conditions, ESCL_unlock_conditions_check,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ESCL_check_is_unlocked, Check_doors_for_lock,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check_car_lock_state, Ext_ID_search_strategy,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Post_lock_check, ESCL_lock_conditions_check,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Check_acc_state, Check_relock_conditions</td>
</tr>
<tr>
<td></td>
<td>ASIC</td>
<td>Decrypt_ID, Check_ID_rights, Check_speed_condition_for_lock,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Extract_ID</td>
</tr>
<tr>
<td>ICM_ECU</td>
<td>ICM_component</td>
<td>Warning_unable_unlock, Warning_unable_lock, Warning_doors_open</td>
</tr>
<tr>
<td>Antennas</td>
<td>Antenna_at1</td>
<td>Detect_ID_position1</td>
</tr>
<tr>
<td></td>
<td>Antenna_at2</td>
<td>Detect_ID_position2</td>
</tr>
<tr>
<td></td>
<td>Antenna_at3</td>
<td>Detect_ID_position3</td>
</tr>
<tr>
<td>BCM_ECU</td>
<td>BCM_component</td>
<td>Detect_doors_states_changes, Doors_locked, Lock_car, Unlock_car, Doors_closed</td>
</tr>
</tbody>
</table>

Table 6.6 – Optimized architecture description

algorithm processing, only 33500 combinations/individuals were treated. Compared to the over $2 \times 10^6$ combinations checked by the exhaustive algorithm while only treating 2 out of the 72 scenarios, the difference in the execution time is easily explained.

The reached solution, as with the exhaustive algorithm, preserve the architecture main topology with minor changes in the components choices and the ASIL allocations.

But, unlike the exhaustive algorithm that ensure that the obtained solutions verifies all the constraints, the genetic algorithm reach the best solution attainable giving the the simulation time (the number of generations). The penalty introduced
in the fitness functions helps leading to fully respect the constraints as it heavily penalizes the non respected constraints. But no guarantee that the reached solution is a “good” solution. A second limit is the alternatives presented. Using the exhaustive algorithm, we were able to propose the top best solutions found. The user can, considering parameters not taken into account, make a choice among these solutions. The genetic algorithm provides though a unique candidate solution. If this solution can not be retained, no guarantee that relaunching the process can provide a different solution. In the same time, we can not recommend making changes to the optimization model (functional architecture) since the obtained results can not affirm if these changes are needed.

![Figure 6.8 – Genetic Algorithm results](image)

### 6.3.4 Variability Impact

In order to take into account the variability and its impact on the architecture, we considered the ESCL management functionalities as optional functions. In some versions where cars do not use electronic steering lock, the system can be adapted accordingly. Removing the ESCL management from the system can have an impact on the architecture. Some components may be removed if they are no longer used. But if the variability is not taken into account, the design and the components choices can lead to high cost solutions considering the remaining functionalities and their ASIL levels.

In our optimization model, we tagged the ESCL management functions as optional. Compared to the optimization without variability, this add a second objective consisting of finding the best architecture where ESCL management is removed from the system. Considering the fact the variability introduction rises the combinations to be checked and the processing time, we decided to use only the genetic algorithms. The exhaustive algorithm application would be unpractical considering the execution time observed and discussed in the previous section.

The Obtained solution, in this case, is different from the obtained previously.
6.3. STUDY CASE: RESULTS AND DISCUSSION

(a) With ESCL Management

(b) Without ESCL Management

Figure 6.9 – Optimization with variability results
In the figure Figure 6.9, we observe that the architecture subsystems remain unchanged. But the system topology is different, the architecture detailed description in the annex. We can observe that the LIN based communications between the PEPS_ECU and ESCL_ECU has been removed. The hardware line connecting PEPS_ECU to the ABS_ECU, too, was replaced with a hardware line between the ABS_ECU and the ESCL_ECU. These differences are the result of a different functions mapping to the subsystems. By comparing the function mappings presented in Figure 6.9 and Figure 6.7, we can see that many of the functions that were allocated to PEPS_ECU and involved in the ESCL management are now hosted by the ESCL_ECU. This change of functions allocation led not only to the observed differences in topology. But also, by allocating the functions responsible of checking the speed condition for the lock and the vehicle movement state to ESCL_ECU, the ESCL_management became more decoupled of the rest of the PEPS system functions. This allows removing this functionality limited to removing the ESCL_ECU and disabling some software functionalities. Contrarily to the solutions proposed before, this will not lead to oversizing in the PEPS_ECU components. In the architecture Figure 6.9, the removal of the PEPS_ECU will result in a LIN interface left unused and the input from the ABS over Hardline unexploited. This is not the case for the proposed architectures in Figure 6.9.

We observe through this example, the advantages of taking the variability into account in the architecture optimization. Although it is more processing time consuming and lead to “full” architecture a little higher in cost, it makes the adaptation of the system less problematic and cost efficient.

6.4 Conclusion

In this chapter, we implemented the approach proposed in Chapter 5 and proved its applicability through the study-case. We used for experimentation Simulink for functional model, HiP-HOPS plugin for the safety analysis and JGAP library for the genetic algorithm simulation. The algorithms for the ASIL allocation scenarios, the architecture synthesis and the constraints verification are implemented in a separate framework that interfaces with these tools.

The obtained results proved also that the approach is capable to propose compliant preliminary physical architectures regarding the safety requirements based on the functional description of the system and given the required inputs. The study case allowed to observe the impact of the problem size on the computing time and the performance of the algorithms. Both the exhaustive and genetic algorithm based approaches have their advantages and limits. On one hand, the exhaustive approach helps to reach multiple solutions and have easier interpretation of the results obtained by the end of the process. On the other hand, the genetic algorithm allows to cope better with the size of the problem but propose a unique solution that can be confusing to exploit if it does not respect all the constraints. The designer can not in this case decide if the functional design need to be reviewed.

By studying the variability impact on the design, we confirmed the benefits of addressing this issue in the optimization approach. We proved that the variability, when taken into account in the design process, can have a considerable impact
6.4. CONCLUSION

on the final design. The impact of the “optional” functions, particularly if safety requirements are involved, can be considerable and difficult to foresee. For the study-case, the obtained optimal designs where variability is introduced and ignored are considerably disparate although the same inputs are used.
Conclusion and future works

In this thesis, we focused on the challenges faced during the design of embedded automotive systems. Our objective consisted of proposing a methodology that can assist in reaching a cost optimal system architecture. With the growing complexity and size of the systems and the new concepts introduced by the ISO 26262, reaching a compliant design without incurring unnecessary costs is a challenge especially for the manually performed process.

The baseline of the research was set with a study of the concepts and properties of the targeted systems as well as the process used for the architecture engineering. We focused particularly on the functional safety concepts from the ISO 26262 point of view. This study allowed us to understand the system design challenges and investigate the key architectural decisions having the highest impact on the design cost. We concluded that the functional design, the safety levels allocation and the components choices in the physical architecture are the main causes of the cost derive. These choices made at different levels of the design process deeply impact the system design and its cost and should thus be addressed.

The optimization of the system design is not a recent subject. It has been studied in different contexts in the literature. We reviewed the optimization approaches and categorized them by their optimization targets, techniques and algorithms. We applied some of these approaches on a study case to analyze the gap and their limits. The results allowed us to conclude that the objective of our research and the automotive system properties can be met with an architecture synthesis approach adapted to the current context.

Based on these conclusions, we proposed a novel architecture synthesis approach for automotive systems. It consists of an architecture synthesis approach taking into account the automotive systems context particularly the ISO 26262 standard.

At the functional safety concept level, we proposed an approach allowing to solve the ASIL allocation and decomposition problem. It identifies the ASIL allocation scenarios applicable based on the safety analysis results at the functional architecture level. It also identifies the additional constraints to be taken into account during the
physical architecture design to ensure the respect of the independence requirement for the ASIL decomposition.

The approach proposes an architecture synthesis process capable of reaching candidate solutions starting from the functional architecture and the ASIL allocation scenarios. It consists of the architecture components choice and the implementation of the functional architecture on the obtained physical architecture. The implementation takes into account the design constraints, the safety constraints and the engineer preferences in the process. While it is inspired from the previous works on the architecture synthesis in the chips design, it differentiate by proposing an automotive specific architecture construction algorithms and an ISO 26262 perspective for the functional safety requirement implementation. It is an innovative approach which proposes new techniques allowing to jointly propose a physical architecture solution and ensure the compliancy with the safety constraints.

The approach was developed with the implementation challenges in mind. To prove its applicability, we developed a framework implementing the synthesis process interfaced with Simulink and HIP-HOPS plugin for the functional model and the safety analysis. The study-case used for the experimentation is the PEPS system. The choice of the study case is motivated by its size, the design constraints and the multiple safety goals it implements. The optimization problem is NP complete and the results from applying the exhaustive and genetic algorithms confirm the scalability issue for the kind of problems. The processing time is not the only point of discussions as the results of these algorithms can not be interpreted in the same way although the same architecture construction logic is used in both. On one hand, the exhaustive approach helps to reach multiple solutions and have easier interpretation of the results obtained by the end of the process. On the other hand, the genetic algorithm allows to cope better with the size of the problem but propose a unique solution that can be confusing to exploit if it does not respect all the constraints. The designer can not in this case decide if the functional design need to be reviewed.

By studying the variability impact on the design, we confirmed the benefits of addressing this issue in the optimization approach. We proved that the variability, when taken into account in the design process, can have a considerable impact on the final design. The impact of the “optional” functions, particularly if safety requirements are involved, can be considerable and difficult to foresee. For the study-case, the obtained optimal designs where variability is introduced and ignored are considerably disparate although the same inputs are used.

The observations from the study case results allow us to confirm that the objective fixed at the beginning of this thesis is met. We proposed and implemented an approach that is capable to provide an optimal design based on the functional description of the system. It takes into account the safety concepts of the ISO 26262 such as the ASIL, the independancy requirements, the mixed criticality cohabitation requirements. It also implement automotive systems design rules such as the ECUs, the busses networks and compare the designs based on specific cost parameters.

Although the study-case results were satisfying, many improvements can be considered for future works.

The assumptions made on functional model limit the design description as in-
formation about timing constraints, operation modes can not be, for example, taken into account.

The ASIL allocation problem can be solved using faster techniques. In the framework this task can be handled using dedicated solvers that are generic with easy implementation and better performance.

Another interesting research direction continuing from this work is to add timing constraints to the architecture synthesis. The schedulability problem of the tasks and the messages on the network can have an impact on the design. Sometimes the design need to be rethought to satisfy a tight timing constraint. Thus adding these constraints to the approach can lead to more precise and complete results.
Annex A -

A.1 Safety Analysis Techniques

a- Dynamic fault tree (DFT)

As mentioned earlier the traditional FTA provides limited features to capture sequencing or timing constraints on failure propagation. The used gates offered only combinatorial logic possibilities. In [Dugan1992] new gates were introduced allowing representing order of occurrence of the events. They are called dynamic gates. They helped to represent the common cause failures and redundancy strategy. Later in [Cepin2002], more temporal notions were introduced. It allowed handling the modular behavior of the system. Example of new introduced gates: PAND FDEP... The dynamic fault tree could be exploited for qualitative and quantitative analysis. The qualitative analysis is based on the techniques used for traditional FT. But it would be first necessary to translate the new gates to the basic AND, OR gates. DFT could also be analyzed using Monte Carlo simulations [RAO]. The use of Markov chains to analyze the DFT makes it vulnerable to state-space explosion.

b- State event Fault Tree (SEFT)

The main idea behind the SEFT is an attempt to add a notion of states and events to FTA. The states and events are modeled using state machines formalism which is widely used for modeling system and software behavior. The graphical notations are based on the familiar FTA and State charts notations. With this extension, the SEFT is capable of modeling temporal and order notions in addition to cyclic events. To analyze the SEFT, it is necessary to translate them into Deterministic and Stochastic Petri Nets. This allows merging the translated component SEFT models into one flat model and using analysis existing tools [Kaiser2007]. The approach was implemented in the ESSaRel project. The limit is the combinatorial explosion due to size and complexity of the analyzed systems. Fusing all the com-
ponent SEFT to produce one DSPN could lead to difficulties in the analysis of the system.

c- Compositional Failure Propagation approaches

In order to deal with size, reusability limits of the classical tools, CFP approaches were proposed. They mainly focus on characterizing the system by design components. They are often based on annotation of the failure behavior of the components which once interconnected allows the analysis of the system as a whole. CFT Component Fault tree is an approach aiming to provide more flexibility for system analysis when using FT. It proposes a modular approach that would allow analyzing the system in a component and ports concept. It was proposed in [Kaiser 03] in an attempt to deal with the size and complexity of the failure tree when analyzing large scale systems. FTA in a modular way was already used. The independent branches could be analysed separately to cope with the complexity. Yet these branches were often logically separated from each other. Local changes could lead to drastic changes in the branches layout. CFT on the other hand consist of focusing on components and ports when analyzing the system. Later these FT are hierarchically connected to form a global tree. In this case local changes at component level lead rarely to wide changes at system level. The approach has the advantage that the classical algorithm used for analysis of the FT could still be used such as BDD technique. Many later works are based on this approach to manage the growing size and complexity of the analyzed systems.

d- NUSMV

Formal safety analysis platform (FSAP) provides a graphical user interface for the NuSMV-SA model checking and safety analysis engine (BOZZANO, M and Villafiorita, A, 2006). FSAP/NuSMV-SA was created to provide a single environment for model design and safety analysis. The underlying engine is capable of simulating the model, in both nominal functional mode and degraded – fault injected – mode. Standard model checking capabilities, such as property verification and counter example generation, are also provided by the NuSMV engine. Additionally, fault trees can be generated from the model where failure modes have been added to the model. The fault trees can include NOT gates for specifying conditions where events are required not to have happened for failure to occur. Tighter integration of the safety analysis and design modelling allow safety analysis to be performed earlier in the design cycle, enabling less expensive design changes. As is common with model checking systems, FSAP/NuSMV-SA is susceptible to state space explosion. Using simplified models can alleviate the problem in some cases.

A.2 Model Based Safety Approaches

a- Failure Propagation transformation Notation (FPTN)

FPTN stands for PROPAGATION TRANSFORMATION NOTATION. This approach, proposed in is based on a description of the propagation of the failures
through the system using logical equations. These equations specify the logical propagation of the failures through the components from input to output. It takes also into account the impact of the internal error on these outputs. Describing the failure propagation through the different components leads to a characterization of the whole system failure behavior. The described logic allows analyzing the impact of the basic failures modes on the system output and consequently the observed behavior. In this approach the failure and the functional model are not strongly linked. The failure model is considered as floating model. It is constructed over the functional model. The ports must though be synchronized. This made it easy to implement and use with different modeling languages since the safety analysis in this case could be conducted separately from the system functional behavior. FPTN’s main advantage is though its main disadvantage. The separation between the failure model and functional model made the implementation easy yet it made the desynchronization between them more often. Minor changes in the system model could lead to complicated changes on the failure model. Since the process is mainly manual. Also only the previewed failure logic are taken into account during failure modeling.

b- HIP-HOPS

HIP-HOPS stand for Hierarchically Performed Hazard Origin and Propagation Studies. It allows in the same way as in FPTN to synthesize system fault trees from an annotated architectural model. HIP-HOPS propose an enhancement to the FPTN approach. The failures models are no longer floating models since the failure expressions are annotated directly to the components and not to a separate floating failure model. On the other hand, for the annotation HIP-HOPS introduced a new approach inspired from the FMEA called Interface Focused FMEA. A first tool made possible the annotation of the models and the extraction of the FTA afterwards. Further works added functionalities to the tool such as : the generation of the FMEA [parker 2006], definition and analysis of DFT [106, 70]. Further works extended the support of the ISO 26262 through automatic safety levels allocation approach.

c- FPTC

Failure Propagation and Transformation Calculus is an approach proposed in [wallace] closely related to FPTN. The approach is fully textual and differs from FPTN by introducing nominal behavior in the model. It also allows analyzing cyclic dependencies of feedback structure which is a limitation of FPTN. It is in fact a challenge for most of the failure logic modeling technique. On the other hand for the analysis, it does not use the generation of FT. Instead, it uses fix-point calculation technique, an inductive strategy.

d- Altarica

Altarica is a formal modeling language developed to support the safety, reliability analysis. It is a dependability language which enables describing the behavior of
systems when faults occur. Altarica allows the hierarchical modeling of the system through ‘nodes’ which represents components or subsystems. Each node could be considered of an interfaced automaton defined over a number of state variables, flow variables and events. Flow variables allow the interaction with the other nodes. As for the state variables along with the events, allow the automata to describe the behavior of the component. The automata description on the other hand is specified through transitions and assertions. The assertions consist of the constraints that need to be verified by the variables. While the transitions, determine the state of the node. Altarica remains a mainly textual modeling language. Some tools provide though graphical support. Among the software suites supporting Altarica, offering modeling, analysis and simulation of the models, we may cite: Safety Designer (Dassault), ALTATOOLS, SIMFIA (Cassidian). MEC V and ACHECK are on the other hand model checkers.

e- AADL Error Annex

Architecture Analysis and Design Language (AADL) is a domain-specific language for the specification and analysis of real-time embedded software. The AADL syntax is very rich and extensible. The AADL Error Annex for example is an extension to allow the modeling the failure behavior of the components. It is modeled as a stochastic automaton consisting of finite number of error states. The AADL Error Annex in its simplest form could be considered as a failure logic approach since it describes the propagation and deviations of faults through the system. But the failure model is not a floating model since it interacts with the nominal system model. It could be considered as a hybrid approach where both the failure and nominal model are specified. It allows the reuse of error types through error libraries. The error behavior state machines could also be reused or refined using a similar to object approach. It is also possible to model the fault mitigation behavior as well as the fail silent behavior. Currently works are undergoing to unify AADL error Annex and Altarica Dataflow.

f- Lustre FM

Lustre is a synchronous data flow language where the behavior is modeled using a set of equations that assign variables. The language is used to model the behavior of the system using nodes. Each node specifies the input and output parameters allowing he interactions between the nodes. The equations are used to specify how the variables are assigned. The nodes are used to specify the nominal behavior. The language was not conceived to model the failure behavior. In [51] an extension is proposed to allow the modeling of the failure behavior leading to LustreFM. The extension allows specifying failure behavior thanks to mechanisms allowing specifying the internal faults and the error propagation rules, modeling non determinism aspects. The extension allow to translate the extended model into a nominal system model that could be analyzed using existing tools such as NuSMV and PVS.
g- SAML

SAML stands for Safety Analysis Modeling Language. It can be used for both qualitative and quantitative analysis. It allows the combination of discrete probability distributions and non-determinism. The framework allows creating a system model comprised of software control, hardware components, environment and failure mode modeling. SAML is a tool-independent specification language with an automatic model transformation framework to tool-specific languages of the verification engines. The SAML model allows capturing different aspects of the system architecture and behavior. The models are translated afterwards for analysis to correspond to the tools input requirements. The SAML models describe finite state automata which are used to characterize the extended system model. This model specifies the nominal behavior, failure occurrences, its effects and the physical behavior of the surrounding environment. From these models qualitative and quantitative analysis could be conducted using model checking tools such as NuSMV or PRISM. [65] proposed an eclipse tool for modeling and analysis of the SAML models.
Annex B

B.1 Study case optimization model

<table>
<thead>
<tr>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detect_acc_change, Check_valid_ID, Check_steering_lock_status, Vehicle_speed,</td>
</tr>
<tr>
<td>Vehicle_mvt_state, Check_IG_state, Unlock_ESCL, Warning_unable_unlock,</td>
</tr>
<tr>
<td>Lock_ESCL, Warning_unable_lock, ESCL_unlock_conditions_check, ESCL_check_is_unlocked, ESCL_lock_conditions_check, Command_lock,</td>
</tr>
<tr>
<td>Save_ESCL_state, ESCL_check_is_locked, Check_speed_condition_for_lock,</td>
</tr>
<tr>
<td>Detect_user_presence, Detect_user_position, Ext_ID_search_strategy, Detect_ID_position1, Detect_ID_position2, Detect_ID_position3, Extract_ID, Decrypt_ID, Check_ID_rights, Check_access_conditions, Check_car_lock_state, Detect_doors_states_changes, Check_relock_conditions, Unlock_car, Lock_car, Doors_closed, Check_acc_state, Warning_doors_open, Check_lock_conditions, Doors_locked, Post_lock_check, Disable_peps, Check_doors_for_lock, detect_door_states_change, detect_pedal_push, detect_start_push, check_crank_state, check_transmission_state, check_engine_state, start_cranking, stop_cranking, power_down_acc, power_up_acc, power_up_powertrain, power_down_powertrain, ID_management, power_down_acc_conditions_check, power_up_acc_conditions_check, power_up_powertrain_conditions_check, power_down_powertrain_conditions_check, start_cranking_conditions_check, stop_cranking_conditions_check</td>
</tr>
</tbody>
</table>

Table B.1 – Functional architecture to DFG - Functions list

<table>
<thead>
<tr>
<th>Flow</th>
<th>Origin function</th>
<th>Target function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acc_change</td>
<td>Detect_acc_change</td>
<td>ESCL_unlock_conditions_check</td>
</tr>
<tr>
<td>Valid_ID</td>
<td>Check_valid_ID</td>
<td>ESCL_unlock_conditions_check</td>
</tr>
<tr>
<td>Steering_lock_state</td>
<td>Check_steering_lock_status</td>
<td>ESCL_check_is_unlocked</td>
</tr>
<tr>
<td>Steering_lock_state2</td>
<td>Check_steering_lock_status</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Flow</td>
<td>Origin function</td>
<td>Target function</td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Steering_lock_state3</td>
<td>Check_steering_lock_status</td>
<td>ESCL_check_is_locked</td>
</tr>
<tr>
<td>Doors_state_change2</td>
<td>Detect_doors_states_changes</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Vehicle_speed_data</td>
<td>Vehicle_speed</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Vehicle_mvt_state_data</td>
<td>Vehicle_mvt_state</td>
<td>Check_speed_condition_for_loc</td>
</tr>
<tr>
<td>IG_state</td>
<td>Check_IG_state</td>
<td>ESCL_lock_conditions_check</td>
</tr>
<tr>
<td>Unlock_ESCL_request</td>
<td>ESCL_unlock_conditions_check</td>
<td>Unlock_ESCL</td>
</tr>
<tr>
<td>Unlock_ESCL_request2</td>
<td>ESCL_unlock_conditions_check</td>
<td>ESCL_check_is_unlocked</td>
</tr>
<tr>
<td>Warning_unable_unlock_request</td>
<td>ESCL_check_is_unlocked</td>
<td>Warning_unable_unlock</td>
</tr>
<tr>
<td>ESCL_ok_for_lock</td>
<td>ESCL_lock_conditions_check</td>
<td>Command_lock</td>
</tr>
<tr>
<td>ESCL_ok_for_lock2</td>
<td>ESCL_lock_conditions_check</td>
<td>Save_ESCL_state</td>
</tr>
<tr>
<td>ESCL_ok_for_lock3</td>
<td>ESCL_lock_conditions_check</td>
<td>ESCL_check_is_locked</td>
</tr>
<tr>
<td>ESCL_lock_request</td>
<td>Command_lock</td>
<td>Lock_ESCL</td>
</tr>
<tr>
<td>Warning_unable_lock_request</td>
<td>ESCL_check_is_locked</td>
<td>Warning_unable_lock</td>
</tr>
<tr>
<td>ESCL_lock_request_confirmation</td>
<td>Check_speed_condition_for_lock</td>
<td>Unlock_ESCL</td>
</tr>
<tr>
<td>user_present</td>
<td>Detect_user_presence</td>
<td>Ext_ID_search_strategy</td>
</tr>
<tr>
<td>user_position</td>
<td>Detect_user_position</td>
<td>Ext_ID_search_strategy</td>
</tr>
<tr>
<td>user_position2</td>
<td>Detect_user_position</td>
<td>Check_access_conditions</td>
</tr>
<tr>
<td>Lock_state</td>
<td>Check_car_lock_state</td>
<td>Check_access_conditions</td>
</tr>
<tr>
<td>Lock_state2</td>
<td>Check_car_lock_state</td>
<td>Check_relock_conditions</td>
</tr>
<tr>
<td>Locking_source</td>
<td>Check_car_lock_state</td>
<td>Check_access_conditions</td>
</tr>
<tr>
<td>Locking_time</td>
<td>Check_car_lock_state</td>
<td>Check_access_conditions</td>
</tr>
<tr>
<td>Locking_time2</td>
<td>Check_car_lock_state</td>
<td>Check_relock_conditions</td>
</tr>
<tr>
<td>Doors_state_change</td>
<td>Detect_doors_states_changes</td>
<td>Check_relock_conditions</td>
</tr>
<tr>
<td>Access_authorised</td>
<td>Check_access_conditions</td>
<td>Unlock_car</td>
</tr>
<tr>
<td>Relock_car</td>
<td>Check_relock_conditions</td>
<td>Lock_car</td>
</tr>
<tr>
<td>Start_search_position1</td>
<td>Ext_ID_search_strategy</td>
<td>Detect_ID_position1</td>
</tr>
<tr>
<td>Start_search_position2</td>
<td>Ext_ID_search_strategy</td>
<td>Detect_ID_position2</td>
</tr>
<tr>
<td>Start_search_position3</td>
<td>Ext_ID_search_strategy</td>
<td>Detect_ID_position3</td>
</tr>
<tr>
<td>ID_position1</td>
<td>Detect_ID_position1</td>
<td>Extract_ID</td>
</tr>
<tr>
<td>ID_position2</td>
<td>Detect_ID_position2</td>
<td>Extract_ID</td>
</tr>
<tr>
<td>ID_position3</td>
<td>Detect_ID_position3</td>
<td>Extract_ID</td>
</tr>
<tr>
<td>ID_found</td>
<td>Extract_ID</td>
<td>Decrypt_ID</td>
</tr>
<tr>
<td>Identifier</td>
<td>Decrypt_ID</td>
<td>Check_ID_rights</td>
</tr>
<tr>
<td>Authorised_ID</td>
<td>Check_ID_rights</td>
<td>Check_access_conditions</td>
</tr>
<tr>
<td>Unlock_car_request</td>
<td>Check_access_conditions</td>
<td>Unlock_car</td>
</tr>
<tr>
<td>Doors_state</td>
<td>Doors_closed</td>
<td>Check_doors_for_lock</td>
</tr>
<tr>
<td>Acc_state</td>
<td>Check_acc_state</td>
<td>Check_doors_for_lock</td>
</tr>
<tr>
<td>Doors_open_warning</td>
<td>Check_doors_for_lock</td>
<td>Warning_doors_open</td>
</tr>
<tr>
<td>Ok_for_lock</td>
<td>Check_doors_for_lock</td>
<td>Ext_ID_search_strategy</td>
</tr>
<tr>
<td>Lock_car_request</td>
<td>Check_lock_conditions</td>
<td>Lock_car</td>
</tr>
<tr>
<td>Doors_is_locked</td>
<td>Doors_locked</td>
<td>Post_lock_check</td>
</tr>
<tr>
<td>Disable_peps_request</td>
<td>Post_lock_check</td>
<td>Disable_peps</td>
</tr>
<tr>
<td>door_states_change_data</td>
<td>detect_door_states_change</td>
<td>ID_management</td>
</tr>
<tr>
<td>pedal_state</td>
<td>detect_pedal_push</td>
<td>ID_management</td>
</tr>
</tbody>
</table>
### B.2 Study case safety analysis results

<table>
<thead>
<tr>
<th>Flow</th>
<th>Origin function</th>
<th>Target function</th>
</tr>
</thead>
<tbody>
<tr>
<td>pedal_state2</td>
<td>detect_pedal_push</td>
<td>power_up_acc_conditions_check</td>
</tr>
<tr>
<td>pedal_state3</td>
<td>detect_pedal_push</td>
<td>power_down_acc_conditions_check</td>
</tr>
<tr>
<td>pedal_state4</td>
<td>detect_pedal_push</td>
<td>start_cranking_conditions_check</td>
</tr>
<tr>
<td>start_state</td>
<td>detect_start_push</td>
<td>ID_management</td>
</tr>
<tr>
<td>acc_change_detected</td>
<td>Detect_acc_change</td>
<td>ID_management</td>
</tr>
<tr>
<td>steering_lock_status</td>
<td>Check_steering_lock_status</td>
<td>start_cranking_conditions_check</td>
</tr>
<tr>
<td>acc_state</td>
<td>Check_acc_state</td>
<td>power_up_powertrain_conditions_check</td>
</tr>
<tr>
<td>acc_state2</td>
<td>Check_acc_state</td>
<td>power_up_powertrain_conditions_check</td>
</tr>
<tr>
<td>ig_state</td>
<td>Check_IG_state</td>
<td>power_up_acc_conditions_check</td>
</tr>
<tr>
<td>crank_state</td>
<td>check_crank_state</td>
<td>power_down_acc_conditions_check</td>
</tr>
<tr>
<td>ig_state3</td>
<td>Check_IG_state</td>
<td>start_cranking_conditions_check</td>
</tr>
<tr>
<td>gear_position</td>
<td>check_transmission_state</td>
<td>power_up_acc_conditions_check</td>
</tr>
<tr>
<td>gear_position2</td>
<td>check_transmission_state</td>
<td>power_down_acc_conditions_check</td>
</tr>
<tr>
<td>gear_position3</td>
<td>check_transmission_state</td>
<td>start_cranking_conditions_check</td>
</tr>
<tr>
<td>engine_state</td>
<td>check_engine_state</td>
<td>power_up_acc_conditions_check</td>
</tr>
<tr>
<td>engine_state2</td>
<td>check_engine_state</td>
<td>start_cranking_conditions_check</td>
</tr>
<tr>
<td>vehicle_speed</td>
<td>Vehicle_speed</td>
<td>power_down_powertrain_conditions_check</td>
</tr>
<tr>
<td>crank_request</td>
<td>start_cranking_conditions_check</td>
<td>start_cranking</td>
</tr>
<tr>
<td>crank_stop_request</td>
<td>stop_cranking_conditions_check</td>
<td>stop_cranking</td>
</tr>
<tr>
<td>power_up_acc_request</td>
<td>power_up_acc_conditions_check</td>
<td>power_up_acc</td>
</tr>
<tr>
<td>power_down_acc_request</td>
<td>power_down_acc_conditions_check</td>
<td>power_down_acc</td>
</tr>
<tr>
<td>power_up_powertrain_request</td>
<td>power_up_powertrain_conditions_check</td>
<td>power_up_powertrain</td>
</tr>
<tr>
<td>power_down_powertrain_request</td>
<td>power_down_powertrain_conditions_check</td>
<td>power_down_powertrain</td>
</tr>
</tbody>
</table>

Table B.2 – Functional architecture to DFG - Flows list
(a) FTA

Showing 1 to 4 of 4 Cut Sets (Sort by Order)

- Start_cranking.failure (34)
  - power_states_management.control_cranking.Start_cranking.failure (126)
    - ID_management.failure (4)
    - check_acc_state.failure (41)
    - check_engine_state.failure (50)
    - check_ig_state.failure (53)
    - check_transmission_state.failure (56)
    - detect_pedal_push.failure (58)
    - detect_start_push.failure (71)
  - check_acc_state.failure (41)
  - check_engine_state.failure (50)
  - check_ig_state.failure (53)
  - check_transmission_state.failure (56)
  - detect_acc_change.failure (62)
  - detect_door_states_changes.failure (65)
  - detect_pedal_push.failure (68)
  - detect_start_push.failure (71)

(b) MCS

Figure B.1 – SG2
B.2. STUDY CASE SAFETY ANALYSIS RESULTS

(a) FT

(b) MCS

Figure B.2 – SG4
FaultTrees

(a) FT

(b) MCS

Figure B.3 – SG5
B.2. STUDY CASE SAFETY ANALYSIS RESULTS

(a) FT

FaultTrees

- Badvalue-power_down_powertrain.Out1
  - Badvalue-power_down_powertrain.Out1
  - Badvalue-power_down_powertrain.Out1 (84)
    - power_down_powertrain.failure (81)
    - power_states_management.control outcome.power_down_powertrain.failure (158)
  - AND [Badvalue-power_states_management.control lg.power_down_powertrain.Out1] (162)
    - detect_start_push.failure (71)
    - check_transmission_state.failure (56)
    - detect_door_states_changes.failure (65)
    - check_vehicle_speed_ABS.failure (59)

(b) MCS

Showing 1 to 3 of 3 Cut Sets (Sort by Order)

- power_down_powertrain.failure (81)
- power_states_management.control lg.power_down_powertrain.failure (158)
- check_transmission_state.failure (56)
- check_vehicle_speed_ABS.failure (59)
- detect_door_states_changes.failure (65)
- detect_start_push.failure (71)

Figure B.4 – SG6
# Contents

1 Introduction 3

2 Automotive Embedded Systems: Basic Concepts 7
   2.1 Introduction ............................................. 7
   2.2 Embedded Systems .................................. 7
   2.3 System architecture .................................. 9
      2.3.1 Logical Architecture .............................. 10
      2.3.2 Physical architecture ............................ 11
   2.4 System Safety ........................................ 13
      2.4.1 ISO 26262 ........................................ 13
      2.4.2 Functional safety concept ...................... 14
      2.4.3 Automotive Safety Integrity Level (ASIL) .......... 15
      2.4.4 ASIL Reduction .................................. 17
      2.4.5 Mixed Criticality cohabitation .................. 18
   2.5 Objective and Research Questions .................... 18

3 Systems Architecture Evaluation and Optimization 21
   3.1 Introduction .......................................... 21
   3.2 Architecture Optimization ............................ 21
      3.2.1 Optimization targets and constraints .............. 22
      3.2.2 Optimization Techniques .......................... 23
      3.2.3 Optimization algorithms .......................... 26
   3.3 Architecture Evaluation ................................ 26
      3.3.1 Safety Analysis ................................... 27
      3.3.2 Cost Estimation ................................... 29
   3.4 Model Based System Engineering ....................... 31
   3.5 Conclusion ............................................. 32

4 Motivating example and Research Methodology 33
   4.1 Introduction .......................................... 33
   4.2 Study case ............................................ 33
   4.3 State of the art: Optimization approaches applicability 36
      4.3.1 Optimization Model ............................... 37
   4.4 State of the art limits and gap analysis ............... 42
   4.5 Conclusion ............................................. 43
# 5 Safety Driven Automotive Systems Optimization Approach

- **5.1 Introduction** 45
- **5.2 Approach Overview** 45
  - 5.2.1 Key architecture definition choices and constraints 46
  - 5.2.2 Architecture Synthesis Flow 48
- **5.3 Functional Design** 49
  - 5.3.1 Functional architecture and variability 49
  - 5.3.2 Assumptions 50
  - 5.3.3 Model 51
- **5.4 Functional Safety Concept** 51
  - 5.4.1 ASIL allocation challenge 52
  - 5.4.2 ASIL allocation scenarios 52
  - 5.4.3 Independence Constraints 56
- **5.5 Architecture Synthesis** 57
  - 5.5.1 Overview 57
  - 5.5.2 Assumptions 58
  - 5.5.3 Architecture Synthesis process 61
  - 5.5.4 Synthesis process 61
- **5.6 Cost estimation** 77
  - 5.6.1 Cost Estimation Parameters 77
- **5.7 Conclusion** 82

# 6 Approach Implementation and Study Case

- **6.1 Introduction** 83
- **6.2 Tool Implementation** 83
  - 6.2.1 Input Model 83
  - 6.2.2 Optimization Engine 85
- **6.3 Study Case: Results and Discussion** 88
  - 6.3.1 Functional model/Failure model 88
  - 6.3.2 Optimization model 92
  - 6.3.3 Optimization results 93
  - 6.3.4 Variability Impact 98
- **6.4 Conclusion** 100

# 7 Conclusion and future works

- **A Annex A -** 107
  - A.1 Safety Analysis Techniques 107
  - A.2 Model Based Safety Approaches 108

- **B Annex B** 113
  - B.1 Study case optimization model 113
  - B.2 Study case safety analysis results 115
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>ISO 26262 Severity Levels</td>
<td>15</td>
</tr>
<tr>
<td>2.2</td>
<td>ISO 26262 Exposure Levels</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>ISO 26262 Controllability Levels</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>ISO 26262 ASIL determination matrix</td>
<td>16</td>
</tr>
<tr>
<td>4.1</td>
<td>PEPS safety goals</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>ASIL Inherited by ECUs</td>
<td>35</td>
</tr>
<tr>
<td>4.3</td>
<td>ASIL ALgebra</td>
<td>39</td>
</tr>
<tr>
<td>4.4</td>
<td>Allocation Scenarios</td>
<td>40</td>
</tr>
<tr>
<td>5.1</td>
<td>Independence Constraint example</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>three Flow implementation categories</td>
<td>68</td>
</tr>
<tr>
<td>5.3</td>
<td>Illustrative example: Safety constraints</td>
<td>75</td>
</tr>
<tr>
<td>5.4</td>
<td>Illustrative example: Failure mode of host H1</td>
<td>75</td>
</tr>
<tr>
<td>5.5</td>
<td>Illustrative example: Failure mode of host H2</td>
<td>76</td>
</tr>
<tr>
<td>6.1</td>
<td>ESCL mnagement: DFG</td>
<td>92</td>
</tr>
<tr>
<td>6.2</td>
<td>Hosts Inputs</td>
<td>94</td>
</tr>
<tr>
<td>6.3</td>
<td>Host attributes example</td>
<td>95</td>
</tr>
<tr>
<td>6.4</td>
<td>ASIL allocation scenarios</td>
<td>95</td>
</tr>
<tr>
<td>6.5</td>
<td>Independence Constraints in the ESCL management functionalities</td>
<td>95</td>
</tr>
<tr>
<td>6.6</td>
<td>Optimized architecture description</td>
<td>97</td>
</tr>
<tr>
<td>B.1</td>
<td>Functional architecture to DFG - Functions list</td>
<td>113</td>
</tr>
<tr>
<td>B.2</td>
<td>Functional architecture to DFG - Flows list</td>
<td>115</td>
</tr>
</tbody>
</table>
# List of Figures

2.1 Embedded System ................................................. 8  
2.2 ISO 15288 Systems Engineering V-model Process ................. 9  
2.3 Architecture Design Process ...................................... 10  
2.4 SEBOK: Functional Architecture .................................. 11  
2.5 Physical Architecture View ....................................... 12  
2.6 ISO 26262: Safety Lifecycle ...................................... 14  
2.7 ASIL Fault Metrics ................................................. 16  
2.8 ASIL Random Hardware Failure Targets ......................... 16  
2.9 ASIL Decomposition Patterns ..................................... 17  
2.10 CCF & CF concepts ............................................... 18  

3.1 Change cost Vs development time [6] ................................. 22  
3.2 Safety Activities at System level .................................. 28  
3.3 Cost models applicability in a project lifecycle .................. 30  

4.1 PEPS Proposed Solution ............................................. 35  
4.2 PEPS Redundancy introduction ..................................... 36  
4.3 Safety goal decomposition ......................................... 36  
4.4 IFMEA failure model for a sensor .................................. 38  
4.5 Fault Tree for SG5 ............................................... 40  
4.6 Abstract Functional architecture ................................... 41  
4.7 Physical Architecture Optimization Startpoint .................. 41  
4.8 Implementations for the PEPS ECU ................................ 42  

5.1 Architecture Synthesis Flow ....................................... 49  
5.2 ASIL allocation scenarios solving algorithm ...................... 54  
5.3 Illustrative Example FT ............................................. 55  
5.4 Illustrative Example ................................................. 59  
5.5 ISO 11898 Data Bus Implementation Pattern ..................... 60  
5.6 Exhaustive design space exploration ............................... 62  
5.7 Variability implementation algorithm ............................. 64  
5.8 Composite gene .................................................... 66  
5.9 Genetic algorithm based approach Fitness estimation ............ 67  
5.10 Flow hosts identification algorithm ............................... 69  
5.11 Node controller choice algorithm ................................ 71  
5.12 Safety level constraints check algorithm ......................... 73  
5.13 Mixed Safety level constraint check algorithm ................. 74  

125
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.14</td>
<td>Independence constraints check algorithm</td>
<td>76</td>
</tr>
<tr>
<td>6.1</td>
<td>Model import process</td>
<td>84</td>
</tr>
<tr>
<td>6.2</td>
<td>From Simulink To DFG</td>
<td>85</td>
</tr>
<tr>
<td>6.3</td>
<td>Optimization Tool</td>
<td>86</td>
</tr>
<tr>
<td>6.4</td>
<td>Simulink model: Functional Architecture - Part 1</td>
<td>89</td>
</tr>
<tr>
<td>6.5</td>
<td>Simulink model: Functional Architecture - Part 2</td>
<td>90</td>
</tr>
<tr>
<td>6.6</td>
<td>ESCL management: Safety Analysis results</td>
<td>91</td>
</tr>
<tr>
<td>6.7</td>
<td>Optimized architecture description</td>
<td>96</td>
</tr>
<tr>
<td>6.8</td>
<td>Genetic Algorithm results</td>
<td>98</td>
</tr>
<tr>
<td>6.9</td>
<td>Optimization with variability results</td>
<td>99</td>
</tr>
<tr>
<td>B.1</td>
<td>SG2</td>
<td>116</td>
</tr>
<tr>
<td>B.2</td>
<td>SG4</td>
<td>117</td>
</tr>
<tr>
<td>B.3</td>
<td>SG5</td>
<td>118</td>
</tr>
<tr>
<td>B.4</td>
<td>SG6</td>
<td>119</td>
</tr>
</tbody>
</table>


[8] L. Azevedo and D. Parker. Automatic Decomposition of Safety Integrity Levels: Optimization by Tabu Search. ... Safety) of the 32nd ..., 2013. 24, 26


[33] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides. Design patterns: elements of reusable object-oriented software, volume 206. 1995. 25


[57] O. Laires. Architecting and modeling automotive embedded systems. 2005. 32


[60] B. Lewis and E. Colbert. Avionics Avionics Architecture Architecture Description Description Language Language. 32

[61] D. Liaigre. ISO 26262 impact on the state of the art of actual automotive safety concepts. pages 1–9, 2008. 15, 52


[66] A. Long. Beauty And The Beast, Use And Abuse Of The Fault Tree As A Tool. pages 1–10. 28


[78] A. Mehiooui, T. Sara, and J. Babau. A Mixed Integer Linear Programming formulations for optimizing timing performance during the deployment phase in real-time systems design. (33), 2012. 23, 26

[79] J. Montgomery and I. Moser. Parallel Constraint Handling in a Multiobjective Evolutionary Algorithm for the Automotive Deployment Problem. 2010 *Sixth


[98] SEBoK. Guide to the Systems Engineering Body of knowledge. 9, 10, 12, 31


Thèse de Doctorat

Mohamed Slim DHOUIBI

Méthodologie d’optimisation de l’architecture des systèmes embarqués critiques dans l’industrie automobile.

Optimization approach for the critical automotive embedded systems

Résumé

Mots clés
Synthèse d’architecture, Optimization, Systèmes embarqués, ISO 26262.

Abstract
The embedded system design is a challenging task. The engineers are faced with technological, cost, complexity and safety constraints. These constraints have a big impact on the system architecture and consequently on the final cost. We propose in this thesis an approach for system design and architecture optimization driven by safety and cost constraints. It consists of an architecture synthesis approach that takes into account the safety constraints in the ISO 26262 context. It allows, at one hand, to reach a system preliminary architecture by choosing the architecture elements that reduce the overall cost. On the other hand, it leads to a functions mapping that respects the safety constraints related to the integrity levels and to the dependent failures. We use exhaustive and genetic algorithm for the design space exploration. By applying it on an industrial study-case we demonstrate its contribution in reaching compliant design and its capability in reducing the safety constraints costs.

Key Words