Institute of Software Technology
Reliable Software Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis

Variant Management for Technical Architecture of Highly-Automated Driving Systems

Jung-A Yoon

Course of Study:	M.Sc. Information Technology (Specialization: Embedded Systems)
Examiner:	Prof. Dr.-Ing. Steffen Becker
Supervisor:	Prof. Dr.-Ing. Steffen Becker, Dipl.-Ing. Christian Bohne, M.Sc. Andreas Hörtling
Commenced:	June 1, 2019
Completed:	November 30, 2019
CR-Classification:	I.7.2

Abstract

As automotive systems become more and more complex and customized, the variability of and within technical architectures also significantly rises. The rising adoption of highly-automated driving features further adds complexity to the type and number of variants to handle. In addressing this issue, effective variant handling via well-structured feature models plays a crucial role. Although many approaches deal with the variability related challenges in the software level, there exists no proven variant management approach for the technical architecture level. The variant management method for technical architecture shall adequately handle different levels of variants as well as cover the needs and requirements of the users in the model-based system engineering (MBSE) environment. This includes not only modelling the variability existing in the system and visualizing architectural variants and dependencies between features but also ensuring traceability and enabling efficient collaboration between different design levels (i.e. system design, functional architecture design, technical architecture design, subsystem design) throughout the V-Model via a consistent a feature modelling concept and smooth exchange of technical information. In this thesis, we present a variant management method for the technical architecture, which addresses the identified variability related challenges and the needs from the user side.

Contents

List of Figures

List of Tables

List of Acronyms

ACC Adaptive Cruise Control

ACU Airbag Control Unit

AD Automated Driving

AEB Automatic Emergency Braking

AUTOSAR Automotive Open System Architecture

BB Building Block

BDD Block Definition Diagram

CAN Controller Area Network

CBFM Cardinality-Based Feature Modeling

ECU Electronic Control Unit

EPS Electronic Power Steering

ESP Electronic Stability Program

FeatuRSEB Featured Reuse-Driven Software Engineering Business

FODA Feature-Oriented Domain Analysis

FOPLE Feature-Oriented Product Line Engineering

FORE Family-Oriented Requirements Engineering

FORM Feature-Oriented Reuse Method

GP Generative Programming

HUD Head-up Display

HWP Highway Pilot

IBD Internal Block Diagram

RSEB Reuse-Driven Software Engineering Business

TJP Traffic Jam Pilot

WP Work Package

WSS Wheel Speed Sensor

Chapter 1

Introduction

Automotive systems are highly-customized and show high variance by nature. As automotive systems become more and more software-intensive, the complexity further arises with the increase in the number of variants [OPS+17]. This trend adds a significant rise in the complexity of the system architecture, and as a result, variant management has been a known issue. As a solution to handle the variability, feature modelling that models variability of the system has been proposed and applied [KCH+90].

As to the variant handling for the technical architecture, there exist no proven approaches or methods. Most of the proposed concepts focus on the software level or functional architecture. The technical architecture shall describe the types and number of physical elements that comprise the system as well as the E/E architecture - the communication network and the power supply system for the system. Hence, in the technical architecture, numerous variances arise from different levels - vehicle feature design, subsystem design, component design, and E/E architecture design which includes both communication network design and power supply system design.

Also, technical architecture describes the system architecture, which consists of many different subsystems. Each subsystem is designed and developed by various development organizations, and this highly-distributed development environment brings many difficulties in regards to variability. System architects and E/E architects who develop the technical architecture of a system needs access to the technical information of the components which affects the design of the technical architecture, such as the number and types of ports. Plus, the architects need information on subsystem architecture which they have no visibility on the system level. Furthermore, for transparent change management, traceability links between design artefacts, such as from requirement specifications, the functional architecture, and to the technical elements the technical architecture, shall be established.

On top of that, for the highly-automated driving systems, the complexity further increases in the design of technical architecture. Performance requirements for such systems are significantly higher; hence, it leads to a rise in the number of variants with different configurations of Electronic Control Unit (ECU)s and various set of combinations of sensors. Besides, such systems are realized with more cross-domain features which not only broaden the scope of the architectural description but also make the system architecture more complex. Furthermore, strict safety requirements which require redundancy concepts and requirements for in-vehicle connectivity add complexity to the system architecture. Thus, the complexity that arises with the adoption of highly-automated driving systems should be taken into account when considering technical architecture modelling and variant handling of such systems.

Therefore, an adequate variant handling method which can handle the challenges on the technical architecture level is necessary. Approaches on the software level can be comparably easily extended to the functional architecture on the system level since both concentrates on the functional blocks which realize the system functionality. However, for the technical architecture, a different approach optimized for its needs and requirements is required.

In this thesis, we aim to develop a variant management method for the technical architecture, that covers the variant handling challenges in the technical architecture, especially for highly-automated driving systems, and that fits into the model-based system engineering (MBSE) environment. The motivation of this study stems from the variability related challenges that were faced during the technical architecture design in Robert Bosch GmbH. The study is, therefore conducted in the organization, and the final purpose of developing this method is to be applied in the system development process in the organization and help to address variant management related challenges. In this context, it is necessary to take into account the needs of the users and reflections from them. For this, we conducted interviews with practitioners for the roles that we defined as stakeholders for the variant management method of technical architecture. The stakeholders include system architects and E/E architects for highly-automated driving systems, system architects and E/E architects for subsystems, function developers and software developers/architects. The purpose of the stakeholder interviews was to identify use cases and to derive requirements for the variant management method of the technical architecture. These requirements, as well as the evaluation criteria proposed in the literature, are used to evaluate the method.

This thesis provides the following contributions to the challenges mentioned above:

C1 Characterizing the current challenges for the variant management of the technical architecture design in the context of the model-based system development for highly-automated driving systems

C2 Proposing a variant management method and a structure of feature models for the technical architecture of highly-automated driving systems which satisfy stakeholder requirements

C3 Implementation and evaluation of the method

Thesis Structure

The content of this thesis is structured as follows:

Chapter 2 – Foundations presents essential concepts that are necessary to understand for the topic of this thesis,

Chapter 3 – Related Work systematically reviews the relevant literature in the related area of this thesis,

Chapter 4 – Research Methods describes the overall process and research methods taken for this work,

Chapter 5 – Research Results presents the results of this study including the development of the variant management method,

Chapter 6 – Implementation shows the implementation results of the variant management method,

Chapter 7 – Evaluation evaluates the method based on the requirements and the evaluation criteria,

Chapter 8 – Discussion poses discussion points for each work package,

Chapter 9 – Conclusion summarizes the results of the work and presents the starting points for future works.

Chapter 2

Foundations

In this chapter, we present basic concepts which are essential for the understanding of the topic and which in turns form the foundations for this topic. It includes Model-Based System Engineering (MBSE) for automotive systems with a focus on system technical architecture design and modelling of it as well as variant management for system technical architecture.

2.1. Definitions

The list below includes the definitions of the technical terms that we continuously use in this thesis:

Feature A prominent or distinctive part, quality, or characteristic of a system-of-interest, which is visible to users and that stakeholders and end-users can understand [Her00; ISO15; KCH+90].

Feature Model A feature model is used as a communication medium between users and developers. For the user side, the feature model displays the standard features and optional features which they can choose. For the developer side, the feature model specifies what needs to be parameterized in the system architecture and in the model, and how the parameterization shall be done [KCH+90].

User Functions A part of an application that provides facilities for users to perform their tasks. User functions form the elementary units of requirements and specifications [ISO08; ISO17; PHAB12].

Functional Architecture Hierarchical arrangement of functions, subfunctions, functional interfaces and external physical interfaces, which define the execution sequence, conditions for control or data flow, the functional and performance requirements, and the design constraints [ISO17].

Technical Architecture The technical architecture specifies technologies and rules that govern the arrangement, interaction, and interdependency of the elements to ensure that the system-of-interest fulfils a specified set of requirements [Def05; She14]. In this thesis, we use this term to refer to the technical architecture for the vehicle-level system, not for the subsystems which are limited to only certain domains or building blocks of a vehicle.

Besides, there exist high complexity with the technical terms that are used in the environment of the automotive system development. The same term could be used in a different setting with different meanings. Therefore, to prevent confusion, we list the technical terms that we use throughout the thesis and clarify the meaning of them we intend as below:

- Vehicle-level System: refers to the systems which are composed of subsystems in different domains of vehicles (i.e. powertrain, chassis, body, etc.), and the realization of whose functionalities shall be approached from the perspective of the whole vehicles, rather than only a subsystem. For examples, systems such as Cruise Control, Park Assist, Autonomous Emergency Braking belong to vehicle-level systems
- Building Block: a part of a domain of vehicles, which on its functions as an independent system
- Subsystem: a part of a vehicle-level system, which usually realizes one building block of vehicles
- Feature: in this thesis, a feature can be any properties or elements of the system that creates variances of the system and can be added to the feature model
- Vehicle Feature: different lines of product for the vehicle-level system. e.g. Highway Pilot, Highway Assist
- System Variant: a product variant of a vehicle feature, which is created as the result of variant configuration. e.g. Highway Pilot Option A, Highway Pilot Option B, Highway Assist Option A
- Component: the physical hardware elements that comprise the system, such as sensors, processing units, actuators

- Technical Element: in this thesis, it has the same meaning as the component
- Technical Architecture Model: the architecture model for the technical architecture, modelled in architecture modelling tool such as Rhapsody
- Variant Architecture Model: the technical architecture model for a certain system variant

2.2. Model-Based System Engineering (MBSE) for Automotive Systems

In automotive systems engineering, increasing complexity became a known challenge. New features such as automated driving caused an increase in the complexity due to higher performance and safety requirements. More and more cross-domain functionality come into place, and engineers now need to handle more functional interfaces from different domains. This trend brought up needs for a consistent cross-domain vehicle system architecture, and thus, MBSE approaches have been proposed and applied as a solution. Model-based approaches allow a flexible and seamless system design which efficiently deals with iterative innovation developments in adopting new features [AP10]. More precisely, MBSE has significant benefits for today's distributed development environment and ensures high traceability and improved quality.

2.2.1. System engineering process

As shown in Figure 2.1 [Rob19a], general design steps of generic automotive system engineering processes starts from defining *System Requirements*. Through the *System Design* activity, the *Functional Architecture* is created from *Use Cases* and *System Requirements*. The step of defining the *System Architecture* follows as next, which is supported by the *E/E Architecture Design* activity. Within this activity, the *Functional Architecture*, which describes functional relations of all different system functions, is converted to the *Technical Architecture*. The *Technical Architecture* includes the *E/E Architecture*, which describes the communication network and the power supply system of the system of interest, and the *High-Level Hardware and Software Architecture*. Until the *Functional Architecture*, the system architecture is independent of technical implementation, and from the *Technical Architecture*, the implementation architecture is started to take into account. From the *Technical Architecture*, *Subsystem Design* each for software and hardware is executed with the output of the *Software Architecture* and the *Hardware Architecture*.

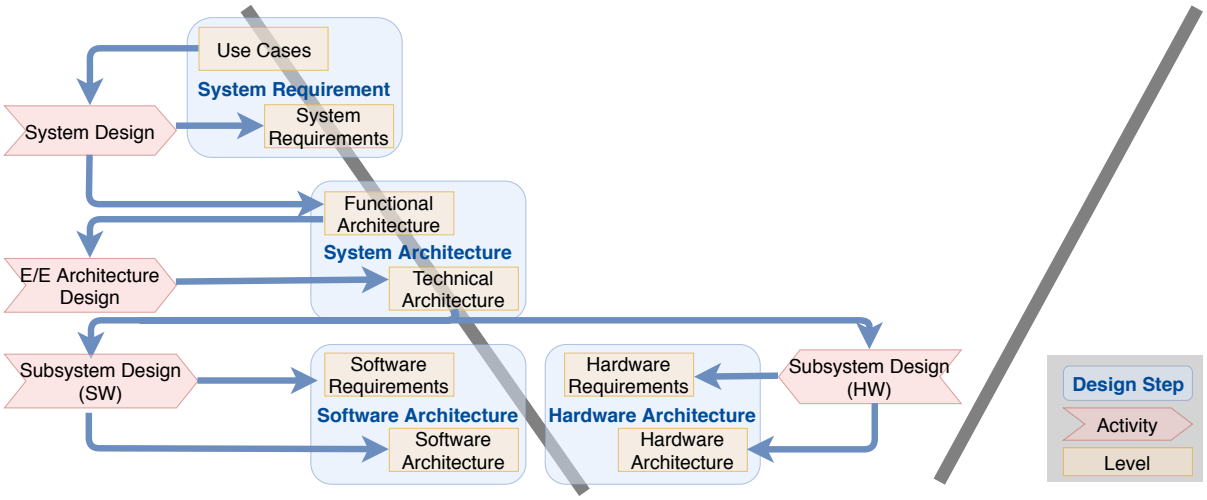

Figure 2.1.: Design Steps for System Engineering [Rob19a]

The introduction of MBSE approaches ensures consistency throughout the overall design steps from system requirements to software and hardware architecture development.

2.2.2. Technical architecture models and point of views

Figure 2.2 [Rob19b] represents the model of E/E system technical architecture for the example system '*System*' in SysML notation. The *Library Model* contains common element blocks which comprise different systems existing in the same vehicles. The *System Model* represents the technical architecture model for the *System*. In *System Model*, the *Library Model* is referenced as read-only, and all its containing element *Blocks* are also referenced together. The *System Model* includes two different views - an external and an internal view. The external view shows the structure of the system; of which components the system consists. In SysML notation, the Block Definition Diagram (BDD) represents the external view. The BDD contains blocks for the common components which reference the original blocks in the *Library Model*. It also includes blocks for the system-specific components which only exist in the corresponding system, but not in other systems within the same vehicle. In the BDD, each element *Block* is connected to the system *Block* through directed composition relations, which represents *Part* in SysML notation. The internal view describes the interconnection of the components, in terms of the communication network and the power supply network. In SysML, Internal Block Diagram (IBD)s represent the communication network and the power supply system of the system. In the IBD, *Parts* coming from directed composition in the BDD are connected with each other through *Connectors* and *Ports*. *Connectors* in communication network represent different communication connections or bus systems such as Flexray

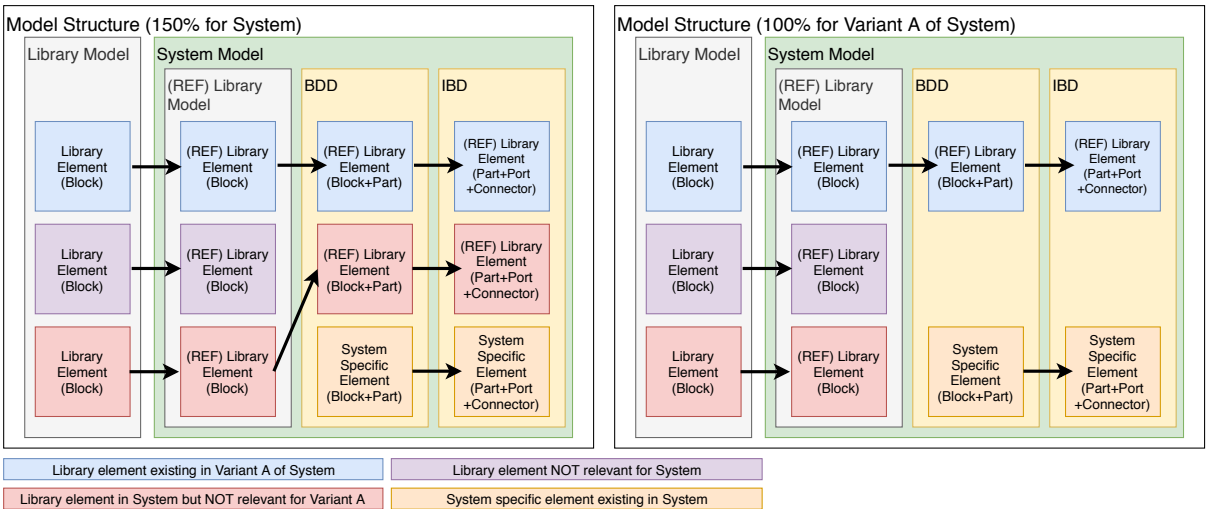

Figure 2.2.: E/E System Technical Architecture Model [Rob19b]

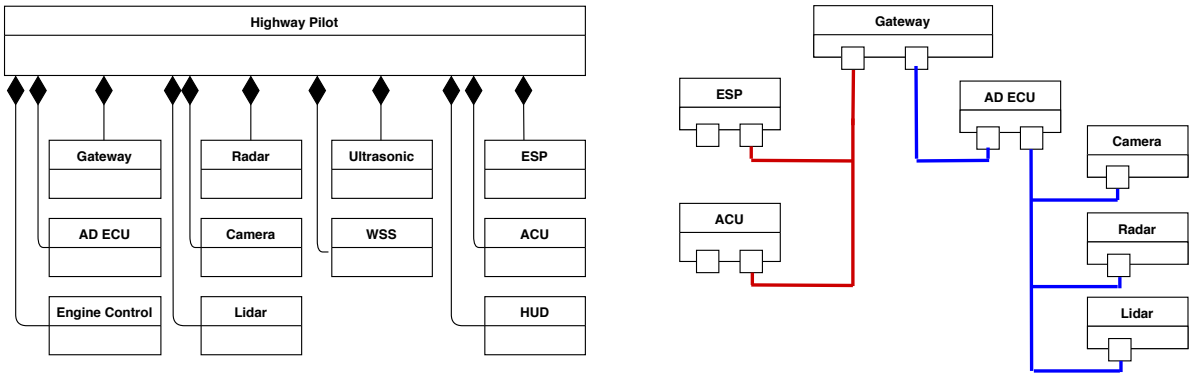

Figure 2.3.: Example of BDD and IBD [Rob19c]

or Controller Area Network (CAN). Figure 2.3 shows a simplified example of BDD and IBD [Rob19c]. In this example, the BDD of the *Highway Pilot* system consists of components including Automated Driving (AD) ECU, Electronic Stability Program (ESP), Airbag Control Unit (ACU), Head-up Display (HUD), as well as Gateway and sensors such as Camera, Radar, Lidar and Wheel Speed Sensor (WSS)s. The IBD represents the communication connections between these components.

It is a common approach of system architects to build 150% system models which contain all system variants that exist in the overall system. With an adequate variant handling method, a variant transformation automatically creates a 100% system model which is specific to one system variant.

2.3. Variant Management in Automotive System Engineering

In the automotive system engineering context, variability is spread out and occurs throughout the whole design steps of V-Model, which Figure 2.1 partly shows. From the system requirements down to the software and hardware architecture design and implementations, different levels and kinds of variants exist. An adequate variant management method that encompasses this variability dispersed in different design steps, documents, models, or any other forms shall effectively handle the variability.

2.3.1. Variant Management for System Technical Architecture Design

The variant management activities considered in this thesis are intended to include the following aspects [OPS+17; Rob19b; Rob19d]:

- to show explicit information about the system variability,
- to represent interdependencies between and the hierarchy of features,
- to derive valid configurations which comprise each variant,
- to ensure traceability for design steps of V-Model,
- to ensure accessibility of product information

This highlights the benefits of variant management which include [OPS+17; Rob19b; Rob19d]:

- Explicit representation of product variability and resulting variants
- Handling of increased complexity in different levels of variants
- Improved traceability and change management for the evolution of product features
- Improvement in product quality and development time
- Increased accessibility of information
- Reduction of local solutions
- Increased efficiency of work especially for a distributed development environment
- Consistency of contents along the V-Model

With the variant management activities listed above, proper tool support further contributes to the system development process. For instance, it enables automatic creation of a variant-specific architecture model that corresponds to a specific variant, and essential information such as which system variants are meant for which customer projects can be displayed.

Figure 2.4 shows the design process for technical architecture design in the model-based system engineering. Firstly, for the *Technical Architecture Design* step, System architects and E/E architects take necessary inputs, requirement specifications and component information. They design the technical architecture based on the input information and produce a model for it. This model is the 150% model which includes all existing technical solutions for the system. The next design step, *System Variant Configuration*, takes this model as an input and configures different system variants based on the needs of each system variant. The 150% technical architecture model contains all variability of the technical solutions which satisfy the requirements. The architects select specific technical solutions for each system variant and perform a variant transformation to produce 100% variant architecture models which correspond to each system variant.

Both these design steps of the technical architecture require different variant management activities. In designing the technical architecture, the primary role of variant management is to gather the necessary information of the components and make it available for the system and E/E architects. The difficulty for system architects and E/E architects in designing the technical architecture is that they have to perform the acquisition of the component information manually. The necessary information of components which affect the variability of the architecture design, such as the product generation or the number and type of available ports for the communication connections, are not readily accessible for them. Therefore, they have to check with the component developers to receive this information manually. Variant management can help on this by having this information about variability modelled in the feature model of the component. System architects and E/E architects then only need to import these feature models and refer to the information about variability for the system technical architecture design. The so-called *Blackbox Concept* can realize this. In the system level, the architects model the part for the subsystem or component as 'blackbox' which they have no visibility on the architectural structure. They then fill these blackboxes with the feature models provided by product owners, which they import into the configuration space of the system.

Another variant management activity for the technical architecture design step is to ensure traceability. The technical elements existing in the technical architecture model are the realization of the technical solutions, which are derived to satisfy the requirements. If we have a different set of technical solutions which meet the same requirement, then we have variability in the technical solutions. The feature model should capture

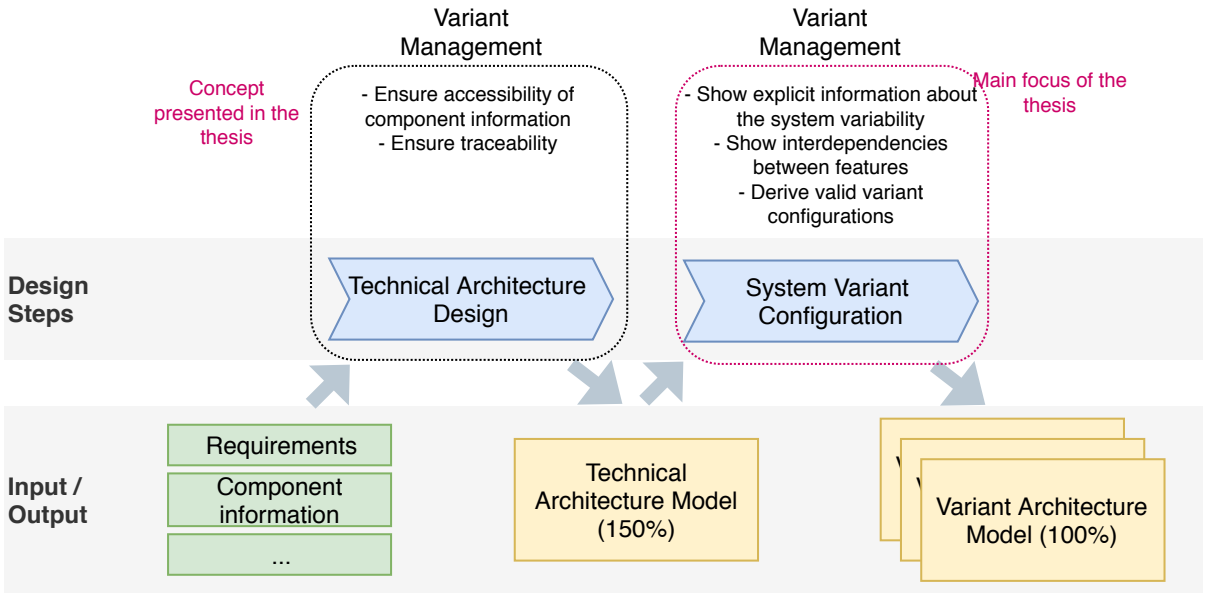

Figure 2.4.: Technical Architecture Design Process and Variant Management Activities

this variability. Through the feature model, we can establish traceability links from requirements to the technical architecture.

For the system variant configuration, the variant management activities concern more about the detailed structure of the feature model. The feature model shall describe the explicit information about the system variability, and it shall show the relations, hierarchy and interdependencies of features. With the proper modelling of the variability information and interdependencies between features in the feature model, the architects can use the feature models to derive the valid combinations of features, which form each system variant.

In this thesis, we mainly focus on the variant management activities for the system variant configuration steps, as it involves fundamental studies of the structure of feature models. Ensuring traceability is also a crucial factor of the variant management for the technical architecture; therefore, we also considered it in developing the method. Also, ensuring accessibility of component information can be solved by the blackbox concept. Hence, in this thesis, we present a concept and ideas to realize it. However, due to the limitation of time, the actual implementation of these concepts is left as future work.

2.3.2. Levels of Variants

When it comes to the variant management for the technical architecture, the issue of having to handle different levels of variants come into place. There exist different levels

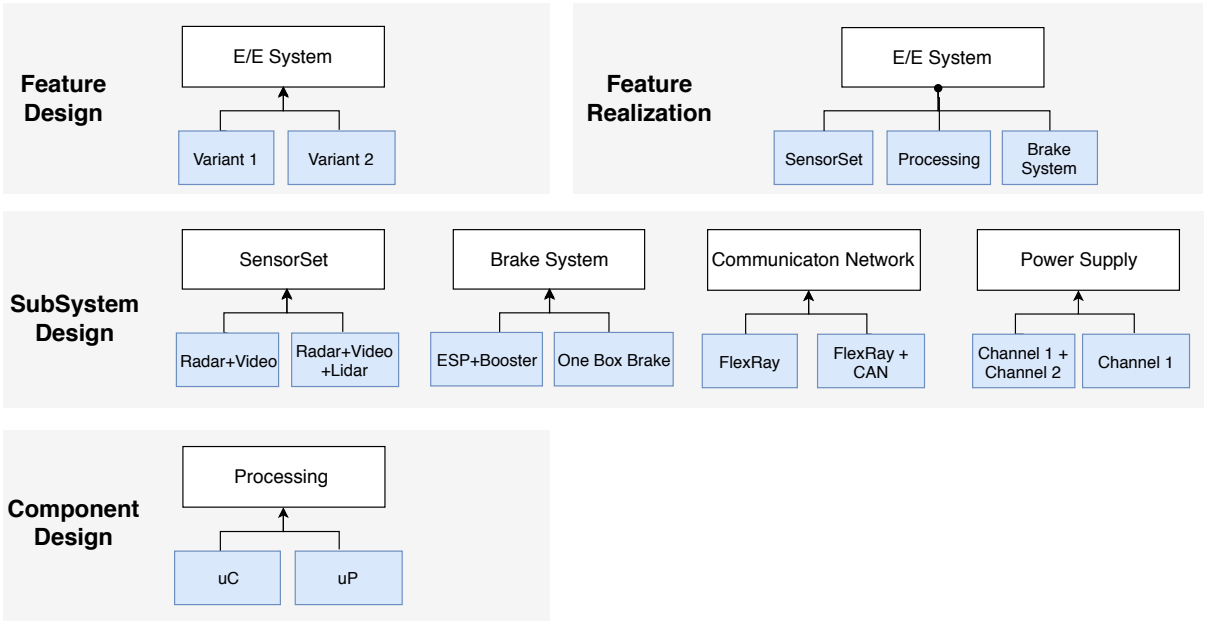

Figure 2.5.: Levels of Variants [Rob19a]

of variants to handle, as shown in Figure 2.5. On the top level of (Vehicle) *Feature Design*, the system *E/E System* can either be designed with only basic mandatory features which correspond to the *Variant 2* or with more advanced functions *Variant 1* which forms another product line variant. The *Feature Realization* shows how the system is composed of, with which different subsystems. The architects can design each subsystem with different design options, which the *SubSystem Design* level shows. In this example, the subsystem *SensorSet* for the *E/E System* can be designed either with only *Radar* and *Video* sensors or with *Lidar* on top of these. The subsystem *Brake System* can be designed with the combination of *ESP* and *Brake Booster* or as *One Box Brake*. Furthermore, for the component like ECUs, different configuration can exist, and this corresponds to the level of *Component Design*. All these levels of variants are dependent on each other and shall be properly handled together.

One critical point of the variant management is that it shall only show where the variability exists. In other words, it focuses on describing and representing variation points in the system, rather than showing the complete architecture of the system, which would be the job of the architecture model.

On top of that, for the highly-automated driving systems, the complexity further increases due to the significantly high-performance requirements. This strict requirement adds more variants, such as having an additional fail-operational ECU or adding more sensors with higher detection capability to the sensorset. Also, such systems require more cross-domain features, and this broadens the scope of architecture description of a system and

makes the system architecture more complex. Lastly, strict safety requirements which require redundancy concepts, and higher requirements for in-vehicle connectivity add further complexity to the system architecture. Therefore, the complexity that arises with the adoption of highly-automated driving systems should be taken into account when considering architecture modelling and variant handling of such systems.

Chapter 3

Related Work

In this chapter, we systematically review the relevant literature in the related area. Systematic literature reviews are conducted to identify a complete and comprehensive picture of the existing evidence [WRH+12]. The contents of the literature review consist of system modelling through meta-models, and variant management and feature modelling. The detailed process of the review is presented in Appendix A – Systematic Literature Review Process.

3.1. System Modelling through Meta-models

Meta-models represent abstract modelling concepts which are utilized to build an integrated system models [RBBS02]. A system model includes all necessary details about the functional logic, the distributed network of ECUs, sensors, actuators and also the environment [RBBS02]. The meta-model structures these details and describes the interrelationships [RBBS02].

In software engineering, meta-modelling is a well-known concept. In the automotive software context, there exists the Automotive Open System Architecture (AUTOSAR) meta-model. AUTOSAR is widely accepted as an automotive industry standard to facilitate the development and integration of software components from different vendors [SBC+13]. It defines meta-model structures with several templates which can be used for the development of automotive software in standardized format [AUT17b]. The AUTOSAR meta-model is the most representative example of domain-specific meta-models used in the industry. After its introduction to the automotive industry, it became the standardized meta-model for the development of automotive software systems [DSTH14]. Figure 3.1 depicts a simple example of the usage of AUTOSAR meta-model, which was used to map software components to different ECU instances.

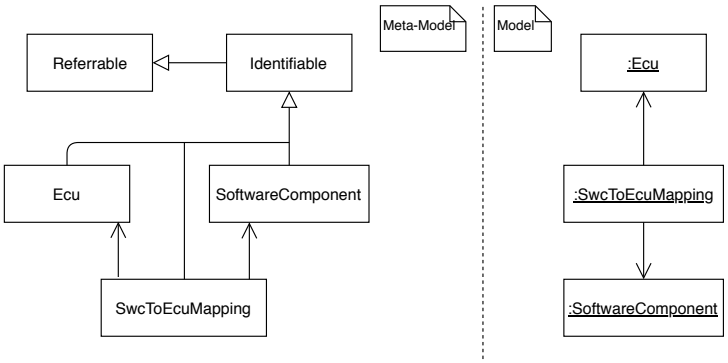

Figure 3.1.: Example of the Usage of AUTOSAR Meta-model [DSTH14]

However, as specified in the goal of AUTOSAR standards, the AUTOSAR meta-model deals with software components mapped to and realized by each ECU instance in the software level, rather than the system architecture over different domains of a vehicle described on the system level. For the functional architecture on the system level, the AUTOSAR Meta-model could be extended since it shares the nature of describing the system as a set of function blocks. However, in terms of technical architectures, there is no proven approach or guidelines yet to design such domain-specific meta-models for automotive systems. As the automotive industry started to adopt MBSE approaches, developing adequate system models based on meta-models became a crucial step. Having a standardized structure of meta-models on the system level is much helpful for the description of complex system architectures, just as the AUTOSAR meta-model is for the software level.

3.2. Variant Management and Feature Modelling

Variant management or handling variability has been an essential challenge in the development of automotive systems. The adoption of new features and requirements for providing customized products are raising the number of possible variants of a system to handle. According to O. Oliinyk et al., a vehicle can typically have around 80 electronic features that can be selected by customers. This number sums up to the fact that there exist 2^{80} possible variants for the vehicle, without considering the validity of the feature combinations [OPS+17]. Variant management provides solutions by offering methods to handle the introduction, use, and evolution of variability to a system [SD07]. Constructing an adequate and efficient variability model is vital since variability models are used in deriving concrete products by product line users [BSL+13].

In the last decades, numerous variability modelling techniques were proposed. These can be grouped into four different families: feature-based, use case-based, UML classes-

based, goals and aspects-based [DS06]. Among these, the feature-based approach is most widely accepted and applied because using features provides the most effective mean to represent the variability [DS06]. Kang et al. explains the major advantage of using features as "Features are essential abstractions that both customers and developers understand [KLD02]". Therefore, the majority of proposed approaches are feature-based. Approaches such as Feature-Oriented Domain Analysis (FODA) [KCH+90], Feature-Oriented Reuse Method (FORM) [KKL+98], Feature-Oriented Product Line Engineering (FOPLE) [KLD02], Generative Programming (GP) [Cza98], Family-Oriented Requirements Engineering (FORE) [Str02], Featured Reuse-Driven Software Engineering Business (FeatuRSEB) [GFd98], Cardinality-Based Feature Modeling (CBFM) [CHE05] are feature-based. Czarnecki et al. defines feature modelling as "the activity of modelling the common and the variable properties of concepts and their interdependencies [Cza98]". Feature modelling has served as the key solution to tackle variant handling issues especially.

However, the most feature modelling methods differ in terminology and process, and this leads to the non-existence of standard feature modelling notations [DS06]. Besides, although many approaches are tested and implemented on different tools, there exist no standard tools. That is why the first challenge when beginning with variant handling is to choose a suitable method [DS06].

3.2.1. Requirements to feature modelling notations

To aid this decision, Djebbi et al. proposed several requirements to feature modelling notations as follows [DS06]:

1. Readability: The notation should visualize the common and variable parts of the system graphically.
    a) Clearness: The graphical arrangement of the elements in the diagram should be clear to comprehend.
    b) Minimality: Each concept in the diagram should be represented only one and once without duplication.
2. Simple and Expressive: The diagram should be expressive by adequately representing the user's needs. It should be simple by containing the minimum necessary number of objects in the diagram.
3. Type distinction: The types of variability (i.e. mandatory, optional, etc.) should be explicitly distinguished.

4. Properties: The properties (i.e. binding times, justification of the variability, etc.) of variation points should be able to be specified.
5. Dependencies: Dependencies between features should be represented.
6. Evolution: The evolution of features which causes models to evolve should be supported.
7. Adaptable: The notation should be adaptable to the company's specific needs.
8. Scalable: The notation should be scalable to model large-scale systems.
9. Supported: The notation should be supported by a feature modelling tool and be integrated into existing toolchains.
10. Unified: The notation should be unified into the entire product line development cycle.
11. Standardizeable: The notation should be standardizable, providing a precise semantics and avoiding conflicting interpretations.

Among these, Djebbi et al. suggested the five most crucial requirements for industrial settings: *Type distinction, Dependencies, Standardizeability, Simplicity and Expressiveness* and *Readability*. For a company, the two most critical criteria for selecting a variability modelling notations are modelling requirements and architectural variabilities along with describing dependencies [DS06]. *Type distinction* is crucial because the types of features (i.e. mandatory, optional, alternative) can be taken into account with cost objectives in the value analysis for the feasibility phase of projects [DS06]. Furthermore, dependencies between requirement variabilities and technical and architectural variability is a central theme in the process of negotiation between a supplier and customers [DS06].

3.2.2. Feature modelling approaches

We summarize the most representative feature modelling concepts and approaches that were identified in the literature review.

FODA FODA [KCH+90] is the most referenced feature modelling approach which was introduced in 1990. Many of later developed techniques are based on FODA. It aimed to capture commonalities and variability at requirements level [OPS+17]. FODA uses features which are properties of a system that have a direct impact on the end-user [Cza98]. The FODA features are described as mandatory or alternative or optional. These features are structured in a tree-style feature diagram. The interdependencies between features are also included as two types of composition rules: *require* and

mutually-exclusive-with rule. However, FODA lacks expressiveness in terms of modelling relations between variants and several extensions were later included as a consequence [OPS+17].

FORM FORM [KKL+98] is an extension of FODA developed in 1998 to incorporate analysis and design issues from a marketing perspective [KLD02]. It extended the product line development process into an asset development process in which product line analysis such as marketing, product plan development, refinement, feature modelling and requirement analysis are performed [KLD02]. Based on the result of this analysis, product development is executed.

FOPLE FOPLE [KLD02] was later introduced in 2002 as a refinement of FODA and FORM. In FOPLE, four different viewpoints which represent different stakeholders' views were introduced [DS06; KLD02]. These views specify features which are classified to corresponding types: capability features (service, operations, nonfunctional characteristics), domain technology features (domain method, standards), operating environment features (hardware, software) and implementation techniques features (design decision, communications) [Rie03]. However, Riebisch et al. assess that FOPLE lacks appropriate definition concerning a clear distinction between the views and the aims and argues that the advantages of having these views are not clear [Rie03].

Generative Programming Generative Programming [Cza98] uses feature diagrams which are extended from FODA with OR-features describing 'one or more' decomposition relations between the parent feature and its sub-features [DS06]. It also advanced FODA by providing a definition of a graphical representation of feature dependencies [OPS+17]. On top of that, Generative Programming allows programs to be automatically generated from precise specifications [DS06]. Czarnecki et al. define the process of feature modelling as following steps [Cza98]:

1. Record similarities between instances to identify common features
2. Record differences between instances to identify variable features
3. Organize features in feature diagrams which classify and group features into hierarchy and characteristics of features (i.e. mandatory, alternative, optional, OR)
4. Analyze feature combinations and interactions by studying feature composition rules (i.e. mutual-exclusion constraints, requires constraints, etc.)

5. Record all additional information regarding features (i.e. short semantic descriptions, rationale, stakeholders, client programs related to each feature, constraints, etc.)

These steps should be performed in a continuous and iterative way as features evolve over time [Cza98].

FeatuRSEB FeatuRSEB [GFd98] combines the methods of FODA and Reuse-Driven Software Engineering Business (RSEB). In FeatuRSEB, UML-like notation is used to construct feature diagrams and explicit representations of variants, feature constraints and dependencies [OPS+17]. Griss et al. clarify an important principle of features that "Not everything that could be a feature should be a feature. Feature description needs to be robust and expressive. Features are used primarily to discriminate between choices, not to describe functionality in great detail; such details are left to the use case or object models [GFd98]". They further elaborate on the difference between use case models and feature models: "A use case model captures the system requirements from the user perspective whereas the feature model organizes requirements from the reuser perspective based on commonality, variability, and dependency analysis [GFd98]".

FORE FORE [Str02] extends feature diagrams with UML multiplicities notations which describe the minimum and the maximum number of features to be chosen [DS06]. In addition, in FORE, the feature diagram is represented as a directed acyclic graph whose edges represent the relations between features and circles at the end show the directions [DS06]. The circles display the types of the features, meaning if the circle is filled, it is a mandatory feature, and if not, the feature is optional.

CBFM CBFM [CHE05] extends FODA methods with a mean of modelling the variability solely by the choices in features [SD07]. In CBFM, features represent any functional or non-functional characteristics in the product family, and each feature can have one attribute that specifies a numeric or textual property of the feature [SD07]. CBFM uses feature cardinalities to describe variability. The feature cardinality shows the number of times the feature can occur in one product (e.g., a cardinality of [0..1] means the feature may exist once or not at all in the product) [SD07]. Djebbi et al. performed a comparative analysis of four feature modelling techniques (FOPLE, FeatuRSEB, GP, FORE) based on the requirements to feature modelling notations which are presented in the previous subsection [DS06]. As a result, FORE was proven to be the most suitable notation since it supports most of the major requirements for the industrial settings as well as the other requirements. However, Djebbi et al. discuss none of the methods is adequate for the modelling of variability of embedded systems [DS06]. This is because

the tested methods which were mainly developed in the academia fail to support the physical dimension in the product lines engineering [DS06].

AUTOSAR Feature Model AUTOSAR has been supporting variant handling from its release 4.0 [AUT17b]. This was first done by introducing variation points to AUTOSAR meta-models and defining means to describe what comprises a specific variant. Later in 2013, AUTOSAR introduced the AUTOSAR feature model to express variation points on a higher level and to capture the dependencies between features [AUT17a]. The AUTOSAR feature model describes a variant handling concept specifically on the software level rather than on the system level. This is because, as previously mentioned, the focus of AUTOSAR lies in modelling aspects of ECU software like software components, ECU configuration and communication topology.

Apart from the automotive industry, although the variability issue in large industrial product lines with thousands of features exist, only a few studies discuss feature modelling and none for the technical architecture description level. As to some notable studies, for the aerospace industry, Gaeta et al. present a system modelling method and the adoption of variability modelling in the product line engineering for aerospace systems [GC15]. The method models systems with variability using SysML to satisfy aerospace systems requirements and software development standards [GC15]. In an attempt for this, they divide the model into two parts, a family model which describes the architecture common to all products in the family, and a variant model which describes the design of a single product [GC15]. Gillan et al. discuss the challenges of applying feature modelling in the telecommunications software industry [GKS+07]. As a result, they reported that there exists no unique way in expressing feature models for a telecommunications system which poses challenges arising from the lack of standardized method [GKS+07].

Although there exist numerous feature modelling methods proposed to address variability handling issues both in the academia and industry, most of them focus on the software level - as a part of software product line engineering. On the system level, several papers discuss feature modelling for the functional architecture description. Grönniger et al. present variability management methods for automotive systems, by using views to describe features and different variants, which is based on generative programming [Cza98; GKPR08]. In this approach, feature diagrams based on the Generative Programming [Cza98] are used to model the feature variability of functional architecture level [GKPR08]. The feature diagram contains functional features such as comfort functions, central locking system and navigation system.

However, there exist no proven or standardized feature modelling approaches for the technical architecture description level. The technical architecture is an integral part

of the whole system engineering process, as it represents the physical architecture of the system and is used for the subsystem design. In other words, from the technical architecture level, an actual implementation which includes subsystem design and component design with the hardware and software architecture becomes relevant. A large number of variants which boosts up the system complexity arise from physical elements and properties such as communication connections between components, power supply channels, and their resulting port configurations of each component, hardware and software components, and configuration and calibration parameters. Also, as mentioned before, variability occurs on different levels throughout the design steps of V-Model. Therefore, having an adequate variant management method which covers different levels and design steps is crucial. Variant management approaches on the software level could potentially be extended to the system functional architecture level, since both the software architecture and the functional architecture focus on the functional blocks which realize the system functionalities. However, as to the development of approaches on the software level, scalability is not considered [SD07]. Therefore, the variant management method which starts from the technical architecture level and encompasses the needs of other design steps is necessary.

Furthermore, most of the proposed approaches lack empirical evidence and reflections from industry [OPS+17]. According to Chen L. et al. [CB11], among 91 identified feature modelling approaches, only 26 are evaluated in the industrial setting. Moreover, it is not clear whether these approaches are applied in practice [CB11].

Chapter 4

Research Methods

This thesis aims to develop a variant management method which encompasses the overall process of system engineering for the technical system architecture of highly-automated driving systems. The research in this thesis began with a systematic literature review. The goal of this review was to identify state of the art in the related. The detailed process and result of the literature review was presented in the previous chapter: Chapter 3 – Related Work. The remainder of the work in this thesis consists of the following four Work Package (WP)s:

- **WP1 Use Case Identification and Requirement Derivation** the stakeholders for the technical architecture modelling and variant management were defined first. To identify their needs, we conducted interviews with these stakeholders. From the interviews, we learned about the current context within the organization about variant management of technical architecture. Besides, the primary purpose of the stakeholder interviews was to identify use cases from which we derived the requirements for the variant management method of technical architecture.
  - RQ1: What is the current situation regarding the variant management of technical architecture? Who are the stakeholders?
  - RQ2: What are the use cases of the stakeholders regarding variant management? What requirements can be derived from the use cases?
- **WP2 Prototype Method Development** considering the requirements derived from the user side, we developed a prototype variant management method.
  - RQ3: What kind of variant management method is suitable in order to satisfy the requirements?

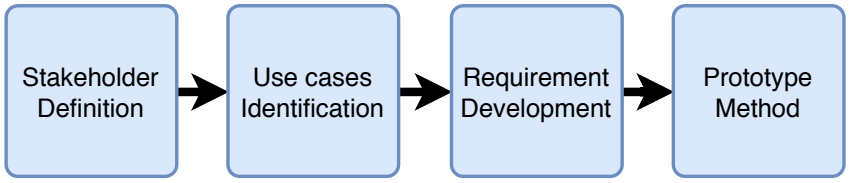

Figure 4.1.: Research Process for WP1 and WP2

- **WP3 Proof of Concept: Implementation and Evaluation** we implemented the prototype method on an example system technical architecture. The example architecture was developed to encompass multiple kinds of variability that exist in automotive systems as well as highly-automated driving systems. The evaluation was conducted based on the requirements developed from the user side and the evaluation criteria proposed from the systematic literature review.
  - RQ4: Does the prototype method satisfy the requirements and evaluation criteria?

Figure 4.1 shows the research process from the stakeholder definition to prototype feature modelling method development, which corresponds to WP1 and WP2.

4.1. Stakeholder Definition

To select appropriate interviewees who can provide meaningful inputs to the technical architecture modelling and variant management in a different point of views, we first defined the levels of abstraction in the system engineering process which are relevant to the system technical architecture design. The (Vehicle) System-Level refers to the development of the vehicle-level system, which requires the perspective of the whole vehicle and which consists of the subsystems spread out onto the whole domains of the vehicle. The example of the vehicle-level system is Adaptive Cruise Control (ACC), Automatic Emergency Braking (AEB), Highway Pilot (HWP) and Traffic Jam Pilot (TJP). On the SubSystem and Component Level, a part of the vehicle-level system is developed. The vehicle-level system functions stand-alone on the vehicle by performing a part of the vehicle control. For the development of the vehicle-level system, the design of system architecture is handed over to the subsystem design parties. The development of each subsystem then starts from there. For example, *Brake system* or *Sensorsets* belong to the Subsystem level. The component here means a physical element which comprises the part of the subsystem, for instance, ESP, Electronic Power Steering (EPS), ACU, video sensors and radar sensors. Although the development of component and subsystem is carried out separately in many organizations, we combined Component level to the Subsystem level, since these are similar from the technical architecture point of view.

Depending on the type of development projects and structure of an organization, only one of them or both levels exist. For each level, there exist different roles as shown in Table 4.1. It also shows the number of stakeholders we interviewed for each role and level as well as their industry experience in years (yr).

Role/Level	SubSystem/Component Level	(Vehicle) System Level
System Architect	2 Stakeholders (4 & 3 yr)	2 Stakeholders (3 & 3 yr)
E/E Architect	2 Stakeholders (5 & 3 yr)	3 Stakeholders (11 & 4 & 1 yr)
Function Developer	-	1 Stakeholder (4 yr)
HW/SW Developer	1 Stakeholder (2 yr)	1 Stakeholder (4 yr)
Product Manager	1 Stakeholder (12 yr)	-

Table 4.1.: Overview of Stakeholders

System Architects and E/E Architects were taken as the most critical stakeholders, as they use the technical architecture model the most as a part of the model-based system engineering. Handling of variants which arise during the engineering process is one of their primary concerns. Therefore, we selected more than one interviewees for these roles to increase the significance of the data. For the other roles, they would refer to the technical architecture model. Still, the importance of the technical architecture model itself to them is lower than for the System Architects and E/E Architects, who both build and use the model in performing their primary tasks. Therefore, the answers from one stakeholder for each role is deemed to be representative to the other level as the principle of the development process for such roles do not significantly differ by the level of abstraction in the system engineering process. Function Developer and HW/SW Developer for the (vehicle) system-level is the same person as he or she is responsible for both roles on this level.

4.2. Stakeholder Interviews

The interview was designed to take about 60 minutes. For a total of 12 stakeholders, we organized an individual interview session. Two interviewers, the author and an E/E architect who is responsible for MBSE process development for the technical architecture, participated in the interviews. During the interview, he or she made sure that the details of the answers are correctly captured and clarified any unclear statements. At the beginning of the interview, we provided the interviewees with a short presentation on the general overview and concept of technical architecture models and variant management through feature modelling. The purpose of this presentation was to help the interviewee properly understand the context. During the interview, we used a

large display in the meeting room to show the template which contains the interview questions. The answers for each question were filled out together in the template to prevent any misunderstanding.

The interview questions include the followings:

1. Role of the stakeholder within the organization (ex: System architect, software developer, etc.)
2. Responsible system (e.g. Brake system, HWP/HWA)
3. Industry experience in the specified role
4. Main tasks in the specified role
5. What process/development steps they perform, for which they need technical architecture models
6. What information should be available in the technical architecture models to perform their roles
7. What kind of variability exist in the area of their tasks
8. If captured at all, where and how this variability is captured today
9. What kind of needs they have, for variant management of technical architecture to perform their roles?
10. Among the typical needs for variant management below [OPS+17]. Which they need for their roles
    - Need to know which variants are meant for which customer projects
    - Need to know what variants can be provided to customers (i.e. which feature combination is valid)
    - Need to have explicit information about product variability (i.e. To see the high-level overview of features, their variants, and interdependencies)
    - Need to work more efficiently
    - Need to improve quality of products
    - Need to increase the accessibility of information
    - Need to handle the increased complexity
    - Clear structuring guidelines for feature models

- Change management: traceability to another feature and explicit interdependency (i.e. impact analysis can be done more transparently, without having to go through each spec, requirements, test cases, configuration data, calibration data and manually change them)

11. What activities they need with the feature model of technical architecture for their roles

After the interview, the interview questionnaire, which was filled together during the interview, was sent to the interviewee. In this way, the interviewees had a chance to review their answers and correct them if necessary.

4.3. Use Case Identification and Requirement Derivation

To identify the use cases for the technical architecture models, we analyzed the interview responses. Since the technical architecture includes variation points existing in the system, the uses cases also include the potential use cases that could further arise in the future with the help of an adequate variant handling method. The identification of the use cases was conducted with several iterations, to spot the missing use cases and to remove redundancy.

After fixing the use cases, we marked the use cases which have relevance to the variant management. These use cases then became the focus of this study. As a next step, we defined the types of variability which shall be covered by these use case. Finally, from these variant management-related use cases, the requirements which need to be satisfied to represent the variability were derived.

During the process of use case identification and requirement derivation, the results were checked with two E/E architects on a weekly basis. In the weekly meeting, each of the use cases identified from the interviews was thoroughly reviewed and confirmed by them. Their roles were to help shape the use cases more concrete and identify redundant use cases. In the requirement derivation phase, they reviewed the content of each requirement and relevance to variant management. Also, prioritizing the requirements, and thus, determining the focus of this thesis were conducted under their supervision.

4.4. Definition of Example Technical Architecture

From the requirements for the variant management of technical architecture, method development was performed by exploring architectural decisions for the feature model,

which can satisfy the requirements of the variant management. For this, a clear problem description was first of all necessary. Thus, we defined an 'example technical architecture' which contains possible variability that exists in automotive systems, furthermore in the highly-automated driving systems.

The purpose of developing the example technical architecture was to capture the variability related issues and to define the architecture, which we can use for the proof of concept by implementing the method on it. In this regards, the architecture shall include different kinds of variability which exist not only in conventional automotive E/E systems but also in highly-automated driving systems. At the same time, the architecture shall be as compact as possible and needs not to include the parts which are redundant in terms of complexity to optimize the implementation time and efforts.

The example technical architecture, as shown in Figure 4.2, describes the system architecture of an example system *AD System 1*. *AD System 1* has two different vehicle features, which corresponds to product line variants - the *Pilot* feature and the *Assist* feature. The *Pilot* feature in the example architecture is intended to represent the L3 automated driving level defined in *SAE J3016 Levels of Driving Automation* [SAE18]. The *Assist Feature* corresponds to the L2 level. According to SAE J3016, the L2 level is driver support features, whereas the L3 level automated driving features. The L2 level requires the drivers to supervise the support features constantly. From the L3 level, the driver is 'not driving' when the automated driving features are engaged [SAE18]. However, when the feature requests, the driver should be ready to step in and take control.

The technical architecture consists of several building blocks of different domains of vehicle. Building blocks that exist in the example architecture are *AD_Processing*, *AD_SensorSet*, *Braking*, *Steering*, and *Localization*. The majority of variability in the technical architecture level arises in these building blocks; therefore, it is sufficient to have these building blocks in the example. The domains and building blocks which were not included in this example architecture (i.e. Body, Powertrain, Infotainment) contain the types of variability which are redundant to the building blocks in the example architecture, hence, additionally including these in the example still do not add complexity. Each component that exists in the system (*AD System 1*) is visualized as a box. The communication connections between components are represented as lines between the boxes. The colour of the line exhibits the types of communication connection. The red line represents *FlexRay*, green is for *Ethernet 1 Gbps*, blue for *Ethernet 100 Mbps*, purple for *LVDS* and gray for *CAN* connections.

Furthermore, each vehicle feature can have different system variants, in this thesis, we have five variants - *Pilot Option A*, *Pilot Option B*, *Pilot Option C*, and *Assist Option A*, *Assist Feature B*. Figure 4.2 shows the architecture of the *Pilot Option A* and *Assist Option A*. Each variant is realized with different design of subsystems, components and E/E

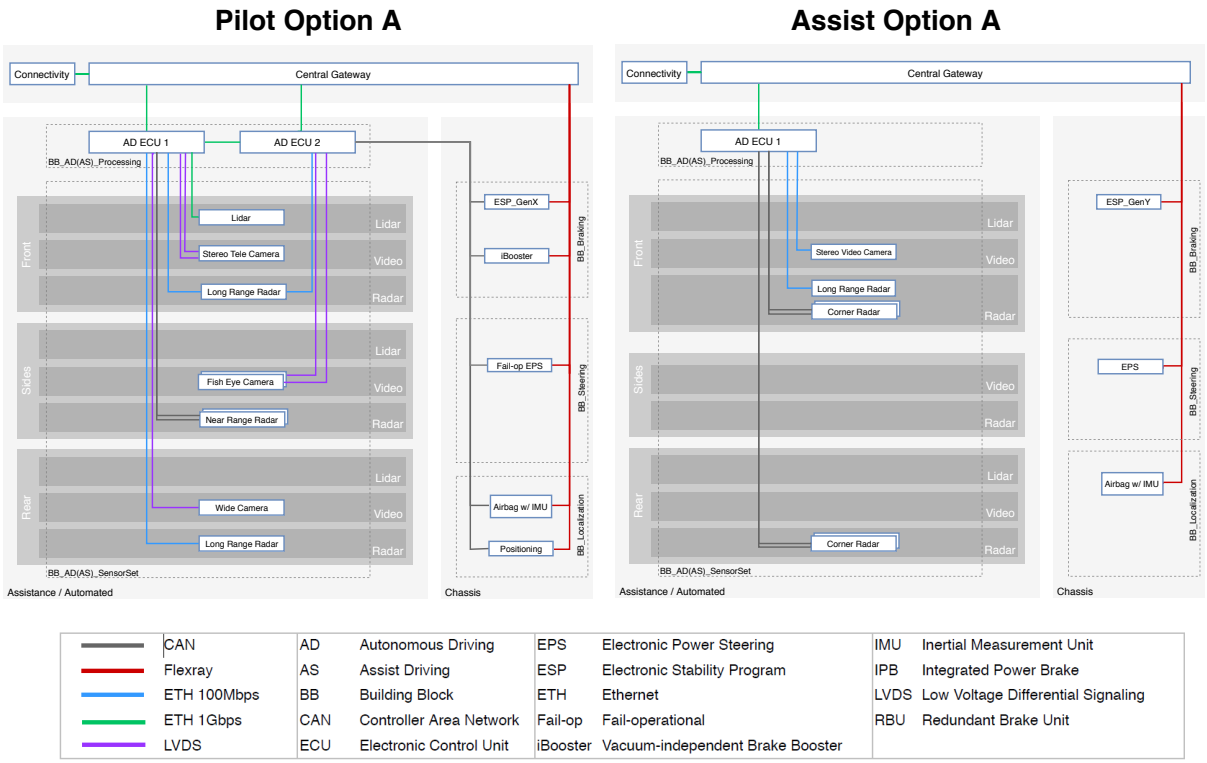

Figure 4.2.: Example Technical Architecture: Pilot Option A and Assist Option A [Rob19e]

architecture. The structure of the example technical architecture for all five system variants are shown in Appendix B – System Variants of Example Technical Architecture.

Table 4.2 summarizes the variations between the *Pilot* feature and *Assist* feature. For example, the front sensor set for the *Pilot Feature* is designed with three different types of sensors - lidar, radar and video sensors. The *Assist* feature, however, is realized with two types of sensors - radar and video sensors.

Subsystems	Pilot Feature	Assist Feature
Front SensorSet	Lidar + Radar + Video	Radar + Video
Processing	AD ECU 1 + AD ECU 2	AD ECU 1
Brake System	w/ redundant actuator	ESP only
Steering System	w/ redundant actuator	EPS only
Localization	IMU + Positioning	IMU
Chassis bus	Flexray + CAN	Flexray

Table 4.2.: Comparison of Pilot Feature and Assist Feature of AD System 1 in the Example Architecture

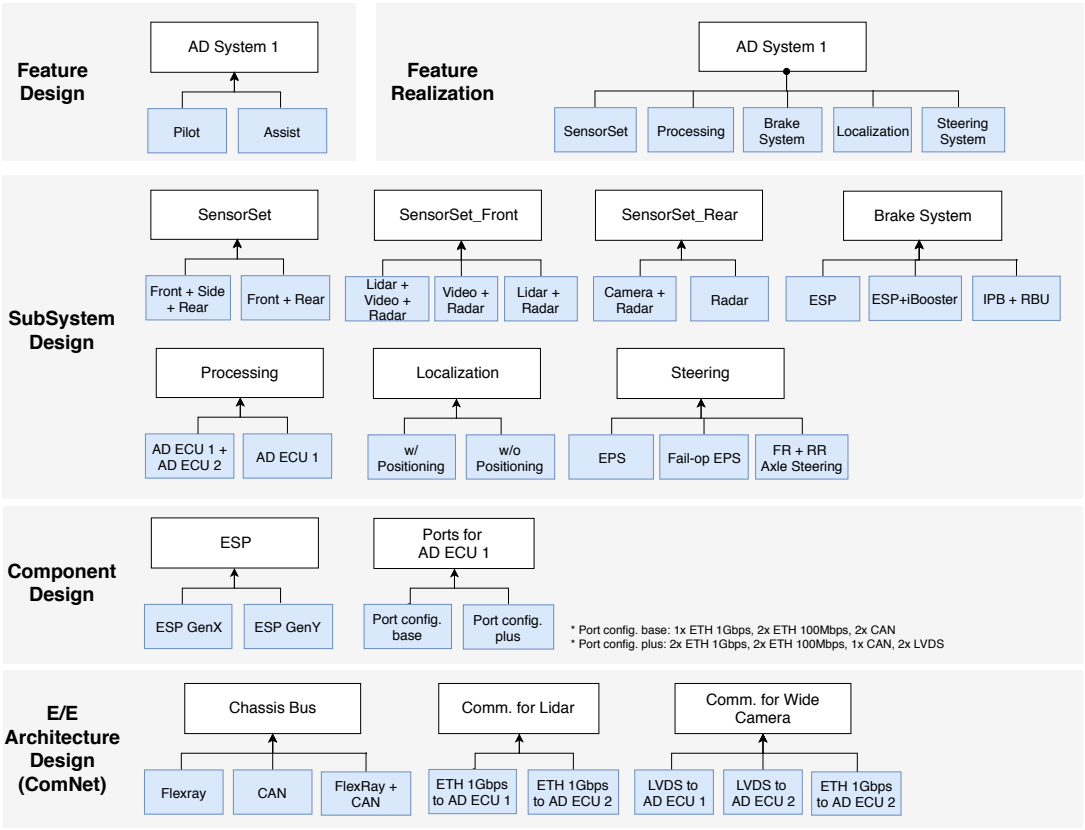

Figure 4.3.: Levels of Variants for Example Technical Architecture

Figure 4.3 shows the levels of variants for the example architecture. In terms of the variability for the E/E architecture, i.e. communication network and power supply system, we first focus on the communication network in this thesis. Handling of the variability of the power supply system is left for the future work, due to the unavailability of the standard modelling method of the power supply system and the limitation of time.

4.5. Variant Management Method Development

In order to develop a method of variant management, we referred to the method proposed in the literature. We identified ten criteria for structuring feature models for software product lines. We studied each of them and selected the ones which can be extended to the vehicle level method. Based on these criteria, we developed a high-level concept for the method and implemented it on a tool in order to prove the concept. In

the course of developing the method, a weekly meeting with the t subsystem-level E/E architects was conducted, to discuss and review the interim result.

It was one of the requirements from the stakeholders to be able to use the standard tool-suite Pure::Variants [Gmbb]. Pure::Variants is already in use in the organization as a variant management tool due to its benefits of having mature modelling techniques and availability of plug-ins into the current architecture modelling tools as well as the requirement management tool used in the organization. Therefore in the development of the method, we followed the Pure::Variants modelling techniques which are based on CONSUL approach [BLPW04; SD07]. In the CONSUL approach, feature models play the core role by representing the problem domain in terms of commonalities and variabilities of the system [BLPW04]. In this sense, the central part of our method development comprised of constructing the structures of feature models which can best address the requirements from the stakeholders.

4.6. Proof of Concept: Implementation and Evaluations

In this work package, we implemented the method on the example technical architecture in a variant management tool-suite Pure::Variants. The purpose of it was to prove the concept of the method. First of all, we modelled the example technical architecture on an architecture modelling tool, i.e. Rhapsody, in SysML notation, with BDDs and IBDs created for each building block. Then we constructed feature models in Pure::Variants for the example architecture. We created necessary features in the corresponding feature models and also modelled the dependency between features by using the relation types available in Pure::Variants. After importing the necessary external feature models to the configuration space in Pure::Variants, we connected this configuration space to the technical architecture model on Rhapsody by using Pure::Variants plug-in to Rhapsody. Then we configured the system variants for each of the five variants of the example architecture. As the last step, we performed the variant transformation on Pure::Variants, and as a result, the 100% architecture models for each system variant were generated. By comparing the generated models with the example architecture, we were able to prove that the method worked correctly on the example technical architecture.

Finally, we evaluated the developed method. We organized a teaching session where we invited stakeholders and presented the method. After that, the evaluation questionnaire was sent to the stakeholders, along with a description document for the method. The evaluation questionnaire contains questions which ask them to evaluate the method based on the requirements for the variant management as well as the evaluation criteria for feature modelling notations.

4.7. Threats to Validity

In the area of empirical software engineering research, four types of validity are recommended to be considered - descriptive validity, theoretical validity, generalizability, and interpretive validity [PG13].

4.7.1. Descriptive validity

Descriptive validity relates to the acquisition of the data, in terms of whether the data is correctly collected. One practitioner whose main role concerns the design of the technical architecture was present in all interviews. The practitioner could clarify the interview questions and answers when necessary helping with the correct and effective recording of the interview answers. During the interview, we used a large screen in the meeting room to display the interview template, which contains all the interview questions. The interviewer and the interviewee filled out the template together. In this way, the interviewee could see how their answers are being recorded and correct it right away if there is any misunderstanding. After the interview, we sent the recorded interview answers to the interviewees, and they were asked to edit or add their answers if necessary.

4.7.2. Theoretical validity

Theoretical validity concerns whether the correct data can be captured as intended. Defining stakeholders to interview was performed by two practitioners in the organization who have been working on technical architecture design in many years. At the beginning of each interview, we asked the interviewees to confirm their roles and the area of work. It was to make sure that we are interviewing the correct practitioners whose roles and experience match with our intention. Before the start of the interview, we explained the background and aim of this thesis, and we also provided a short orientation on the technical architecture models and variant handling activities in order to make sure the interviewees understand the scope of the thesis. For the evaluation of the method, we conducted a 1-hour method presentation, a method teaching session, for the stakeholders, where we explained the concept of the method in detail. The result of the implementation was also shown in this session. Also, after this teaching session, we provided the stakeholders with a hand-out document, which includes a detailed explanation of the method. Only the stakeholders who conducted the training or at least who properly read the hand-out document were invited to participate in the evaluation.

4.7.3. Generalizability

Generalizability is about to what degree we can generalize the results. A threat to generalizability concerns that the scope of the data captured in this work is limited to chassis domain of vehicle. Besides, the example technical architecture only includes the building blocks in chassis domains. However, most of the variability that newly adds complexity to the automotive system come from the adoption of automated driving features, and this mostly belong to the *Processing* or *Sensorset* building blocks. Therefore, variability existing in the chassis domain reaches a good level of representability, but further domains shall be included in future work to improve the validity further. Furthermore, for better generalizability of the interview answers, at least two stakeholders for the same role and the same design level shall be interviewed. In this way, the answers could be cross-checked and achieve higher representability.

Another threat is that the feedback on the usage of the method was not yet considered. Since it is still an early phase introducing a systematic variant management method in the organization, our work focuses on developing the concepts of the method. Further extension of the method shall follow in order to apply the method in the organization. It includes aligning the structure of the *Blackbox* feature model with subsystem design departments, agreeing on the type of data that shall be included in the feature model, defining protocols for the exchange of feature models. Evaluating the method based on the feedback of the usage is therefore left as future work.

4.7.4. Interpretive validity

Interpretive validity deals with whether the conclusions or inferences drawn are reasonable, given the collected data. After the evaluation, the evaluation results are analyzed and discussed with the practitioners in the company. These practitioners have roles in designing technical architecture and have many years of experience in this area. Therefore, they possess a good level of expertise to determine and confirm the direction of the evaluation. The practitioners reviewed the content of the analysis and helped interpret the additional comments provided by the respondents.

Chapter 5

Research Results

In this chapter, we present the results of the study which were obtained by applying the research methods introduced in the previous chapter. The results of the study include - definition of stakeholders, understanding of current context, the use cases for the technical architecture models, the requirements for the variant management method for the technical architecture, and development of the variant management method.

5.1. Profile of Stakeholders

The list below describes the primary roles of the stakeholders identified during the interview:

1. *(Vehicle) System-level / System Architect*
    - Vehicle-level system development and engineering
    - Development of requirements/functional/logical/technical viewpoints for a vehicle-level system
    - Model-based system engineering (MBSE) methodology development for the pre-development phase of vehicle-level system development projects
2. *(Vehicle) System-level / E/E Architect*
    - Development of E/E architecture for vehicle-level
    - MBSE methodology implementation for E/E architecture design on vehicle-level
3. *(Vehicle) System-level / Function and Software Developer*

- Development of software for a model-based controller for multi-actuator usage (in order to generate new vehicle behavior and multi-actuator vehicle control)
- Analysis of activity models and requirements for user-function development
- Realization of software for prototype vehicle

4. *Subsystem-level / System Architect*
   - Subsystem development
   - Development of modelling/variant management methodology (contents, method, process, guideline) for subsystem level
5. *Subsystem-level / E/E Architect*
   - Development of E/E architecture for a certain domain of a vehicle
   - MBSE methodology development for E/E architecture specific to the domain
6. *Subsystem-level / Software Architect*
   - Software architecture development according to software product line approach
   - Development of software solutions (i.e. timing delay, signals, etc.) for the interfaces between components as well as for new functionalities/concepts
7. *Subsystem-level / Product Manager*
   - Generation of product roadmaps
   - Definition of top-level requirements coming from markets
   - Definition of modular sets for the product (determination of sub-component combination, building blocks of the component)

For a system architect on the (vehicle) system-level, the product manager for the system-level provides function description packages which include the descriptions of use cases for the system. From this, the system architect then derives system requirements to the technical layer based on which they design the system architecture. The E/E architect performs requirement engineering for the design of E/E architecture and then derives technical solutions to satisfy the requirements. The E/E architecture design is realized by the design of the communication network and power supply system, including determination of dimensions and property values for the elements in the architecture. (Vehicle) system-level function and software developer analyzes activity models for the system and requirements for user-functions. A function developer determines the functionality of the controller to realize the requirements and user functions on the

vehicle. With this, the activity diagram for the functionalities of vehicle controller which was initially designed with black boxes develops to grey boxes and then finally to white boxes. A software developer derives software solutions to realize the requirements and user functions on the vehicle.

On the subsystem-level, a system architect takes information from the technical architecture and system-level requirements and derives requirement specifications for the subsystem. The information from the technical architecture serves as input for the requirement derivation for the subsystem-level. The subsystem architect then makes design decisions to satisfy the subsystem requirement specifications and derives subsystem architecture, which is then used for the software/hardware/ECU design for the subsystem components. The subsystem-level E/E Architect also develops E/E architecture for a specific domain, such as chassis or powertrain, depending on how the organization is structured. A software architect for a subsystem derives software requirement specifications from functional requirement specifications and requirement specification of the subsystem. Then the software architect develops the software architecture following the software product line approach which satisfies this software requirement specification. The software architect is also responsible for finding software solutions for the interfaces between the components in the subsystem. A product manager analyzes markets and technical demands for the product of interest. Then the product manager considers business cases for the identified trends and demands and decides whether to proceed to development. As a next step, the product manager defines requirements for the product to be developed and brings it into the organization for the start of development.

5.2. Context Characterization

The current status for the variant management and feature modelling within the organization could be understood through the stakeholder interviews. There exist several approaches being developed and studied in subsystem design level for some subsystems. The attempt to introduce feature modelling activities onto their development cycle has already begun on the subsystem level. However, these approaches are confined to deal with variability related problems in the subsystem development mostly for the software design. In other words, the scalability to the higher-level of the system engineering, for example, system design or system architecture design, is not considered. There exist no variant handling method which could encompass the whole levels of system engineering from system design level down to the subsystem development, which could be extended to cover the implementation level as well.

5.3. Use Cases For Technical Architecture Model

The use cases (UC) that were identified through the stakeholder interviews are as followings:

UC1 Ensuring consistency: ensuring the consistency in terms of modelling notations and concepts between domains and between levels (i.e. vehicle/sub-system levels) including interfaces.

UC2 Model-based system architecture design: system architects and E/E architects design system architecture and E/E architecture for the system with MBSE approach.

UC3 Feature engineering: by using domain knowledge, create/determine features that realize the functionality of different system variants

UC4 Visualize technical architecture and its variants: using the model to visualize the technical architecture in order to see high-level system architecture including communication and power supply network and variants of the system architecture

UC5 Show technical information of components and features: using the model to show technical information of components and features explicitly

UC6 Show traceability requirements to technical solutions: traceability from requirements or user-functions to technical solutions in the technical architecture

UC7 Show dependency between features: dependency and interrelations between features

UC8 Definition of interfaces: a consistent interface between different domains (i.e. chassis domain and AD domain)

UC9 Feature-driven selection for each variant: derivation of a valid combination of components/features for each system variant

Use case 1: Ensuring consistency This use case refers to deriving consistent chains of action over defined building block elements across different domains and levels. This use case can be characterized as a general use case which all other use cases need to inherit. For better clarity, we categorize the remaining use cases under three categories - use cases *For system design*, *For using model* and *For further improvement of architecture design process*.

5.3.1. Use cases for system design

The use cases in this category refer to creating and revising the technical architecture model as a part of the system development process. These use cases are mainly used by the system architect and E/E architect, who are the designers of the technical architecture.

Use case 2: Model-based system architecture design As a core part of MBSE process, system architects and E/E architects design the system architecture from the use cases and requirements, which in turn serves as technical solutions that satisfy the requirements. This use case is about designing/developing system architecture as part of the V-model of MBSE process. For the design of system architecture, architects build and use the models. For the design of the technical architecture, a product manager provides function description packages, where use cases are described. Then the system architect derives system requirements to the technical layer. Based on this, the system architect designs the technical architecture. This use case also includes the design of E/E Architecture. An E/E architect first analyzes the use cases, requirements and functionality of the system and maps functionality to the technical architecture and the components existing in the architecture. The design of E/E architecture includes the following activities - designing a bus topology/communication network including designing connectors and communication network connections, building a power supply topology and validating it on safety requirements, dimensioning power supply components. This use case is mainly used by the (vehicle) system-level system architect and the E/E architect both in system and in subsystem-level.

Use case 3: Feature engineering An AD system can have different variants in the vehicle feature design - premium, plus and base variants. Each variant differs in the functional features such as lane change capability or velocity limits. Feature engineering refers to designing system variants with different combinations of functional features. Deriving technical solutions which can realize the functionality of these system variants is the primary activity in the previous use case *Model-based system architecture design*. This use case is used by the system architect and the E/E architect in both levels.

5.3.2. Use cases for using model

The use cases in this group refer to when the users use the available technical architecture model for their needs. These use cases speak more from the user side rather than the creator side.

Use case 4: Visualize technical architecture and its variants This use case is about showing the structure of the technical architecture. It shows what components comprise the system, the communication network and power supply system for the system of interest. Also, for the different variants of the system, variation points which occur in different levels of variants need to be represented in the model. This use case shows further potential usage of the technical architecture model. For instance, it could be used for the comparison of architecture variants, and the subsystem-level users use the model to understand the vehicle-level requirements. For example, the information such as customer requirements, markets, vehicle types, development and production timeline could be understood by observing the model. Similarly, in order to identify stakeholders for a specific subsystem, it can also be used in the way of understanding the interfaces and neighbouring system. Lastly, in some cases, depending on the users' needs, a description of a high-level technical architecture which abstracts unnecessary details is useful. For instance, for a high-level structure review with the management of an organization, or for a reporting purpose, a high-level structure view without detailed technical information is necessary.

This use case is used by the system architect in both levels, E/E architects in both levels and product managers. However, it can be speculated that all stakeholders may need this use case since they all are interested in understanding the overall structure of technical architecture in order to gain a broader view of the system from the vehicle level.

Use case 5: Show technical information of components and features This use case means referring to the technical architecture model in order to retrieve necessary technical information of a specific part of the system. The technical information which shall be available in the technical architecture model includes any information for the elements of the system which stakeholders need. It includes the source and destination of signals, power supply concepts, component properties, port information. This use case was captured mainly to be used by the subsystem-level software developer and subsystem-level system architect. A system architect, for instance, takes information from technical architecture for the system integration tests, and this information is then used as the basis of ECU design as well. Having these kinds of the necessary information in the model also helps improve the accessibility of information.

5.3.3. Use cases for further improvement of architecture design process

This category describes the use cases which are not yet applied in practice but identified as potential use cases. The stakeholders mentioned that they would potentially use the

technical architecture model as specified in these use cases with the help of a variant management method.

Use case 6: Show traceability requirements to technical solutions This use case refers to the definition and representation/visualization of relations between high-level requirements and the solution space. The requirements here include the aspects and contents from safety and redundancy concept for AD systems, such as whether a redundant brake system is needed or fail-operational processing is required. To fulfil these requirements, the system architects design technical solutions, and the technical solutions are realized by the capability of the technical elements that comprise the technical architecture. The relation between these requirements and the technical solutions and elements shall be captured. It can be achieved by establishing links from requirements to technical solutions, and then to the technical elements in the technical architecture in order to ensure the traceability. This link lies not only between requirements and technical architecture but also the design steps between them. That is, the link shall be formed from requirements to the functional architecture, then to the logical architecture which shows the logical elements to realize the functionality, and finally to the technical architecture. Having traceability enables more straightforward and transparent impact analysis.

This use case was identified from the answers of the system-level function and system developer, system architect, E/E architect and the subsystem-level system architect. However, we expect that all the stakeholders would be interested in this use case because it helps better understand the complete picture of the system architecture in terms of showing requirements for each part of the technical solutions.

Use case 7: Show dependency between features This use case concerns describing dependencies, interrelations and hierarchy between features existing in the system. The subsystem-level E/E architect is the main stakeholder for this use case, but all other stakeholders may also have an interest in it. It provides an overview of features in a system, which is useful to understand the relations between features of a particular subsystem and neighbouring parts.

Use case 8: Definition of interfaces This use case refers to defining a consistent interface between vehicle domains. Subsystems are developed in different organizations; hence, consistent interfaces between different domains are necessary, considering the increased number of cross-domain features. This use case was identified from the answer of the system-level function and system developer and subsystem-level system architect. We expect that all system architects and E/E architects would be relevant to this use case as well, because the vehicle-level system and the E/E architecture covers the whole

vehicle domains, having to deal with many cross-domain features. In this sense, ensuring consistent interfaces between the models from different design steps or domains are essential.

Use case 9: Feature-driven selection for each variant The last use case deals with the system variant configuration. A system variant is configured with a combination of features and in order to derive a valid feature combination, the variant composition rules, as well as the interdependencies between features, should be thoroughly considered. In the traditional document-based system development environment, this information is spread out over different development parties, and much of this information resides in the head of the product owner. Thus, it was not only difficult to gather this information but also quite probable to miss out an essential piece of information which leads to an invalid combination of the features. However, if this information is modelled and the users can get this information by simply taking the model, the variant configuration can be done more effectively in a feature-driven way. This use case is the interest of the system-level system architect and E/E architect.

5.3.4. Relevance of Use Cases to Variant Management

The use cases above include both for the technical architecture modelling and variant management for the technical architecture. For the technical architecture model, there exist ongoing activities within the organization to define the standardized modelling method. However, variant management for the technical architecture level has not been explored despite the needs to handle different system variants. Therefore, we deduce that the needs for method development of variant management are higher and thus more urgent, which sets the rationale for the focus of this thesis - variant management for the technical architecture. In this sense, among the use cases listed above, it is reasonable to select the ones that have aspects of variant management and continue for the next step of requirement derivation on them.

Out of the nine use cases, six use cases (Use case 2, 3, 4, 6, 7 and 9) were identified as relevant to variant management.

Use case 2: Model-based system architecture design In designing the technical architecture, the variability occurs when different technical solutions can realize the same functionality. This variability occurs in terms of:

- Components

  - Different type of components: for example, the subsystem *Brake System* can be designed with the combination of *ESP* and *Brake Booster* or with *One Box Brake*. It corresponds to the *Subsystem Design* level, which was shown in Figure 2.5.
  - Different number of the same components: for some systems, the number of the same components may differ in the system variants.
  - Different version of the same products: for instance, the same ECU can be configured with different combinations of the parts within the ECU, such as different communication interfaces or processing cores. This case also covers different product generation of a component. It conforms to the *Component Design* level.

- Communication connections
  - Different communication connections for the same component: for example, *EPS* can be connected to the *Gateway* via *FlexRay* or *CAN*. Also, concerning this, ports usage shall be considered. It occurs when the same port is used for different communication connections in different variants.

- Power supply connections
  - Different power supply channel for the same component: for instance, some components can be connected to dual power supply channels in one variant and only one channel in another variant.

Furthermore, the following additional variability is captured during the interviews, which exist outside of the technical architecture.

- Markets, regions, legislation, customer, customer projects
- Different types of vehicles (e.g. long-haul, vehicles for construction site)
- Timeline for development activities (e.g. series production start)
- Different sources/sinks for signals

Use case 3: Feature engineering The same system can be designed with different system variants with different combinations of features. For example, the highway pilot (HWP) system can have three different system variants, as shown in Table 5.1. It corresponds to the *Feature Design* level.

Variants	Lane change functionality	Velocity supports
HWP base	X	up to 80 kph
HWP plus	O	up to 80 kph
HWP premium	O	up to 120 kph

Table 5.1.: Example variants for Feature Design level

Use case 4: Visualize technical architecture and its variants For the visualization of technical architecture, there exist different types of variants. First of all, different system variants for the same system (ex: HWP Variant 1, HWP Variant 2) shall be shown with the help of variant management. Also, stakeholders need architecture models or views which abstract the details of the technical architecture based on their needs. For instance, some stakeholders mentioned that they need a high-level architecture view which only shows the name and type of the component for the management review. Furthermore, most of the stakeholders, especially in the subsystem level, mentioned that they would like to have an aid for comparing different architectural variants in order to understand vehicle level requirements better. It requires the capability to identify common and varying parts of the architecture.

Moreover, there were needs to display necessary information regarding variability. It includes which variants are offered to which customer as well as the overall developmental timeline for each variant. Further extensions of variant handling capability which enable the automatic display of possible topologies of technical architecture based on the input selection of user-functions were also suggested.

Use case 6: Show traceability requirements to technical solutions In ensuring traceability between different design artefacts, utilizing variant management approach has a significant advantage. Variability occurs throughout the overall system design process; therefore, a feature model can be a useful medium to build traceability between artefacts from different steps of the design process.

As to the technical architecture, traceability exists in the following types:

- Traceability from requirements: a high-level requirement, such as a redundancy requirement, shall have traceability links to the technical elements, which satisfy such requirement.
- Traceability from functional architecture: there exist variability when different technical solutions realize the same function. This aspect shall be examined together with the variant management approach for the functional architecture, or when considering extending the method for the technical architecture to the functional architecture.

- Traceability to subsystem architecture: traceability from a (vehicle) system-level feature to the technical elements in the subsystem-level shall be visible in the feature model. This factor is already covered by expressing the levels of variants in the feature model. At the system design phase, there exist cases when system architects do not have a clear idea of the design of the subsystem which satisfies the requirements. For this case, the so-called blackbox-concept, which enables importing the subsystem feature model from the subsystem owners, with pre-defined and aligned interfaces between the two models.

Use case 7: Show dependency between features There exist dependencies between features in the system. It includes a particular component requires the inclusion of the other components or a particular component conflicts with another component already existing in the system. For example, *Brake Booster requires ESP* or *Chassis _Bus_Flexray conflicts with Chassis_Bus_CAN*. This information shall be modelled in the feature model.

Use case 9: Feature-driven selection for each variant With the types of the features (i.e. mandatory, optional, alternative, OR) and dependencies modelled in the feature models, the configuration of system variants can be performed.

Figure 5.1 shows the use case diagram, which depicts the stakeholders, use cases and the allocations of use cases from the stakeholders. The relevance to the variant management is represented as a stereotype *«VMRelated»* for the corresponding use cases.

Figure 5.2 shows only the variant management-related use cases, for the better readability. For the remaining part of the thesis, we will focus on these use cases.

5.4. Requirements to Variant Management Method of Technical Architecture

From the variant management-related use cases, we derived the following requirements (RQ) for the variant management of technical architecture.

RQ1 Level of variants: it shall show different levels of variants (i.e. system feature design, subsystem, component) of the technical architecture.

RQ2 Show relations between high-level requirements and technical solutions: it shall show relations between high-level requirements and technical solutions.

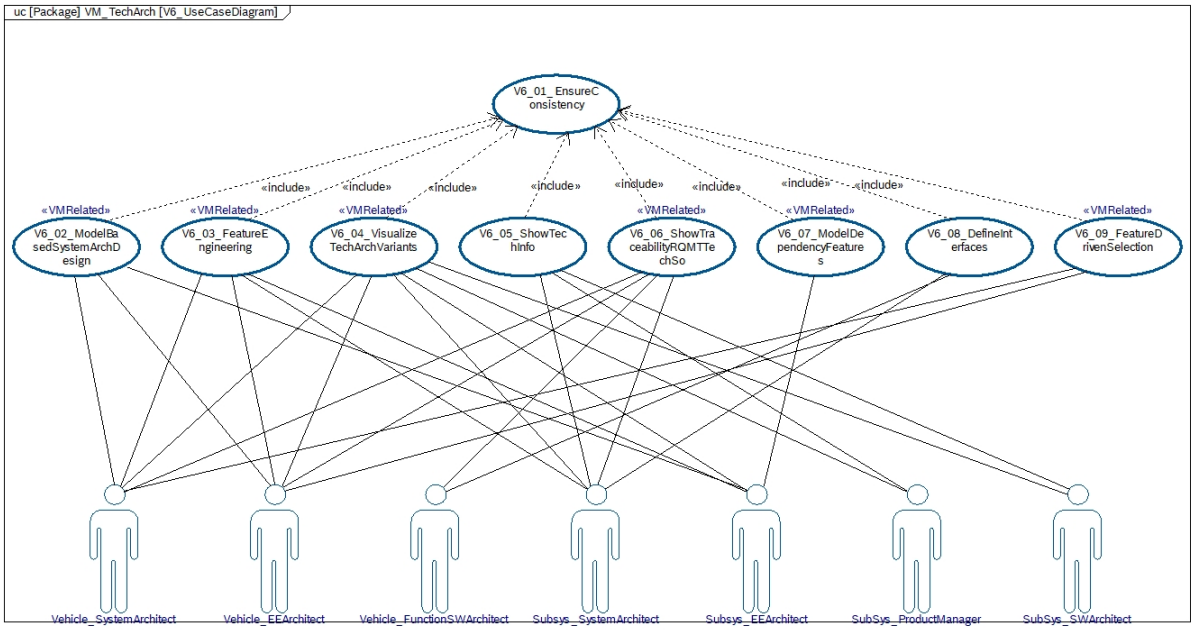

Figure 5.1.: Use Case Diagram

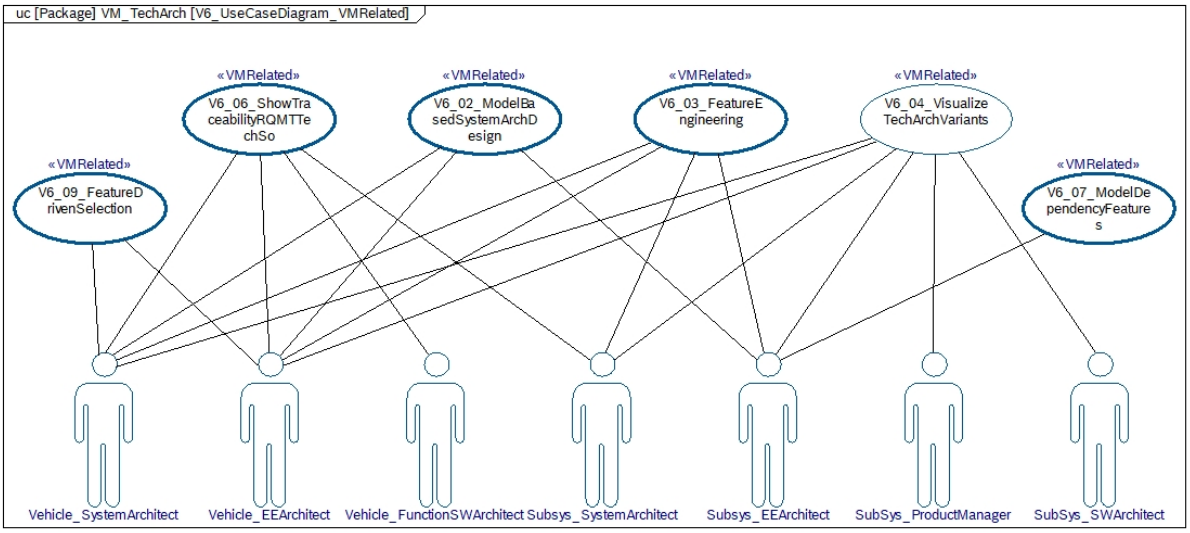

Figure 5.2.: Use Case Diagram for Variant Management Related Use Cases

RQ3 Interdependency of features: it shall model the interdependencies between features. Interdependency here refers to a relation between two features that requires a feature to be revisited if the other feature changes.

RQ4 Ensure traceability via feature model: it shall have traceability links between different design artefacts or viewpoints (e.g. from requirements to technical solutions to the technical architecture).

RQ5 Blackbox concept for the subsystem parts: it shall be able to model a certain part of the feature model as a blackbox and to import a feature model to fill the blackbox, which is created by the part-owner.

RQ6 Automatic generation of variant architecture model: Shall automatically generate 100% architecture model for system variant.

RQ7 Show validity of a feature combination: it shall determine if a certain feature combination is valid or not, based on the interdependency between features and composition rules

RQ8 High-level abstraction view: it shall show a high-level abstraction view of technical architecture for users' needs (e.g. management review).

RQ9 Comparison support: it shall offer supports to compare architectures by showing commonality and differences.

RQ10 Usability: it shall be easy to use.

RQ11 Readability: it shall be easy to read.

Due to the limitation of time and the consideration of priority, RQ8 and RQ9 are left for future work. The usability and readability shall be ensured at the very best in exploring solutions to satisfy the other requirements. Therefore the RQ10 and RQ11 are kept common and general in the overall process of the method development.

5.5. Variant Management Method for Technical Architecture

We categorized the primary requirements of focus based on the four subject matters that need to be solved:

1. Variability in the system: from RQ1, RQ3, RQ6, RQ7
    - Variability in vehicle feature design

- Variability in subsystem design
- Variability in component design
- Variability in E/E architecture design

2. Variability outside of the system: from RQ1, RQ3, RQ6, RQ7
3. Traceability: from RQ2, RQ4
   - from requirements to technical elements in the technical architecture
4. Blackbox concept: from RQ5

By exploring solutions for these subject matters, we developed the concept for the variant management method.

5.5.1. Variability in the system

As discussed previously, there exist different levels of variants in the system. As shown in Figure 5.3, subsystem design level represents the variability in the technical solutions for subsystem design. The component design level is the same way for the variability in the technical solutions for component design, and E/E architecture design level is for the variability in the technical solutions for E/E architecture design. These technical solutions are spread out over different domains of a vehicle and existing in the horizontal direction. On the other hand, the feature design level represents the vehicle feature design, in our example, either Pilot or Assist features. The design of the vehicle feature is performed by choosing particular technical solutions from subsystem design, component design and E/E architecture design levels. It means that the design of a vehicle feature is performed vertically, by vertically incorporating design decisions for the three design levels, which exist horizontally. Due to these two directions perpendicular to each other, we need to separate feature models into two, one for the horizontal direction, the other for the vertical direction.

Figure 5.4 shows the two feature models for each direction - one for vehicle feature design (vertical) and the other for the design of technical solutions in subsystem, components and E/E architecture design (horizontal).

The feature model for the vehicle feature design consists of features representing the vehicle features of the system - in our example, Pilot and Assist feature. The technical solutions feature model has several layers, each for subsystem design, component design and E/E architecture design. By having separate feature models, we can independently configure vehicle features with the technical solutions existing in the technical solutions feature model in each layer.

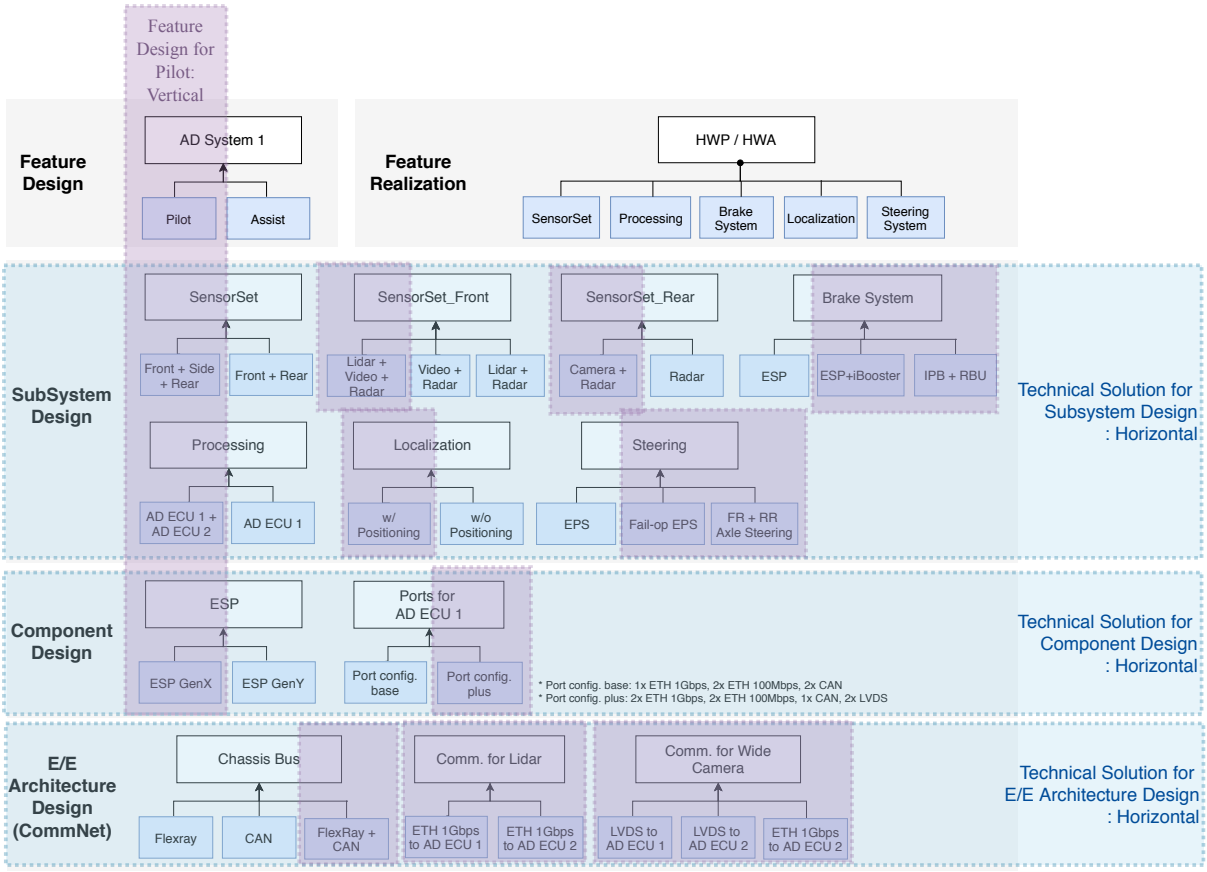

Figure 5.3.: Directions for Design Levels

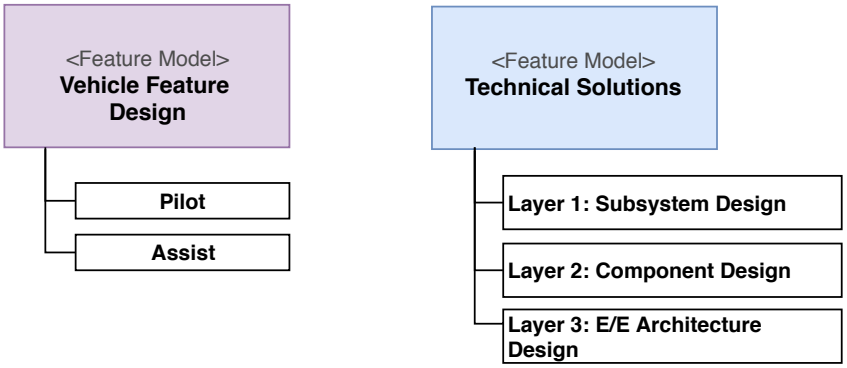

Figure 5.4.: Structure of Feature Models - Vehicle Feature Design and Technical Solutions

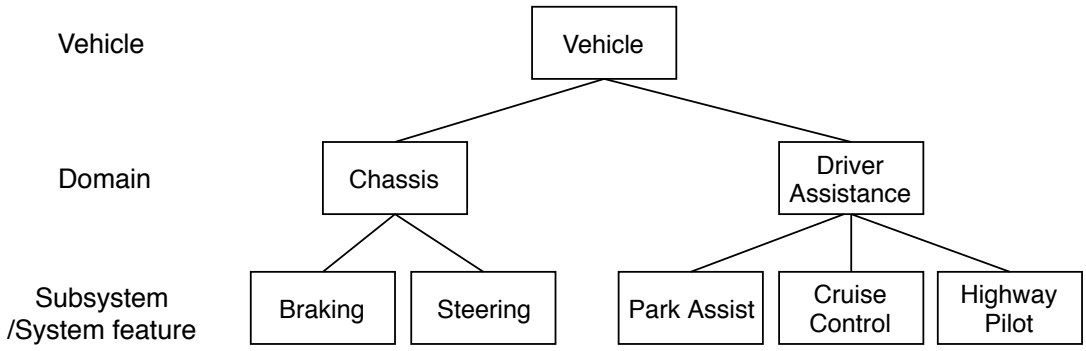

Figure 5.5.: Functional Decomposition of Vehicle [KLD02]

Technical solution feature model shall include technical solutions spreading out over the different domain of a vehicle. For this, we need to apply a categorization, and the most common way of classifying it is to structure the feature model by the functional architecture of the vehicle [KLD02]. Figure 5.5 shows the functional decomposition of a vehicle, in which it decomposes the vehicle into domains, and then subsystems or system feature.

This functional decomposition is the basis of all development process in the automotive industry, and therefore it is easy to understand [KLD02]. Also, it is well-suited to the automotive development settings, where the development for different subsystems and components are distributed over different organizations and locations. However, this shows weaknesses when having to apply variables which are global, not specific to the system, such as brand, country, vehicle types. Also, ensuring traceability between different levels is difficult in this decomposition [OPS+17]. Furthermore, it is not consistent in the subsystem/system feature levels, where one is representing physical building blocks of the vehicle, and the other is for the functional features. For our technical solution feature model, we adopt this functional decomposition with some modifications. Rather than having all domains and subsystems existing in the vehicle which do not possess variability, we only include building blocks (e.g. braking, steering, sensorset) which have variability in the technical solutions. This is because, the essence of variant management is to model the variability in the feature model, not the whole architecture of the system, which would be redundant to the information existing in the technical architecture models. Figure 5.6 shows the structure of the technical solutions feature model with functional decomposition applied. The weakness of the functional decomposition presented above is compensated by adopting other concepts which are presented in the next sections.

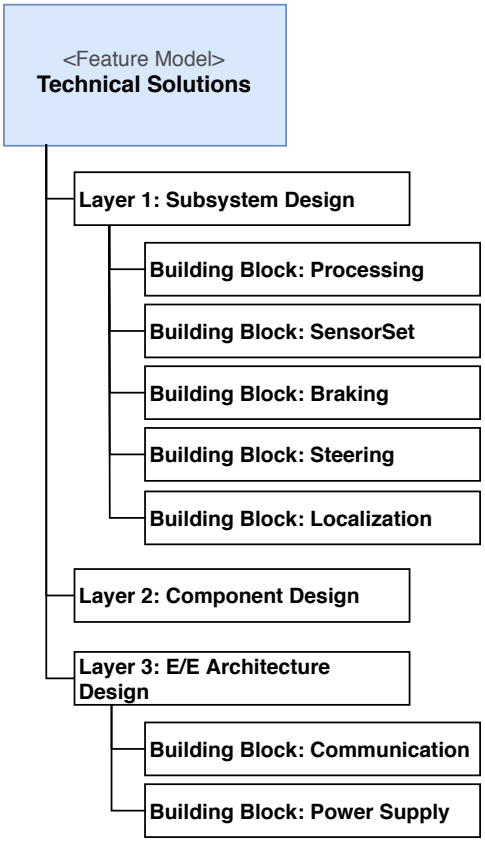

Figure 5.6.: Structure of Feature Model - Technical Solutions with Building Blocks

5.5.2. Variability outside of the system

Not only the variability in the system, there exist variables which are global to the systems. The examples of these parameters would include regions, markets, customers, vehicle types and driving types. H. Hartmann introduced the structuring criteria by context variability, which is shown in Figure 5.7 [HT08].

The advantage of this criteria is that it reduces redundancy and contributes to consistency, and also it is well applicable to functional decomposition [HT08; OPS+17]. However, its weakness is that it possibly has a high number of dependencies with the system models [OPS+17].

In our method, we created a separate feature model for the context variability, so that it can be flexibly combined with other feature models that describe a system, rather than having the same context variability parts in each of the feature models for every system. Figure 5.8 shows the structure of the context variability feature model.

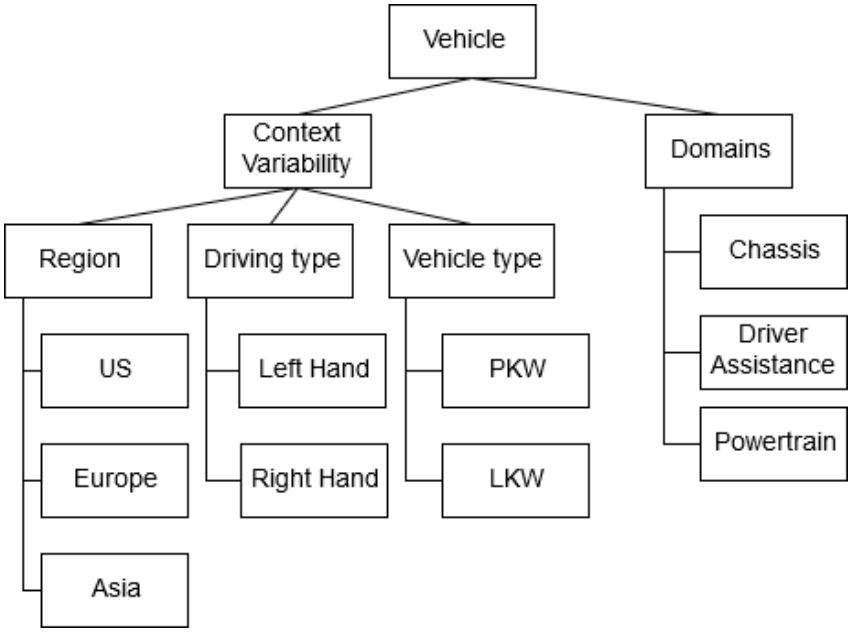

Figure 5.7.: Structuring Criteria by Context Variability [HT08]

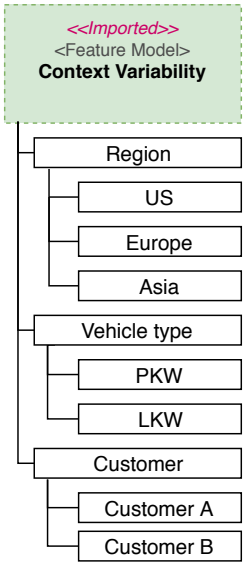

Figure 5.8.: Structuring Criteria by Context Variability [HT08]

5.5.3. Traceability

As mentioned earlier, ensuring traceability from requirements to technical architecture is an essential requirement for variant management. The most significant benefit of having traceability is transparent change management. Through a well-established traceability link, the changes in one part can be successfully traced to the affected parts. It includes the changes in system requirements as well as the changes in a subsystem or a component. Also, traceability helps users better understand the vehicle-level requirements. In the automotive system development setting, where the whole system development is divided into many different sub-levels of development, this is an important point. It enables the development engineers for subsystems or component levels to have vehicle-level system requirements and information.

In this sense, in examining the detailed structure of the feature models, we profoundly considered securing traceability, mainly from the requirement to the technical architecture, which corresponds to the design process. System architects derive system requirements from use cases, and then they derive technical solutions which can satisfy the system requirements. Each technical solutions show the types and numbers of components that shall exist in the system, which comprise the technical architecture.

The structuring guideline, presented by F. Bachmann, discusses structuring feature models by asset types in domain [BN09]. Asset types refer to different design aspects in the process of system development, such as requirement specifications and architecture design. This guideline is adopted in our method to decide to structure the feature models based on technical solutions derived from the requirement specifications. However, having a separate feature model for the architecture design produces redundant information to the technical architecture model.

As a result, the technical solution feature model contains the variances in the technical solutions and the high-level requirements for these technical solutions. Technical solutions contain technical elements which realize the solutions. The high-level requirements existing in the technical solution feature model can be linked to the system requirements, which are normally documented in a requirement engineering tool such as DOORS. In the same way, the technical elements under the technical solutions can be connected to the technical architecture model, which is modelled in the architecture modelling tool like Rhapsody. Figure 5.9 shows the detailed structure of the technical solution feature model, and Figure 5.10 is an example of it.

Figure 5.11 shows the complete structure of the technical solution feature model, from the highest level - layers, down to building blocks, high-level requirements, technical solutions, and technical elements.

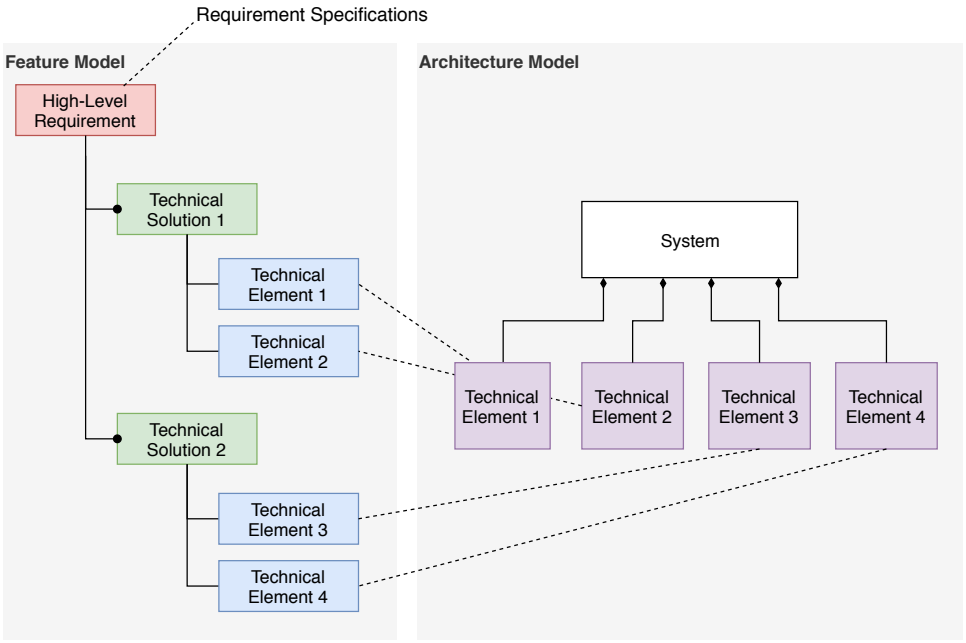

Figure 5.9.: Detailed Structure of Technical Solution Feature Model

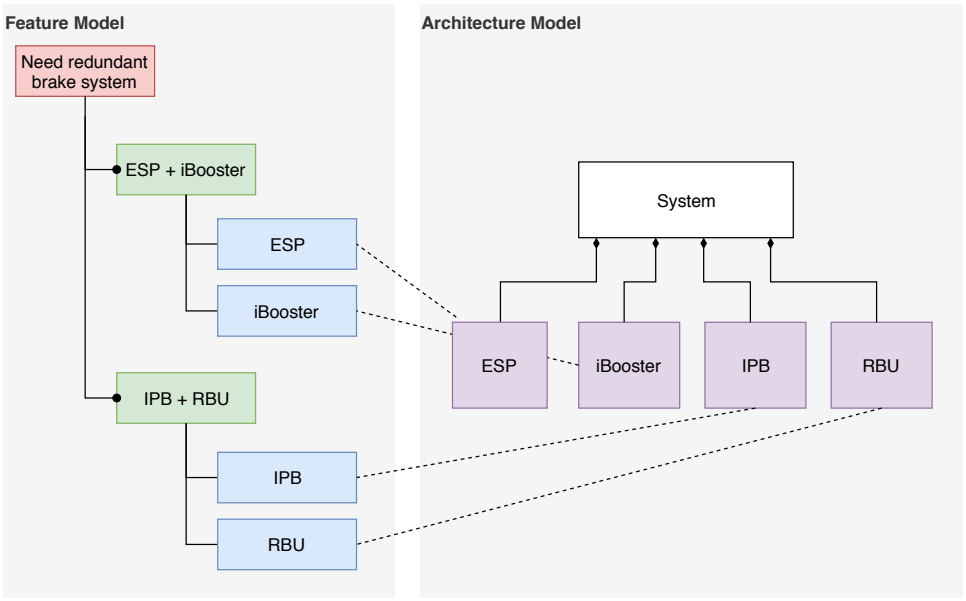

Figure 5.10.: Example of Detailed Structure of Technical Solution Feature Model

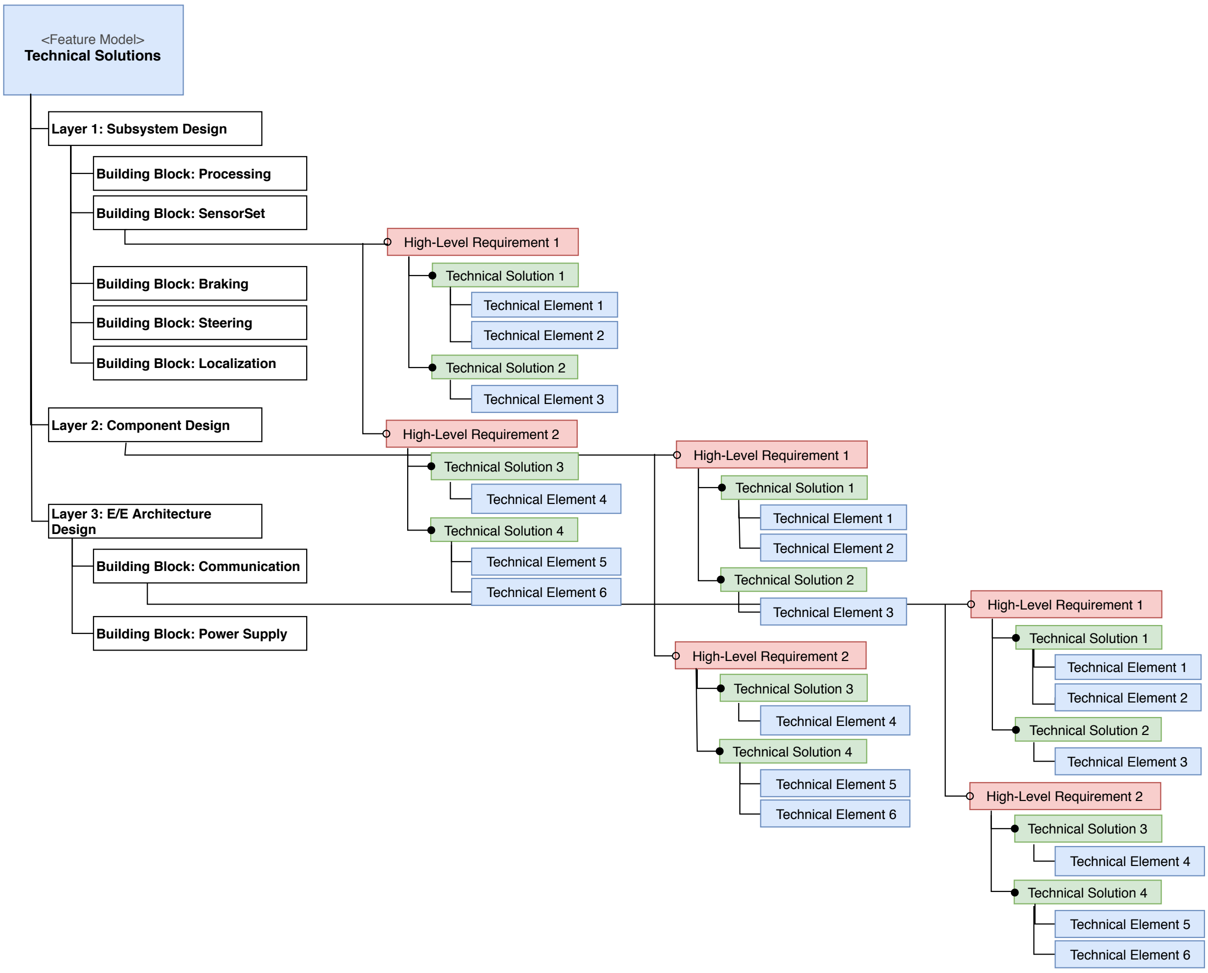

Figure 5.11.: Overall Structure of Technical Solution Feature Model

In summary, the feature model is structured based on technical solutions. The most significant advantage of this is traceability, from the requirements to the technical solutions and then to the technical architecture. In this way, we can also document the technical solutions for the design decisions, which usually only reside in the brain of system architects. Besides, this enables a more straightforward system variant configuration, due to the dependency modelled between vehicle level features and high-level requirements. With this, by selecting the vehicle feature (e.g. either Pilot or Assist), the majority of high-level requirements are automatically selected. It reduces the number of selections that the users have to make to configure a system variant and helps to reduces the possibility of making mistakes by missing out some necessary selections. However, a disadvantage of this method is that it is quite complicated.

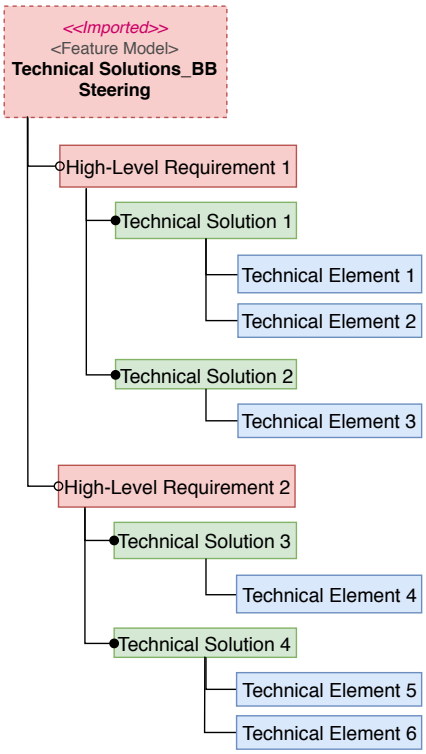

Figure 5.12.: Structure of Technical Solution Feature Model - Blackbox Concept

5.5.4. Blackbox Concept

For the blackbox concept, the feature models for the contents of the blackbox, i.e. a particular subsystem or component, should be able to be imported and fill the part which was left as blackbox in the technical solution feature model of the system. The product owner organizations or departments would create these feature models and provide it to the system architects and E/E architects. The structure of the feature model shall follow the technical solution feature model, as these models also represent the technical solutions specific for particular subsystems or components. Figure 5.12 shows the structure of these feature models for the blackbox concept.

After importing these blackbox feature models into the project space of the system, the complete picture of the technical solutions for the system shall be available, and the users can now configure the system variant by selecting particular technical solutions for each variant.

It should be considered that each development parties for the subsystem or components might already have different modelling structures or notations for their feature models. In order to minimize the modelling efforts and preventing having to create an entirely new feature model for this blackbox concept, an adequate alignment between these two

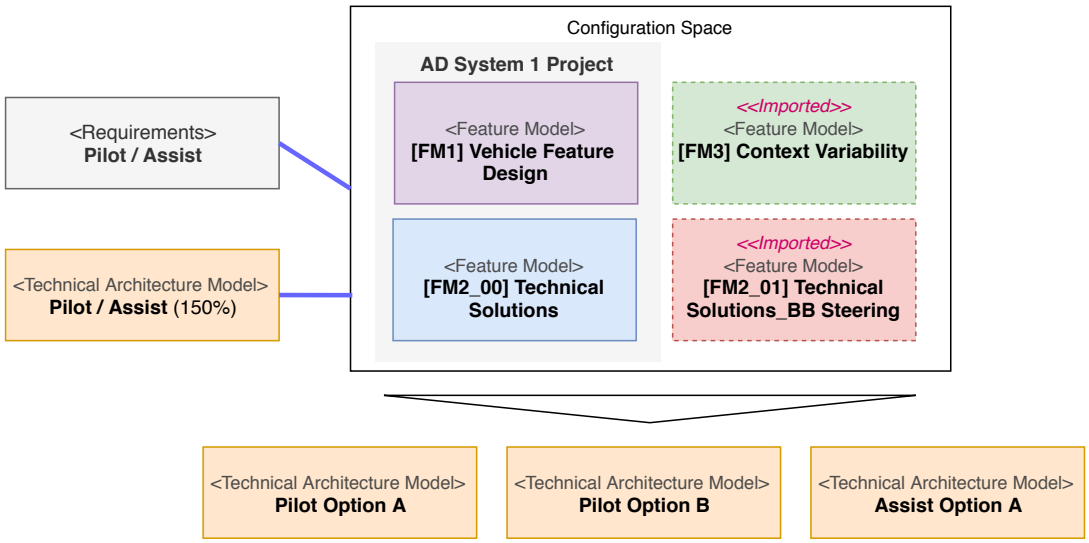

Figure 5.13.: Overview of the Overall Concept

levels, i.e. subsystem/component feature model and system feature model, shall follow. For the alignment, the protocols to exchange the feature models and the compromised structure of the feature model which can be agreed by both parties shall be determined.

5.5.5. Overview of the overall concept of the method

Figure 5.13 shows the overall concept of the variant management method. We numbered the feature models based on the order of the usage in the system variant configuration. The user shall follow the number of feature models, from 1 to 3, when selecting features for a particular system variant. The steering system is used as an example for the blackbox concept (FM2_01).

The configuration space here means where the corresponding feature models relevant for the system are added and configured together for the system variant configuration, which produces resulting product models for each system variant (e.g. Pilot Option A, Pilot Option B). According to the user manual of Pure::Variants, a variant management tool which we used for the implementation of this method, the configuration space is "used to combine models for configuration purposes [Gmb19]". In the configuration space, there exist the first two feature models, *FM1 Vehicle Feature Design* and *FM2_00 Technical Solutions*, which are specific for the system (i.g. AD System 1 in our example). These two feature models abide in the system project, in our example, AD System 1 Project. Outside of this project, in the same configuration space, the other two external feature models are imported - *FM2_01 Technical Solutions_BB_Steering*, so-called *Blackbox feature model*, and *Context Variability*. For the traceability, certain features in the *Technical Solution* feature

models are connected to requirement specifications which usually reside in requirement management tool and to the technical architecture model in the architecture modelling tool. To be more specific, the high-level requirements which are modelled as features in the technical solution feature model, i.e. *FM2_00 Technical Solutions* and the blackbox feature model, can be linked to requirements in requirement management tool. Also, the technical elements which are also modelled as features in the same feature models are connected to the corresponding model elements in the technical architecture model. Finally, the feature models added in the configuration space can be jointly configured to produce each different system variant (e.g. Pilot Option A, Pilot Option B).

5.5.6. Dependencies between feature models

For better usability, we also model dependencies between feature models. Figure 5.14 shows example dependencies existing between the feature models. Between *FM1 Vehicle Feature Design* and *FM2_00 Technical Solutions*, a dependency between the vehicle feature, i.e. *Pilot* or *Assist*, and the high-level requirements or technical solutions that are specific to the vehicle feature shall be added. For example, *Pilot* feature requires to have a redundant braking system. Then a dependency from the feature *Pilot* in *FM1 Vehicle Feature Design* to the high-level requirement feature *Need Redundant Braking System* in *FM2_00 Technical Solutions* shall be added. The same dependency applies between *FM1 Vehicle Feature Design* and *FM2_01 Technical Solutions_BB_Steering*.

Additionally, between *FM2_00 Technical Solutions* and *FM2_01 Technical Solutions_BB_Steering*, a dependency shows that a specific building block in *FM2_00 Technical Solutions* is modelled as blackbox, and this part corresponds to a separate feature model which is imported from outside, *FM2_01 Technical Solutions_BB_Steering*.

For the *FM3 Context Variability*, no dependencies were added in this thesis, but potential dependencies about the global variables could be added. For example, a dependency showing that a particular feature is specific to a certain market or is intended for a particular vehicle type can be added. However, it is crucial to make sure that the direction of dependencies goes only from *FM1 Vehicle Feature Design* or *FM2_00 Technical Solutions*, which are the feature models specifics to a particular system, to the imported feature models. In other words, the system-specific dependenceis shall only be added to the system-specific feature models, but no to the external feature models which shall be globally used. System-specific dependencies such as "*Pilot* feature requires *Redundant Steering System*" shall not be added to the imported feature models but only to the system-specific feature models, such as *FM1 Vehicle Feature Design* or *FM2_00 Technical Solutions*.

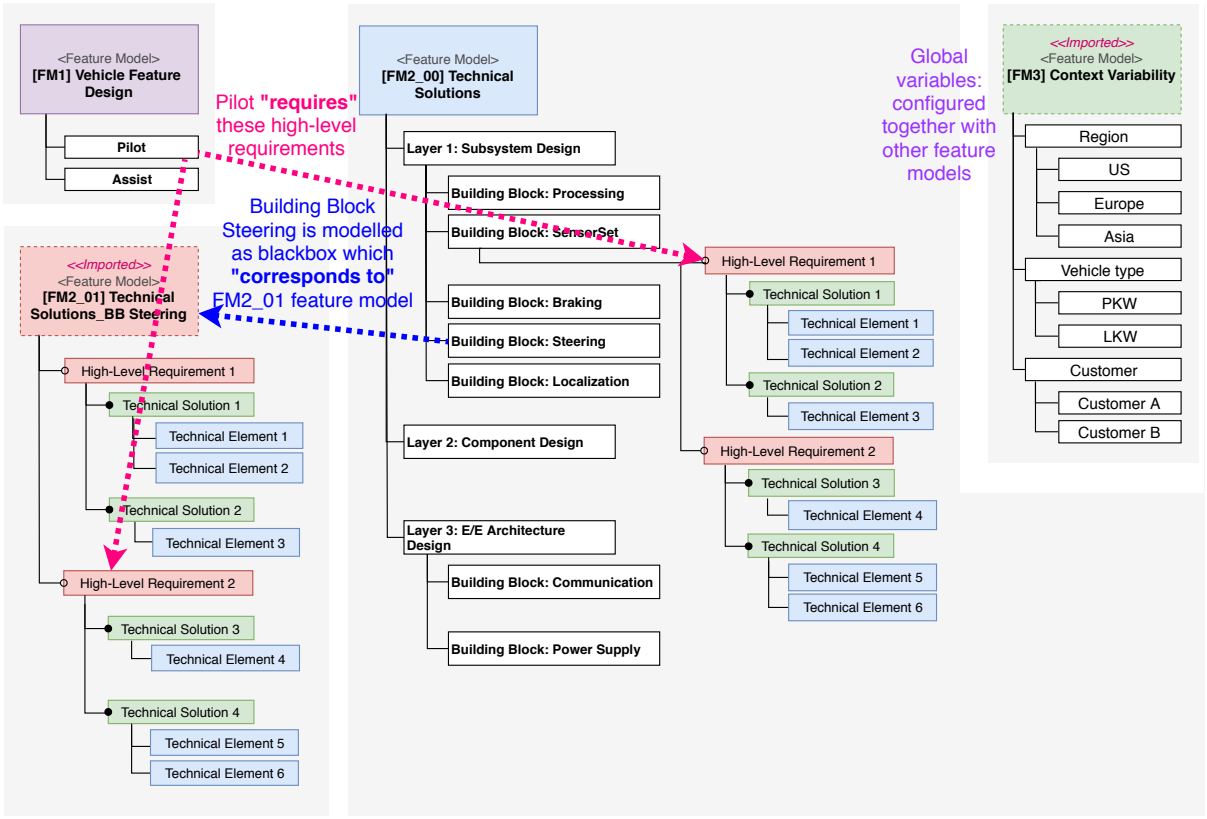

Figure 5.14.: Example Dependencies Between Feature Models

5.5.7. Overall process of using the method

In this section, we present the process of using the method from constructing the feature models and using the models for the system variant configuration, as shown in Figure 5.15 and described in steps below:

Step 1 Build *FM2_00 Technical Solutions*

- Create the technical solution feature model for the system of interest, add layers, building blocks, high-level requirements, technical solutions and technical elements as features.

Step 2 Import *FM2_01 Technical Solutions_BB_Steering*

- Import the blackbox feature models that is created and offered by product owners into the Configuration Space.

Step 3 Build *FM1 Vehicle Feature Design*

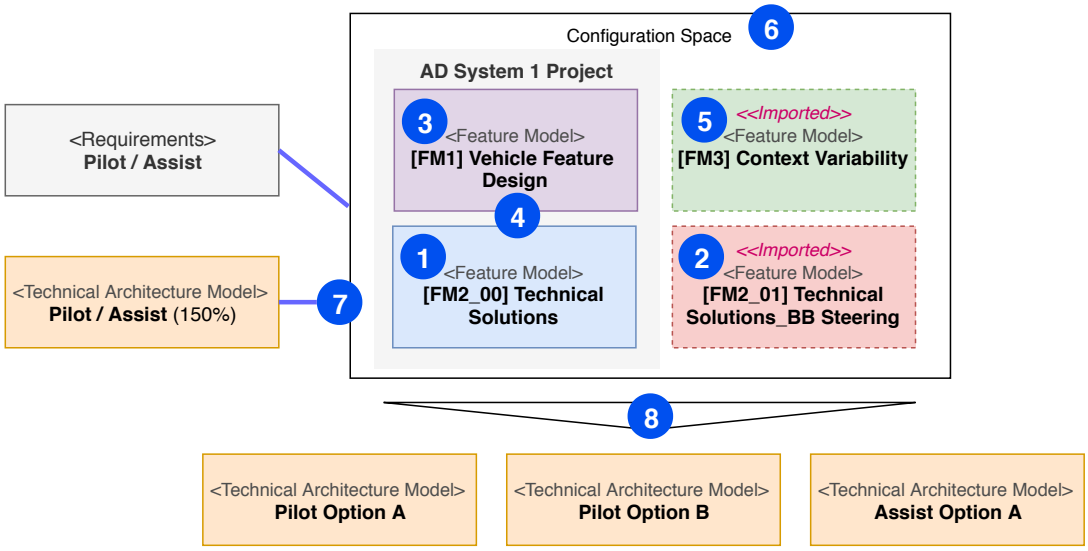

Figure 5.15.: Process of Using the Method

- Create the vehicle feature design feature model and add vehicle features. (e.g. *Pilot* or *Assist*)

Step 4 Add relations (dependencies) to *FM1 Vehicle Feature Design*

- Add dependencies to *FM1 Vehicle Feature Design* for the high-level requirements or technical solutions specific to the vehicle feature. (e.g. *Pilot* requires the high-level requirement "*Need Redundant Braking System*")

Step 5 Import *FM3 Context Variability*

- Import the context variability feature model into the Configuration Space.

Step 6 System variant configuration (product configuration)

- Select the technical solutions to create a system variant. For example, if we would like to configure *Pilot Option A*, we need to do the selections of technical solutions for the high-level requirements that is specific to *Pilot* feature. For this, in this step, we select the technical solution "*Lidar + Radar + Video sensors at front*" which is for the high-level requirement "*Need to detect objects in front*".

Step 7 Connect the Configuration Space (feature models) to the technical architecture model

- The Configuration Space which contains the feature models shall be connected to the technical architecture model. After this connection, the technical elements which exist under the technical solutions in this feature model shall be assigned to the corresponding model elements in the technical architecture model. For

instance, the technical element "*Lidar*" which is under the technical solution 1 "*Lidar + Radar + Video sensors at front*" shall be linked to the component block existing in the technical architecture model.

Step 8 Variant transformation

- Based on the configuration performed for each system variant, the 100% architecture model shall be automatically created.

Chapter 6

Implementation

The variant management method that we presented in the previous chapter was implemented on the example technical architecture in a variant management tool-suite Pure::Variants. In this chapter, we present the result of the implementation.

6.1. Pure::Variants Modelling Concept

Pure::Variants [Gmba] is a variant management tool-suite by Pure-Systems GmbH that is developed based on the CONSUL approach [BPS04]. It was primarily developed to support each phase of the software product-line engineering [Gmb19] Pure::Variants is regarded as a practical feature modelling language or method developed and proposed by the industry side, whereas most of the feature modelling approaches, listed in Section 3.3.2, are presented by academia [BSL+13]. M. Sinnema et al. classifies Pure::Variants as a mature variability modelling technique that differ from the modelling techniques such as CBFM [CHE05] and FeatuRSEB [GFd98] which are based on FODA [KCH+90] [SD07].

Pure::Variants models variability with the combined usage of four types of models: *Feature Model, Variant Model, Family Model*, and *Result Model*. The *Feature Model* contains FODA-like features, which means that it models variability as features [SD07]. In addition to the type of features, the *Feature Model* also contains composition rules which represent dependencies between features. The *Variant Model* describes a configuration of features to comprise a specific variant of a system. The *Family Model* is necessary because it describes the family in terms of architectural elements [SD07]. It includes components and parts which are associated with source models [SD07]. Source models can be system architecture models which describe the product in the system level or programming language elements such as classes, objects, flags or variables in the software level which

determine how the source code for the specific part is generated [SD07]. Finally, the *Result Model* indicates which elements that are associated with the *Family Model*s are included in the product [SD07]. Based on the type of features and composition rules specified in the *Feature Model*, and the elements selected in the corresponding *Variant Model*, the *Result Model* is generated through variant transformation.

The advantage of Pure::Variants is that, firstly, it describes a modelling method that is suitable for describing embedded system architectures in the industry. Also, it provides adequate tool support which allows users to produce an outcome which can directly be used for product descriptions.

6.2. Implementation of Variant Management Method

In this section, we describe the process and results of the implementation of the method. The developed variant management method was implemented on the example technical architecture in Pure::Variants. The implementation follows the 'Overall process of using the method', which we presented in Section 5.4.7. For the architecture modelling of the example technical architecture, we used Rhapsody.

6.2.1. Preparation: Creation of Technical Architecture Model

In order to correctly implement the variant management method, the *FM2_00 Technical Solutions* feature model needs to be connected to the corresponding technical architecture model for the system of interest. Also, to prove the concept of the method, we need to observe whether the correct architecture models for each different system variant (e.g. Pilot Option A, Pilot Option B, Assist Option A) are produced as the result of system variant configuration and variant transformation. For this, we created the technical architecture model for the example technical architecture in Rhapsody. As explained in Chapter 2 Foundations, the technical architecture is modelled with different views each represented in BDD and IBD. We modelled each building blocks of the example architecture as separate BDDs and IBDs, as shown in Figure 6.1. Figure 6.2 and Figure 6.3 shows the screenshots of BDDs and IBDs of the example technical architecture that we modelled in Rhapsody. This is the 150% model, which include all variation points existing in all system variants. For the architecture modelling, we followed the MBSE modelling guideline in the organization.

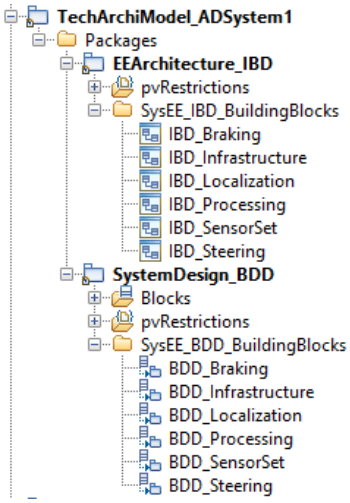

Figure 6.1.: Structure of Technical Architecture Model for the Example Architecture

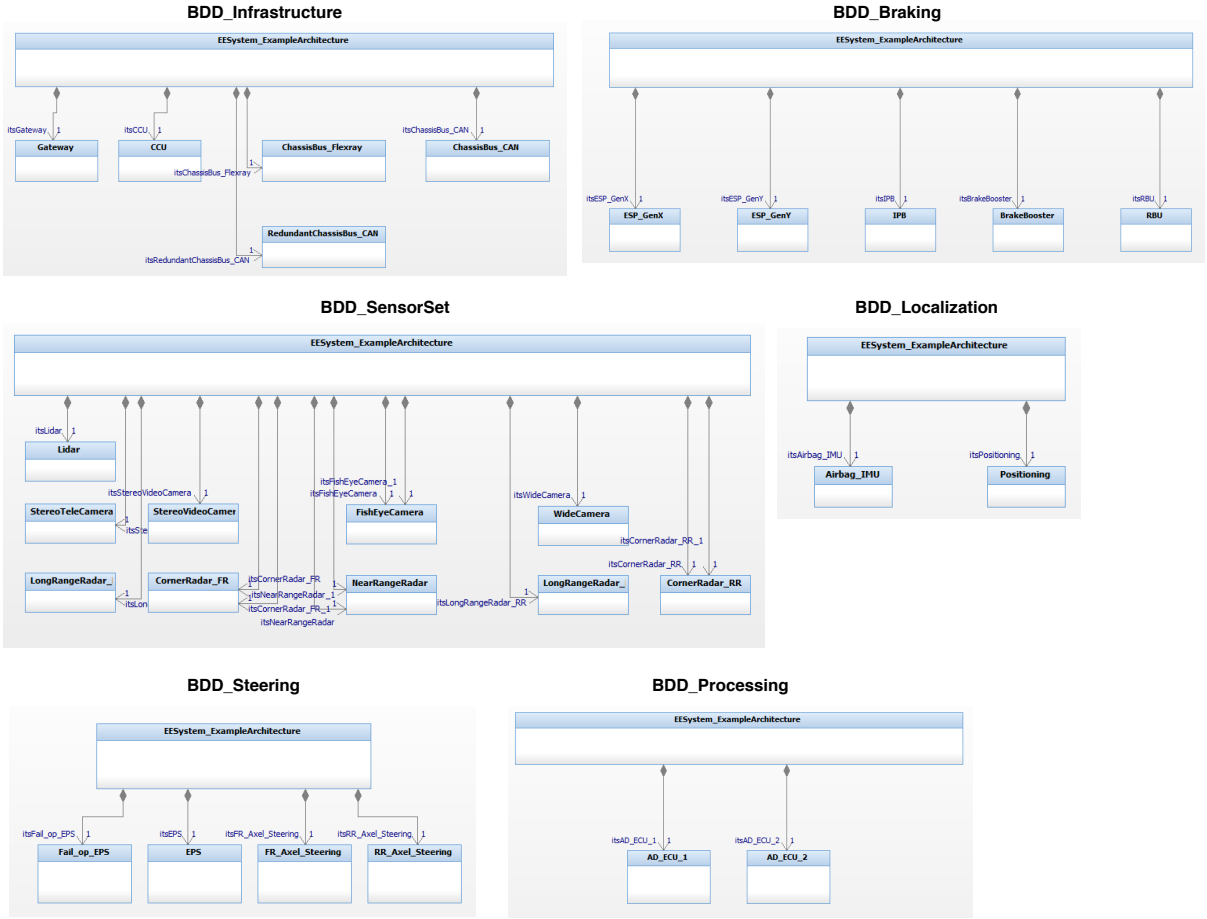

Figure 6.2.: BDDs of Example Architecture

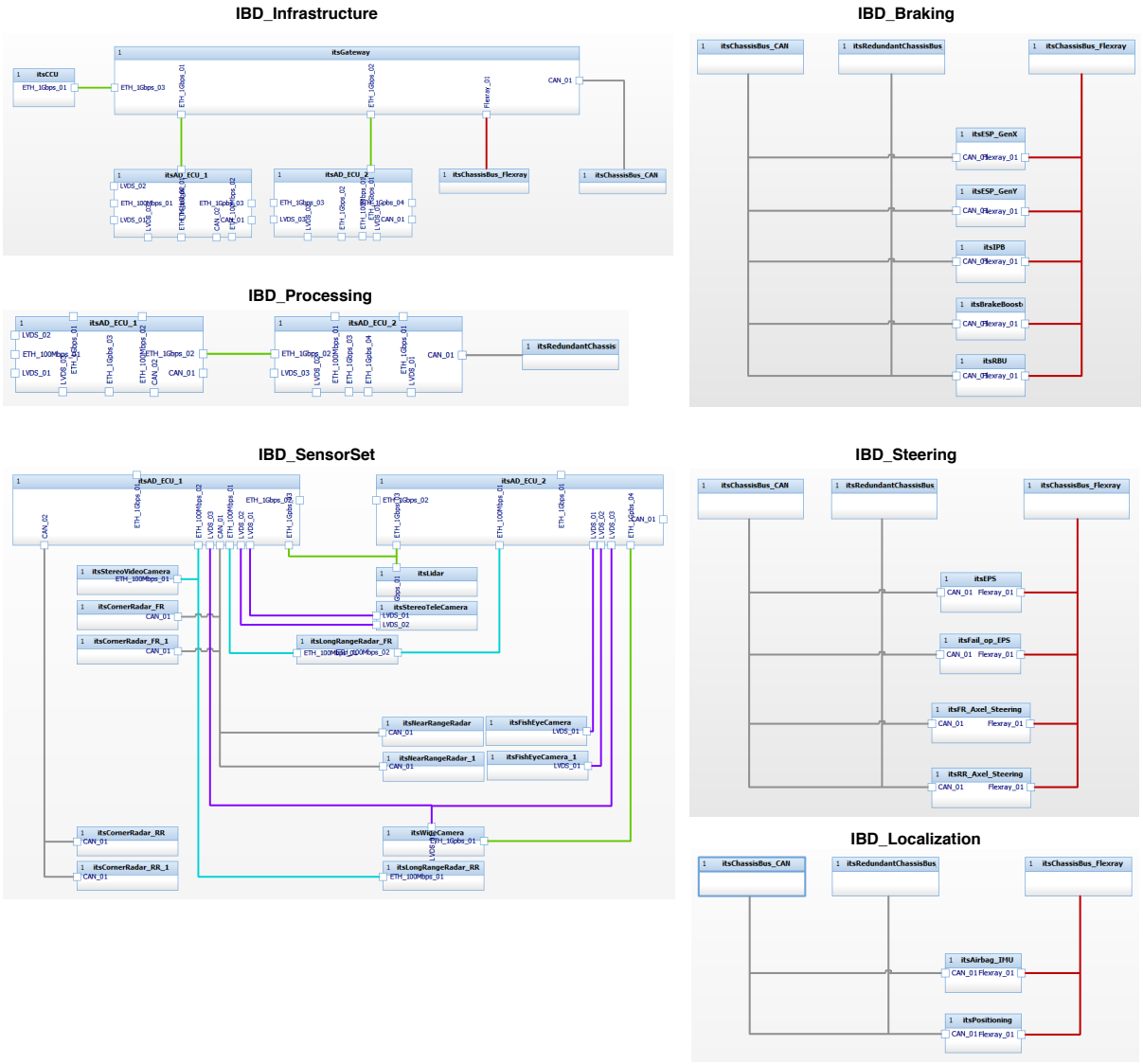

Figure 6.3.: IBDs of Example Architecture

6.2.2. Step 1 Build *FM2_00 Technical Solutions*

The example project *PROJECT1_ADSystem1* for the example system *AD System 1* of the example technical architecture was created in the Pure::Variants project space. Under this project, we created the feature model *FM2_00 Technical Solutions*. Figure 6.4 shows the screenshot of *FM2_00 Technical Solutions*, and Figure 6.5 shows the same feature model in graph format.

Pure::Variants supports four different types of features - Mandatory (symbol: exclamation mark), Alternative (1 out of n, symbol: bi-directional arrow), OR (i..j out of n,

Figure 6.4.: *FM2_00 Technical Solutions*

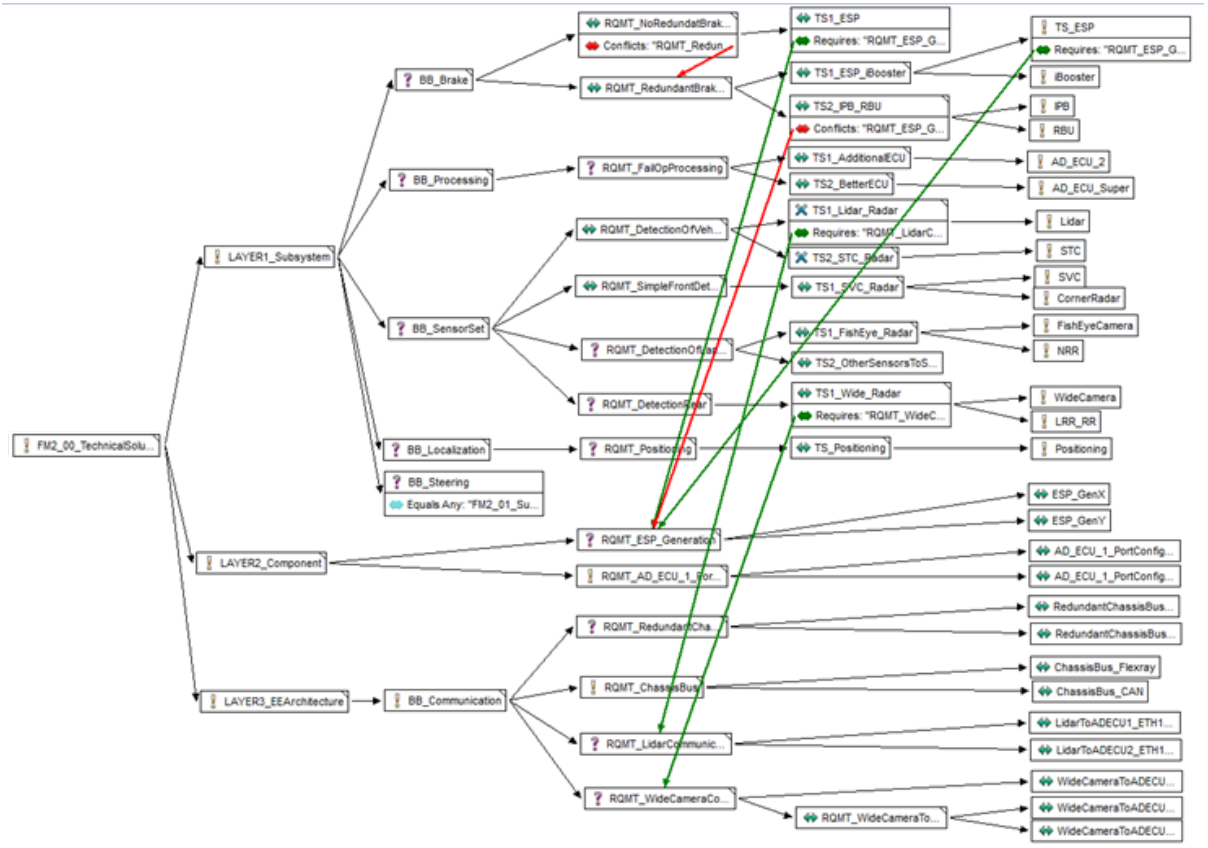

Figure 6.5.: *FM2_00 Technical Solutions* in Graph

symbol: cross mark), and Optional (0..1, symbol: question mark) [SD07]. The legend for this notation is added in the right upper corner of Figure 6.4.

On the top level, there exist three layers - *LAYER1_Subsystem*, *LAYER2_Component* and *LAYER3_E/E Architecture*. Under each layer, Building Block (BB)s are added where necessary. For instance, *LAYER1_Subsystem* has building blocks of subsystems. However, for *LAYER2_Component*, only a few variabilities exist for some of the technical elements modelled in the upper layer, i.e. *LAYER1_Subsystem*. Therefore, for simplicity, we did not add building blocks to this layer. For *LAYER3_E/E Architecture*, the building block of *BB_Communication* (Communication network) exist, and *BB_Power Supply System* could be added. The building block features are modelled as *Optional* so that the user can add only the necessary building blocks for the system of interest.

Under this top-level structure, as explained in the previous section, in the technical solution feature model, high-level requirements, technical solutions and technical elements are modelled as features. Also, there exist dependencies between the layers. For instance, under *LAYER1_Subsystem* and *BB_Brake*, the high-level requirement *RQMT_RedundantBrakeSystem* is added. The two possible technical solutions for this

high-level requirement, *TS1_ESP_iBooster* and *TS2_IPB_RBU* are modelled as features. Each technical solution has the corresponding technical elements as child features - *ESP* and *iBooster* for *TS1_ESP_iBooster*, *IPB* and *RBU* for *TS2_IPB_RBU*. For the technical element *ESP*, there is variability in the component design, product generation - i.e. ESP GenX, ESP GenY. This is modelled as dependency *Requires* which points to the high-level requirement *RQMT_ESP_Generation* under the *LAYER2_Component*.

By having this dependency, not only the user sees the relations between the features, but also the user gets help in the system variant configuration step by being guided to the part where a selection of technical solutions is necessary. In this case, when the user selects the *TS1_ESP_iBooster* for the redundancy requirement, he or she is then notified, due to the dependency, that he or she needs to make a selection for *RQMT_ESP_Generation*. Otherwise, the feature combination for the system variant configuration would be invalid. Having these dependencies reduces the possibility of making mistakes by leaving out some necessary selections.

High-level requirements and technical solutions modelled here are examples created for this implementation. The technical plausibility or the completeness was not considered.

6.2.3. Step 2 Import *FM2_01 Technical Solutions_BB_Steering*

As a next step, we imported *FM2_01 Technical Solutions_BB_Steering*, so-called blackbox feature model, into the project space, as shown in Figure 6.6. The screenshot on the left shows the project space and the feature model is shown on the right, both in tree and graph format.

In this thesis, *Steering System* is used as the example of blackbox concept. *FM2_01 Technical Solutions_BB_Steering* was created in a separate project (*PROJECT2_BB_Steering*) in advance. The structure of this feature model follows the *FM2_00 Technical Solutions* in this implementation, being composed of high-level requirements, technical solutions and technical elements.

The *BB_Steering* in *FM2_00 Technical Solutions* is modelled as blackbox, meaning no content is added but only the relation *EqualsAny* to *FM2_01 Technical Solutions_BB_Steering*. In this way, the blackbox is filled with the high-level requirements and technical solutions offered by the product owner, solving the issue of having no visibility on the design options of the subsystem from the technical architecture level.

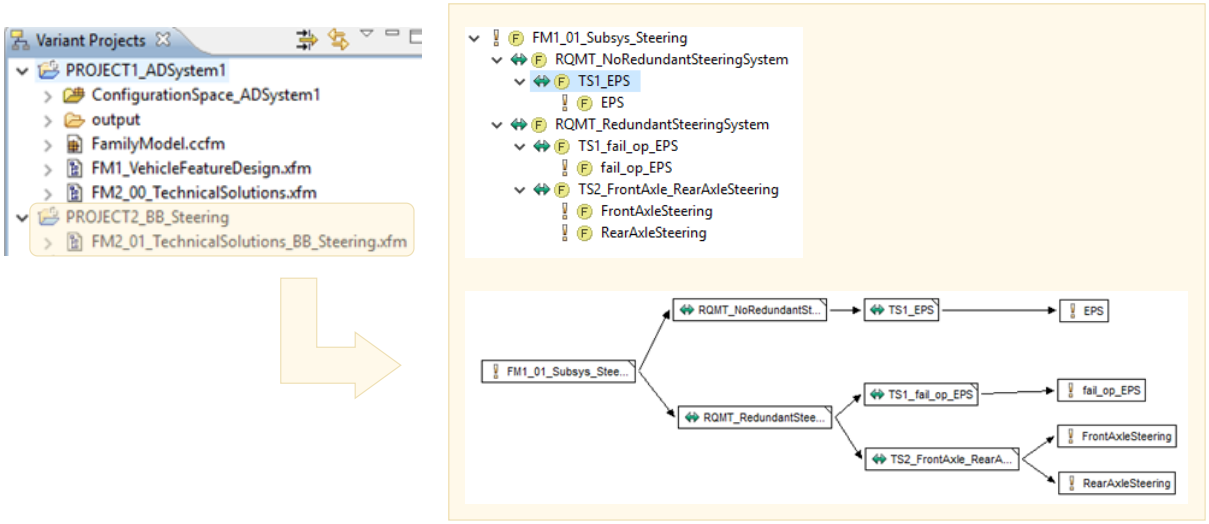

Figure 6.6.: *FM2_01 Technical Solutions_BB_Steering*

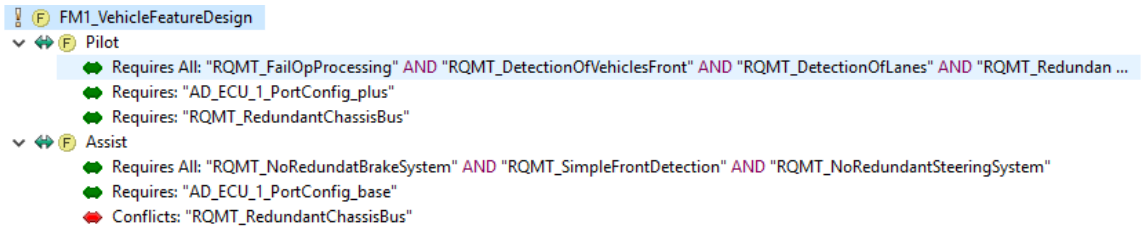

Figure 6.7.: *FM1 Vehicle Feature Design* with relations

6.2.4. Step 3 Build *FM1 Vehicle Feature Design*

This feature model is also created under the *AD System 1* project, since it is also project-specific. The structure of this feature model is rather simple - it only has vehicle features, in our case *Pilot* and *Assist*.

6.2.5. Step 4 Add relations (dependencies) to *FM1 Vehicle Feature Design*

Now that we have feature models both for technical solutions and vehicle feature design, we can add dependencies. The dependencies *Requires* and *Conflicts* are added to *FM1 Vehicle Feature Design* as shown in Figure 6.7.

These dependencies point to the high-level requirements or technical solutions, which are modelled in the feature models for technical solutions, i.e. *FM2_00 Technical Solutions* and *FM2_01 Technical Solutions_BB_Steering*. For instance, to design the *AD System 1* with *Pilot* feature, for the subsystem design level, we need redundancy both in the

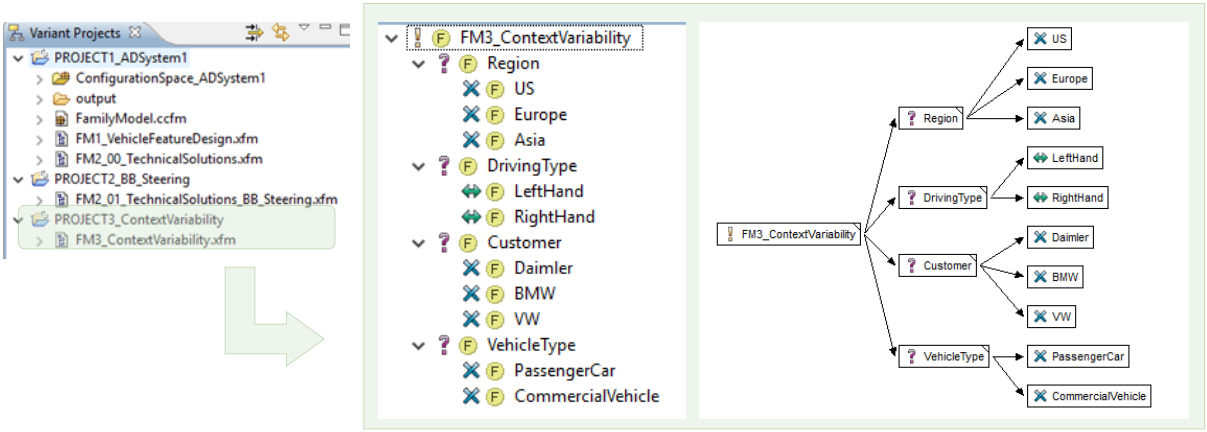

Figure 6.8.: *FM3 Context Variability*

brake system and in the steering system, fail-operational processing of AD ECUs, ability to detect objects in front as well as the detection of lanes. On the component design level, *Pilot* feature requires to have *Port configuration plus* for the main AD ECU. Lastly, for the E/E architecture, we need a redundant chassis bus. Therefore, the *Requires All* dependency directing to the corresponding high-level requirements are added.

6.2.6. Step 5 Import *FM3 Context Variability*

In the same way that we imported the blackbox feature model, we added context variability feature model to the project space. Figure 6.8 shows the project space and the *FM3 Context Variability*.

6.2.7. Step 6 System variant configuration (product configuration)

Now that all necessary feature models are added into the project space, we can configure the system variants. In the configuration space of Pure::Variants, variant models for each system variants were created (e.g. `Pilot_OptionA.vdm`). Variant models would show the feature trees of all the feature models that exist in the configuration space. The users need to make necessary selections of the technical solutions. Here, the numbering of the feature models comes into use. The users can follow the numbers of feature models, from the *FM1* in the increasing order to the *FM3*, which guide them to the next feature model that they need to check.

For example, the users would first need to go to the *FM1 Vehicle Feature Design* part in the variant model and make the selection for the vehicle feature - *Pilot* or *Assist*. Due to

the dependency added in the *FM1 Vehicle Feature Design*, the high-level requirements in the *FM2_00 Technical Solutions* and the *FM2_01 Technical Solutions_BB_Steering* which are required by the vehicle feature are then automatically selected, and the conflicting parts are deselected. Then the users simply need to make the selections in the required parts based on the needs of the system variant. As explained before, this makes the variant configuration simpler and more straightforward, as the necessary parts are automatically activated and the user is informed whether there still exist more selections that to do. Lastly, the global variables for the context variability available in the *FM3 Context Variability* can be selected.

Figure 6.9 shows the configuration steps of the variant model `Pilot_OptionA.vdm` from the beginning, selecting vehicle feature in *FM1* to choosing values for global variable in *FM3*. As a result, the system variant *Pilot_OptionA* looks like the screenshot shown in Figure 6.10.

In Pure::Variants, although the feature models are numbered in the alphabetical order based on the order of usage for the variant configuration, they are not alphabetically ordered in the variant models. This point can confuse the user, making the users going up and down to find the next number of feature models for the selection. As one can see in Figure 6.10, the result is correctly shown in the alphabetical orders of the feature models - from *FM1* on the top, and down to *FM3* at the bottom. Thus, this point shall be improved from the tool side.

6.2.8. Step 7 Connect the Configuration Space (feature models) to the technical architecture model

One of the advantages of using Pure::Variants is that it provides plug-ins to the tools that are used in the system engineering process. We modelled the technical architecture model on Rhapsody, and through the Pure::Variants plug-in to Rhapsody, the architecture model and the configuration space in Pure::Variants which contain feature models are connected. The screenshot on the left in Figure 6.11 shows the plug-in window of Pure::Variants on Rhapsody.

After connecting the configuration space and the architecture model, we need to assign the technical element features from the feature models onto the model elements in the technical architecture, such as *Block*s, *Connector*s and *Port*s. In SysML notation for technical architecture, physical components such as ECUs, sensors and actuators are modelled as *Block*s in BDDs, and communication connections are modelled as *Connector*s linked between *Port*s of the components in IBDs. The features from Pure::Variants feature models are assigned to *Constraint*s and these shall be attached to model elements through *Attach*.

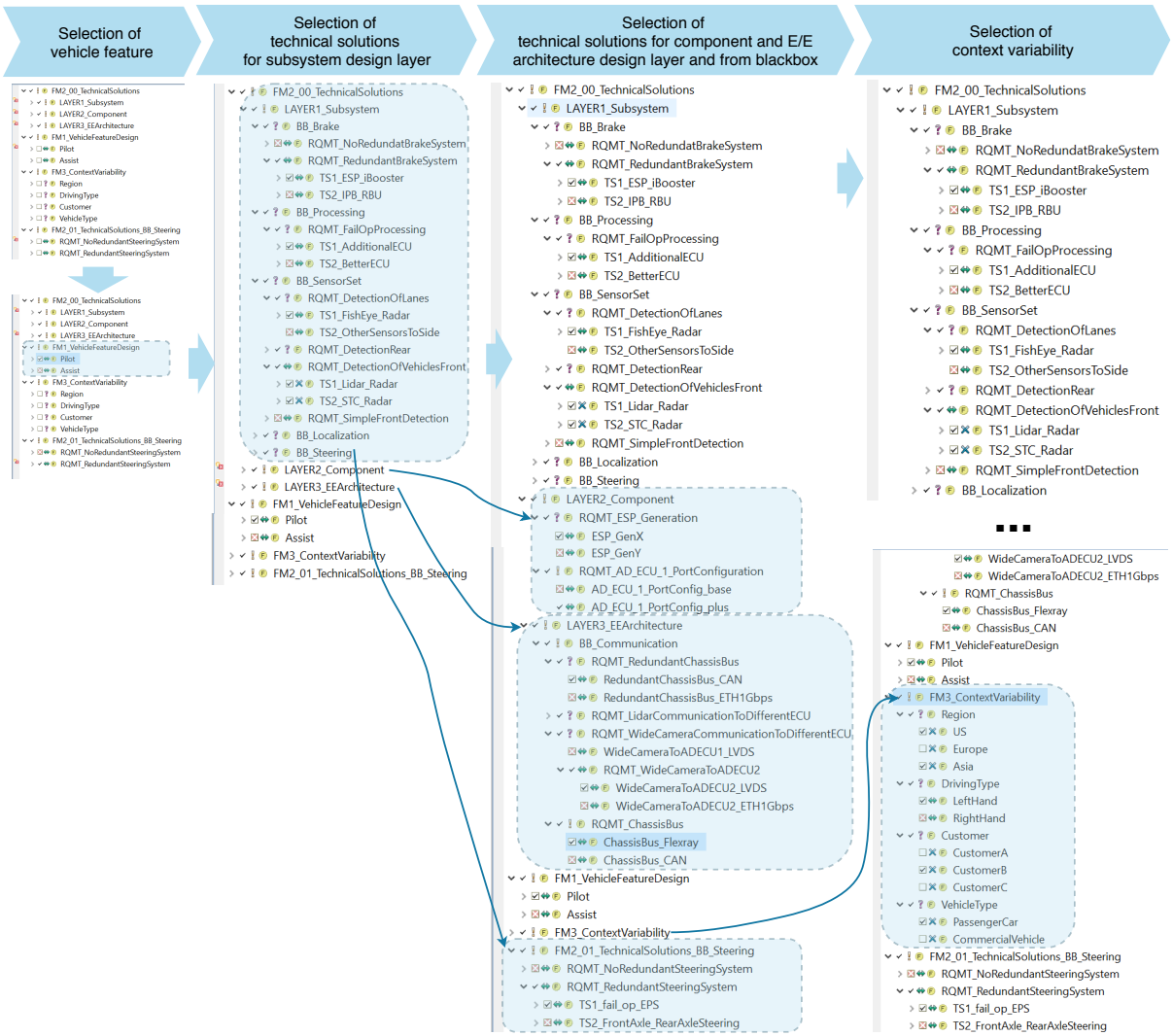

Figure 6.9.: System Variant Configuration of *Pilot_OptionA* in its Variant Model `Pilot_OptionA.vdm`

Depending on the types of variability, the model element to which the *Constraint* shall be attached differ as follows:

- To *Blocks* in BDD: for component variability, i.e. whether a particular component exists in the system or not, mostly in *LAYER1 Subsystem* and also in *LAYER2 Component* of technical solution feature models
- To *Ports* in IBD: for port configuration variability of ECUs in *LAYER2 Component* of technical solution feature models
- To *Connectors* in IBD: for the communication network variability in *LAYER3 E/E Architecture* of technical solution feature models

Figure 6.10.: Configuration Result of *Pilot_OptionA* and *Assist_OptionA*

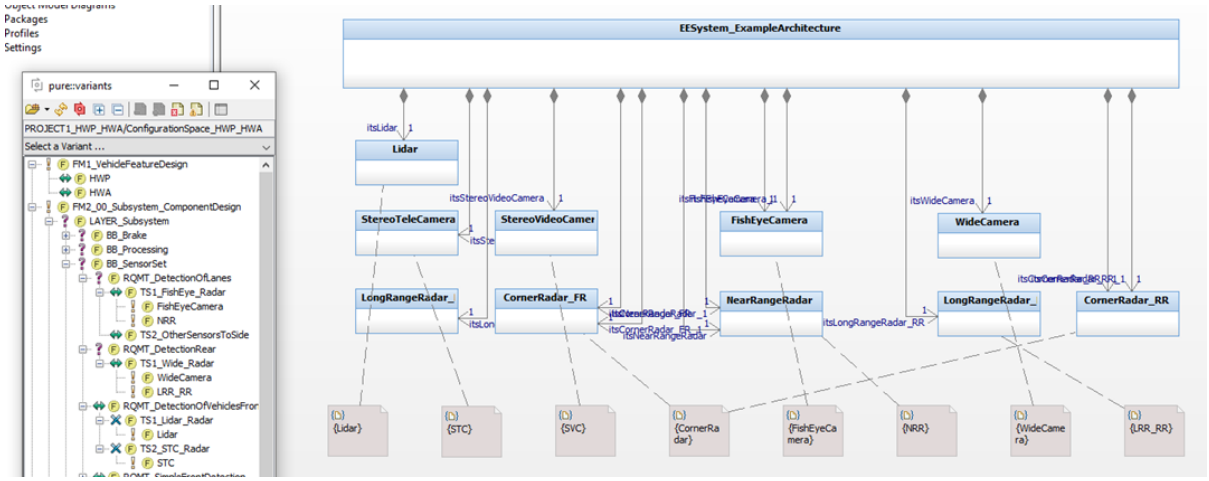

Figure 6.11.: Pure::Variants Configuration Space Connected to Technical Architecture Model

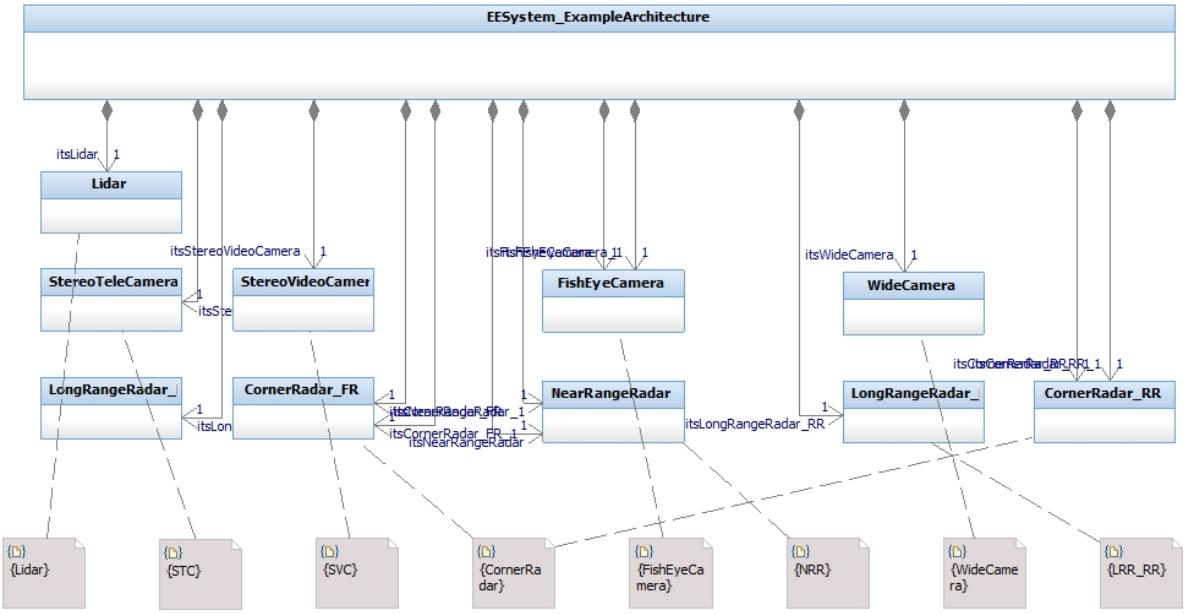

Figure 6.12.: BDD_SensorSet: *Constraints* attached to *Blocks*

Figure 6.12 shows the BDD_SensorSet of the example architecture where the *Constraint* referring to the technical elements in the feature models are attached to the corresponding *Blocks* in the architecture model.

In Figure 6.13, the features *AD_ECU_1_PortConfig_plus* and *AD_ECU_1_PortConfig_base* are attached to the corresponding *Ports* of *AD_ECU_1*. In addition, the features referring

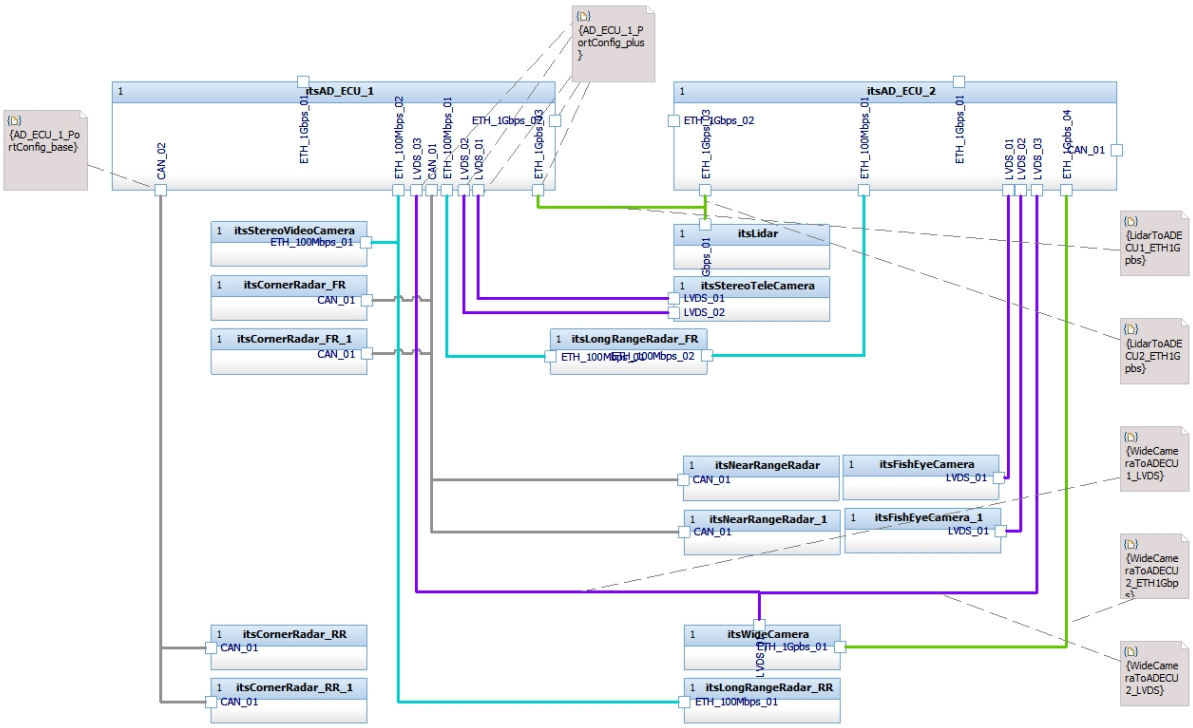

Figure 6.13.: IBD_SensorSet: *Constraints* attached to blocks

to the communication network variability, e.g. *LidarToADECU1_ETH1Gbps*, are attached to the corresponding *Connectors* in the architecture model.

6.2.9. Step 8 Variant transformation

The final step is to perform the variant transformation in Pure::Variants for each system variants. As the results of the variant transformation, the technical architecture model (100% model) which corresponds to the system variant is automatically produced. It means that only the model elements (*Blocks*, *Ports*, *Connectors*) that exist in the specific system variant remain in the resulting transformed model and the non-relevant elements are removed. Figure 6.14 shows a part of the transformed model for *Pilot Option A* and *Assist Option A*. For simplicity and easier comparison, here we only show BDDs and IBDs for the *SensorSet* building block. The transformed models that include the whole building blocks are shown in Appendix C – Transformed Model of Example Technical Architecture.

We checked the transformation models for all system variants (*Pilot Option A*, *Pilot Option B*, *Pilot Option C* and *Assist Option A*, *Assist Option B*), and we could find that they were correctly transformed. Since, in performing this implementation, we followed the exact

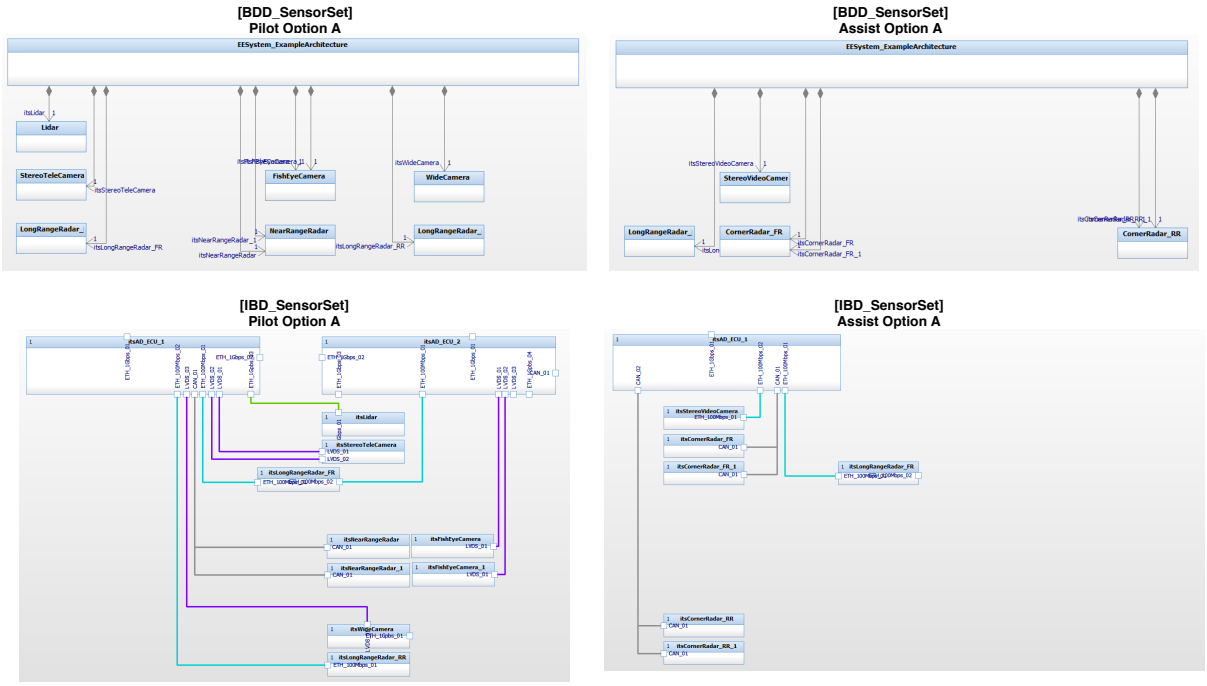

Figure 6.14.: Transformed Model: BDDs of Pilot Option A

'Process of using the method' introduced in Section 5.2.7., we could conclude that the concept of this method was successfully proved on the example technical architecture.

Chapter 7

Evaluation

As the final step of this study, we evaluated the method based on the feedback from the stakeholders. We arranged a teaching session and invited the stakeholders that we had interviewed. The teaching session lasted about one hour, and there we explained the concept of the method and presented the results of the implementation. In order to obtain concrete feedback for the essential attributes of the method, we created an evaluation questionnaire. After the teaching session, the questionnaire was sent to the stakeholders, along with an additional document, which contains a detailed description of the method. In this chapter, we present the results of the evaluation.

7.1. Evaluation Questionnaire

In the design of the evaluation questionnaire, we considered the two aspects - scientific evaluation and practical evaluation. Since the method shall be applied and used in practice and the method was developed based on the requirements derived from the needs of the stakeholders, it is crucial to evaluate the method on the practical criteria. Therefore, in the practical part of the questionnaire, we asked the respondents to evaluate the method in terms of the primary concepts of the method, which we derived from the requirements. The scientific evaluation was also added in order to assess our method based on the proven criteria for feature modelling. The evaluation criteria are formulated based on the recommendation for "Requirements to feature modelling notations [DS06]"

The evaluation questionnaire consists of three parts with the following contents:

- General

  0-1 Conduct of training (Yes / No)

0-2 Industry experience (in years)

- Part 1. Evaluating the method and the process of using it
  - 1-1 Evaluation in terms of "modelling variability of the system"
  - 1-2 Evaluation in terms of "ensuring traceability"
  - 1-3 Evaluation in terms of "blackbox concept"
  - 1-4 Evaluation in terms of "modelling variability outside of the system"
- Part 2. Evaluating the feature modelling notation and the structure of feature models
  - 2-1 Evaluation in terms of the criteria "Type distinctions"
  - 2-2 Evaluation in terms of the criteria "Dependencies"
  - 2-3 Evaluation in terms of the criteria "Simple and Expressive"
  - 2-4 Evaluation in terms of the criteria "Readability"
  - 2-5 Evaluation in terms of the criteria "Standardizeability"
  - 2-6 Properties of the method that would increase the standardizeability (Multiple choices)
    - Traceability links from requirements to technical architecture
    - Technical solution feature model which is structured based on technical solutions
    - Context variability feature model
    - Blackbox concept
    - Using the standard tool Pure::Variants
- Part 3. General Feedback
  - 3-1 Positive aspects of the method
  - 3-2 Negative aspects of the method
  - 3-3 Evaluation on the applicability of the method in the stakeholders' work areas

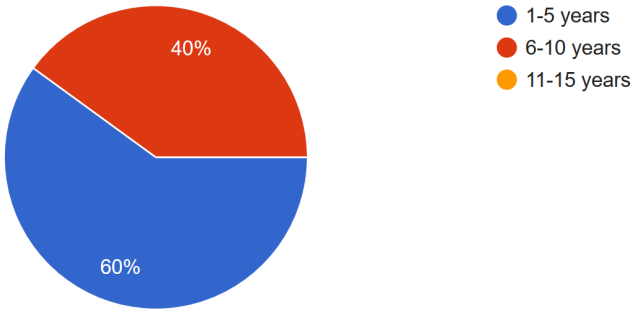

Figure 7.1.: Industry Experience of Respondents

7.2. Evaluation Results

One week after the questionnaire was given to the stakeholders, we analyzed the answers. There was a total of 10 respondents, and 8 of them conducted the training, and the other two were not able to participate but learned the method by reading the detailed hand-out document. Figure 7.1 shows the respondents' industry experience in years.

The answers for the main part of the evaluation, which we asked them to evaluate the method based on a scale from 1 (Very Bad) to 5 (Very Good), are shown in Figure 7.2.

To most of the evaluation criteria, more than 80% of the answers were positive, either 4 (Good) or 5 (Very Good). The remaining 20% was neutral, 3 (Normal). This fact shows the high level of general satisfaction on the method. Out of the practical criteria in Part 1, the respondents evaluated *Blackbox concept* with the highest score, 90% positive (70%, Very Good, 20% Good) and 10% Normal. One of the respondents provided comment saying that he or she "*finds the* Blackbox concept *very helpful because it will improve the collaboration process between different architects and stakeholders*". The answers for *Modelling variability* both for inside and outside of the system were also positive, 90% Very Good or Good. Together with 80% of positive answers to the criteria *Type distinctions* and *Dependencies* in Part 2, we could see that the respondent evaluated the most fundamental function of variant management positively - modelling variability and showing dependencies between features. As to the *Traceability*, 60% of the answers were positive, 30% neutral, but there was one negative answer indicating 2 (Bad). We were able to relate the non-positive parts of the answers to the comments provided in the General section. One respondent found that "*although it is important, ensuring traceability from requirements to the architectures over different business units in the organization may be difficult. Since each business unit is currently using different approaches and methods, aligning these would require quite an effort*". However, this is more about the organizational issue rather than the feedback to the method itself. In

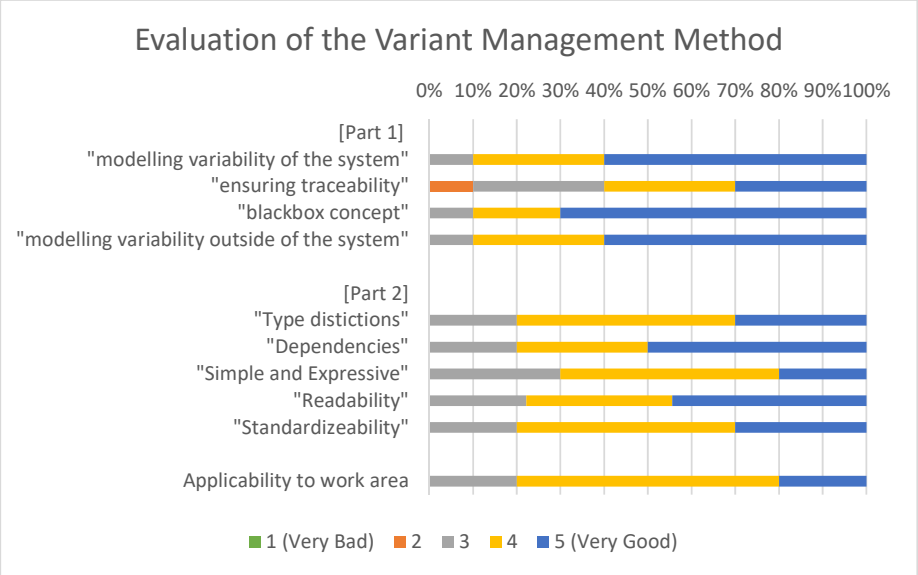

Figure 7.2.: Evaluation Results for Part 1 and Part 2

this method, we presented the concept on how the traceability could be ensured through the feature models, and alignment and applying this approach to different business units and subsystems were not a part of this thesis but left as future work.

The answers to *Simple and Expressive* and *Readability* were also mostly positive. It shows that the structure of the feature models and feature modelling notations in this method are simple and straightforward enough and easily understandable. Several respondents provided comments about the structure of the feature models; they mentioned that "*the feature models are clearly structured*" showing the hierarchy of features and components in the system. They also stated that "*the tree structure of the feature models gives an easier overview of the interconnections and interdependencies of the features, which would also help in the standardization of the way of modelling variability*". Besides, there was a comment stating that "*the method improved the way of showing constraints and dependencies of the features, which are, in today's setting, quite implicit and hard to capture*".

About the overall structure of the method and process of using it, the respondents mentioned that "*the method is clearly structured and simplified the variant configuration steps*". The respondents further indicated that "*the method enables the possibility to be extended down to the whole hardware and software design steps*" which exist in the levels below the technical architecture design step in the V-Model, by providing information

on the reasonable feature combinations transparently. They stated that "*having this method in the technical architecture level would benefit the organization overall since it provides a clear overview of the valid feature combinations on a higher level. This could help the architects and developers in other domains or subsystems/components to keep a wider perspective from the vehicle level*". It would enable them to focus on the development, which is reasonable for the wider perspective of the system, or of the vehicle at all.

Furthermore, there were comments about the overall research of this thesis. The respondents also noted that "*the analysis of the variability related problems in the vehicle level itself was helpful, especially dividing the technical solutions into different levels which are existing in the horizontal direction and the vehicle feature design which is performed in the vertical direction*". They indicated that "*The structured approach to get the requirements (to the variant management) from the stakeholders and afterwards collect their feedback was very helpful to keep us aligned*". Besides, a respondent commented that he or she "*found the implementation of the method on the tools especially helpful, which took the variant handling challenges into practice*".

However, some respondents pointed out "*high tool complexity*". In today's development environment, there are already different tools used for different purposes in the whole system engineering process, e.g. a requirement management tool, architecture modelling tools. The way this method was implemented was introducing yet another tool for feature modelling, requiring the users to learn to use it with the existing tools jointly. Plus, one respondent stated that it requires "*high effort in creating feature models*" with consistent notations and constraints. Further, one comment notes that "*Although the process of using the method is clearly defined, it could be complicated for the users who do not already possess knowledge on the topic*". Plus, the importance of "*ensuring consistency of the imported feature models*" were also mentioned. A few respondents noted that the feature models do not include other types of features, for example, functionality, technology, or property that the users interact with. However, since the focus of this method was the technical architecture, the features in our feature models should be technical features - mainly technical elements rather than functional elements, which are part of the functional architecture. This part can be further explored in the future work, extending this method to neighbouring design steps - i.e. the functional architecture, subsystem architectures.

Moreover, there was feedback pointing out that "*the application of the method could be limited to the systems which already achieved a certain level of maturity in the feature dependency*". This is because, the method requires a quite strict set of dependencies between features. However, in the early phase of development or for an immature system, the dependencies could not be formulated yet to the required level. One respondent also indicated the lack of *"review and validation methodology"* to check whether the feature models that were created are correct or not.

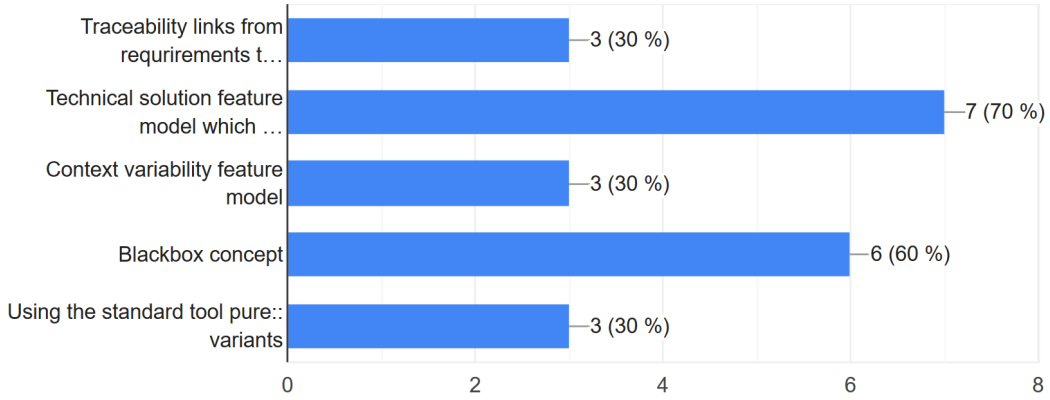

Figure 7.3.: Evaluation Results for Standardizeability

As to the *Standardizeability*, 80% of the responses was positive, 50% Good, 30% Very Good, and the remaining 20% was Normal. Standardizeability is an especially important factor for automotive systems. The automotive systems are developed in a highly-distributed environment over different business units and organizations; therefore, it is crucial to have a standardized method or a modelling concept across domains and different design steps in V-Model. Also, more and more cross-domain features are applied to vehicles due to the increasing number of automated driving features. Hence, in order to obtain more in-depth feedback in terms of the standardizeability, we further asked the respondents which exact properties of the method would help the most. The result is shown in Figure 7.3.

More than 70% of the respondents found the structure of the feature models based on technical solutions would increase standardizeability, and 60% said *Blackbox Concept* would be helpful. Structuring the feature model based on the technical solution was the core principle and the significant contribution of the method, which we aimed to help improve traceability from high-level requirements, technical solutions to technical elements in the technical architecture. Also, the purpose of *Blackbox Concept* was to help on the collaboration process between different parties from different design levels or different domains. Additionally, the answers to the *Applicability to work area* were also positive, 80% positive and 20% neutral. Considering that the respondents evaluated the method from a broader perspective in the organization, already considering the alignment that needs to be made with other stakeholders, this positive answer for applicability shows a good potential of this method, that can be extended over further design steps and other domains to increase the efficiency and quality of automotive system development.

Chapter 8

Discussion

In this chapter, we discuss and interpret the results and findings of the research.

8.1. Understanding Context and Stakeholder Definition

To gain understanding of the context, we asked the following research question: *RQ1: What is the current situation regarding the variant management of system architecture? Who are the stakeholders?*

Through the stakeholder interviews, we were able to gain an in-depth understanding on the current situation, especially regarding the difficulties that the system and E/E architects face when designing the technical architecture of the system. In designing the technical architecture of a highly-automated driving system, the system architects and E/E architects need to make design decisions on different levels. The system can be designed with different vehicle feature, for example, either with L3 level pilot feature or L2 level assist feature. The system is then realized with different subsystems in different domains of the vehicles such as braking, steering, sensorsets and processing. In order to determine the design decisions for each part of the system, the architects need to decide 1) what components they would choose to realize the functionality, 2) what communication connections they would select for these components, and 3) how the power supply system should be structured. For this, the system architects need technical information of the components such as product generation, power supply levels, signal messages, the number and types of ports, message datarate, voltage levels, energy consumption, safety level, and much more. However, the problem is that this information is distributed over different sources, mainly available in the product owner/developer side, so the system architects and E/E architects would need to contact

these departments and manually get this information, in order to design the technical architecture.

More importantly, we have numerous variants arising in different levels. Subsystems can be designed with different design options, and we have variability in the component design level as well. Also, we have variability in the E/E architecture design – both for communication network and power supply system design. Here, the necessity of an adequate variant management method, which can address these issues is high. However, there exists no proven variant management method for the technical architecture level. This thesis was therefore aim to develop a variant management method for technical architecture, whose concrete activities include the followings - increasing accessibility of information, showing explicit information about system variability, showing interdependency between features, deriving valid feature combination, and ensuring traceability from requirements to design artifacts.

In the organization, the variant management method for the technical architecture needs to be understood in the context of adopting model-based system engineering (MBSE) approach. Due to the inefficiency and disadvantages of the traditional document-based system development process, more and more business units in the organization have been adopting the model-based approach. The high-variance nature of automotive systems which is being further strengthened with the new automated driving features is adding importance to having an adequate variant management method for technical architecture. In this sense, in order to identify who the relevant stakeholders are, we asked ourselves the question "*who is working with the technical architecture model*". The system architects for the vehicle level systems such as Highway Pilot or Blind Spot Detection and the E/E architects for these systems shall be the creators of technical architecture models. Then these models would be taken as an input to the later design steps, such as subsystem and hardware/software architecture design. Considering this, we first defined the three design levels - vehicle level (vehicle-level system development), subsystem level and component level. For each levels, we selected the roles of stakeholders that we should take into account in identifying use cases and deriving requirements - *System Architect*, *E/E Architect*, *Function Developer*, *Product Manager*, and *Software Architect*. As mentioned, the *System Architect* and the *E/E Architect* were the main stakeholders as they are the main parties designing the technical architecture, creating the models and dealing with variability issues. The *Function Developer* was considered because functional architecture design is usually the step taken before the technical architecture design; hence, it is necessary to take into account the neighbouring design steps. The *Product Manager* was considered in order to include inputs from the general user side. The *Software Architect* for the vehicle-level system was unfortunately not available; therefore, only the *Software Architect* for the subsystem level was interviewed. However, during the interview, we found that they normally take subsystem-level requirements which were formulated by the system architect in the subsystem level, and taken it as

an input for the software architecture design. Therefore, the relevance to the technical architecture model itself was rather low for these stakeholders.

8.2. Use Case Identification and Requirement Derivation

In order to identify the use cases and to derive the requirements for the variant management method, the following research question was asked: *RQ2: What are the use cases of the stakeholders regarding variant management? What requirements can be derived from the use cases?*

During the interviews, we were able to gain an in-depth understanding of the needs and the challenges that the stakeholders have. Beyond the use cases such as *Model-based system architecture design*, *Visualize technical architecture and its variants* and *Show technical information of components* which are quite expected, considering the model-based system engineering approach, we could learn about further use cases. Especially, the use case *Show traceability from requirements to technical solutions* played a major role in structuring our feature models. It became the motivation to establish the traceability links through the feature models.

Moreover, from the use case *Feature engineering*, we were able to find out that two different directions of design activities are involved to design the technical architecture. These refer to the design of technical solutions for subsystem, component and E/E architecture design levels which are spread out throughout different domains of vehicle in the horizontal direction, as well as the vehicle feature design which is performed by selecting specific technical solutions for the required functionality of the vehicle feature - in the vertical direction (shown in Figure 5.3). It was essential to identify this point so that we know how we should structure our feature models.

Also, we were able to learn about stakeholders' needs to use the models for review purposes. Several stakeholders mentioned that they need supports in producing the technical architecture model on the desired level of abstraction. Based on needs, some information in the technical architecture could be irrelevant or unnecessary. Furthermore, the stakeholders in the subsystem or component design level use the technical architecture model to understand vehicle level requirements better. In this case, they need supports for a useful comparison of different variants of technical architecture, so that they could easily compare and contrast the variants. Further tool supports could realize these, but since we are focusing on developing the concept of the method in this thesis, it was excluded from the scope and left as future work.

Through the interviews, we could identify the stakeholders' needs to have the variability existing outside of the system also modelled. Having this information available in the

model helps not only the stakeholders who design the vehicle-level system but also the ones for the subsystem and component design, as they could gain a better understanding on the vehicle-level requirements in the broader perspective.

In the organization, it was already clear that they need an adequate variant handling approach; however, the unavailability of this approach could limit the stakeholders to recognize their needs for the variant management. In order to help on this, during the interview, we show them the typical needs for variant management which are presented in literature and asked them whether they would also have the same needs [OPS+17]. After the analysis of their answers, we were able to identify further needs for the variant management as follows:

- Providing high-level overview of different variants for different customer projects: to know which are major variants, which are used more, to know which combination is valid
- Showing high-level overview of features: what feature combination is possible for customers, type of control units that can be offered to customers, mandatory combination of other components for a certain component, to show links to the constraints that need to be considered for a certain component (ex: Parking Pawl as one of the constraints that need to be considered for ESP), to get information about the features of neighboring systems, helpful to have this overview in deriving product strategies for subsystems, to understand how a system is shaped to fulfil a certain functionality
- Increasing work efficiency: helpful on product configuration, automatic creation of a variant model, gathering information about different needs for E/E architecture design without having to check all documents by having a centralized information base
- Increase quality of products: help not to oversee some variants in technical solutions, enables early change and error detection, enables to crate product variants with valid feature combination, can have an impact to time-to-market, by ensuring consistency via a standard variant handling approach
- Increase accessibility of information: especially helpful for the distributed development environment, not having to rely on the availability of experts
- Handling complexity: also helpful for the subsystem level, by knowing only certain combinations are valid, the number of variants can already be reduced from the vehicle level
- Traceability and impact analysis: to know which features or components would be impacted when there is a change, in order to see whether specific function is still in use by any customers, helpful to have links between functions and

components to analyze impacts and influences in both directions, e.g. "if we have these components in the system, what functions can we have" and "if we need this function, what components should we have", current approach in DOORS is complicated because there is no graphical overview on dependencies

Out of these and analyzing the use cases relevant to the variant management, we derived the requirements for the variant management method of the technical architecture, which were presented earlier.

8.3. Variant Management Method Development

The following question was asked to develop the method: *RQ3: What kind of variant management method is suitable in order to satisfy the requirements?*

In the software product line engineering, the *product line asset development process* is explicitly distinguished from the *product development process*. In the phase of the *product line asset development*, developing architectures based on the analysis of a product line takes place, and the analysis is performed, which includes the requirement analysis and the feature modelling for the product line [KLD02]. The product line assets generated from the *product line asset development process* are then provided as input to the *product development process*. In the *product development process*, product requirement analysis, feature selection and architecture selection are performed in order to create products [KLD02].

In our case, product line assets here would refer to the vehicle feature for the system of interest, e.g. *Pilot* or *Assist* feature. Also, products correspond to system variant for each of the vehicle feature, e.g. *Pilot Option A*, *Pilot Option B*, *Assist Option A*. In the organization, the vehicle-level system development is still at the pre-development research phase; therefore a concrete development process is not yet defined. Consequently, in our method, we described the whole vehicle-level system development in one chain of a process rather than already dividing it into the product line and the product development process. In our process of using the method, the Steps from 1 to 5, where the necessary feature modelling is performed, correspond to the *product line development process*, and the remaining Steps 6 to 8, where the system variant is configured through feature selections, equate to the *product development process*. When the vehicle-level system development process develops beyond the pre-development research phase, the method can be extended and divided into two groups, based on the needs of the organization at that time.

Another reason why we have not yet divided the process into the *product line development* and the *product development*, at the current stage, it is hard to document requirements

for the system variants as in the way that the requirements for the vehicle feature are documented. This is because, the requirements to configure the system variant depend on many different external factors, such as markets needs, legal and commercial requirements in different regions, and also varies from customer to customer. Also, in many cases, the system variant shall follow the requirements from the customer side, for example, a customer can request to use a particular type of component in the system, or a specific component that is manufactured by another organization. Therefore, at this stage, the product configuration step in our method only requires the users to select necessary features based on the requirements that he gathered for that specific system variant.

As stated in Section 2.3.1, our focus was more on the variant management activities for the system variant configuration rather than the ones for the technical architecture design. However, the *Blackbox Concept* of our method conceptually covers the variant management activity *Ensure Accessibility of Component Information*. With this concept, we have a mean of exchanging necessary information modelled in a feature model. As explained, the alignment of how the model exchange shall be done and what is the agreed structure of the feature model is required. After this, this approach can be further proved and extended in the practical setting.

Moreover, during the stakeholder interviews, we have learned about different kinds of technical information that the stakeholders need to see in the models. They include component properties such as energy consumption, voltage level, message datarate, latency, and port information such as number and types of ports, as well as safety concept such as ASIL rating. Furthermore, this relates to the variability issue as well. When the technical information for the same component differs, this is also the component level variability that needs to be handled. In this sense, the kind of information that should be contained in the feature model and the level of detail of this information shall also be discussed in the alignment process.

One of the most significant contributions of this work is to present the possibility of structuring feature models in a different way. The traditional approach is to model the physical components under the categories following the functional decomposition of the vehicle. Our method structured the feature models based on high-level requirements and technical solutions so that we could establish traceability from the requirement to the technical architecture. It provides users with a clear understanding of the design of the system by documenting the rationale for a particular design decision. Considering how many and how often changes occur in an automotive system, both expected and unexpected, it plays a crucial role in effective change management. Plus, another benefit of this structure is that it corresponds to the technical architecture design process, where the system architects analyze system requirements and derive technical solutions to

satisfy them, which determine what technical elements shall exist for each different technical solutions.

8.4. Implementation

In order to prove the concept of the method, we implemented the method in Pure::Variants. As noted earlier, Pure::Variants has been widely used as a feature modelling tool in the organization; thus, we implemented the method in a way that could work on Pure::Variants. However, the development of the concepts of the method was performed independent of the tool functionality. Therefore, the method does not limit the selection of the tool for implementations.

In Pure::Variants, only one kind of feature exists in the feature model. Although the feature has different name and types such as mandatory or optional, there is no classification for the kinds of features. For example, in a case like our method, where we model different attributes as features in the feature models - high-level requirements, technical solutions and technical elements, further classification of the feature would be helpful. In the current version, the factors that can distinguish the features are only the name. Hence, when the user try to attach the constraint referring to the features onto the model elements in the technical architecture model (as described in Step 7 of the process of using the method), they need to find which features are referring to the technical elements, not the high-level requirements or technical solutions. Classifying the kinds of features with different icons would be much helpful in this process when the constraint manually needs to be attached to the correct model elements.

Using a separate feature modelling tool provides benefits by being flexibly incorporated into the existing toolchain and handling variability relevant challenges. On the other hand, introducing yet another tool to the current toolchain for the system engineering further increases tool complexity. It was also pointed out by a number of participants of the evaluation. As a possible solution, combining architecture modelling and variability modelling could be offered. In the coming years, the Object Management Group (OMG) releases SysML 2.0, which also covers variant modelling [Gro]. Implementing the concept of this method in the existing architecture modelling tool with SysML 2.0 standard could be performed as a feasibility study.

8.5. Evaluation

The following question was asked for the evaluation phase: *RQ4: Does the prototype method satisfy the requirements and evaluation criteria?*

Overall, from the answers, we could see that the respondents have answered the evaluation questions not only for the method itself but also from a broader perspective in the organization. This could be observed from the question regarding traceability. We could identify the respondents who have evaluated the traceability of this method with relatively low scale have commented that "*establishing traceability over different business units would be difficult because they are using different approach*". This point is more of an organizational matter, not feedback about the method itself, which should be discussed and achieved during the alignment between actual developmental parties from different business units, which shall take place in the future. Our method focuses on presenting a concept that could establish traceability. The average score for the *Applicability of the method* is 4 (Good), and this shows that the respondents found the method applicable.

As mentioned in Chapter 7, the satisfaction of the stakeholders was the highest for the *Blackbox Concept*. The needs for this concept could also be clearly identified during the interviews with the system and E/E architects from the vehicle level. From their perspective, having no visibility about the technical solutions for a particular subsystem was a big pain-point, and there were also difficulties regarding the low accessibility of the component information. In addition, for the stakeholders in the subsystem or component level, they also need a way to exchange models with the vehicle-level departments. The *Blackbox Concept* shows how this can be done by enabling a part of the system to be modelled as blackbox and filing this it by importing a feature model which is created by the product owners themselves.

An interesting finding is that the respondents who have shorter industry experience have given higher scores by 14% than the ones with longer industry experience. 14% could be not a meaningfully large number, but this difference occurs throughout all the scoring questions. The largest difference in the score was around 30%. From this, we could speculate that the stakeholders who are relatively new to the area tend to accept a new method relatively more willingly.

Chapter 9

Conclusion

In this final chapter, we conclude our work by summarizing the main topics and outcomes. Here we answer the research questions and suggest the areas where future research could take place concerning this work.

Conclusion

The goal of this thesis was to develop a variant management method for technical architecture. The motivation of this research arose from the industry needs in Robert Bosch. In the automotive system development environment, the number of variants that needs to be managed has significantly increased with the newly introduced highly-automated driving features. Especially in the technical architecture, there exist different levels of variants and variability occurs in different levels - vehicle feature design, subsystem design, component design and E/E architecture design, throughout the whole system engineering process. These variants have dependencies on one another, but the development for each design levels is performed in a distributed environment. In other words, the development of the system, the subsystems that comprise the system, and the components that consist of the subsystems are performed in different domains and development parties. Therefore, a need for a variant management method which effectively handles these variability and addresses variant management related challenges in the technical architecture design level arose. In the organization (Bosch), the adoption of model-based system engineering (MBSE) approach is widely taken place, replacing the traditional document-based approach. In this sense, the variant management method should also correspond to and work well with the MBSE process.

The main part of the research consisted of the use case identification, requirement derivation, method development, and implementation and evaluation. In order to

understand the context and learn about needs for the variant management method for the technical architecture, we conducted interviews with practitioners in relevant roles in the organization. Through the interviews, we identified a total of nine use cases for the technical architecture model, among which six of them have relevance to variant management. From these variant management-relevant use cases, we derived eleven requirements for the variant management method. After the prioritization and classification, we extracted four subject matters that we need to be addressed in the development of the method - *Variability in the system*, *Variability outside of the system*, *Traceability*, and *Blackbox Concept*. By solving each of the subject matters, we structured our method which utilizes the four feature models - *Vehicle Feature Design*, *Technical Solutions*, a so-called *Blackbox* and *Context Variability*. As a next step, in order to prove the concept, we implemented the method on an example technical architecture. Finally, the evaluation of the method was conducted based on the feedback from the stakeholders.

In each phase of the work, the following research questions were asked. Here we present the answers which we were able to find during the course of our work.

RQ1: What is the current situation regarding the variant management of technical architecture? Who are the stakeholders?

In the organization, although there exist some variant management methods under development, these are mostly only confined to subsystem or software development level. However, more and more variability arises in the technical architecture with the adoption of highly-automated driving features, and we need an adequate variant handling method to effectively address the variability related challenges on the technical architecture level.

Considering that the variant management method shall be applied to the model-based system development process in the organization, we selected the stakeholders to be the creators and the users of the technical architecture model. We defined three different levels of abstraction in the system engineering process - (vehicle) system-level, subsystem-level, and component-level. For each level, we specified the roles of the stakeholders who need to use the technical architecture model - system architects, E/E architects, function developers, software architects, and product managers.

RQ2: What are the use cases of the stakeholders regarding variant management? What requirements can be derived from the use cases?

Through the stakeholder interviews, we identified a total of nine use cases. The first use case is *Ensure Consistency*, which shall be a general use case for all other use cases, since ensuring consistency through a consistent modelling concept or notations should be achieved in all cases. We classified the remaining use cases into three categories as use cases for system design, for using the model, and for further improvement of the architecture design process. The use cases for further improvement of the architecture design process are the ones which are not yet applied in practice; however, the needs were identified from the stakeholder interviews. The use cases *Model-based system architecture design*, in which the model is used for the technical architecture design process itself, and *Feature Engineering*, in which the vehicle feature is designed with a different combination of vehicle functionality, belong to the first group. The second group include two use cases - *Visualize technical architecture and its variants* and *Show technical information of components*. The remaining four use cases belong to the last group - *Show traceability requirements to technical solutions*, *Show dependency between features*, *Define interface*, and *Feature-driven selection*. Among these use cases, we selected six use cases that have relevance to the variant management and derived the requirement for the variant management method from them.

Total eleven requirements are derived, and from these, we extracted four subject matters that need to be addressed in the development of the method. The first two subject matters - handling *Variability of the system* and *Variability outside of the system* are to satisfy the requirements *Levels of Variants*, *Interdependency of features*, *Automatic generation of variant architecture model* and *Show validity of feature combination*. The third subject matter was ensuring *Traceability*, which was for the requirements *Show relations between high-level requirements and technical solutions* and *Ensure traceability via feature model*. The last subject matter was *Blackbox Concept* for the requirement *Blackbox concept for the subsystem parts*. The method development is performed in the way of finding the structure of the feature models that can address each of the subject matters.

RQ3: What kind of variant management method is suitable in order to satisfy the requirements?

Our method utilizes the four feature models - *Vehicle Feature Design*, *Technical Solutions*, a so-called *Blackbox* and *Context Variability*.

The main feature model is the *Technical Solutions*, and here the *high-level requirement*s, *technical solution*s and *technical element*s are modelled as features. For the subject matter

Variability of the system, we defined the high-level structure of this feature model. It first has three layers each for different levels of variant - *Subsystem*, *Component*, *E/E Architecture* design layers. Under this layer, the building block structure is applied based on the functional decomposition of the vehicle. In addition, we also added dependencies between the features in different layers in order to model the existing interdependencies. For the subject matter *Traceability*, we came up with the detailed structure of this feature model, which is based on the high-level requirements and the technical solutions. Under the three layers, high-level requirements are first modelled as features. Under this, the technical solutions that satisfy the high-level requirements and then lastly the technical elements realize the technical solutions are added as features to the feature model. The high-level requirements in the feature models can be connected to the requirement specifications documented in the requirement management tool like Doors, and the technical elements are connected to the corresponding model elements in the technical architecture model. In this way, we established traceability links from the requirements to technical solutions and finally to the technical elements in the technical architecture.

The *Vehicle Feature Design* feature model has vehicle features, which correspond to the product lines, of a system. This model is used to design the vehicle features with a combination of the technical solutions which are available in the *Technical Solutions* feature model. In this model, the dependency onto the *high-level requirements*, *technical solutions*, which are specific to the vehicle feature is modelled. Having these dependencies enable simpler and more straightforward system variant configuration, reducing the possibility of making mistakes.

The so-called *Blackbox* feature model was created to address the subject matter *Blackbox Concept*. This feature model refers to the feature model, which contains technical solutions for a particular subsystem or a building block. Due to the invisibility of the technical solutions for a certain subsystem from the vehicle level, the system architects model the corresponding part of the feature model as blackbox. This *Blackbox* feature model, which was created by the product owner, is provided to the system architects. System architects then can import this feature model to the configuration space in order to fill the blackbox. This concept not only solves the visibility issue of the technical solutions for a sub-part of the system but also provides an approach to increase the accessibility of the information. This approach is helpful for the system and E/E architects by enabling them to easily gain the necessary technical information of the components, which they need for the design of the technical architecture.

Lastly, for the *Variability outside of the system*, we model the global variables which represent the variability outside of the system, such as markets, regions, vehicle types, and customers, in an external feature model, called *Context Variability* feature model. This feature model is also imported to the configuration space, and the values of

the global variables for each system variant can be designated during the variant configuration step.

RQ4: Does the prototype method satisfy the requirements and evaluation criteria?

The evaluation of the method was conducted based on the requirements for the method and the evaluation criteria for the feature modelling notation. For the requirements part, the respondents were asked to evaluate the method based on the subject matters of the method - *Modelling variability of the system*, *Ensuring traceability*, *Blackbox concept*, and *Modelling variability outside of the system*. The evaluation results were mainly positive, showing a high level of satisfaction. 90% of the respondents evaluated the subject matters *Modelling variability of the system* and *Modelling variability outside of the system* as positive - 60% as 'Very Good (5)', and 30% as 'Good (4)' and the remaining 10% was neutral. The *Blackbox concept* was evaluated mostly highly, 70% 'Very Good (5)', and 20% as 'Good (4)' and 10% was neutral. The result for *Ensuring Traceability* was 60% positive, 30% neutral and 10% negative; however, we were able to conclude that these non-positive answers come more from the organizational issue making the application hard in the organization, rather than the feedback on the method itself.

As to the evaluation criteria, we have selected the five most important requirements for the feature modelling notations [DS06]. For the criteria *Type distinctions* and *Dependencies*, which relate to the ability of the feature model describing necessary levels of variability, 80% of the respondents rated 'Very Good (5)' or 'Good (4)'. For *Simple and Expressive* and *Readability*, which are about the understandability of the feature model, more than 70% of the answers were positive, the remaining part being neutral. For *Standardizeability*, 80% of the respondents evaluated it as positive.

Finally, we also asked the respondents whether they think the method would be applicable to their work area. 80% of the respondents evaluated the applicability as positive (20% Very Good, 60% Good), and the remaining 20% was neutral. All in all, considering the majority part of the evaluation results were positive, it could be concluded that the method satisfies the requirements and the evaluation criteria well.

Throughout this thesis, we provided the following contributions:

C1 Characterizing the current challenges for the variant management of the technical architecture design in the context of the model-based system development for highly-automated driving systems

C2 Proposing a variant management method and a structure of feature models for the technical architecture of highly-automated driving systems which satisfy stakeholder requirements

C3 Implementation and evaluation of the method

One of the critical contributions of this work is to present the possibility of structuring the feature models based on technical solutions. The primary benefit of this structure is that we can establish traceability from the requirements to the technical solutions and then to the technical architecture, which is crucial considering change management of features. Moreover, in this way, the technical solutions can be documented in the feature model, whereas before this method, they only abide in the brain of the architects. Lastly, this structure corresponds to the design process of the technical architecture, which presents a possibility of supporting the design process better.

Future Work

In developing the variant management method for technical architecture, it was an essential factor to take in to account the requirements from the subsystem design level. Due to the limitation of the availability and time, we interviewed the practitioners from one subsystem development department - braking system. Other subsystem departments would have different kinds of requirements, and also some of them already have their own variant handling approach. Therefore, as future work, further requirements form other subsystem development parties shall be gathered and applied in the extension of this method.

In addition, applying perspectives from safety experts is also important. This is because, automotive systems are safety-critical, and the granularity for the technical architecture modelling shall be defined by the safety point of view. In this sense, taking the requirements from the safety side into consideration shall also be conducted as part of future work.

Furthermore, for the variability in E/E architecture, we only handled the variability in the communication network. The power supply system could not be covered in this thesis due to the lack of a clear architecture modelling concept. Therefore, extending this method to handle variability in power supply system shall also be covered in future work.

Among the requirements for the variant management method, due to the limitation of time and tool supports, we were not able to handle the requirements that require high-levels of further tool supports. These include providing supports for comparing architecture variants and creating views of the technical architecture model in the different abstraction levels based on the user needs. These can also be handled as future work.

Lastly, as the extension of the method, several matters can be considered. These include considering the early phase of the system or immature system where the dependencies between features are not yet fully determined. Also, extending this method to other design steps in the system engineering process, such as functional architecture shall be explored. In this case, the types of features that shall be included in the feature model should further be examined.

Appendix A

Systematic Literature Review Process

The process of the systematic literature review follows the detailed steps presented by C. Wohlin et al. in their book [WRH+12]. The review is structured into: *planning* and *conducting* steps.

A.1. Planning the Review

This section discusses plans and structures of the review. It presents necessary concepts which need to be clearly defined before conducting the systematic review.

A.1.1. Need for a review

Before diving into developing method, we need to identify whether there are existing techniques and to understand the state-of-the-art in the subject area. This is fulfilled by conducting a systematic review.

A.1.2. Research questions for the review

For this work, it is crucial to identify the state-of-the-art in the context of system modelling through meta-model and feature modelling. Since the modelling concept originates from in the area of software engineering, the existing techniques in the software level needs to be studied. In addition, since our focus lies on the system level which encircles both hardware and software levels, the proposed methods for the system level need a rigorous review. For these areas, we set the research questions for the review as follows:

RQ0.1 What is the state-of-the-art in meta-modelling for automotive systems?

RQ0.2 What is the state-of-the-art in feature modelling for automotive systems?

RQ0.3 Are there proposed methods for system level architectural description?

A.1.3. Review protocol

The review protocol for the systematic review defines important concepts that needed to be clarified before moving onto conducting the review.

Search strategy for primary studies For the identification of relevant literature, we conducted our search on the database as follows:

- IEEEXplore
- ACM Digital Library
- Google Scholar
- AUTOSAR Website

Table 3.1 shows the list of search terms that we used for the search of relevant literature. Search terms were selected in order to identify as many possibly related papers as possible which were filtered according to the relevance to the topic in the next step.

No.	Search Terms
1	variant OR variability OR variance
2	metamodel OR meta-model OR meta-modelling OR metamodelling
3	feature model OR feature modelling
4	variant handling OR variant management OR variant modelling
5	(automotive OR autonomous OR automated) AND system

Table A.1.: Search Terms

Study selection criteria After the search in the databases, the relevance of each paper to the topic was evaluated. We selected papers if at least one of the answers to the following questions is positive. We formulated the questions with bottom-up approach, so that the most specific question which have the highest relevance to the topic is asked first, then move upwards to the more general questions. In this way, we could broadly research approaches applied in other industries or other levels in the system engineering process (i.e. software architecture design level) on top of identifying the most relevant papers.

1. Does the paper discuss variant handling methodologies or feature modelling for the system architecture development of automotive systems?
2. Does the paper discuss variant handling methodologies or feature modelling for other levels in the system engineering process of automotive systems?
3. Does the paper discuss variant handling methodologies or feature modelling in other industries besides automotive?
4. Does the paper discuss variant handling methodologies or feature modelling in general? (i.e. not developed for industrial uses, more of presenting research in the academic arena)
5. Does the paper provide other relevant information to this topic (i.e. system modelling through meta-models, requirements for highly-automated driving systems, etc.)?

Study selection procedures For the selection of relevant studies, titles of the studies are observed first, and then keywords followed by abstract, introduction and conclusion. In addition, the list of references of selected papers were investigated in order to identify further relevant resources.

Data extraction strategy From the selected works, the following data is extracted:

- Domain of application
- Method description
- Summary of main results
- Definitions of important concepts which are necessary to understand the topic

Synthesis of the extracted data After the extraction of the target data, we analyzed the content to decide whether the approach proposed in the work can be applied to our research.

A.2. Conducting the Review

Based on the plans, the review protocol and the criteria created in the previous section, the systematic review was conducted.

A.2.1. Selection of primary studies

Selecting the relevant papers was performed by applying the study selection criteria and the process which were defined in the planning stage of the review. In addition to the primary search that was conducted on the database, additional manual search was carried out according to the further clarification of relevant terms and concepts that arose during the course of the review process. After the search on the databases, based on the selection criteria, 38 papers were selected as primary studies and further reviewed in the next steps.

A.2.2. Data extraction

From the selected primary studies, the data that was specified in the planning phase is extracted. The extracted data can be categorized into: model-based system engineering for automotive systems, system modelling through meta-models, variant management and feature modelling. The content from data extraction is presented in detail in the next section.

A.2.3. Data synthesis

For the synthesis of extracted data, we used the *Thematic analysis* method [CD11; WRH+12]. *Thematic analysis* method is suitable when synthesizing inhomogeneous studies with a goal of identifying and analyzing patterns or themes in the primary studies [CD11; WRH+12]. As a result, we identified the lack of proposed and proven variant handling method for the technical architecture design of automotive systems.

Appendix B

System Variants of Example Technical Architecture

Here, we show the detailed structure of the example technical architecture. As explained in Chapter 4, in the example technical architecture, *AD System 1* is realized with five different system variants.

Table B.1 - B.4 summarize the variances existing between system variant of the vehicle features of *AD System 1*. Table B.1 is reprinted to help the users in the comparison of the system variants.

Subsystems	Pilot Feature	Assist Feature
Front SensorSet	Lidar + Radar + Video	Radar + Video
Processing	AD ECU 1 + AD ECU 2	AD ECU 1
Brake System	w/ redundant actuator	ESP only
Steering System	w/ redundant actuator	EPS only
Localization	IMU + Positioning	IMU
Chassis bus	Flexray + CAN	Flexray

Table B.1.: [Reprinted] Comparison of Pilot Feature and Assist Feature of AD System 1 in the Example Architecture

Figure B.1 - Figure B.6 shows the example technical architecture of each variant.

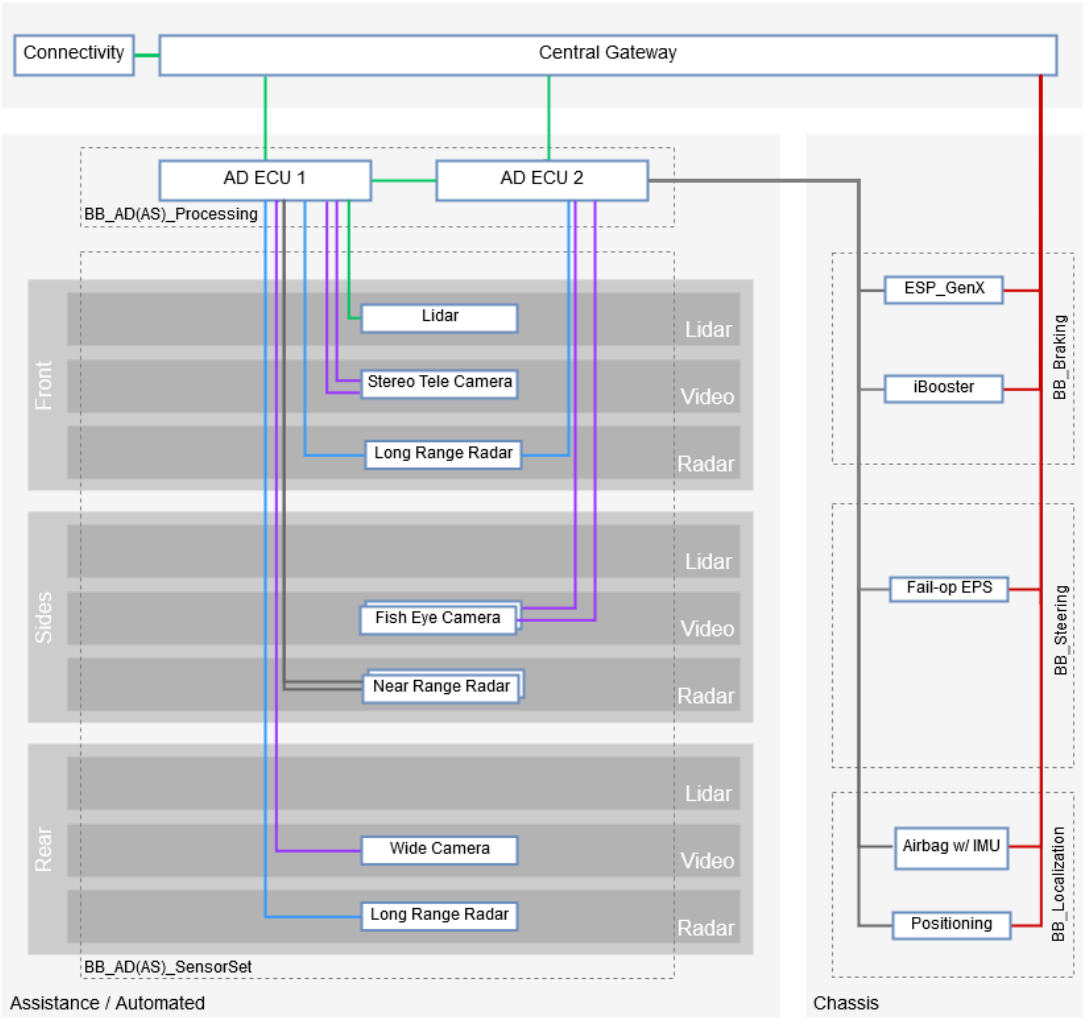

Figure B.1.: Example Technical Architecture: Pilot Option A [Rob19e]

CAN	AD	Autonomous Driving	EPS	Electronic Power Steering	IMU	Inertial Measurement Unit
Flexray	AS	Assist Driving	ESP	Electronic Stability Program	IPB	Integrated Power Brake
ETH 100Mbps	BB	Building Block	ETH	Ethernet	LVDS	Low Voltage Differential Signaling
ETH 1Gbps	CAN	Controller Area Network	Fail-op	Fail-operational	RBU	Redundant Brake Unit
LVDS	ECU	Electronic Control Unit	iBooster	Vacuum-independent Brake Booster		

Figure B.2.: Legend

Subsystems	Pilot Option A	Pilot Option B
Front SensorSet	Lidar + Radar + Video	Lidar + Radar
Brake System	ESP + iBooster	IPB + RBU
Steering System	Fail-op EPS	Front + Rear Axle
Comm. of Lidar	ETH 1Gbps to AD ECU 1	ETH 1Gbps to AD ECU 2
Comm. of Wide Camera	LVDS to AD ECU 1	LVDS to AD ECU 2

Table B.2.: Comparison of Pilot Feature Option A and Pilot Feature Option B

Subsystems	Pilot Option B	Pilot Option C
Comm. of Wide Camera	LVDS to AD ECU 2	ETH 1Gbps to AD ECU 2

Table B.3.: Comparison of Pilot Feature Option B and Pilot Feature Option C

Subsystems	Assist Option A	Assist Option B
Chassis Bus	Flexray	CAN

Table B.4.: Comparison of Assist Feature Option A and Assist Feature Option B

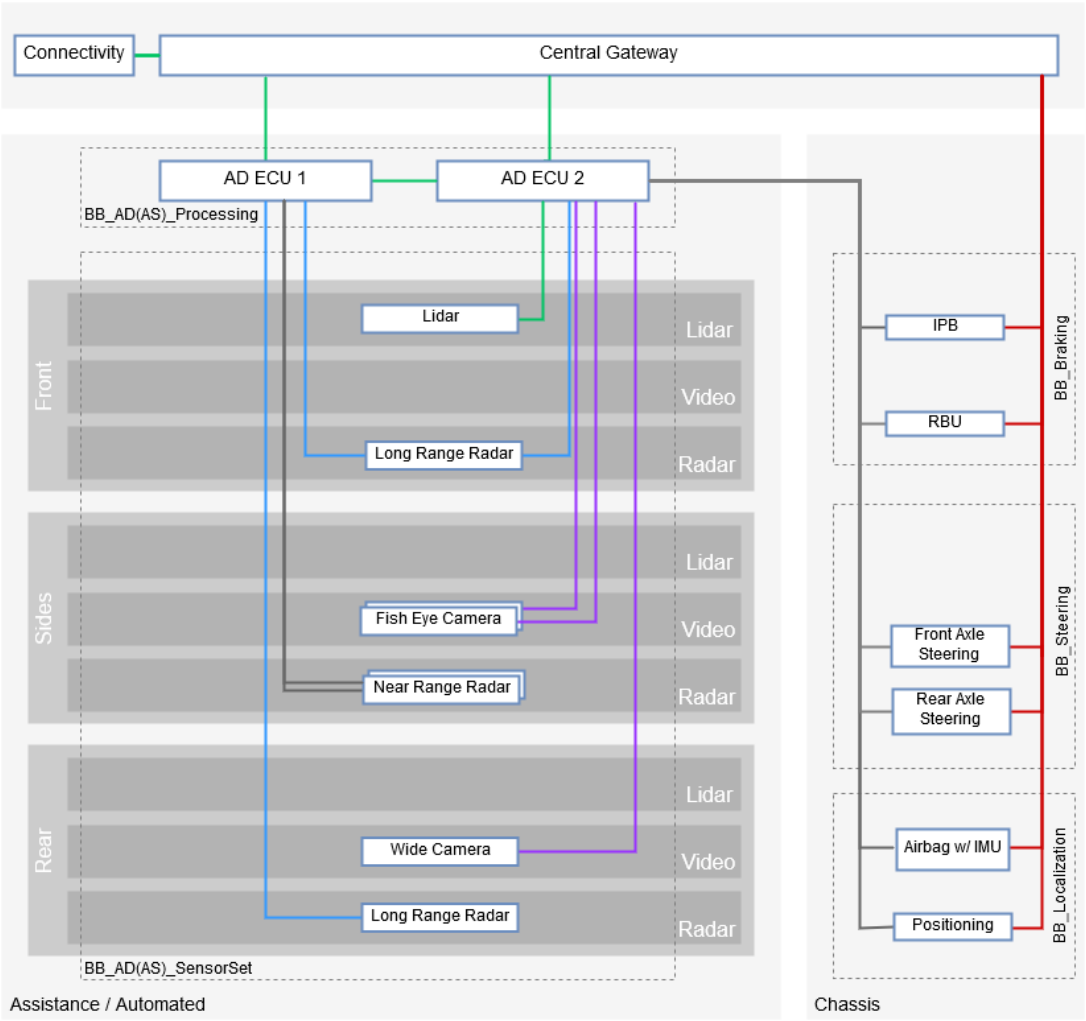

Figure B.3.: Example Technical Architecture: Pilot Option B [Rob19e]

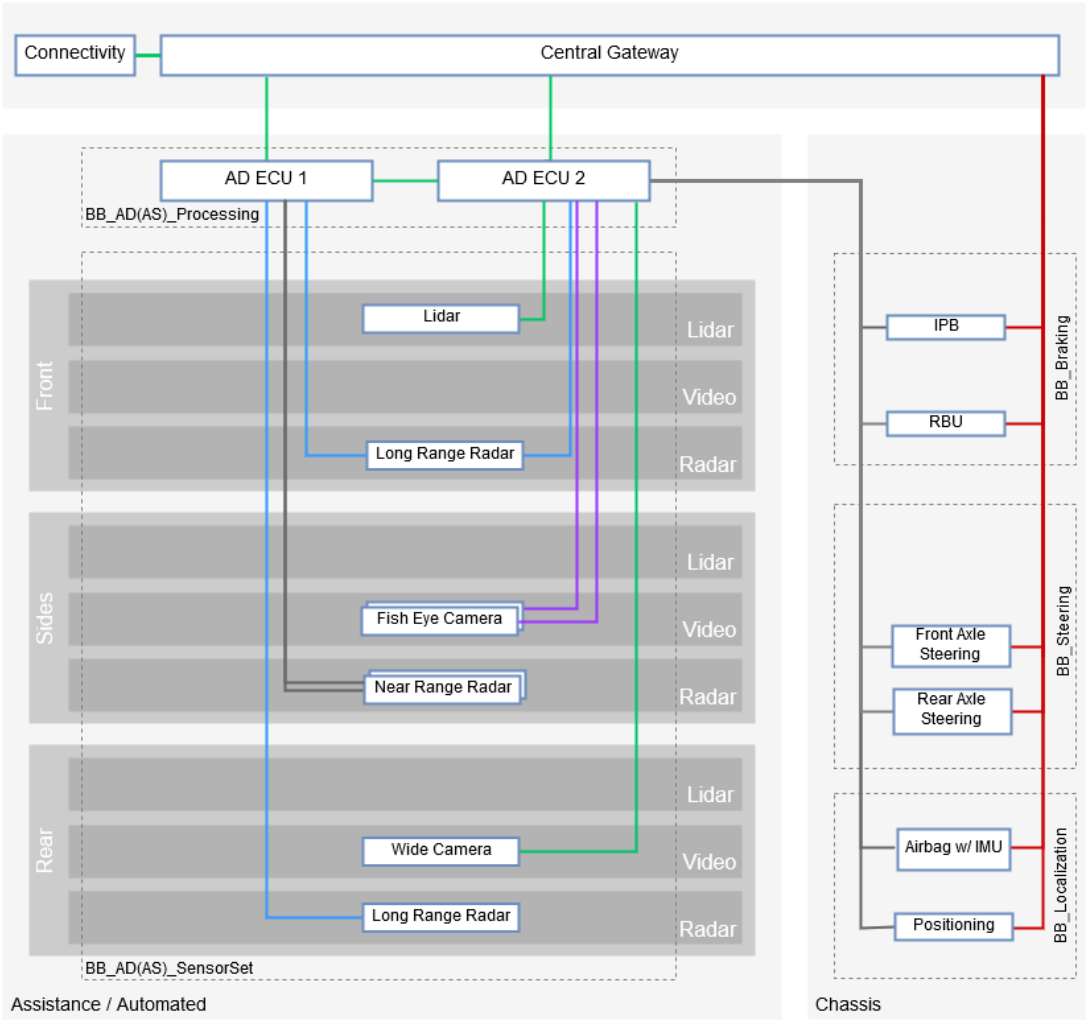

Figure B.4.: Example Technical Architecture: Pilot Option C [Rob19e]

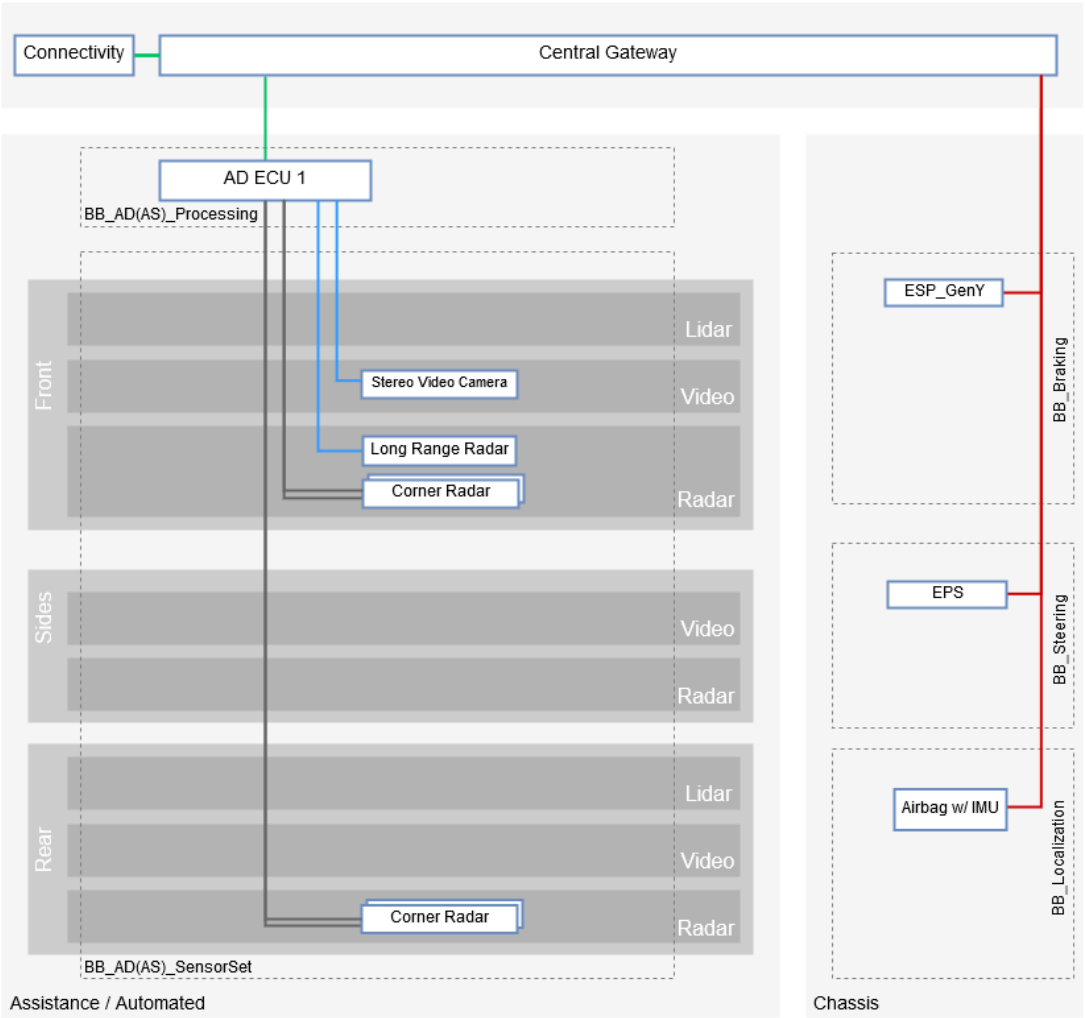

Figure B.5.: Example Technical Architecture: Assist Option A [Rob19e]

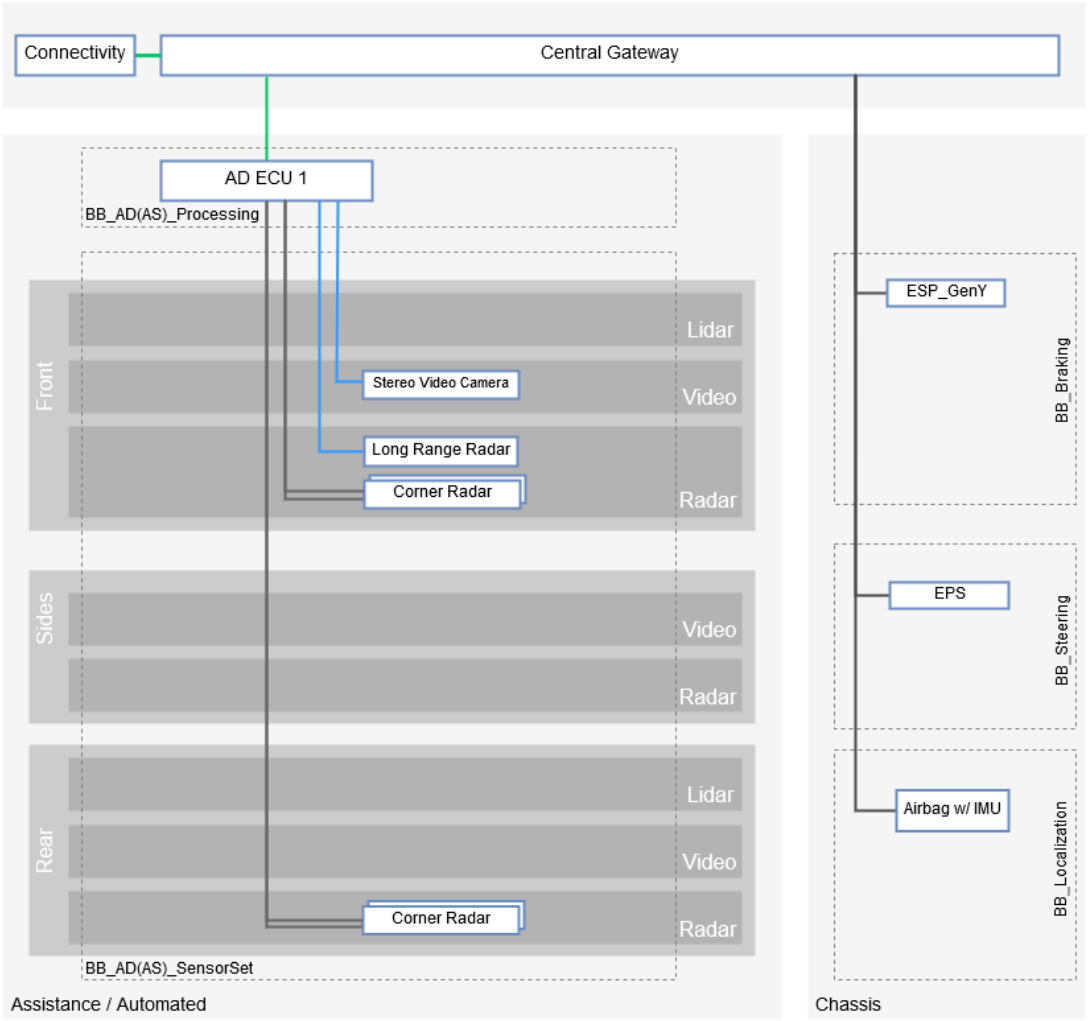

Figure B.6.: Example Technical Architecture: Assist Option B [Rob19e]

Appendix C

Transformed Model of Example Technical Architecture

In this section, we show the transformed model for *Pilot Option A* variant and *Assist Option A* variant. As explained before, for the implementation, we followed the 'Overall process of using the method' presented in Section 5.2.7. This is the result of the 'Step 8 Variant transformation' of the implementation which was described in Section 6.2.9.

Figure C.1 and Figure C.2 show the transformed model for Pilot Option A variant - each for BDDs and IBDs. Figure C.3 and Figure C.4 are the transformed model for Assist Option A variant. We determined to show the models for these two variants in this section, since the variability in the technical architecture can be best seen and compared in these two variants. The transformed models for the other variants have also thoroughly checked and confirmed that they are correctly generated.

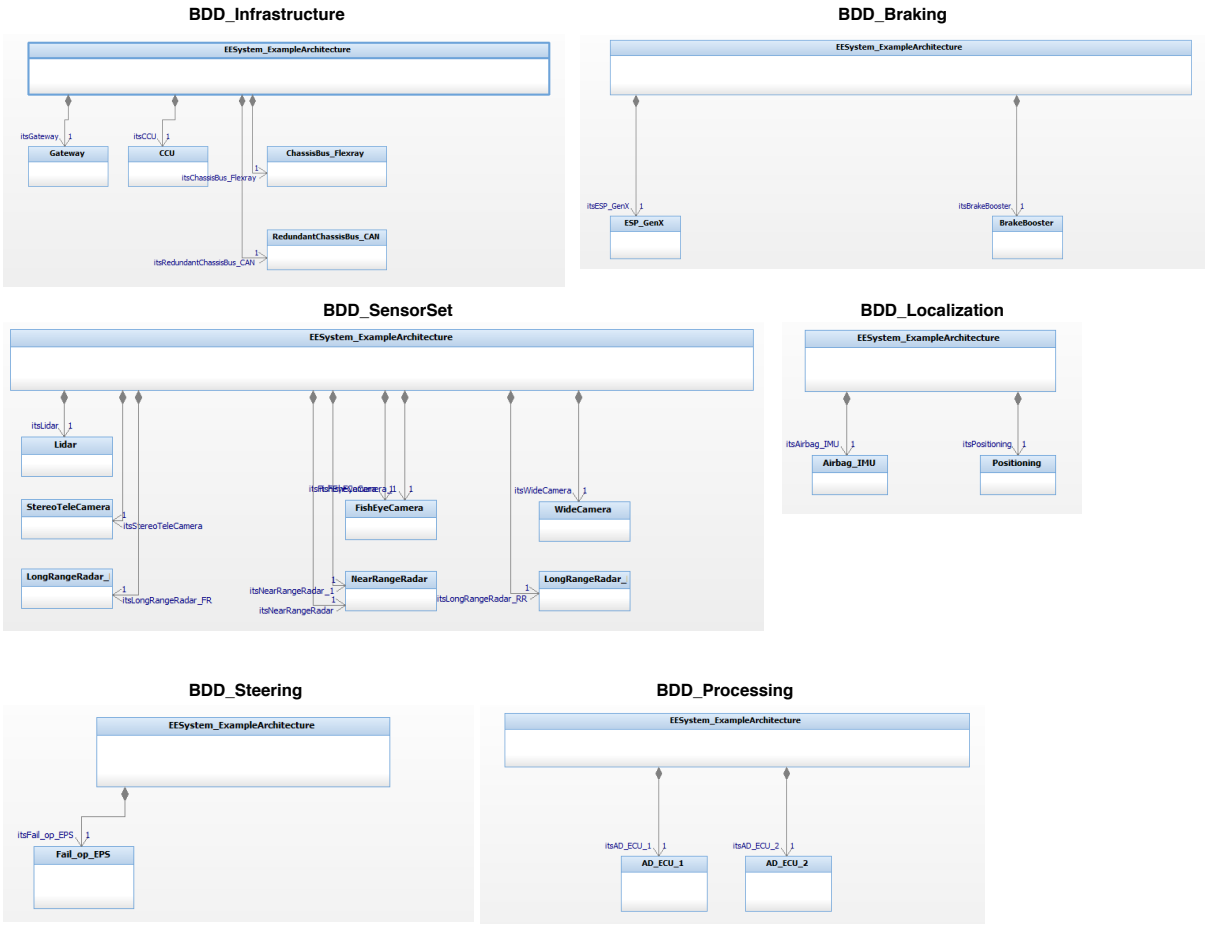

Figure C.1.: Transformed Model: BDDs of Pilot Option A

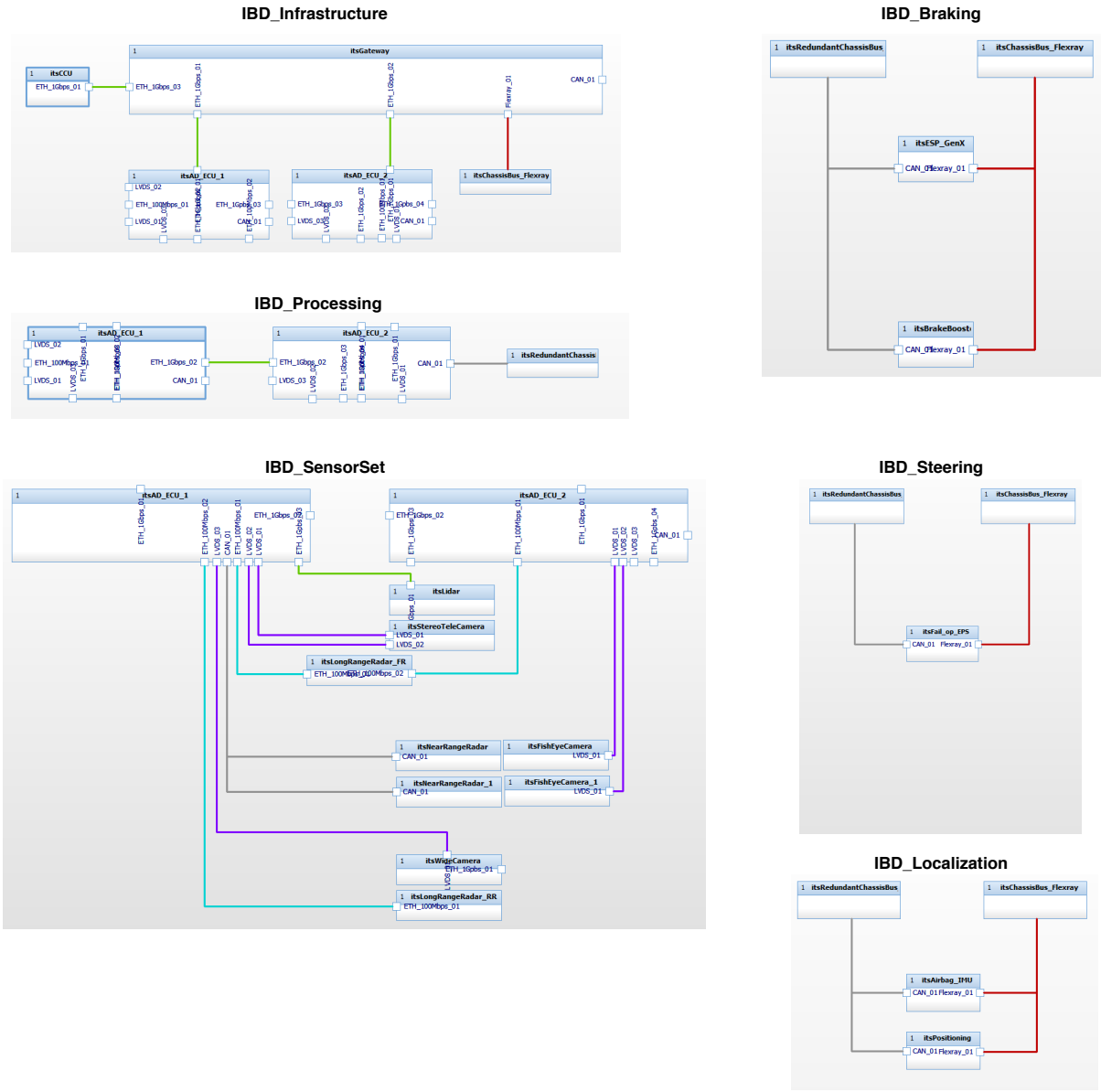

Figure C.2.: Transformed Model: IBDs of Pilot Option A

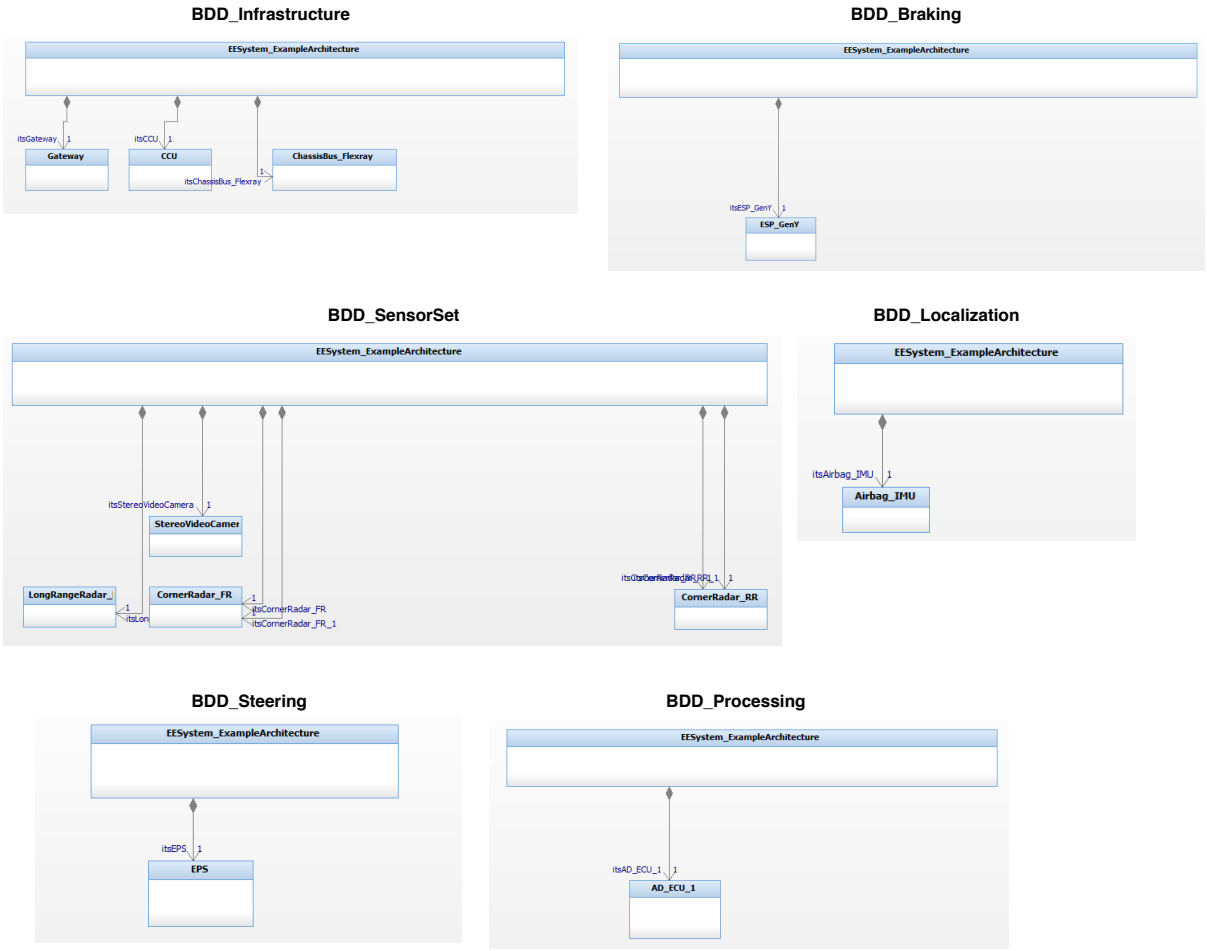

Figure C.3.: Transformed Model: BDDs of Assist Option A

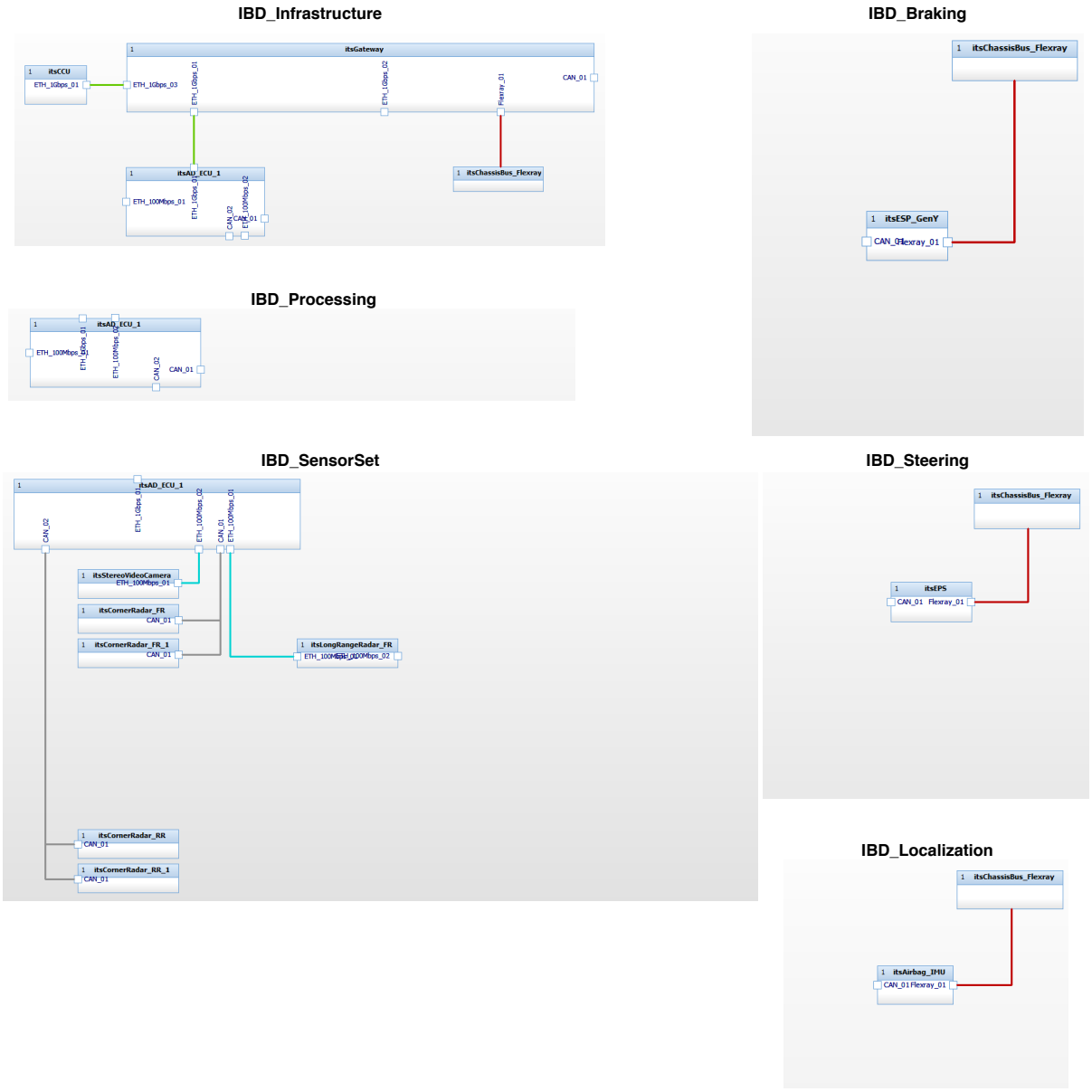

Figure C.4.: Transformed Model: IBDs of Assist Option A

Appendix C

Bibliography

[AP10] E. Andrianarison, J.-D. Piques. “SysML for embedded automotive Systems: a practical approach.” In: *Conference on Embedded Real Time Software and Systems. IEEE*. 2010 (cit. on p. 7).

[AUT17a] AUTOSAR. *AUTOSAR Feature Model Exchange Format, 4.3.1*. Tech. rep. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_FeatureModelExchangeFormat.pdf (cit. on p. 21).

[AUT17b] AUTOSAR. *AUTOSAR Generic Structure Template, 4.3.1*. Tech. rep. 2017. URL: https://www.autosar.org/fileadmin/user_upload/standards/classic/4-3/AUTOSAR_TPS_GenericStructureTemplate.pdf (cit. on pp. 15, 21).

[BLPW04] S. Bühne, K. Lauenroth, K. Pohl, M. Weber. “Modeling features for multi-criteria product-lines in automotive industry.” In: *Workshop on Software Engineering for Automotive Systems (SEAS), at ICSE*. Vol. 4. 2004 (cit. on p. 31).

[BN09] F. Bachmann, L. Northrop. “Structured variation management in software product lines.” In: *2009 42nd Hawaii International Conference on System Sciences*. IEEE. 2009, pp. 1–7 (cit. on p. 53).

[BPS04] D. Beuche, H. Papajewski, W. Schröder-Preikschat. “Variability management with feature models.” In: *Science of Computer Programming* 53.3 (2004), pp. 333–352 (cit. on p. 63).

[BSL+13] T. Berger, S. She, R. Lotufo, A. Wasowski, K. Czarnecki. “A study of variability models and languages in the systems software domain.” In: *IEEE Transactions on Software Engineering* 39.12 (2013), pp. 1611–1640 (cit. on pp. 16, 63).

[CB11] L. Chen, M. A. Babar. “A systematic review of evaluation of variability management approaches in software product lines.” In: *Information and Software Technology* 53.4 (2011), pp. 344–362 (cit. on p. 22).

[CD11] D. S. Cruzes, T. Dybå. “Research synthesis in software engineering: A tertiary study.” In: *Information and Software Technology* 53.5 (2011), pp. 440–455 (cit. on p. 104).

[CHE05] K. Czarnecki, S. Helsen, U. Eisenecker. “Formalizing cardinality-based feature models and their specialization.” In: *Software process: Improvement and practice* 10.1 (2005), pp. 7–29 (cit. on pp. 17, 20, 63).

[Cza98] K. Czarnecki. “Generative programming: Principles and techniques of software engineering based on automated configuration and fragment-based component models.” In: (1998) (cit. on pp. 17–21).

[Def05] U. D. of Defense. *Dictionary of Military and Associated Terms*. US Department of Defense, 2005 (cit. on p. 6).

[DS06] O. Djebbi, C. Salinesi. “Criteria for comparing requirements variability modeling notations for product lines.” In: *Fourth International Workshop on Comparative Evaluation in Requirements Engineering (CERE’06-RE’06 Workshop)*. IEEE. 2006, pp. 20–35 (cit. on pp. 17–21, 79, 97).

[DSTH14] D. Durisic, M. Staron, M. Tichy, J. Hansson. “Evolution of Long-Term Industrial Meta-Models–An Automotive Case Study of AUTOSAR.” In: *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE. 2014, pp. 141–148 (cit. on pp. 15, 16).

[GC15] J. P. Gaeta, K. Czarnecki. “Modeling aerospace systems product lines in SysML.” In: *Proceedings of the 19th international conference on software product line*. ACM. 2015, pp. 293–302 (cit. on p. 21).

[GFd98] M. L. Griss, J. Favaro, M. d’Alessandro. “Integrating feature modeling with the RSEB.” In: *Proceedings. Fifth International Conference on Software Reuse (Cat. No. 98TB100203)*. IEEE. 1998, pp. 76–85 (cit. on pp. 17, 20, 63).

[GKPR08] H. Grönniger, H. Krahn, C. Pinkernell, B. Rumpe. “Modeling variants of automotive systems using views.” In: *Proceedings of Workshop Modellbasierte Entwicklung von eingebetteten Fahrzeugfunktionen (MBEFF)*. Citeseer. 2008, pp. 76–89 (cit. on p. 21).

[GKS+07] C. Gillan, P. Kilpatrick, I. Spence, T. J. Brown, R. Bashroush, R. Gawley. “Challenges in the application of feature modelling in fixed line telecommunications.” In: *Proceedings of the First International Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2007), Lemrick, Ireland, Jan 16-18, 2007*. 2007 (cit. on p. 21).

[Gmba] P. S. GmbH. *Technical White Paper Variant Management with pure::variants*. Tech. rep. URL: http://www.pure-systems.com/mediapool/pv-whitepaper-en-04.pdf (cit. on p. 63).

[Gmbb] P.-S. GmbH. *Pure-Systems Website*. http://www.pure-systems.com (cit. on p. 31).

[Gmb19] P.-S. GmbH. *Pure::Variants User's Guide*. Tech. rep. 2019. URL: https://www.pure-systems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf (cit. on pp. 57, 63).

[Gro] O. (O. M. Group). *OMG SysML Portal*. http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-roadmap:sysml_assessment_and_roadmap_working_group (cit. on p. 91).

[Her00] A. Heritage. *The American Heritage Dictionary of the English Language. Boston*. Houghton Mifflin, 2000 (cit. on p. 5).

[HT08] H. Hartmann, T. Trew. "Using feature diagrams with context variability to model multiple product lines for software supply chains." In: *2008 12th International Software Product Line Conference*. IEEE. 2008, pp. 12–21 (cit. on pp. 51, 52).

[ISO08] ISO/IEC. "ISO/IEC 26514:2008 Systems and software engineering — requirements for designers and developers of user documentation, 4.21." In: 2008 (cit. on p. 5).

[ISO15] ISO/IEC. "ISO/IEC 26550:2015 Software and systems engineering — Reference model for product line engineering and management." In: 2015 (cit. on p. 5).

[ISO17] ISO/IEC. "ISO/IEC 24765:2017 Systems and software engineering – Vocabulary." In: 2017 (cit. on pp. 5, 6).

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson. *Feature-oriented domain analysis (FODA) feasibility study*. Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990 (cit. on pp. 1, 5, 17, 18, 63).

[KKL+98] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh. "FORM: A feature-; oriented reuse method with domain-; specific reference architectures." In: *Annals of Software Engineering* 5.1 (1998), p. 143 (cit. on pp. 17, 19).

[KLD02] K. C. Kang, J. Lee, P. Donohoe. "Feature-oriented product line engineering." In: *IEEE software* 19.4 (2002), pp. 58–65 (cit. on pp. 17, 19, 50, 89).

[OPS+17] O. Oliinyk, K. Petersen, M. Schoelzke, M. Becker, S. Schneickert. “Structuring automotive product lines and feature models: an exploratory study at Opel.” In: *Requirements Engineering* 22.1 (2017), pp. 105–135 (cit. on pp. 1, 10, 16, 18–20, 22, 26, 50, 51, 88).

[PG13] K. Petersen, C. Gencel. “Worldviews, research methods, and their relationship to validity in empirical software engineering research.” In: *2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement*. IEEE. 2013, pp. 81–89 (cit. on p. 32).

[PHAB12] K. Pohl, H. Hönninger, R. Achatz, M. Broy. *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer Science & Business Media, 2012 (cit. on p. 5).

[RBBS02] M. Rappl, P. Braun, M. von der Beeck, C. Schröder. *Automotive software development: A model based approach*. Tech. rep. SAE Technical Paper, 2002 (cit. on p. 15).

[Rie03] M. Riebisch. “Towards a more precise definition of feature models.” In: *Modelling Variability for Object-Oriented Product Lines* (2003), pp. 64–76 (cit. on p. 19).

[Rob19a] C. S. C. Robert Bosch GmbH. *Internal presentation slides from Robert Bosch GmbH*. 190326_MBSE_EEA_Approach_Guideline_OpenPoints.pptx. Robert Bosch GmbH, 2019 (cit. on pp. 7, 8, 13).

[Rob19b] C. S. C. Robert Bosch GmbH. *Internal presentation slides from Robert Bosch GmbH*. 190326_MBSE_CCEYX_VariantHandlingApproach.pptx. Robert Bosch GmbH, 2019 (cit. on pp. 8–10).

[Rob19c] C. S. C. Robert Bosch GmbH. *Internal presentation slides from Robert Bosch GmbH*. MasterThesisProposal_JungAYoon_R7_190701.pptx. Robert Bosch GmbH, 2019 (cit. on p. 9).

[Rob19d] C. S. C. Robert Bosch GmbH. *Internal presentation slides from Robert Bosch GmbH*. 20190710_Variance_Management_along_PEP_and_V_model.pdf. Robert Bosch GmbH, 2019 (cit. on p. 10).

[Rob19e] C. S. C. Robert Bosch GmbH. *Internal presentation slides from Robert Bosch GmbH*. 190701_ExportEngine_WS2.pptx. Robert Bosch GmbH, 2019 (cit. on pp. 29, 106, 108–111).

[SAE18] O.-R. A. D. c. SAE. “SAE J3016 Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles.” In: 2018 (cit. on p. 28).

[SBC+13] G. M. Selim, F. Büttner, J. R. Cordy, J. Dingel, S. Wang. “Automated verification of model transformations in the automotive industry.” In: *International Conference on Model Driven Engineering Languages and Systems*. Springer. 2013, pp. 690–706 (cit. on p. 15).

[SD07] M. Sinnema, S. Deelstra. “Classifying variability modeling techniques.” In: *Information and Software Technology* 49.7 (2007), pp. 717–739 (cit. on pp. 16, 20, 22, 31, 63, 64, 68).

[She14] R. Sherman. *Business intelligence guidebook: From data integration to analytics*. Newnes, 2014 (cit. on p. 6).

[Str02] D. Streitferdt. “Integration of current models towards family oriented requirements engineering.” In: *Proceedings of the 3rd international workshop on software product lines: economics, architectures, and implications*. 2002 (cit. on pp. 17, 20).

[WRH+12] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, A. Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 15, 101, 104).

All links were last followed on July 25, 2019.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature