CO-OVM: A Practical Approach to Systems Engineering Variability Modeling

Cosmin Dumitrescu

CO-OVM: A Practical Approach to Systems Engineering Variability Modeling

– PANTHÉON - SORBONNE –
UNIVERSITÉ PARIS 1
OMNIBUS SAPIENTIA UNICUIQUE EXCELLENTIA

A thesis submitted by

Cosmin Dumitrescu
Centre de Recherche en Informatique
Université Paris 1 Panthéon Sorbonne

In partial satisfaction of the requirements for the degree of

Doctor of Philosophy in Computer Science

Members of the jury:

Reviewer: Nicole Levy, Proferssor at *CNAM (Conservatoire national des arts et métiers)*
Reviewer: Jean-Claude Royer, Professor at *École des Mines de Nantes*
Supervisor: Camille Salinesi, Professor at *Université Paris 1 Panthéon-Sorbonne*
Joint Advisor: Raul Mazo, Associate Professor at *Université Paris 1 Panthéon-Sorbonne*
Examiner: Alexis Aubry, Associate Professor at *Université de Lorraine*
Examiner: Daniel Diaz, Associate Professor at *Université Paris 1 Panthéon-Sorbonne*
Industrial Advisor: Alain Dauron, *Renault*

Date: June 2nd, 2014

Résumé

L'industrie automobile est confrontée aujourd'hui à un impact toujours plus grand de la personnalisation des produits, une conséquence des efforts pour défier la concurrence et exploiter des marchés de niche à l'échelle mondiale. C'est pourquoi la complexité induite par la diversité des produits est un enjeu majeur aujourd'hui dans l'industrie automobile, où la conception et fabrication des familles de produits franchissent les frontières des constructeurs individuels. Les alliances entre constructeurs permettent de partager la conception des véhicules, de faire du cross-badging , partager des plateformes. La complexité augmente également avec la variabilité induite par les fournisseurs de composants concurrents, avec la prise en compte des dépendances techniques et celles de l'offre client.
Nous croyons que la solution aux problèmes soulevés par la diversification se trouve dans la façon dont les produits sont conçus et documentés pour répondre aux exigences variables des clients. Le but de ce projet est d'étendre les pratiques de l'ingénierie systèmes à base des modèles vers une modélisation des familles de systèmes, tout en capitalisant sur les éléments existants de l'industrie automobile. Le défi est de comprendre les besoins d'un environnement industriel complexe et aborder les questions clés dans deux domaines - l'ingénierie systèmes et l'ingénierie lignes de produits - qui mènent á des résultats immédiats validés sur des cas d'études, mais aussi á l'identification des nouvelles questions de recherche à long terme chez Renault.
Le contexte fournit une base solide pour les principaux sujets de ce projet: un cadre de modélisation des systèmes en place chez Renault, les outils de modélisation fournis par le CEA LIST, les connaissances en programmation par contraintes et ingénierie des méthodes du CRI - Université Paris 1. Les résultats de la recherche de plusieurs thèses de doctorat au CRI ont été essentiels et étroitement liés à ce projet: le travail de Olfa Djebbi sur la configuration des modèles des lignes de produits, le travail de Raul Mazo sur la représentation à base de contraintes des modèles des lignes de produits multi-vues, et le raisonnement automatique sur ces modèles, le travail de Raouia Triki sur les techniques de recommandation pour la configuration des produits. La collaboration avec le CEA LIST a eu également une très importante contribution, et les outils issus ont été élaborés dans ce contexte.
Les principales contributions sont: (i) *CO-OVM* - un méta modèle de gestion des variantes de modèles SysML, (ii) *la dérivation des modèles orientée*

vue - une approche méthodologique pour la configuration des modèles de systèmes, (iii) un ensemble d'heuristiques pour améliorer les processus de configuration de la ligne de produit, (iv) l'identification des cas d'utilisation pour un outil de gestion de variantes des systèmes dans le cadre du processus d'ingénierie de Renault.
Les résultats ont été appliqués, avec le soutien des outils développés par le CEA LIST, dans des cas d'étude , révélant leur utilité. Tout aussi important est le fait qu'ils ont contribué au développement de l'expertise de l'équipe "Renault MBSE" en ce qui concerne la gestion des variantes, qui à son tour aura un impact sur les pratiques de développement des véhicules en aval et les améliorations futures des processus et des outils d'ingénierie.
La personnalisation des produits a fait l'objet de recherches dans des différents domaines de l'ingénierie. Dans les dernières années, les Lignes de Produits Logiciel ont démontré des résultats intéressants concernant les processus et les techniques de modélisation de la variabilité. L'approche adoptée consiste à tirer sur les éléments de l'ingénierie des lignes de produits et de les appliquer dans le contexte de l'ingénierie des systèmes, avec l'accent sur les concepts de la variabilité, la définition du méta-modèle, et l'intégration avec les systèmes d'information existants pour la configuration des véhicules par les clients.
Les résultats sous forme de publications ont atteint les deux communautés - l'ingénierie lignes de produits logiciel, et l'ingénierie systèmes. Compte tenu du fait que le " *logiciel*" a toujours été un élément central dans tous les contextes de l'ingénierie des lignes de produits, le projet actuel a permis de réduire l'écart entre ces deux domaines distincts - IS et LP.

Abstract

The automotive industry is faced today with an ever-increasing trend towards product customization, a consequence of the efforts to challenge competition and exploit niche markets on a global scale. This is why complexity induced by product diversity is a major challenge today in the automotive industry, where product configurations cross the boundaries of single manufacturers. Car manufacturer alliances, cross-badging, platform sharing, concurrent component suppliers, market and technical dependencies and of course variable customer requirements - are just o few of the themes that increase complexity through variety and inter-dependencies.
We believe that the solution to the problems brought by diversification stands in the way products are engineered to meet variable customer requirements. The aim of this project is to extend Model Based Systems Engineering practices to families of systems, while capitalizing on existing elements from the automotive engineering industry. The challenge is to understand the needs of a complex industrial environment and address key issues in the area of systems engineering (SE) and product lines (PL), that would on the one hand yield immediate results running on case studies, and on the other hand open new research questions for a longer timeframe at Renault.
The context provides a solid basis for the core themes of this project: a system modeling framework in place at Renault and modeling tools provided by the CEA LIST, constraints programming support for product lines and PL processes in the existing research at CRI, Paris 1 University. The research results of several PhD disertations at CRI were essential and tightly linked to this project: Olfa Djebbi's work on configuration of product line models, Raul Mazo's work on constraint-based representation of multi-view product line models and the automatic reasoning on these models, Raouia Triki's work on recommendation techniques for product line configuration. The collaboration with the LIST, CEA LIST had an important contribution, and the resulting tools were developed in this context.
The main contributions consist in the following: (i) *Co-OVM* – a variant management metamodel for SysML models, (ii) *viewpoint oriented product line derivation* – methodological support for configuration of family of system models, (iii) a set of heuristics for improving product line configuration processes, (iv) identification of use cases for a tool for system variant management in the context of Renault engineering processes.

The results were applied, with the support of the tools developed by the CEA LIST, in case studies, revealing their utility. Equally important is the fact they contributed to the development of the expertise of the "Renault MBSE Team" in respect to variant management, which in turn will impact downstream vehicle development practices and future improvements in engineering processes and tools.
Customization of products has been a research subject in different areas of engineering. In the past years Software Product Lines have proven interesting results regarding processes and variability modeling techniques. The adopted approach consists in drawing on elements from product line engineering and applying them in the context of systems engineering, with the main focus on variability concepts, meta-model definition, and integration with Renault legacy systems for vehicle configuration support.
Results in the form of publications reached both communities – systems engineering and software product lines. In light of the fact that "*software*" was always present in all product line engineering contexts, the current project explicitly addressed and *bridged* the gap between the two distinct domains – SE and PL.

Acknowledgements

I would never have been able to finish my dissertation without the guidance of my supervisors, help from colleagues (friends), and support from my family. I thank all those who helped me on my path.
I would like to express my sincere gratitude to my advisors. As an academic advisor Prof. Camille Salinesi helped me through his continuous support of my study, his patience, steadfast motivation for research, and knowledge. I would like to thank Alain Dauron, who, as an industrial advisor, supported my every action and initiative, provided me with the right opportunities for understanding my subject in the context of the organization, promoted my work, and trusted me throughout this long journey. Their guidance helped me in all the time of research and writing of this thesis.
I also thank my thesis reviwers, Prof. Nicole Levy and Prof. Jean-Claude Royer, for their insightful comments and guidelines for improving this document, their availability on such short notice, and the rest of the jury for all their support.
My sincere thanks go to my colleagues who took part in the Model Based Software and Systems Engineering project: Patrick Tessier for his guidance in the insights of Papyrus UML modeling and for an excellent collaboration, Emmanuel Laurent for his efficiency and motivation in the organization of the project, Sebastien Gérard for the paper reviews and support. As for the Systems Engineering expertise, no question would be left without answer when Thierry Gaudre and Hugo Chalé Gongora were around. I thank them for their dedication to this domain, and I truly hope I rose to their standards.
I am grateful for Raul's enthusiasm and positive energy, which is more than enough to power the whole CRI lab. His name will be undoubtedly found in many acknowledgements from now on... I wish to thank Daniel Diaz, without whom a part of this work would not have been possible.
I thank Jean-Marc Astesana and Laurent Cosserat for their undisputed knowledge about the Renault Documentary Language, and for providing me with the necessary means for understanding the "variability" at Renault. I also thank Frederic Delrieu and his successor Nicholas Godlewski, who provided me with all the needed documentation for the case studies, and for revealing some variability issues they faced with the current systems engineering approach and tools.
I am grateful to all of my colleagues at Renault, CRI, and the AFIS product lines working group, for their support. Surely, I have not forgotten any of them, but if I were to continue I would soon run out of writing space for the dissertation itself.

Contents

List of Figures

1 Introduction

Systems Engineering (SE) has emerged as an approach that enables the successful realization of individual complex systems. Product variety in the automotive industry today introduces a complexity that is impossible to manage without proper tools. For example, there are up to 10^{21} physical configurations possible for the Renault Traffic van [43]. In this context, although it is a necessity, extending variant management to all SE assets may prove to be a challenging problem.
Standards that apply to SE activities leave issues such as reuse between projects or concurrent development of a family of systems to the ad-hoc means of the system engineer. As time to market continuously decreases, capitalization and reuse have a great impact on the development of reliable, customer oriented systems. In the automotive industry, where product diversity represents a key factor in the success of the company, the need for managing families of systems sharing common assets is revealed as a major concern. As SE have gained acceptance in the automotive industry, processes must be adapted to take into account the transition from traditional industry practices. In particular, the diversity that characterizes this environment, from vehicle components up to client offer, impacts SE in the large. Furthermore, in today's context of changing, disruptive technologies related to the electric vehicle and migration towards renewable energies, efficient management of technology diversity through reuse, and controlled variability across families of systems becomes an essential asset for the industry.
The theme of creating customized products has been studied in several domains, ranging from software to complex mechatronic systems and from multiple perspectives, such as Software Product Lines (SPL), product platforms, modular design or market related studies. In the automotive industry the problem is particularly complex as the vehicle range[1] is very wide, the degree of options available potentially leading to the

[1] Gamme vehicule

customization of unique vehicle configurations. At the same time product diversity has a wide impact on the overall company and industry: automotive diversity is driven by the vehicle manufacturers, but suppliers are also affected by the costs and risks of diversification [115].

1.1 Context: The automotive industry

The automotive industry has been associated for years to mass-production of vehicles offered to end-users with which it has developed a strong, almost passionate relationship. On a world-wide market with high competition on costs, focused product features for particular types of customers as well as product customization play an ever increasing important role. Coupled with a context of increased market uncertainty, customization oriented organizations are pushed to diversify the product range even further.
Renault's product range is very large and has a huge number of vehicles for each model. In engineering, the variety of technical contexts and requirements for which a system needs to be specified and designed represents a challenge from the perspective of management and integration to the organization information system. Elsewhere in the organization diversity is managed effectively: online product configurators provide the customer front end and send the customer orders to flexible production lines that support the manufacturing of customized products. While product diversity has an impact on all organization activities, one of the areas that could benefit from improved variability management is SE.
SE bridges the gap between customer requirements and vehicle components, which is why it is important to introduce variability management as a mean of early identification and specification of variability, but also of management of solution alternatives and configurations in respect to vehicle features.
While dealing with variety has been a reality of every-day practice in automotive SE, a lack of formalization and methodological support leads to poor documentation of variability of SE artifacts and deliverables in relation to product components and company commercial offer. Our purpose was to extend Model Based Systems Engineering (MBSE) formalism to support variability modeling and integrate these to the context of the organization. Meanwhile, a formalization of the activities for managing variability for systems was needed, including derivation [3] and evolution of domain models. This led us to study the engineering scenarios of development of systems and the needs of the organization (and perhaps of mass customization automotive industries in general), with a SE approach. The formalization takes into account the point of view of product line engineering, and generalizes many of the concepts applied to the software domain, bringing them closer to the systems context.
Product Line Engineering emerged as a viable and important reuse based development paradigm that allows companies to realize important improvements on time to market, cost, productivity, quality and flexibility [63]. The methods proposed in product line engineering (PLE) provide valuable information for approaching variability manage-

ment in MBSE. As one of the critical aspects of automotive MBSE, variability was among the first challenges to overcome for the adaptation and adoption of model based engineering practices.

1.1.1 Platform sharing, customization and reuse

All passenger cars are built on architectures that share a common set of elements within a range of vehicles from the same family. Platform development costs represent up to 50% of the total product development cost of vehicles. According to an evaluation performed by "Evalueserve" [90],only 20 of the vehicle platforms in production in 2010, cover 40% percent of the global vehicle market, revealing reuse as a core asset of the automotive industry. The same report reveals that 5 vehicle Original Equipment Manufacturers (OEMs) in regard to platform based reuse, Renault-Nissan, General Motors, Volkswagen, Toyota and Ford, fully rely on sharing platforms within their own group as a mean of achieving economies of scale.
What sets Renault-Nissan Alliance apart from the other manufacturers is the involvement and ability to harness synergies with other manufacturers. They also rely on platform sharing with manufacturers external to their own group through co-development, or through various agreements. For instance the Dacia/Renault Duster SUV platform is to be shared with Nissan for the indian market [28].
Modular architectures appear as yet another strategy, in extension to platforms, exploiting standardization and reuse of parts. Within the manufacturers evaluated in the same report, General Motors and Volkswagen qualify as the organizations which best exploit product modularity to leverage reuse. In order to enforce the alliance synergy, Renault-Nissan recently launched the "Common Module Family" approach [19], which aims at extending manufacturing commonalisation to vehicles from both OEMs. Deployed to more than 10 countries across 5 continents, it is estimated that this approach will generate up to 40% reduction of cost per model [19]. The approach consists in creating a common "parts bank" and shared architectures that can be used on a variety of vehicle models.
Some years earlier, Alizon et al. noted that "the full potential of reusing manufacturing knowledge has not been realized" [35], and that Renault, at that time did not have a method for the systematic reuse of experiential knowledge (in respect to manufacturing processes).
Today, many of the problems related to reuse have been solved, in the most critical areas for reducing cost through the effect of mass production : common architectures (platforms), modules, component standardization, production line/manufacturing flexibility. However, with the increasing product diversity and especially interdependence between systems, the problem of reuse is ever more present in the specification of systems requirements and architectures, at Renault. On the one hand, systems engineers need to provide generic documents to describe a whole family of systems, where each specific system configuration depends on the whole vehicle configuration. On the other hand, reuse of existing knowledge, about system architectures and requirements, is not

fully exploited today, at Renault.
Many of the existing variability modeling techniques fall into the categories of feature based [123] or variation-point based techniques [163]. Only a few of them focus on the systems level, in a context of complex systems comprising software, mechanical, electronic parts with dependencies to external systems, regulation and market characteristics. Trujillo et al. [203] rely on feature based representation and mapping to variation points defined within the SysML models though stereotypes. While, this approach enhances SysML and system architecture expression support for variability, it does not address, in our context, some aspects of variability orthogonal to system models, such as traceability, complex constraints modeling, integration with legacy models, multiple levels product of decomposition, shared features among families of systems. The SYSMOD [24] approach focuses on the systems model structure, by introducing stereotyped (UML) packages for variants and variations, in the same model with the system.
Modeling techniques for systems [23][195][78] enable the representation of different systems aspects, like structure, behavior or interactions, in conjunction with a specific method or to the methods employed by each organization. At Renault we often refer to the framework defined by Krob et al. [129][58] in connection with SysML models. Methodology aspects become even more critical when variability is involved, because this affects all domains of activity within the organization.

1.1.2 Mass customization

Reuse strategies related to the product manufacturing processes, product portfolio selection and optimization and product design paradigms, have also been proposed in "the context of product platforms". The term "product platform" was defined by Meyer & Lehnard [137], from a strategy point of view, with a focus on sharing core elements (components, interfaces, processes) among different variants of a product family and achieving reductions in development cost and time, while diversifying a number of products.
According to Simpson, T. [182], the process of developing product families can be approached in two different ways: the top-down approach where a company develops the product family based on a common platform, and the bottom up approach where a company consolidates a group of products by component standardization and reuse. The process comprises three main domains of activity: product definition (establishing the product portfolio), product design (designing products based on a single platform) and process design (cost and manufacturing decisions). However, the proposed approach does not take into account an explicit model of variability and engineering decisions that impact product variety.
Yet another coined term related to reuse and design processes is "mass customization", defined according to Tseng and Jiao [204] as the process of "producing goods and services to meet individual customer's needs with near mass production efficiency". It represents at the same time a wider subject that can be decomposed into several sub problems: customization and reuse in product architectures, client requirements

identification and marketing, client order reception and configuration (online product configurators), production process optimization (by reuse of production lines across product families, reuse of tools, processes etc.). In the context of mass customization, variant derivation is defined as the instantiation of a product from a product family. The instantiation is based on the model of the product family, product variety description, through the following actions: parameter propagation (impact of configuration parameters), include condition (dependencies) and variety generation (attaching, swapping, scaling) [204].

1.1.3 How this document is organized

This document is organized in 7 chapters, structuring the variability management theme as follows:

- *Chapter 1*: The rest of the current chapter introduces the research questions and methodology adopted in order to identify the concepts appropriate for variability in MBSE and sets the objectives for this project.
- *Chapter 2* describes existing works in variability modeling and model based systems engineering. The variability models are presented briefly through a classification based on the main variability concepts.
- *Chapter 3* defines the needs in systems engineering and in the automotive industry (based on the observations at Renault) in respect to variability management. The analysis is materialized in: (i) a series of "engineering scenarios" that aim at capturing the way product line artifacts may be transformed during the system conception and development, (ii) use cases for a systems engineering and variability management tool, (iii) requirements for a model based systems engineering approach that supports reuse.
- *Chapter 4* introduces the concepts that support variability modeling activities in SE (c.f. Appendix D), as well as a simple example to illustrate these concepts. The detailed Electric Parking Brake case study is available in Appendix E along with two other examples.
- *Chapter 5* consists of two themes: (i) methodological support for the derivation of SysML models through staged refinement and inclusion of new artifacts, and (ii) improvement of the configuration process based on constraint programming heuristics in order to reduce the overall time and number of steps.
- *Chapter 6* provides the validation of the approach in respect to requirements and use cases, by modeling the examples detailed in Appendix A.1.2. In addition, the initial requirements for the Electric Parking Brake are included in Appendix C.
- *Chapter 7* draws conclusions and identifies future work subjects for variability management in systems engineering.

- In addition, some details of the implementation of the tool extensions are available in Appendix F, in the form of source code examples and UML Profile diagrams.

1.2 Challenges

The challenge is to adopt variability modeling techniques and adapt SE methods to integrate the Product Line (PL) paradigm, while taking into account, and without disrupting global organization processes, some of which already deal effectively with product diversity.
Numerous approaches of Product Line Engineering (PLE) already exist and propose different formalisms, many in the context of software product line engineering. the complexity problem which we are confronted with in the context of SE applied in the automotive industry, is rarely addressed: it is difficult today to specify, integrate and analyze the variability issued from multiple sources, as for example in the automotive environment - the environment where the system shall be used, vehicle features (external variability, observed by the client), technical vehicle characteristics that impact the studied system, the internal behavior and structure of the system etc. Reuse in the automotive industry targets most often component reuse through vehicle architecture strategies (e.g. modular, platform), production line flexibility or interaction with the market and customer preferences through product configurator. An effective framework for reuse in the MBSE development process can bridge the gap between client needs or the marketing offer and the final vehicle production phase in plants and focus also on reuse of all systems life-cycle assets, not only on components.
The main *research question* question is to a large extent related to the industrial setting at Renault, but can undoubtedly be found in other industries where variability is a major concern.

> How can MBSE explicitly support reuse in the specification and design of systems, while successfully linking marketing options and physical components diversity?

Today, many of the variabilities are poorly documented, or are documented by ad-hoc means. This makes documentation of systems difficult, when there are many possible configurations, and could introduce inconsistencies in respect to marketing and component variability. The MBSE approach provides the engineer with a useful set of tools to support the system analysis from customer requirements to detailed design. The aim is to document variability as part of the system analysis, where the documentation of variability is not only a methodological requirement with a later benefit, but an immediate support for the system engineer. The details of the research questions and existing issues related to variability at Renault, are further detailed in chapters 1.3 and 3.
Our research methodology relies on the analysis of a Renault case study, and is based

on the development of engineering scenarios for a vehicle system, in which variability is involved. The adopted theme is that SE models need to be extended with a formalism that expresses variability on a family of systems level, complementing the already existing marketing offer descriptions towards a more detailed design level. The scenarios refer to the development process, in which different actors and stakeholders are involved, as well as to the specific methods employed in the case of SE practices applied to a system family. They allow us to define a set of requirements, that the formalism and the methods that use it, must satisfy. The variability management theme emerges in the context of the development of a tailored MBSE tool suite, as suggested in Figure 1.1. Safety analysis, project management or numerical simulation are some

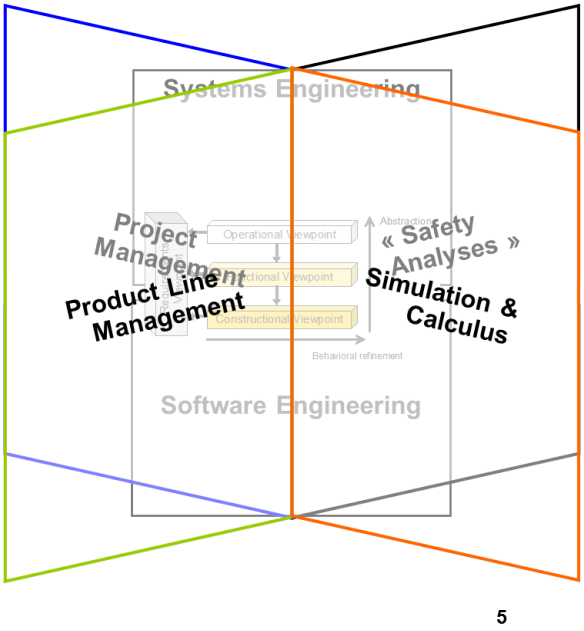


Figure 1.1: Product Line Management context and integration with Renault SE framework [58]

of the extensions that come and "plug-in" to the MBSE framework [58, 194] in use at Renault.
Before dwelling into the questions of "systems variability", we introduce the reference domains that provide a stable foundation for this research.

1.3 Research Questions & Methodology

Industrial contexts provide rich information along with constantly emerging problems and challenges, providing opportunities for case studies or survey based research in a realistic setting. However, sometimes they also generate constraints: proprietary data formats, which may further need specific domain knowledge for interpretation; ethical

issues [172] related usage and/or publication of data; availability of resources and time constraints. One of the challenging problems during this research project was to successfully synchronize research planning with projects and tasks within the organization. This chapter goes into the details of the main research question introduced in the previous chapter. It also provides a link to the industrial setting and problems identified at Renault, presents the plan for research and the methods and tools that were used. Although the planning had to be repeatedly adjusted on the way, the planned (or sometimes unplanned) events contributed successfully to the development of ideas that resulted in this thesis.

1.3.1 Adopting case study based research

Engineering activities rely on appropriate tools for support, guidance, data accessibility, sharing etc. One of the recently emerging themes in SE is the "model-based" paradigm [89]. This comes with its challenges and opportunities. *The opportunity*: In the automotive industry management of families of systems is essential for reuse and plainly for coping with the diversity of contexts a system is designed for. Models provide a convenient way to share and reuse information. *The challenge:* Often, existing information systems are already in place to deal with diversity in other areas of the organization, such as manufacturing or defining the commercial offer. Thus the final goal is to identify the specifications of a tool that is both compatible with legacy systems and supports SE methods. Software engineering is a multidisciplinary field, involving the investigation of tools, processes and human factors. Due to the human aspect of this research, empirical methods, which often draw from other fields [83], are also relevant for software engineering.

When adopting a research approach, one needs to take into account that increasing the degree of control can reduce the degree of realism, while controlled experiments are a time consuming task. Conducting controlled experiments may be a difficult task to achieve in an organization where human and time resources are almost always associated with the most immediate tasks and projects.

According to Eisenhardt [85], a case study is a "research strategy which focuses on understanding the dynamics present within single settings". They are a useful research methodology in software engineering, where the objects of study are phenomena difficult to study in isolation, occurring for example in large organizations. The term itself appears in software engineering with slightly different meanings and in parallel with with other terms. Elements from other research methods may be used [172] within a use case.

The system which is used extensively throughout the research is the Electric Parking Brake system (EPB) [86], which we believe is representative in the study for different reasons, such as the existence of multiple solution alternatives, of variable customer requirements, and medium complexity in respect to the other vehicle subsystems. From a practical point of view, the EPB system is well documented and there are no issues regarding intellectual property or copyright. The documents regarding stakeholder and

technical requirements provided valuable information, along with different documents for the presentation or analysis of the alternative designs. Furthermore, documents describing the architecture of related systems and vehicle options provide all necessary information about the EPB system context. Interviews with the system architect responsible for the "brake" family of systems, or with experts in the organization information systems, as well as document or archival analysis, and literature study provided complementary information, completed by a final survey addressed to domain experts[1]. Finally, for validation purposes, we included two complementary examples, which covered some modeling aspects that the EPB case study did not provide. The EPB system is described in detail in Section A.1.

1.3.2 Case study research questions

In the software engineering literature, case studies vary, from well organized field studies, down to simple examples. According to Robson [168][172], a case study needs to answer to some of the following questions.

What to achieve? The objective of our case study research is to explore requirements for the means (methods and tools) to manage system requirements and architectures, during the development process, achieving reuse and supporting early specification of variability, in relation to the commercial offer and component diversity, at Renault. The case study results in (a) research questions, (b) better knowledge of practical situations for the methods to develop, and (c) validation material.

What is studied? Due to the nature of the subject, we can distinguish between two types of case study perimeters:

- The organizational & industrial context,
- The developed system.

At Renault, the essential elements regarding variability management consist in (a) the legacy information systems dealing with the expression of the commercial offer based on the so called Documentary Language, as well as the management of the bill of material, and (b) the activities in the vehicle development process that address the documentation of variability. As explained later in Chapter 3, the SE process was first introduced at Renault, on an intermediate level, focusing on vehicle subsystems. The organization following a SE approach is presented in Figure 1.2. Thus, the socio-technical dimension of the research is focused on the development of vehicle systems, involving internal and external stakeholders and the (Research) System Architect[2], who represents the main actor in charge of the development of systems. Figure 1.2 presents the organization around the SE process in place at the beginning of the study. The system architect is

[1] Group de travail "Lignes de Produits", AFIS
[2] Architecte Systéme (Amont)

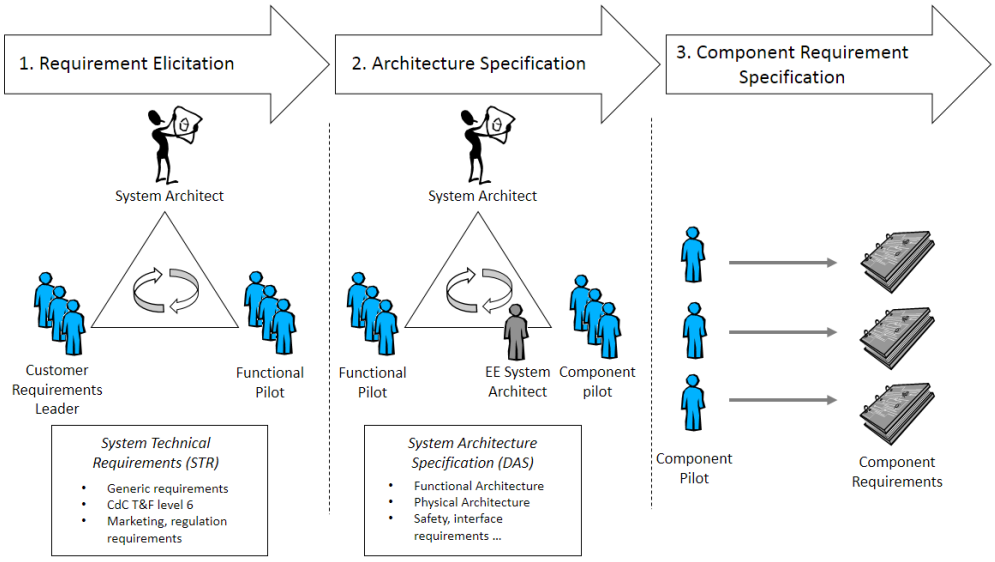


Figure 1.2: Systems engineering actors and produced documents

responsible of requirement elicitation from all stakeholders, select the best alternative and design the system architecture, and finally allocate requirements to components. The documents produced are: the System Technical Requirements (STR) and the System Architecture Description (DAS[1]). These documents need to contain all information in order to obtain specific component specification for each different context or system configuration (e.g. different equipment levels, countries). The problem stemming from a lack of variability formalization is twofold: (a) the management of system configurations across a single system "V" cycle, and (b) reuse and capitalization across multiple "V" cycles for systems from the same family. The difference between the two aspects, a and b, is the time frame (temporal dimension). In the first case (a), the engineer needs to design a system with multiple configurations, in a single "V" cycle to cover the needs of a range of vehicles. Much of the requirements can be used for future generations of the system, thus, in the second case the generic requirements and system architecture items need to be capitalized for future reuse.
The vehicle is decomposed into subsystems, that are developed according to a MBSE approach. In this context, engineers need to represent the different concepts specific to the SE domain at Renault, while taking into account methodological aspects. For example, some of the systems that are developed as part of the larger vehicle system are: temperature control system (CLIM), Connectivity (CONEC), Rear View Camera (RVC), Ultrasonic Park Assist (UPA), Hands Free Parking (HFP), Lane Departure Warning (LDW), Adaptive Intersection Lighting (AIL), Adaptive Cruise Control (ACC), Around View Monitor (AVM). A systems architect is responsible for each of

[1]Dossier d'Architecture Système

these systems. Depending on their complexity they may also be under the responsibility of a single person. The concepts to support the representation of variability are discussed in Chapter 4. In SE, we believe that some of these concepts need to address variability introduced by alternative supply components and engineering decisions. Design decisions lead from a system concept to a detailed technical solution, potentially introducing variety independent from marketing options, as a consequence of technical constraints of particular configurations.

Reference framework Software Product Line Engineering is a well established domain both in industry practice and research [131], [63], [12]. The SPL practice has shown benefits which consist in reduced costs and time-to-market [54], with an initial upfront investment for the development of reusable assets.
The definition of SE varies, but can be summarized as "a robust approach to the design, creation, and operation of systems. In simple terms, the approach consists of identification and quantification of system goals, creation of alternative system design concepts, performance of design trades, selection and implementation of the best design, verification that the design is properly built and integrated, and post-implementation assessment of how well the system meets (or met) the goals."[1]. The essence of the SE domain is captured by the INCOSE definition, as an "interdisciplinary approach and means to enable the realization of successful systems" [2].
Our problem is situated between these two domains: SE and product line engineering. The two domains have similarities and our belief is that it is possible to adopt a product line approach in SE to leverage reuse of various engineering assets. However, each approach appeared and evolved in a different setting, possibly leading to different perspectives over the theme of "variability", which we are going to explore in our study. For instance in:

- software engineering, approaches for the development of (program) families are mentioned as early as 1976 [158], in order to leverage reuse by "module specification";
- SE can be traced back [25] to Bell Telephone Laboratories [178] and the US Department of Defense [18] and industry in general [103], emerging as a result of the need to recognize and manage the properties of complex systems as a whole.

On a system level (comprising software, electronics, mechanical parts), SE was less confronted with the problem of reuse since the developed systems were out of the context of mass production and scale industries, and rather focused on complex, often unique systems tailored to specific stakeholder requirements.
In mass production industries, like the automotive, the focus of reuse was initially elsewhere - on physical components, through standardization and product platforms [137], which brought important cost reduction in manufacturing. Recently, this approach was further extended across brands and car manufacturers. For instance, the

[1] NASA Systems Engineering Handbook, 1995
[2] INCOSE Systems Engineering Handbook

Renault-Nissan alliance shares multiple platforms [22], and the Dacia-Renault models are branded differently across different markets [20]. Other examples can be found in the automotive industry, as this practice brings mutual benefits for manufacturers, joining expertise and manufacturing facilities on limited market segments or specific vehicle platforms [26]. This is a structural point of view on reuse, which is closer to the final product, but not extensively exploited and extended upstream, to all development assets.
Finally, while many approaches and variability languages exist, we report to the work and concepts defined by Pohl et al. [163], and explain our choices further, in Chapter 4.

Detailed research questions generated by case study The questions raised address a wide range of issues regarding modeling, derivation in MBSE or Renault tools and practices.

1. What are the forms of variability in SE ?
2. Why do variations occur with SE assets? (Which are the sources of variations?)
3. Can design decisions be exploited for reuse of SE assets in Research & Advanced Engineering (R&AE)?
4. How do product line processes (separation between domain and application) apply to SE (and to the organization in place at Renault)?
5. How does product line derivation fit with the SE methodology in order to leverage reuse?
6. Can early documentation of variability completely eliminate the need for manual specification of component configurations for manufacturing processes?

The proposed solutions answer to most of these questions. The last question was partially treated in the current implementation, but a definitive answer shall be addressed future work.

How and where to collect data? The information taken into consideration for our case study concerns the two areas of applied MBSE for vehicle mechatronic systems and the enterprise information system.
In respect to the studied system, we considered the Electric Parking Brake system, for multiple reasons, including coverage of the different phenomena present in SE in respect to variability, moderated complexity, availability of information, lack of particular constraints for confidentiality or publication of content. The elements that served for the purpose of our study are, among others:

- System specifications of the Electric Parking Brake for specific vehicles (FPA X91[1]);

[1]Renault notation for vehicle platforms

- Specification for the Hill Start Assistant (HSA) function (DAS HSA[1], HSA System Technical Requirements v0.7 A3 - 2007, Component Specifications for HSA - STRComp_HSA - 1269-2010-65514 (V1), HSA Function for ESC Technical requirements - 31-03-018—A);
- Regulations in regard to the automotive parking brake
- Braking functions diversity and global impact - ad-hoc document proposed by the "DIESC" department (made available through the e-room for Systems Engineering for the BRAKE family of systems) at Renault;
- Electric Parking Brake solution alternatives - ad-hoc document proposed by the "DIESC" department (made available through the e-room for Systems Engineering for the BRAKE family of systems);
- Electric Parking Brake training (internal Renault support documents and training)

These are the main documents that served to build the case study around the parking brake. Other documents, meeting minutes, interviews contributed to the ideas developed through the case study. In particular, meeting minutes were kept for the record. For the Braking system the most immediate problem that surfaced, was the management of variants for a single vehicle platform. Due to the large number of possible configurations of the system corresponding to different configurations of the vehicle, it became almost impossible to maintain a different file for each situation. Methodological aspects also proved to be problematic in respect to variability, since each architect found local ad-hoc solutions, not necessarily aligned to a global framework. I was also exposed to these problems through direct involvement in a short mission in accompanying the system architect responsible for the Brake system in specifying requirements and the architecture for the EPB and ESP systems. Obviously for the follow-up the support of the SA was essential for the accessibility to documents and meetings, as well as validation of ideas.
In respect to the enterprise system, the way variability is expressed and exploited is well documented. The elements that served for the study were:

- Specification of the Renault language for the description of product diversity (Documentary Language). The documentation is structured in short files for each theme, containing definitions, and rules on how to define items based on the Renault product diversity language (made available internally for Renault-Nissan - lex*.doc, lex*_en.doc).
- Training support on the following themes: parts and bill of material (BOM), commercial offer, vehicle diversity, lexicon and Documentary Language, presentation of the Documentary Language comity (organizational entity with authority over the Documentary Language).

[1]Dossier Architecture Système HSA

- Exports from the tool BOM-C, which contain the commercial offer description, defined as configuration tables, using a specific format (called TCS tables).

Again, accessibility and support for the documentation would not have been possible without the help of the engineers form the "Applied Artificial Intelligence" department at Renault. The way diversity is already expressed and exploited at Renault was essential for our purpose for two main reasons: to provide from compatibility with existing practices and legacy information systems, and to benefit and draw on the solutions readily available at Renault [42].

1.3.3 Research path and enabling tools

The research protocol relies on the context at Renault for the identification of engineering scenarios for families of systems [2]. Besides, an ontology for variability aggregates concepts of engineering practices from within the organization, but also from relevant literature on product lines. Figure 1.3 presents the approach, which takes into account existing practices and concepts within the organization, but also from the product line literature, with the purpose of aligning them with our specific process requirements (process stemming from the collection of scenarios).

The variability management ontology provides the specifications for developing tool support for MBSE (based on Papyrus SysML modeler[1]). Requirement elicitation and derivation related to variability management was performed by interviewing system architects in different domains of application within the organization (mechatronic systems, embedded software systems, information systems) and by developing a set of engineering scenarios for product lines adapted to the organization context.
Our purpose is to integrate variability management in our model based engineering framework [58] by : (a) providing specifications for model based development tools and (b) introducing specific activities in SE processes. The analysis of different engineering development scenarios involving reuse contributes to understanding and formalizing these methods and processes. Finally a particular process scenario (or a subset of scenarios) provides the means for validating tools for system variability management that should be built according to the defined specifications.
The use of ontologies for variability has several benefits in our context. Ruiz and Hilera [110] define the objectives for using ontologies in software design and development. Among others they mention the following applications: specification – to formalize the requirements, define components and system functionality; knowledge acquisition – by using ontologies as a guide for the knowledge acquisition process. Also, according to Hesse [109], ontologies can support the definition, re-use and integration of software components.
One of the scenarios captured our attention because it appears in most of the projects for developing automotive systems. These often rely on solution carry-over (previous projects) and carry-across (reuse across the vehicle range) - SE based on reuse of generic

[1] http://www.eclipse.org/papyrus/

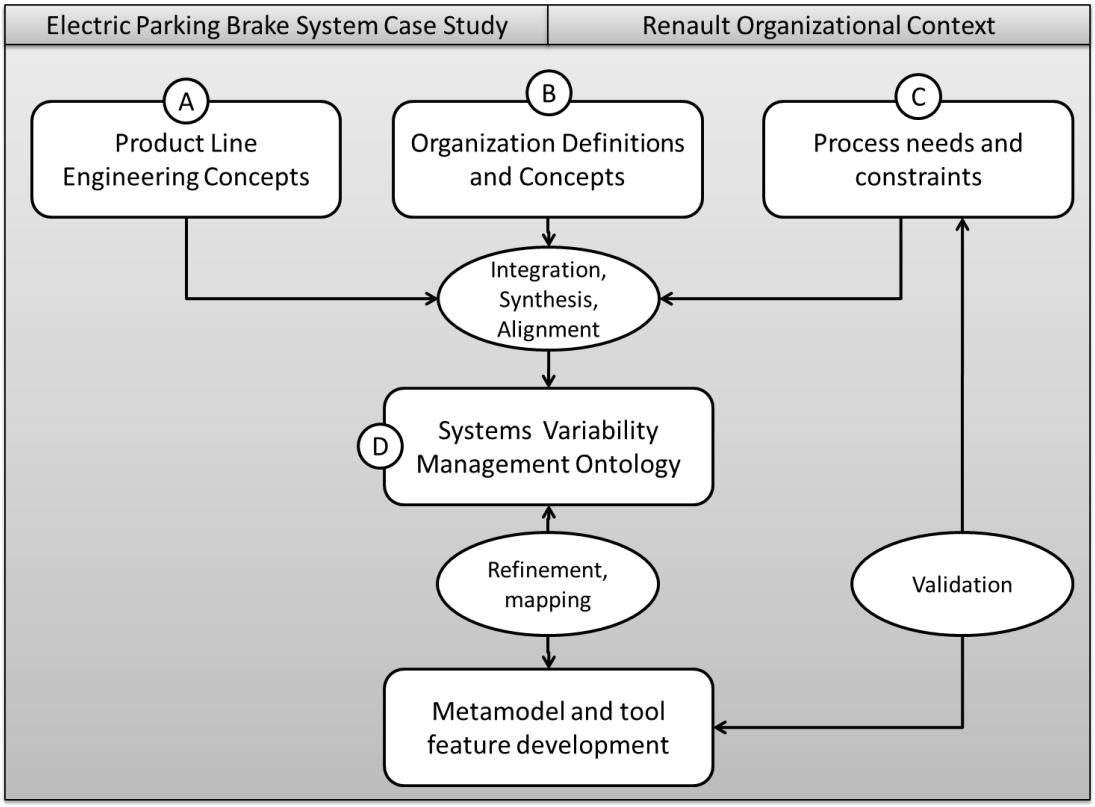


Figure 1.3: Approach for the discovery of required variability concepts

architecture descriptions. Finally, experimenting with the example of the Electric Parking Brake System (EPB), allowed us to confirm and better understand the issues that we identified through interviews and exploration of the company practices and tools.

1.3.4 Tools for knowledge synthesis and integration

Ad-hoc means to manage variability are applied for the SE deliverables. As a consequence for the EPB case study, there was not a single document with a complete system specification, but separate documents, for each solution alternative or functionality of the braking system. These contained inconsistent, complementary and overlapping specifications. The first step was to integrate these into a single consistent description, and prepare the case study for further exploitation. Thus, a single requirements textual document was created, containing stakeholder and technical requirements. The SysML model of the system architecture was first developed using Artisan Studio[1], with annotations for variable elements and family of systems constraints. For an explicit, separate description of variability, we created an OVM [163] model of the Electric Parking Brake

[1] www.atego.com/artisan

System edited with the VarMod-Prime Tool-Environement [162], and added annotations for types, attributes, classifications based on viewpoints or any complementary information needed. The EPB models and specifications are detailed in Chapter 6.
We integrated definitions regarding concepts from the product line literature and Renault documentary language and apply the approach introduced in Figure 1.3. This was achieved using the "CmapTools knowledge modeling kit", developed by the Florida Institute for Human & Machine Cognition [153]. This has proven to be a valuable support for acquiring, capturing and sharing expert knowledge [57].
As an example, Figure 1.4 presents some of the core concepts of the product diversity at Renault. Each node, corresponding to a concept contains:

- The name of the concept, visible on the diagram;
- The textual definition corresponding to the concept;
- The attached document containing the actual source of the definition and further details (if available).
- Optionally, if there was no offline copy of the source document, a link towards the actual source was appended to the definition.

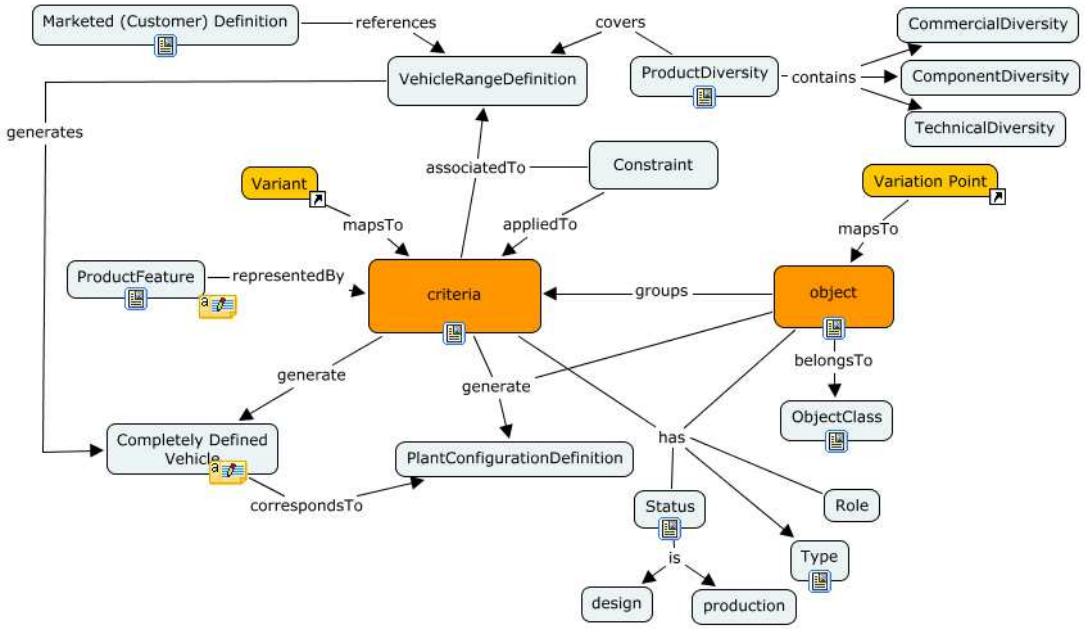


Figure 1.4: Documentary language and Renault product diversity core concepts

Some of the concepts were either implicit, or they lacked a formal explicit definition, although they were often used through the available documentation. The product diversity contains three types of variations: commercial, technical and components. The central concepts are : (a) *the criterion*, which expresses vehicle characteristics (or features) and (b) *the object*, which classifies the characteristics according to the variation subject. It is interesting to point out that, apart from the terminology, the concepts are close to variation point - variant definitions from software product line engineering. The next chapter presents the problem statement that resulted from the Electric Parking Brage (EPB) case study and ontology analysis.

1.4 Objectives for variability management in Systems Engineering at Renault

Mass customization industries have developed strategies of diversification of products and support in areas like customer interaction - through online product configurators, component reuse - through modular and platform based products, flexible production lines (manufacturing). SE bridges the gap between stakeholder needs, and in particular marketing features and component realization and integration, but it lacks the support for reuse and diversification that has already been introduced in other activities. SE is a problem solving approach that enables the successful realization of systems: existing configuration techniques and variability models, have some shortcomings when they need to be implemented in large organizations and they are not perfectly suited for

representing and tracing variability in a problem solving paradigm, like SE.
The main needs that we identified and approached in our study concern the following aspects:

- Methodological support for introducing variability in SE formalizing activities for domain and (application) systems engineering and developing tool support for these activities, in particular for product line derivation through the definition of system viewpoints for derivation. The techniques that we introduce for product line derivation enable reuse in SE activities by relying on a staged configuration process combined with regular engineering activities and consistency checking of system models. It also provides support for (document based) generation of deliverables at different stages of the SE process.

- Capturing variability in MBSE, through a metamodel that addresses some aspects that we consider critical for systems variability : integration of different viewpoints on variability, traceability of decisions, unified representation of different types of variability (stakeholder, marketing, design decisions generating variability, replaceable component level variability), variation characteristics related to system elements (change rate across a vehicle range, impact of variability etc.).

- Preparing component configuration specifications. By enabling interaction with legacy systems providing support for component configuration, and by introducing change propagation techniques for the analysis of system variability, we are able to generate component diversity use cases": the set of system configurations where a particular component can be deployed.

2

Variability Management in Model Based Systems Engineering

2.1 Introduction

This chapter introduces existing techniques that provide visual[1] means for representing variability and explains how model driven development integrates these concepts. Derivation is also presented, as it is of particular interest in model based approaches, where reuse can be achieved in multiple ways and on different levels of abstraction. As SE adopted the model based paradigm, reuse and variability needs to be adapted to its specific needs. Existing reuse approaches in SE are introduced along with a personal perspective on the gap between product line and SE.
The literature review focuses on the following aspects of variability management: (i) variability modeling techniques, (ii) variability in model based approaches (iii) product line derivation and methodology aspects in SE. The core subject of this study lies in the MBSE domain.
Although models have been used in different areas of SE for a long time, the explicit adoption of the MBSE paradigm is a relatively new practice, that uses an integrated set of system representations. This is why this literature review extends in the directions of "Software Product Line Engineering" and (software) "Model Driven Engineering". This approach can be regarded as somewhat natural, given the commonalities between these domains and SE, and the fact that other MBSE "tools" were built onto existing knowledge from these areas. (e.g. definition of SysML as a UML extension). However, some may argue that specific SE needs are overlooked. SE modeling tools are not adapted for non-software engineering users [161], and in consequence the adoption of

[1] i.e. as opposed to textual languages

external tools may become cumbersome or meet resistance.

2.1.1 Context & method

The method employed for the collection of references is based on two components: (a) *tracing* through existing reviews and (b) *direct identification* through specific keywords and thematic conferences.
Given the many approaches for variability modeling, this chapter draws on the existing reviews, for an initial identification and dissemination of references. These reviews could be classified based on their purpose:

- *Classification*: Czarnecki et al. [65], Ahn et al. [31], Sinnema et al. [183], J.M. Jézéquel [121]
- *Comparative evaluation*: Djebbi and Salinesi [75]
- *Survey, perspective*: Chen et al. [59], Kyo C. Kang [126]

Some of these findings reveal that in model based approaches, variability modeling techniques are used both as an external orthogonal model, and as embedded concepts in a general purpose or domain specific language. This key aspect is further considered for introducing relevant product line literature in this chapter. While most of the existing approaches are based on feature models [123] which is a well established practice in product line engineering, Chen et al. [59] point out that most of the variability modeling techniques are insufficiently evaluated: "(95.6%) were evaluated by only one study", and "75.43% of them have never been evaluated in an industrial setting" [59]. This could be explained by the fact that the exhaustive evaluation of existing techniques in a specific industrial setting may encounter potential barriers: availability of tool support for each approach, availability of case study data in usable formats (e.g. compatible with the tools, not specific to a single organization, consistent), openess and availability of the industrial party and ressources, since a thorough evaluation can be a time consuming activity. These issues are also partially behind the choice of this study to pursue *adaptation and adoption* of a variability modeling technique, starting from requirements to Co-OVM implementation: requirements narrow down the solution space and *adoption* alternatives, while *adaptation* of concepts tailors the solution to our SE needs.
The second mean for tracing references focused on Model Driven Engineering and SE. It included the following keywords for the search of primary studies: (*variability* OR *product lines*) AND (*Model Based Systems Engineering* OR *Model Driven Engineering* OR *Model Driven Design*), as well as the use of alternative spellings or acronyms. Relevant conferences, papers, books or researchers already known, in regard to these terms, were also taken into account: the AMPLE project, Gomma et al. [96] book on model driven software product lines, MDPLE workshop.

2.1.2 Chapter structure

The next parts of this chapter introduce each of the approaches as follows. Section 2.4 introduces variability modeling techniques. It starts with a general overview of variability representation mechanisms, and further focuses on feature based and variation point based modeling in Section 2.4. Section 2.5 presents use case based and embedded variability approaches, while its subsection 2.5.4 discusses product derivation in relation to the engineering processes. Section 2.6 presents some existing works in SE and explains why none of the existing approaches was considered for *direct* adoption. Finally, Section 7.3 concludes the chapter by anticipating the contributions of this study, that are to be introduced further in Part II of this document.

2.2 Software Product Lines

Product Line Engineering is a viable and important reuse based development paradigm that allows companies to realize improvements in time to market, cost, productivity, quality, and flexibility [63]. According to Clements & Northrop [62] product line engineering is different from single-system development with reuse in two aspects. First, developing a family of products requires "choices and options that are optimized from the beginning and not just one that evolves over time. Second, product lines imply a preplanned reuse strategy that applies across the entire set of products rather than ad-hoc or opportunistic reuse. The product line strategy has been successfully used in many different industry sectors, and in particular, in software development companies [162][62][124][213]. One example of product lines is the vehicle product line of the French manufacturer Renault, which can lead to 10^{21} configurations for the "Traffic" van product family [41]. Many different kinds of artifacts can be reused: from requirements, to model fragments, code, test data, etc, which can be embodied as patterns, libraries of classes and meta classes, services or parameterized components. Reuse can be achieved in many different ways: instantiation, integration, composition or setting up parameters.
PLE explicitly addresses reuse by differentiating between two kinds of development processes [162]: domain engineering and application engineering. The aim of the domain engineering process is to manage the reusable artifacts participating in the product line and the dependencies among them [188]. The reusable artifacts, called domain artifacts (e.g. requirements, architectural components, pieces of processes, methods, and tests) are related in a model representing the legal combinations of the reusable artifacts (called product line model). The aim of the application engineering process is to exploit product line models in order to derive specific applications by reusing the domain artifacts. To generate new products, product line engineering takes into account the customer requirements but also the constraints of the product line domain.

In the automotive industry reuse is a core asset that allows car manufacturers to develop products faster and stay competitive. One of the most promising ways to manage reuse is by means of a product line approach. In the PL approach, the valid

combinations of reusable artifacts are represented by means of a product line model. Variety, variability and diversity are some of the recurring concepts in the context of families of systems. The way they are used is often different depending on the context or enterprise. The following conventions are used throughout the text:

- **Variety** – represents the absence of uniformity, sameness, or monotony". It is the property of a *collection of existing, completely defined objects* or products, such as a vehicle range, or components sharing similar features. For instance, the vehicle range at Renault exhibits a lot of variety, in order to satisfy a wide range of customer requirements.
- **Variability** – represents the degree of being changeable of *a single object* or product, in respect to specific descriptive or constituant elements (e.g. requirements, parts, properties). Furthermore, these elements are defined by Pohl et al. [163] as *the variability subject* – a "variable item or property' of the real world".
- **Diversity** – a synonim of *variety*, which is used at Renault.

A well established practice in the context of SE, *configuration* management is an engineering process maintaining consistency of a single product's performance, functional and physical attributes with its requirements, design and operational information throughout its life cycle [106]. In product line engineering, where multiple products can be obtained from variable assets, a product *configuration* represents a completely defined product, for which all choices for specifying the product and eliminating variability have been made. When the two practices come together, the complexity of managing product assets throughout the lifecycle increases. Both dimensions have to be taken into account: system life cycle phases (time) and product configurations, in the PL sense (space). Since the question of merging these approaches is not the subject of this project, the PL meaning of product configurations will be used consistently.

2.3 Product line derivation

The product line engineering paradigm separates two processes [213]: Domain engineering and Application engineering. The advantage of this split is that there is a separation of the two concerns with respect to variability [213], [162]. Domain engineering is responsible for (i) defining and realizing the variable and common artefacts of a particular domain, (ii) defining the variability and commonality relationships among these artifacts, and (iii) ensuring that the available variability is appropriate for producing new and correct products. Application engineering is the process by which the products of the product line are defined and built by reusing domain artefacts through the product line configuration process. In product line engineering, a configuration process is a step-wise process, its objective being to deliver configurations that both satisfy the constraints specified during domain engineering, as well as the stakeholders' requirements [76]. Configuration constraints can be completely specified (i.e., to have a complete configuration, otherwise called configurations in which all variables have a

value) or partially specified (i.e., to have partial configuration, in other words, configurations in which some decisions remain to be achieved) [213]. Once a complete or a partial configuration is specified, it evolves in its own project with the aim to become a new product.
Different stages of this process may involve actions on different types of variability. External variability [163] represents the perception of the customer on what is variable, enabling the definition of customized products according to different customer profiles. This is in contrast to internal variability, which is hidden from the customers.
Dealing with PL constraints in application engineering has been the subject of extensive literature that suggests different approaches, mostly based on boolean satisfiability problem (SAT) solving [175]. It has been demonstrated that the SAT approach can also be used to deal with domain engineering issues too [148][140][174][144].

2.4 Variability modeling

Traditionally, the focus of SE projects is on unique systems, systems that serve a specific purpose or type of mission. However, in respect to modeling, we turn to software engineering practices. In software engineering, reuse can be traced a long way back [158], emerging as a need for a better way to organize information in order to avoid re-developing previously created functionalities and accelerate product development. Variability modeling has proven to be an effective approach to analyze and organize domain information, and enable later reuse, as pointed out by Kang et al. [123]. Since then, many variability modeling techniques have been proposed, in a variety of contexts and covering different needs. These approaches can be roughly classified from an abstract syntax point of view, as : "feature based" and "variation point based" approaches. Concrete expression of these models takes different forms: standalone concrete syntax, embedded in general purpose languages (e.g. UML) and textual [60]. From a SE point of view, it is interesting to understand how the concepts of these approaches fit into a general SE framework and in what way they need to be adapted.

2.4.1 Feature based modeling

Feature modeling is by far the most widespread notation in software product line engineering, offering a simple and effective way to represent variabilities and commonalities in a product family. A feature is defined as a "prominent or distinctive user-visible aspect, quality, or characteristic of a software system or system"[123]. The modeling approach enables the representation of variability and commonality early in the product life cycle, as a support for the domain analysis process. Many dialects have been proposed based on the original Feature Model (FODA) [123], extending the initial set of FODA concepts, which include the following: *mandatory*, *optional*, *alternative*, *multiple choice* features, *requires* and *exclude* dependencies.
Czarnecki et al. [67] extend feature models with cardinatilites. This extension has obvious practical implications: e.g. clear and simple expression of alternative constraints.

Two types of cardinalities are proposed:

(i) feature cardinality - expresses the number of allowed instances of a specific feature in the final product;

(ii) feature group cardinality - allows to limit the number of choices of features within a group of features.

Kang et al. [125] extend the FODA model to include attributes attached to non-functional product properties, proposing the FORM notation. These attributes are usually expressed by *name*, *domain* and *value*, although there is no widespread, commonly used notation. They provide an improved expression of how the variability model impacts the final product configuration. Non-functional system properties are an important concern in SE analysis. They enable the quantification of SE measures: efficiency, effectiveness, performance, quality, conformance to standards, use of ressources [94] etc. General purpose languages used in MBSE don't always include domain specific concepts for non-functional system properties. For instance, Espinoza et al. [88] propose a Profile that extends Unified Modeling Language (UML) with non-functional properties focused on temporal verification. Thus, depending on the application context of feature models in SE, their attribute extension can provide a useful overview of system non-functional properties, as well as easy access for configuration. However, the presence of non-functional system properties in MBSE models can have further implications on their coupling with a feature based variability model (e.g. to ensure consistency, avoid information redundancy). Subsequent approaches on feature modeling also use the notion of non-functional properties.
The feature models are typically used to express customer visible variability, as a collection of optional and mandatory, structured elements. However, solution variants, and solution space in general, is not taken into account in feature reasoning. Thüm et al. [201] make the distinction between "abstract" and "concrete" features. The former allow to structure the feature model by introducing auxiliary elements, with no direct impact on the implementation.

2.4.2 Variation point based modeling

The Orthogonal Variability Model (OVM) is a language that documents variability though variants, variation points and variability dependencies among them. This model can be used across all artifacts, both in problem and solution spaces.
Figure 2.1 illustrates the concrete syntax of OVM.

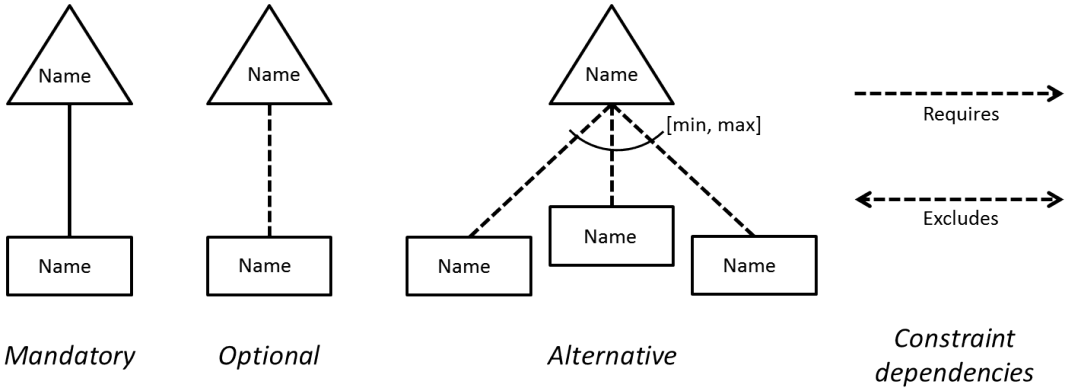


Figure 2.1: Legend of the OVM representation of variability

As the figure shows it, variation points are represented as triangles, and variants as rectangles attached to them. The figure also shows the five types of variability dependencies that can be used in OVM to specify how variants of a product line must or can be selected: mandatory, optional, alternative, requires, excludes.

- A *mandatory variability dependency* between a variation point and a variant describes that this variant must always be selected when the variation point is considered for the configuration at hand.
- An *optional variability dependency* between a variation point and a variant describes that this variant can be selected but does not need to be selected.
- An *alternative choice* is a specialization of optional variability dependencies. An alternative choice group comprises at least two variants which are related to a variation point by optional variability dependencies. The *[min, max]* bounds define that at least min and at most max variants can be selected for the product at hand.
- *Additional dependencies* between variation points and variants, e.g. to enforce that two variants of different variation points cannot be selected together.
- *Artifact dependecies* connect the variability model to artifacts, enabling reuse of engineering representations.

Like the feature model, OVMs are also extended for practical purposes. For instance, the Quality Aware OVM model [169, 170] extends existing OVMs to include non-functional attributes and takes them into consideration in the product line constraints and configuration processconfiguration processes.
The COVAMOF approach [184] follows the same principle of orthogonality and encompasses the concepts of variation points and variants. It documents variability in a uniform manner through first-class representation of variation points and complex dependencies. Variability is captured in two variability views (e.g. variation point view, dependency view) through the following concepts:

- *Variation points* are associated to engineering artifacts, with the following types: optional, alternative, optional variant, variant and value. The variation point also contains additional information such as the realization in the engineering artifacts, binding rationale etc.
- The *variation point realization* captures dependencies between variation points on different levels of abstraction, when there is no immediate realization in the engineering artifacts.
- *Dependencies* between variation points have the following types: logical, numerical and nominal.
- *Associations* between variation points and dependencies have the following types: *predictable*, *directional* and unknown. These allow a more flexible expression of dependencies, even when it is known that a variation point has an impact on other elements, but the precise information is not yet determined.
- *Dependency interactions* define how two or more dependencies interact.

One element that distinguishes this approach from other variability modeling techniques is the representation of complex dependencies as first class citizens, through a graphical modeling language. Although the language addresses the modeling of software variability, the concern of representation of complex dependencies remains valid in the context of SE. However, languages like SysML can cope with numerication relations (e.g. parametric diagrams), thus in our opinion the main question that remains open in MBSE is the separation of dependencies relevant from a variability perspective and their coupling with variability in system architectures.
Other extensions and uses of OVMs come in the context of model based engineering, as explained in Section 2.5.

2.4.3 Decision based modeling

Decision models are another modeling technique that reasons about variability in terms of open decisions in a derivation process. DOPLER, introduced by Dhungana et al. [73], is one of the most representative decision-oriented approaches. Its aim to provide guidance for the derivation process by exposing choices available at a given stage (e.g. through *visibility conditions*) and to link the problem and solution spaces: stakeholder variability is represented using decision models, and architecture and components variability is represented through an asset model. Some of the relevant concepts in the DOPLER approach are the following:

- *Decisions* are the core concept of DOPLER. Values can be assigned during derivation, based on the type of the decision: *boolean*, *string*, *number*, and *enumeration*.
- Dependencies described through *visibility conditions*, *decision effects* and *asset dependencies*. Visibility conditions define a hierarchical structure, whereas decision effects describe decision inter-dependencies. Finally assets dependencies describe the impact on the engineering artifacts.

In this context, decisions play a central role in the derivation process and to support reuse. In SE the decision is the main element that leads towards solution refinement. According to Parnell et al. [159] "the systems decision process is a general problem solving process". The general decision process needs to be tailored to the system, the studied problem, and the stage of the system life cycle [159].

Thus, in SE its meaning is much broader, and although it can have important implications for reuse, it does not necessarily target reuse of engineering artifacts in a product line. From a systems engineer's perspective it can be interesting to properly capture those design decisions with implication for product differentiation in a product line (e.g. architecture alternatives), and that are relatively stable (reoccurring, reusable) throughout the product line lifecycle.

2.5 Model based approaches and product lines

While different classifications are possible, the following discussion makes a clear distinction between approaches that manage variability (i) *by coupling* of general purpose languages with external tools and notations, and (ii) *by extension* of the general purpose language to include concepts from well established variability modeling techniques. A similar distinction was proposed by J.M. Jézéquel refered to as "Amalgalgamated Approach" and "Separated Approach" [121].

2.5.1 Coupling variability modeling with general purpose languages

FeatuRSEB Griss et al. [99] integrate feature modeling with Reuse-Driven Software Engineering Business (RSEB) approach, while later the proposed FeatuRSEB method is extended with SE concepts [92]. Their comparison of software and SE practices reveals some noteworthy facts [92]:

1. Requirements are first class citizens that are elicited from stakeholders but also result from analysis and trade-off studies.

2. Rationale and decision making are first-class citizens.

3. The analysis is on a higher level of abstraction ("functional analysis has a more general meaning" [92])

4. Clear determination of subcontracting boundaries based on component idenfication (including software components).

FeatuRSEB uses a feature model that is linked to use case model in UML, but in the SE extension, they are also linked to other system model elements: (i) parametric diagram elements - that allow to capture systems constraints; (ii) requirements in requirement diagrams; (iii) trade analysis alternatives as class diagrams with related Measures of Effectiveness (MOE). Design alternatives that result from the trade-off analysis are

themselves a source of variability: they have obvious reuse implications in the realization of the system since they can lead to different system architectures. Furthermore, they represent alternative choices that can be easily represented in a variability model. Thus, a possible downside of this approach may be that these "trade-off" decisions are not directly exposed to the engineer as variability (and in consequencen in the derivation process), unless they are explicitly modeled as features, instead of being only linked to features.

PLUSS The PLUSS approach [87] is based on FeatuRSEB. It was introduced by Eriksson et al. [87] and combines use case modeling with feature modeling. It manages variability in both software and SE models by passing through high level artifacts: use cases and scenarii. It distinguishes among several kinds of variations, with a limited scope: use cases; scenario, flow of events and actors in scenarios [87]. All these are coupled with a feature tree that provides an overview of all variability. Instead of using a feature model to provide a synthesis of variability, features allow to instantiate use cases based on their selection. Thus the main role of the feature model is in the derivation process. The feature model is based on the initial FODA notation, but adapted to suit specific needs.

Common Variability Language Common Variability Language (CVL) description The CVL is the answer to the request for proposal of the Object Management Group (OMG) for standardization of variability modeling. The approach focuses on the definition of a generic language that integrates variability concepts. According to the request, the language must include a technique for representing variability separate from the engineering artifacts (represented here by a *base model*), as well as a mechanism for relating the variability specification to elements from the base model. A feature like notation (called *VSpec*) is integrated within the concrete syntax of CVL. The structure of CVL is the following (revised submission [13]):

- The *base model* represents a collection of engineering artifacts represented as an instance of a MOF metamodel.
- The *foundation layer* supports the definition of variability abstraction, which covers *VSpecs*, *Constraints*, *Resolutions* and that of variability realization, through *variation points* related to engineering artifacts. VSpecs, which are similar to features models, provide an overview of variability in the artifacts and are related to variation points. Variation points describe the impact of a single configuration step on the base model, and allow the expression of several kinds of variability in the engineering artifacts: existence, substitution, value assignment, opaque variation point.
- The *compositional layer* provides ways to encapsulate and reuse variability specifications through *Configurable Units* and *Variability Interfaces*.

To summarize, CVL integrates a feature-like concrete syntax with useful concepts for variability modeling, providing the means to relate variability models to engineering artifacts, to express constraints through OCL, reuse and encapsulate variability models, specify impact of configurations on the base model. The purpose is to define variability in a way that is independent from the engineering artifact model.
The *variation point* concept captures the impact of choices on the engineering artifacts. In spite of the similar name, this appears to be different from the orthogonal variability modeling approach, which uses *artifact dependencies* to relate variants and variation points to artifacts.
Other works and tools, both academic and commercial, follow the approach of coupling the feature model with a wide range of engineering artifacts, including model based representations ([51, 130]). However, these do not address a specific methodology and tend to address a wider range of product line support functionalities.

2.5.2 Extending general purpose languages

Model driven approaches provide the means for representing artifacts on a higher level of abstraction in software and SE. Models are also good candidates for reuse, since they document engineering artifacts with more uniformity on different abstraction levels. Model driven and product line approaches both address common concerns [171]: increased productivity and quality of products and automation of design and development processes as much as possible. As Royer and Arboleda point out, these are complementary [171] approaches, and provide an opportunity for improved productivity by leveraging commonality through modeling and automation.
In the context of a model driven approach, product line concerns can be represented by the means of a general purpose language, like UML and SysML. In general, there are multiple ways of extending UML [14]: (i) feather weight extension - visualy distinguish items by using keywords; (ii) lightweight extension - involves using UML Profiles [207] (iii) middleweight extension - involves specialization of UML meta-types (iv) heavyweight extension - involves creation of domain specific meta-types and reuse of existing language units (through copy and merge).

2.5.2.1 Introducing variability through Use Cases

In software and SE, Use Cases define interactions between a role and the analyzed system. In SE they are used to represent the system mission or goals. Use Cases are used as a mean to introduce variability in many of the existing approaches [87, 95, 99, 104, 119]. Since variability is first introduced to analyze variations in stakeholder requirements, Use Cases are perfect candidates to identify domain variability. Gomma H. identifies three types of Use Cases, which are tagged with the corresponding stereotypes: $<<kernel>>$ - for Use Cases present in the core artifacts of the product family, $<<optional>>$ and $<<alternative>>$ Use Cases. They also introduce variation points in Use Cases in two ways: (i) as small variations - in the (textual) description of the

use case itself, by providing alternatives and conditional statements; (ii) by using the <<*extend*>> and <<*include*>> relationships.
John and Muthig [119] define generic Use Case models, which capture the constraints (e.g. optional, alternative and value ranges) outside the Use Cases, so that diagrams are not overloaded with information. They use the stereotype <<*variants*>> with Use Cases and Actors to introduce variability in system modeling.
Von der Massen and Lichter [211] define a metamodel that extends Use Cases to support representation of variation points and variants. The simplified metamodel is presented in figure 2.2.

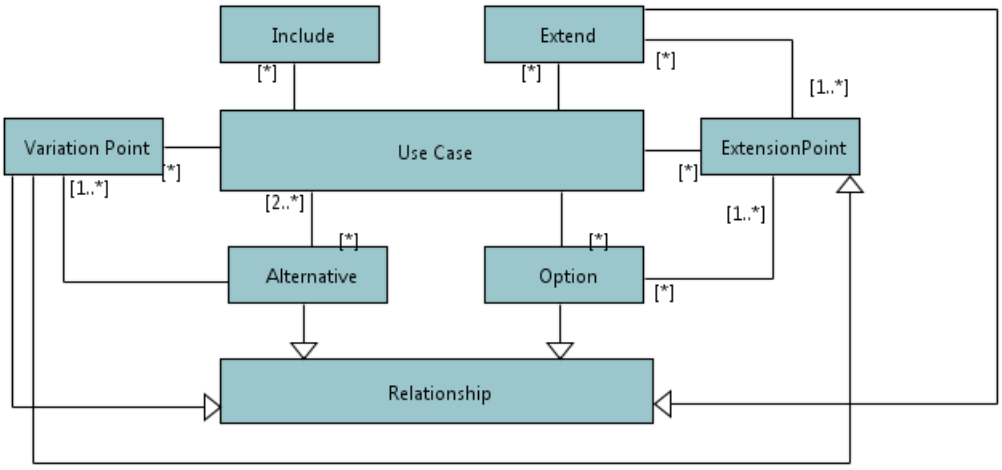


Figure 2.2: Use Case extension to represent variation points and variants [211]

Two types of relationships are used to include or extend use cases with *alternatives* or *optional* use cases.
Although they are a widely accepted modeling technique, Use Cases are less suited for non-interaction based and non-functional requirements, and this can introduce the same limitation to variability, when the "Use Case" concept is not extended, and is used within the boundaries of its initial (UML concept) purpose.

2.5.2.2 Embedding feature models

In several approaches UML models are extended to include feature modeling. Most approaches which involve UML models favor lightweight extensions (e.g. profiling) to introduce variability concepts. The following extend UML models using stereotypes to introduce feature based concepts:

- Claus et al. [61] uses UML Class diagrams to represent feature trees using the following elements: (i) <<*Mandatory*>>, <<*Optional*>> and <<*Context*>>

stereotypes (applied on classes) to represent the corresponding feature types; (ii) $<<requires>>$ and $<<mutex>>$ stereotypes (applied on dependencies); and (iii) adnotations (as UML notes) to represent constraints.

- Gommaa et al. [95] take a similar approach, but also use stereoypes for other concepts: (i) representation of constraints between features (e.g. $<< alternativeFeature >>$); (ii) $<< defaultFeature >>$ to represent default elements, when no selection is performed; and (iii) $<<parametrized\ Feature>>$ to represent mandatory features with variable attributes.

- Possompes et al. [165] introduce a UML profile to model software product line concepts, where the "feature" concept extends the "Component" UML metaclass. Contraints within a feature group are represented with the stereotype $<<featureSet>>$. Binding to model artifacts is also addressed at this level through the stereoype $<<BindingPredicate>>$ [164] .

2.5.2.3 Introducing variants as a core variability concept

The OVM provides the core concepts to represent variability throughout the product life-cycle, and can be easily used through extension in general purpose languages. For the approaches that extend UML, there are often differences in the core elements that are extended. The use of OVMs is flexible and it depends little on the process adopted by the organization, since they contain a limited set of concepts, and their relations do not overlap to existing UML modeling practices.
Ziadi et al. [216] introduce a UML profile to support product line modeling [217]. They use the following concepts:

- The stereotype $<<optional>>$ applied on classes is used to represent static architecture elements in class diagrams.

- The stereotypes $<<variation>>$ and $<<variant>>$ applied to abstract classes and subclasses

- *Generic constraints* (UML dependencies, also formalized in OCL when they are between optional elements or variants) and *specific constraints* (OCL meta level contraints).

Moon et al. [150] develop a tool that supports specifying domain variability and commonality in requirements. They define a metamodel including variability concepts around requirements. Variability is specified on different levels of detail:

- *First level decidable variation.* Variability is realized through common and optional properties (CV Property).

- *Second level decifable variation.* Variability is represented as *variation points* and *variants*. Variation points specify cardinality, which restricts how many variants can be selected: exactly one variant must be selected, at least one variant must be selected, one variant may be selected, zero or more variants may be selected.

- Furthermore, variants have the following types: computation, external computation, control, data. These specify the nature of the variation (function, pattern, data structure).

Halmans and Pohl [104] introduce additional graphical constructs to render variation points explicit in Use Case diagrams, in addition to the use of stereotypes explained in section 2.5.2.1. Thus, the construct is a tree-like structure, that precedes OVMs. The concrete syntax is presented in figure 2.3.

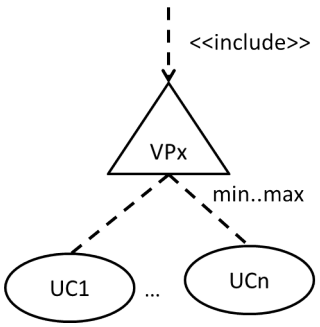


Figure 2.3: Variation points and variants in a Use Case diagram [104]

The variation points are represented as a triangle, with the cardinality specified below as a $[min..max]$ interval. Variants are represented as Use Cases with the stereotype $<<variant>>$. Variation points can be

- *Optional* : if the min element that specifies cardilality is zero.
- *Mandatory* : if the min element that specifies cardinality is greater than zero.

In the same way, depending on cardinality, relationships to variants can be:

- *Optional* : if $min = 0$, or $min > 0 \wedge max < n$, where n is the number of variants associated to a variation point.
- *Mandatory* : if $min = max = n$, where n is the number of variants associated to the variation point.

Like in FODA [123], the optional relationships are marked with an empty circle, while mandatory elements are marked with a filled circle (e.g. attached on the variation point end of the relation).
Halmans and Pohl [104] distinguish between two types of variability: *essential* and *technical* variability. The *variation point - variant* representation deals with essential variability, including concerns like: functionality, system environment, quality etc. Technical variability is software engineering already contains mechanisms for representation specific to the means of implementation depending on the level of abstraction: programming languages (encapsulation, inheritance, instantiation, if-defs etc.), model

based development and code generators, modules, static & dynamic libraries etc. Variation points in use cases can be mapped to corresponding variations in implementation, since in software engineering the product often shares the same production environment (e.g. operating system, software platform).
SE deals with artifacts on a higher abstraction level and includes software, electronics, mechanical subsystems. The implementation often passes through intermediate formalisms and development environments: 3D component specifications, simulation, embedded software artifacts, safety analysis etc. In consequence, direct mapping is not always possible, or it has to deal with tool interoperability issues. Orthogonal variability, formalized in the OVM model takes a step towards the solution, by specifying variability across all engineering artifacts.

2.5.3 Some specific concepts of variability in MDE, MBSE and at Renault

Multiviews. One of the fundamental roles of MBSE is bringing multiple "sources of truth" together, reconciling different points of view to reach the best system solution for a given context. In consequence, one essential requirement for a MBSE modeler is to integrate multiple viewpoints a consistent way. They are often related to project phases, organization, types of analysis (e.g. structural, functional), and specific domain knowledge of each stakeholder (e.g. electrical, mechanical).
A view addresses a subset of stakeholder concerns as defined by the viewpoints conventions and rules. There are many approaches to provide multi-view support in variability modeling [30, 34, 111, 136, 152]. These typically cover the following use cases: specification of variability models within views, generation of views that hide part of the variability, visualization of variability, ensuring model consistency. According to Hubaux et al. [111] three fundamental issues can be addressed (in multi-view feature modeling): view specification, view coverage, and view visualization. The dynamic generation of views is useful when dealing with large complex models, as is the case of the Renault vehicle range. Specific in house tools exist for this particular use case, that are able to generate a matrix-based dependency model within a scope of variants defined by the user. However, in MBSE our focus was on the specification of models consistently through views and on leveraging existing views in derivation.
In environments based on UML, view representation is already well supported by the modeling tools, and consistency can be ensured : (i) on the metamodel level (ii) by specification of constraints (e.g. through constraint languages like Object Constraint Language (OCL)). We combined both approaches in the case of the variability model, while for the system model elements we applied variability propagation techniques as proposed by Tessier et al. [196]. Instead of checking consistency of already defined optional elements, the reasoning is reversed, In the case of variability propagation in UML models [196]:

1. An initial set of optional elements is defined in the system model.
2. New optional elements are *discovered* based on rules that take into account the

static semantics of the system model.

3. New constraints are automatically generated, such that the new constraint model is valid (e.g. at least one solution exists that satisfies the constraints).

We relied on the Sequoia (CEA) constraint model in the MBSSE project, which already includes tools for verification in connection with an external constraint solver.

Level of detail of variability. Arboleda, H. proposes the coordinate use of model driven and product line engineering paradigms to achieve reuse and support the derivation mechanism. The approach relies on the notions of *fine grained* and *coarse grained* variability, which enables users to configure model elements individually [40]. During derivation the concepts of *FineCondition* and *CoarseCondition* allow the user to define the configuration in terms of selected/ non-selected features and constraints in the binding model [38].
Indeed, according to our experience at Renault, vehicle configurations cannot always be specified with enough detail based on high level features, depending on the type of artifacts managed. In this context of the Renault Documentary Language, the current practice is to introduce intermediate variables that are labeled as "technical", since they are not necessarily exposed to customer or stakeholders.
In the context of Bill of Material (BOM) at Renault, the relation between vehicle features and physical parts is known as a (diversity) "use case"[1] [42, 43]. The diversity[2] use case is a logical expression in conjunctive normal form, that defines in which vehicle configurations a physical part can be used. However, the level of detail that should be captured in the variability model and the structure of the model is an open question, and part of this project. For example a configuration variable that is relevant for requirement reuse, is not necessary for the manufacturing processes.

Model transformations. In order to obtain derived product members, models in model driven engineering are transformed, sometimes in several stages (e.g. Arboleda proposes a 4 stage transformation process [39]). Two types of transformations are possible:

- In *vertical transformations* the source and resulting models belong to different levels of abstraction. Models are instantiated based on domain metamodels and information specified by the user.

- In *horizontal transformations* both source and resulting models are situated on the same level of abstraction. Reuse is typically achieved in this way by removing, adding or replacing model elements, according to user specifications.

[1] "cas d'emploi"

[2] The term is *diversity* is used here in order to avoid confusion with the notion of *use case* in MBSE.

Engineering design decisions. The INCOSE MBSE Challenge Team [127] defines a framework that leverages SysML and integrates variability modeling through the application stereotypes:

- A <<*variation point*>> represents any system item subjected to variation, including requirements, blocks, etc.
- A <<*variant element*>> is an artifact that is specific to a specific system configuration, thus it is optional depending on the context.
- The <<*variant*>> stereotype applied on UML Package elements allows to define specific system configurations and contain the elements in the corresponding package.
- <<*Variations aspect*>> package contains all <<*variations*>> and <<*variants*>>, while the <<*structure aspect*>> package contains all variation points (or structural core elements) [114].
- Finally, constraints are expressed using notes, as illustrated in the example available online [16].

In opposition to other approaches, the most of the variability is not represented as fist class artifacts, but as a specialization of other system items (e.g. a requirement, a block, a system can have the stereotype <<*variation point*>>).
Ryan et al. [173] apply the same approach in regards to modeling of variability, but further leverage variability to support flexibility of architecture definitions for trade-off analysis. Variants of the system are assessed through optimization software against the MOEs defined in the SysML model. Their approach brings proof of the utility of the coupling of variant management and trade-off analysis, by using both design parameters and optional architecture elements to represent variability.

2.5.4 Methodological aspects in product derivation

Derivation is the process of system conception and development based on existing product family artifacts through reuse, adaptation and complementation of the product definition, in accordance with the new stakeholder requirements. Derivation requires a complex coordination of activities ranging from *configuration* of variability models, transformation of system models, decision making, conception and development of new artifacts, capitalization. Current tools are able to automate a part of these activities.
While different definitions exist, often referring only to the configuration process, and in MBD addressing model transformation [216], we adopt the aforementioned definition. In consequence, derivation is closely related to organization processes. This fact is even more important in SE, where engineering activities follow a well established process, with deliverables at various project milestones.

2.5.4.1 Basic variety generation

Variability generation techniques enable diversification through changes on the product level, which are linked to variability model configurations in Product Line Engineering Engineering (PLE). According to Tseng & Jiao [205], "*variety generation* refers to the way in which the distinctiveness of products can be created". They introduce three basic methods: attaching, swapping and scaling, as presented in figure 2.2.

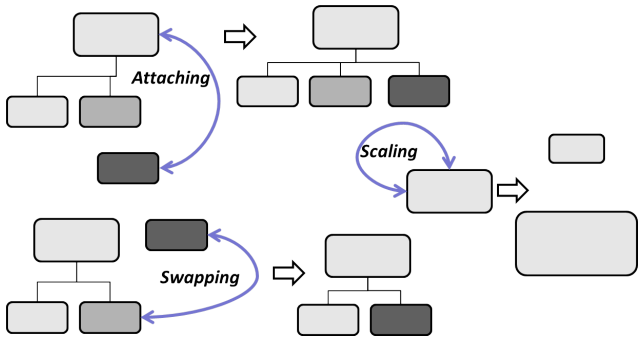


Figure 2.4: Basic variety generation techniques [205]

While these concepts are oriented towards physical structures (e.g. scaling) as found in the BOM, they can be generalized, and applied in a wide range of contexts:

- *attaching* - refers to the presence or absence of an artifact in a given context.
- *swapping* - refers to the potential use of several interchangeable artifacts in the same context.
- *scaling* - refers to the change of parameters that characterize a given artifact. For physical components this is indeed equivalent to a scaling in size.

Furthermore, more complex variety generation techniques can be created based on these three basic techniques.
A similar approach is used in CVL, in the context of Model Driven Engineering (MDE). The foundation layer of CVL, implements four types of variation points: *existence*, *substitution*, *value assignment*, and *opaque variation point*. The *opaque variation point* allows for domain specific variability to be associated with the product artifacts.

2.5.4.2 Process related concerns

The process of reusing existing systems definitions in new projects is dependent on the organization processes. Typically product derivation approaches rely on configuration techniques and automated reasoning on variability models based on different technologies: CSP [49, 146], SAT solvers [147] and more recently SMT solvers [149]. However,

in complex systems with large variability models, derivation can become cumbersome and error prone due to: (a) the large number of available choices, (b) flexibility - need of changes in the variability and binding models to reflect new product constraints. The later issue is especially true for the automotive industry, where products are constantly evolving, through incremental innovation, each new system being developed bringing new improvements and options in respect to the previous one. However, a large amount of elements is still common between products of different generations, as illustrated in figure 2.5.

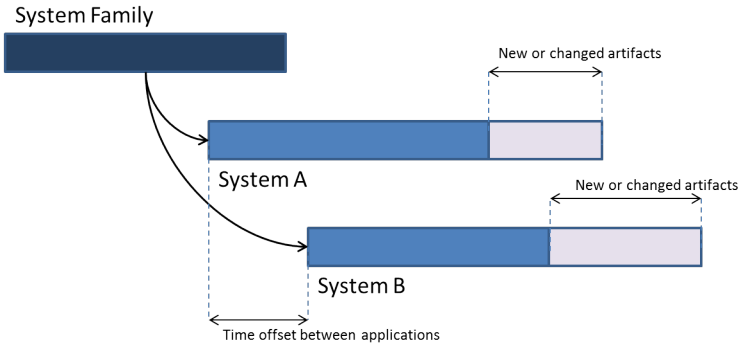


Figure 2.5: Reuse and time offsets of product/system lifecycles (adapted from Boas et al. [53])

Boas et al. [53] point out the impact of divergence and product lifecycle offsets on reuse in several industries through seven case studies. Among these, they argue that the highest divergence impacts the following sectors: automotive, military aircraft, and communication satellites. This is evidence that commonality and variability are not static, even after most design decisions for a product family are made.
From a product derivation perspective both time offsets and changes in family artifacts are poorly supported by current tools due to lack of flexibility. According to Perrouin et al. [160] automated product derivation approaches are inflexible and cannot cope with unforeseen requirements. They introduce a process to allow more flexibility:

1. *Pre-configuration* is done based on feature models and a set of related core assets, through model transformation.

2. *Product customization* adds new elements to the product model.

3. Finally, the product is validated by checking against illegal combinations in the product model. Core Asset Instantiation Constraints allow to define constraints in the product model based on OCL.

Guidance and methodological support can also improve the derivation process. O'Leary et al. [154, 155] introduce *A-Pro-PD*, an agile approach for product line derivation, which is intended to provide guidance for small, co-located teams. The approach is divided in two layers:

- *Phase increments layer* : short units of work that focus on a particular aspect.
- *Iteration life-cycle layer* : structures phase increments in order to deliver stable builds. Builds are delivered in an iterative way until they reach the established objectives.

The two mentioned approaches each deal with at least one of tool support and process aspects. However, they are neither tailored for a SE process, nor for a particular MBSE approach.

2.6 Discussion: A perspective on variability management in Systems Engineering

Variant modeling covers, in most of the existing approaches, aspects related to customer perception of variability (e.g. product options), communication of requirements variability among stakeholders, mapping to variability implementation in software artifacts (e.g. language specific variability mechanisms). However, since SE manages artifacts on a higher level of abstraction, there are some key aspects that variability modeling needs to take in to account in MBSE, that currently have little or no support:

- *Issue 1 (methodology)*: The use of variability models needs to be clarified from a methodological standpoint: how variability is used in each of the system analysis phases, how product line based reuse can be achieved in newly developed products (e.g. products that include both new and existing SE artifacts).
- *Issue 2 (redundancy)*: In MBSE, redundancy between the relations and concepts in the variability and system models should be avoided, both through an appropriate methodology and a metamodel that is complementary to SE concepts. For instance, from a method point of view, the level of detail of documentation of variability needs to be established, to avoid duplication of optional artifacts in both system and variability models (e.g. a feature is created specifically to represent only one optional system function). From a modeling perspective, the meaning of concepts although different, can sometimes overlap (e.g. decision vs. variant, feature vs. function), and concept relations be remodeled by the user in both variability and system models (e.g. hierarchies, system allocations vs. variability dependencies between different levels of abstraction).
- *Issue 3 (gap)*: While the way variability is implement in software artifacts is well known and documented, in SE there is a gap between customer and stakeholder variability and variability implemented in engineering artifacts. The intermediate SE artifacts in MBSE are, on the one hand, high level system architecture descriptions, on the other hand decisions and decision rationale that support trade-off analysis. If variability in the architectures can be implemented using some known mechanisms (e.g. presence, replacement), the relation between design decision and variability needs disambiguation. For instance existing decision

oriented variability models do not address the same issues as SE decisions. However SE decisions may have deep implications on reuse.

The following discussion takes into account the issues identified above, but also the more specific needs for variability management in SE at Renault , described in Chapter 3.

Product line concepts in the automotive domain Variety has an impact on the whole of the product life-cycle: from the definition of the customer offer and stakeholder requirements elicitation, to manufacturing and after-sales services. Although each activity potentially has its specificities in respect to variability management, we can at least distinguish between two types of variability definitions in the automotive sector, introduced in SPL by Pohl et al. [163] : (i) external variability - a client oriented definition, enabling customer to grasp the offer of different manufacturers and request custom configurations, and (ii) internal variability - definitions of variability used in the engineering and documentation of products, internally, hidden to external stakeholders. Before analyzing variability specific to any of the internal processes (e.g. SE), it is important to understand the concepts that enable the expression of the internal and external variability.
The automotive commercial offer is usually formulated based on a limited set of concepts. An example of configuration options from the Renault France and UK websites is presented in Figure 2.6.

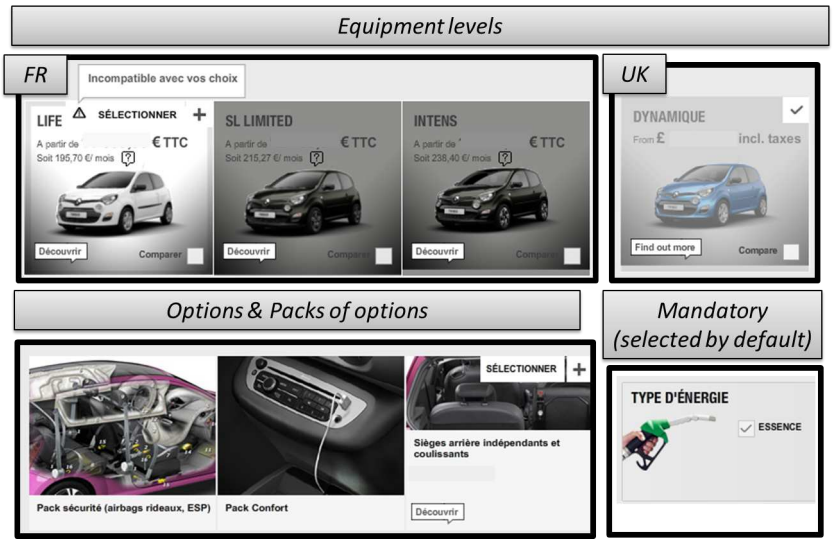


Figure 2.6: Elements from the Renault online configurator, with specific options for France and the United Kingdom

The commercial offer in the configurator is defined based on:

- *options* - essentially optional vehicle features (e.g. airbags, ESP)

- *packs of options* - a set of features that can be selected together;
- *equipment levels* - marketing concepts used to predefine partial configurations of features
- *features selected by default* - mandatory vehicle features, that are always included in the product configuration and cannot be modified by the customer.

Dependencies and constraints defined internally, can invalidate available options at any step in the configuration process, in order to avoid invalid product configuration. This is also visible in Figure 2.6, where the "Life Pack" is marked as incompatible, due to previously selected features.
It is important to note that vehicle features, as expressed in the online configurator, become mandatory depending on the context of the commercial offer. For example, in the case from Figure 2.6, the "cruise control function" is mandatory for UK (included in the single pack of options available), while in France in can be selected by the customer, individually.
In engineering, the prominent strategy applied in the automotive world today revolves around *commonalization*, materialized through vehicle platforms, modular structures, common manufacturing processes, but also shared assets between different manufacturers. In response to the need for customization, and to constant refreshing and improvement of the vehicle range, engineers shifted the design approach towards a common core of elements reusable for multiple vehicles in a vehicle family.
According to Siddique [181], some of the variety design concepts applied in the automotive industry are: standardization, delayed differentiation, modularity, robustness (seen here as insensitivity to variation), and mutability (capability of the system to be reshaped, reconfigured). It is interesting to notice, that all these approaches applied in the engineering of vehicle today focus on commonality (e.g. reusable modules, standardization) and minimization of the impact of change both in time and space (e.g. delayed differentiation, robustness, mutability).
We define *differentiation* as the sum of changes in both time - across product versions and space - across the product family.
In contrast to a typical product line approach, where both variability and commonality are defined a priori, in the automotive industry it is mostly commonality that is designed as the core of vehicle family. In conclusion, internal and external product deal with variability in different ways:

- *Customer visible* product definitions focus on the definition of variability, as vehicle options, which can become mandatory depending on the context (e.g. country, regulation). Commonality is not always exposed explicitly in vehicle configurators.
- *Internal* product definition focuses on commonality, which is at the core of the product line. Differentiation is managed both across the vehicle range and in time only within the limit of existing approaches and tools.

Cardinalities in variability and engineering artifacts Variability is expressed by means of cardinalities in some variability modeling techniques, such as Cardinality Based Feature Models (CBFM) [67] and OVM models, in order to constrain the number of choices in a group of options. In feature models they are also applied to individual features. However, in model based languages cardinalities are also present.
In external variability, some constraints that can be formalized with cardinalities are often used, when options are grouped in packs of options, we have identified two cases (at Renault) in current definitions:

- [0..n] - Individual option can be selected in a pack of n options.
- [n..n] - Only the pack of options can be selected as a whole (individual options are not allowed).

Obviously, marketing rules with other cardinality based restriction could be defined (e.g. at least m out of n options), depending on the product domain.
In SE, system models typically cover the needs for expression of cardinalities. While examples may be scarce in physical architectures, some straightforward examples include:

- [4..5] wheels in a vehicle configuration, taking into account spare wheel, which is optional.
- [1..2] electric motors can be included in the architecture configuration of the electric parking brake system.
- [2..n] (at least 2) braking functions must exist in a vehicle, for safety reasons. (main and emergency brake)

As explained previously, physical products are designed for reuse by means of product platforms, modular structures, scalable artifacts, delayed differentiation. Typical variety generation techniques can be used to represented variability in architectures: presence, replaceable, design parameters. We did not identify any important examples that require the use of cardinalities.
From a configuration point of view, the need for cardinalities depends on the nature of artifacts represented in the variability model: stakeholder and client options, design decisions, component supplier decisions, design parameters. In the solution we proposed, concepts in the variability model do not include explicit parameters, which are represented in the system model. Thus, assigning values to a set of parameters defined in the system model can only be done through the selection of "design decisions" in the variability model. This was done on the one hand for clarity, on the other hand to avoid frequent changes in the variability model. There are however much more detailed parameters that depend to can depend on each individual application (e.g. calibration parameters). However, these appear in models on lower levels of abstraction, and are outside the scope of reuse of systems architecture models.

Hierarchical structures A well established approach in the analysis of complex systems is the hierarchical decomposition of a problem into subproblems. According to Alfaris et al. [33], research on this subject was initiated in the work "Notes on the Synthesis of Form" by Alexander C. [32], who argued that the increased complexity of some design problems is beyond the human cognitive ability. Today the approach is present in many systems analysis methodologies [129] and modeling languages (e.g. SysML [193], OPM [78]), which apply this approach to structural and functional decompositions.
Decision trees are a tool for decision support used in a variety of domains. They consist in the graphical representation of decisions, ordered and in the form of a tree, where each branch leads to an outcome that depends on the set of previously taken decisions. They too are based on a hierarchical structure, with a different semantics, where branches correspond to choices that are mutually exclusive.
Yet another way to structure knowledge hierarchically is through *"is a"* relationships. In MDE and object-oriented languages, this type of relation is also known as subtyping: "every A *is a* B, if every object A is always a B object, too". An illustration of this concept in UML/SysML is the *Class Diagram* [207].
Finally, hierarchies are applied in variability modeling in the form of feature trees [123], enabling the representation of product variability and commonality, along with configuration constraints and dependencies. Since the concept of "feature" is very broad, it is possible to include different type of engineering items, as suggested in the example from Figure 2.7.

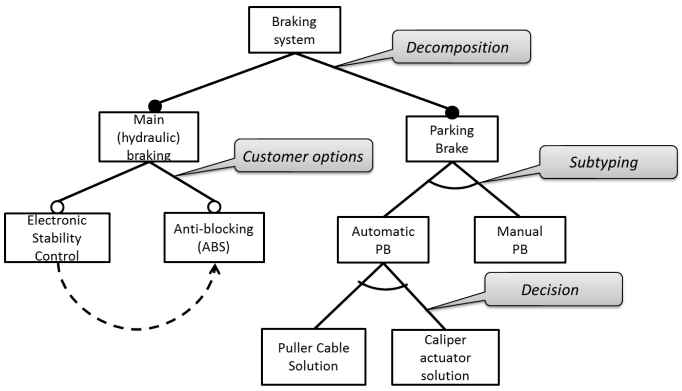


Figure 2.7: Example of hidden relations between features

The simple example captures some of the variability and commonality of the braking system of a vehicle:

- The braking system *is decomposed* in two subsystems realizing different (main) functions: (i) the hydraulic brake enables the vehicle to control the decrease in velocity (and trajectory depending on the options) and (ii) the parking brake immobilizes the vehicle when it is parked (at a standstill, with the engine turned off). Although there are multiple ways of representing variability with feature

trees, such features allow for a more structured, readable digram (e.g. Electronic Stability Control can also be attached to the root). As pointed out by Thüm et al. [201], in practice, not all features are mapped to engineering artifacts. Furthermore, some tools explicitly forbid the mapping of the abstract features [201]. *Composition* relations in feature models were also analyzed by Oster et al. [156] (in respect to Classification Trees), who point out that they always regroup mandatory features, as is the case in the example in Figure 2.7.

- The hydraulic brake includes two options, that are visible to customers [21]: the ABS (Anti-blocking) - prevents wheel locking and skidding, and the ESP (Electronic Stability Control) - adjust the vehicle trajectory when needed during in cornering.

- The parking brake function can be divided in two *types*: the *Automatic Parking Brake* (also known as Electric Parking Brake) and the Manual Parking brake. The latter refers to the typical lever found on many vehicle, that also serves as an emergency brake. Both manual and automatic parking brakes can be generally classified as "parking brakes" . According to Schürr et al. [180], classifications in feature trees are always optional. In the provided example the features optional and in addition, altenative, meaning that only one of them can be present. The mutual exclusion constraint is obviously specified for practical reasons, since the manual parking brake function would be redundant if the automatic option was present.

- Depending on the level of detail captured by the feature diagram, technical solutions may also be included. For instance the Automatic Parking Brake option can be implemented in two ways, both solutions proposed by automotive suppliers: (i) as single electric (DC) motor coupled with the typical puller cable and (ii) with electric actuators placed directly near the calipers. Depending on the technical context, from a SE point of view this is can be seen as make (design) or buy (supplier selection)decision with two alternatives.

The same example can be illustrated partially using an OVM model. For instance, the "Parking brake" can be represented as a variation point, regrouping two variants: the "Automatic PB" and the "Manual PB".

2.7 Closure

Many variability modeling techniques exist in the literature, but there is no solid empirical evidence that a single representation can cover all industrial needs [151]. According to available survey results, feature models are by far the most used among the techniques proposed in product line engineering research. However, in-house and even ad-hoc modeling techniques are often used (e.g. matrix representation, DSMs), which could be viewed as a solution to specific needs of different domains or organizations. Furthermore, in non software-intensive domains like SE, there is less visibility on the

tools and techniques used, which is also due to the fact that traditionally SE was focused on the realization of single systems.
In mass customization industries, variability impacts most of the organization activities, where each of these may have its specific needs. Thus, we believe there is a need to adjust modeling techniques to SE processes and practices, especially in the context of the automotive industry, where there is developments in managing variability in some activities already exist.

3

Problem Identification and Refinement

The automotive industry is faced today with the challenge of developing increasingly complex systems, but it also needs to integrate new emerging technologies while leveraging on existing designs of previously developed products. The concept of reuse is well described in different domains, such as software product line engineering, production, product development or marketing (as a mass customization strategy). In the automotive industry, we see SE as the domain that can bridge the gap between high-level customer requirements diversity and basic component development. The challenge is to be able to ensure traceability from customer down to components while being able to match requirement variability to component diversity and to ensure an effective management of variability at each level of abstraction or decomposition. We aim at developing a comprehensive framework for managing system variability under three main themes : development process conception, variability description and integration with current methods concerning vehicle development.

3.1 Overview of product diversity at the organizational level

Very few systems in the automotive industry are created from scratch, thus the engineer is confronted with the reuse of documents, models or, in more general terms, knowledge from previous experiences.
One automotive industry specific issue that increases the complexity of the problem, and emphasizes on the need to adopt proper computer aided development tools, is the specific product diversity. The challenge is to be able to deal with complexities coming from two different sources : on the one hand the vehicle range that can lead to a large number of possible configurations for a single vehicle family. On the second hand,

the need to be able to describe and manage variations in system description efficiently across different projects. We place our problem in the context of Renault MBSE activities which need to take into account vehicle diversity for the development of systems. While currently there are different efforts to specify a single variability language [192] there is to our knowledge no single variability model that is used along with SysML, nor any model that is used in MBSE and outside the software product line engineering scope. Furthermore, the problem of reuse has to deal with multiple aspects that depend on the context of the company, what the general process for product development is and particular methods used in development. We considered the typical development of a vehicle system, which we used as a case study. We distinguished between the different areas where items related to a system are placed using a simple three dimensional notation that makes reference to the level of detail where the asset is placed, the level of abstraction, and specific viewpoints by which this knowledge is captured. This intermediate framework allows us to track items through their lifecycle and across the different products that include them. Meanwhile, it can also allow us to map variation model elements to product models more easily by keeping a relatively loose coupling with other system models that are used in the context of model-based SE activities.
Reuse in a SE organization is often opportunistic. In the automotive industry efforts of reuse were focused on the component level, through strategies like product platforms or modular structures. Volskwagen, for example, establishes a platform as a common set of components, "a unit that has no impact on the vehicle's outer skin and that is a chassis including the inner wheelhouses".
Another critical aspect that has to be taken into account by a company that develops product lines is the client offer, and how the offer is structured in relation to the product definition and the development process. Most vehicle manufacturers use online product configuration applications that allow clients to express their requirements in respect to a predefined set of options. Each vehicle is built according to the vehicle configured online by a given client. The order then passes on to the plant, which assembles the desired product. The physical structure of the product and the production line are designed to support from the start the required client offer.
In SE (SE), the physical aspect, based on a structural approach is extended to other dimensions. SE has to take into account systems behavior, as well as requirements or model fragments (in the case of MBSE). Reuse has to take into account all these elements and has to be extended to all development items.
Innovation can be implemented in an organization in two main ways: through disruptive innovation, which involves a higher degree of risk by proposing products and designs of significant novelty, and incremental innovation, which leverages existing designs to create new improved products, and involves less risk for the company. Carryover, which is based on incremental innovation, plays a central role in automotive development. Even in more disruptive designs like the electric vehicle, which requires a rethink of many on-board and off-board vehicle systems, the conception is based on existing system requirements.
In this context, the role of the SE development activities is to enable the successful re-

alization of systems corresponding to the customer requirement diversity and enabling reuse of items along the development cycle. Figure 3.1, which is based on Blecker et al. [52], presents the general structure of a mass customization organization that integrates the SE process. Two possible declinations of the client offer are possible: the first ($ConfigurationSystem \rightarrow ProductDevelopmentSustem$) passes through the development activities in order to create the product definition covering the client requirement diversity; the second ($ConfigurationSystem \rightarrow ManufacturingSystem$) ensures the link between the client options and the production line (in plant) where the product will be assembled.

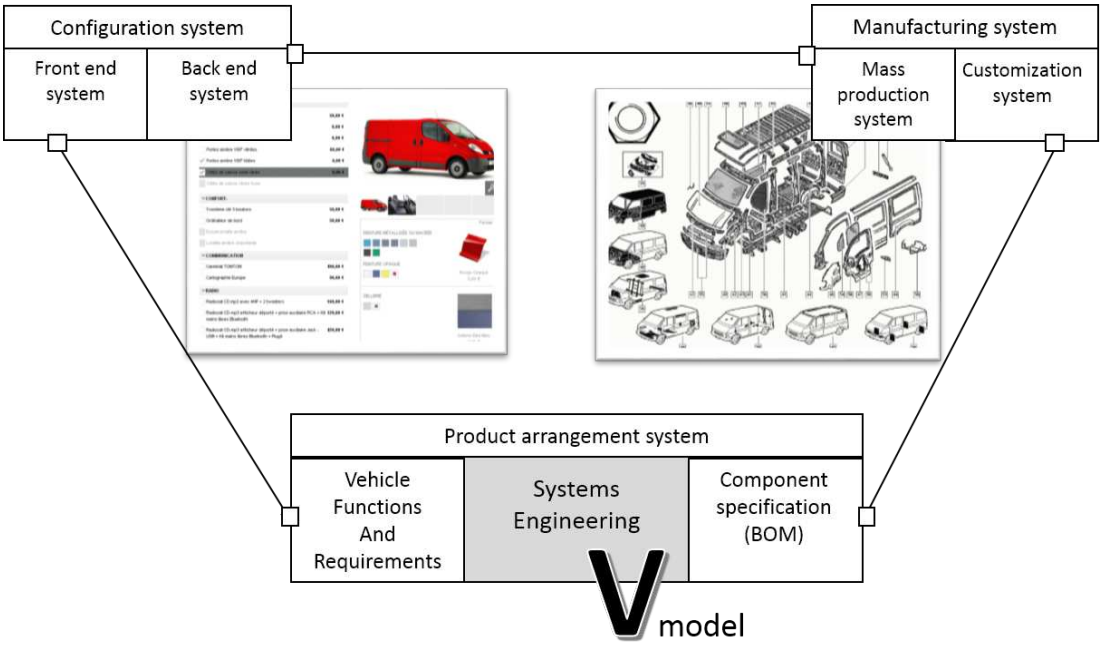


Figure 3.1: Product diversity at the organizational level

Since there is much commonality between different client offers both in time and space for a certain vehicle range, we can adopt a proper development strategy to reuse existing knowledge.

3.2 Product diversity at Renault

In mass customization, industries are driven by the customer variety, with companies adopting appropriate strategies to decrease time to market and production costs. A survey realized by KPMG International [115],concerning the automotive industry, reveals that there are multiple reasons for product diversification. In an industry where competition is strong and the market conditions are difficult, diversification is seen as

a means of keeping up to the competition, of keeping vehicle models up to date, and to integrate state of the art innovations and technologies. According to the same survey, 40% of automotive actors consider competition as a reason for increasing product variety. Diversification is a mean of keeping up to the constantly changing markets and searching for new potential of growth. Since the car is such a widespread good around the world, its markets are subject to influence from a large number of stimuli. These changes manifest as: cultural changes (e.g. preferences for green technology, urban vehicles etc.), purchasing power which depends in turn on local economic conditions, changes in law and regulation (e.g. carbon tax, parking taxes, insurance), local infrastructure (road conditions), etc.
Because most of the company activities have to deal with product variety, there are multiple aspects to consider on an organizational level in variability management: processes, information systems, tools and methods.

3.2.1 Processes for managing product variety at Renault

Product diversity covers different product needs, which in turn impact physical product implementation and components from suppliers. At Renault, three main types of processes for managing product variety and structuring the product families, as suggested in Figure 3.2.

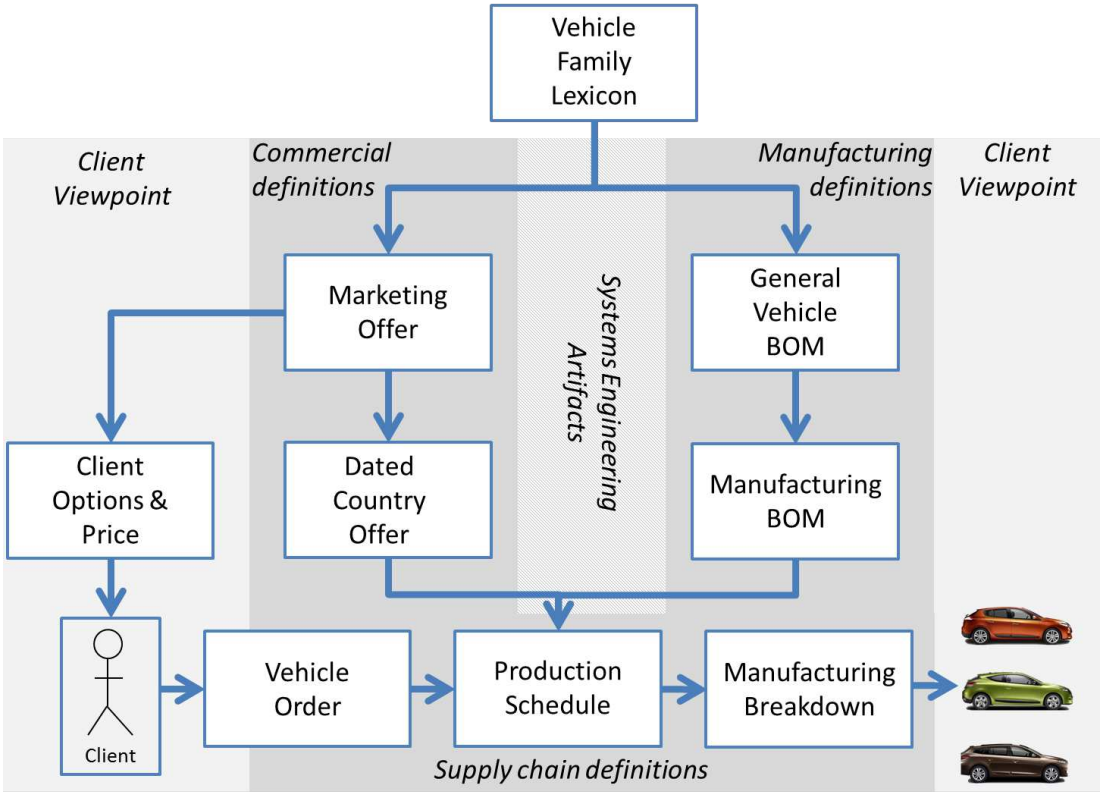


Figure 3.2: Variability management across different processes in the organization

One process (commercial definition management, in Figure 3.2) deals with structuring the commercial offer. The features of each vehicle model are defined based on the geographical location of the destination market. For a product feature to be introduced for a given vehicle, three criteria are considered (lexicon rule *lex064*) :

- The feature needs to induce diversity for the vehicle family, meaning that it is optional at least one context or market (e.g. *combustion engine* does not need to be a feature as long as there is no alternative, such as *electric propulsion*, *hybrid* etc.)
- The feature is offered as a choice to the client (final client or marketing network)
- The feature needs to have its own code/name unless: it can be implicit in a feature grouping some other characteristics (e.g. : *equipment level* requires a bunch of features), or the feature can always be expressed based on another feature already defined (e.g. : *heat-reflective glass* implies *heat-reflective windshield*).

Based on these definitions, a customer visible definition is also created, using informal language, which facilitates customer interaction. The mapping from customer language to vehicle features is shown in a simple example in Table, for a *MODUS* vehicle sold in

Feature name	Client expression	Feature code
Country	(implicit)	FRAN
Model	Renault MODUS	MDU
Version	1.5 Dci (diesel) 80cv	LXP15DCI80
Color	Mint Blue	396
Seat Covering	Leather Seats	CUIR02
Opening Roof	Opening Roof	TO
Manual Air Conditioning	Air Conditioning	CA
Tires	default choice	BA
ARA date	delivery before week 3 2005	0503

Table 3.1: Mapping between customer visible options expressed in natural language and vehicle feature codes

France with the following client options: *1.5 DCI (diesel) 80CV, Luxe Privilge, Mint Blue, with Opening Roof and Air Conditioning, leather seats, standard mounting, for week 3 2005.*

Some features, like the *Version* require other features that are not directly exposed to the customer as individual options, but through intermediate options.

A second process (Engineering Data Management) enables documentation of configuration specifications for components from the bill of material (BOM). These enable the configuration and manufacturing of individual vehicles based on customer feature selection.

Finally, a third process links customer orders in terms of vehicle features to plant manufacturing activities, to produce customized vehicles. The commercial and engineering definitions are needed to produce the specific parts list and manufacturing constraints for the particular vehicle configuration.

3.2.2 Variability in technical product definitions

The conception and development of each vehicle subsystem is realized by a group of actors representing different functions: sourcing and purchasing, engineering research and development, costs of goods sold, design, industrialization etc. Each of these functions relies on the definitions of vehicle features for managing variability in their own domain. Currently, the product definitions are oriented towards the description of vehicle parts or the BOM [58]. As shown in Figure 3.3, the information system supporting the definition of components in the context of vehicle features is related to many of the organization's functions.

In order to integrate variability management functionalities, the BOM (Figure 3.3) needs to provide alternatives and optional components. It is structured in the following way:

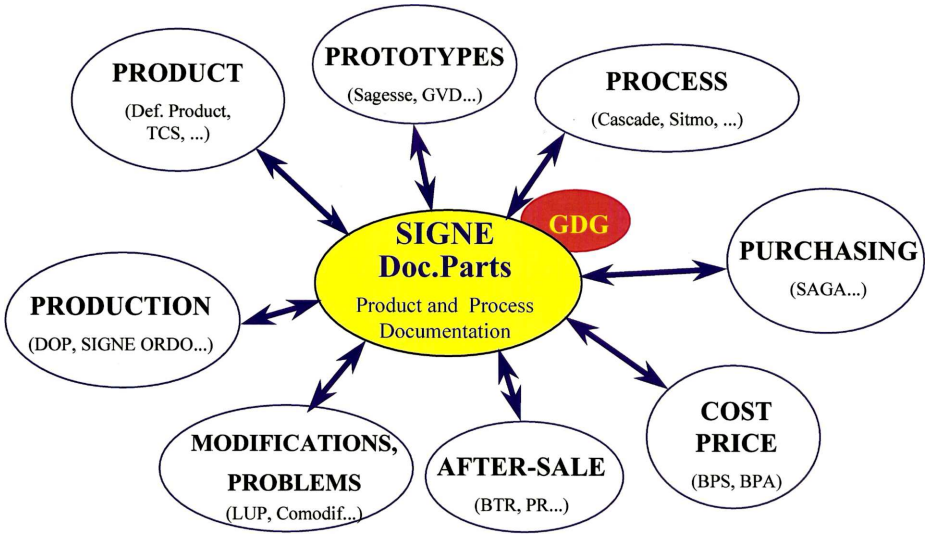


Figure 3.3: (Current) Variability Management Information System

- Function (here the term is used outside SE context) : is an element or part of the vehicle, e.g. *headlight*
- Variant : identifies the vehicle part in respect to vehicle features. Variants need to cover all vehicle feature configurations (also called at Renault vehicle diversity), e.g. *simple lighting*, or *double lighting*
- Generic component (solution element) : is an element that defines a specific technical solution corresponding to the specified vehicle configuration, e.g. *right headlight*, *left headlight*
- Component (concurrent elements) : defines all the alternatives of components that are compatible with the given generic component definition or technical solution. Usually this refers to alternatives from multiple suppliers for the same product (e.g. part no. 80xxxx supplier A, no. 90xxxxx supplier B). If the same supplier offers alternatives for the component, it can also be a part number for a specific component.

One of the shortcomings of a system focused on the physical structure of the product is the lack of support for fine grained representation of variability during conception and design of vehicle subsystems, as explained further in this chapter. Figure 3.4 explains how variability is gradually documented (at Renault) during development phases, with the purpose of manufacturing customized vehicles based on customer preferences. Most of the gains are based on scale economies and sharing common elements among different vehicles. The documentation of variability is tightly integrated into the development process, as suggested in Figure 3.4.
The most important milestones in the documentation of variability are:

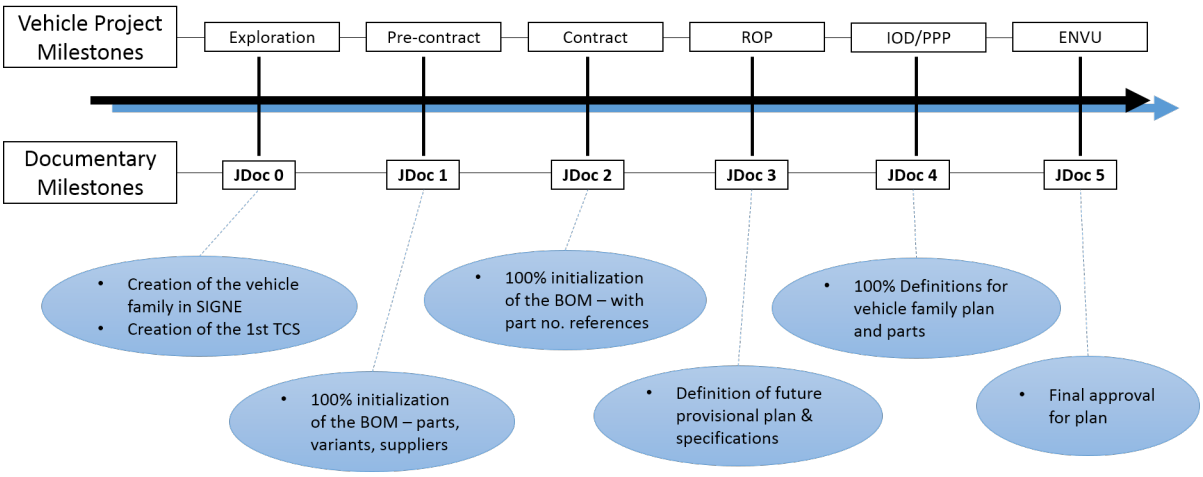


Figure 3.4: Development milestones for variability documentation

1. JDOC0/Orientation : the vehicle family is defined in terms of vehicle features and desired partial configurations (configuration tables/TCS).

2. JDOC1/Pre-contract : first definition of the BOM, at least in terms of solution elements and configuration definitions in respect to vehicle features.

3. JDOC3/Contract : first complete definition of the BOM, including specific components with part numbers, and configuration definitions in respect to vehicle features.

From a design and development perspective, variability definitions represent on the one hand requirements for what is to be developed (e.g. desired partial configurations), and on the other hand a deliverable associated to vehicle component specifications, in preparation for the manufacturing.

3.3 MBSE: example in the automotive context

Many publications at the intersection of product lines and SE actually focus on software. Of course, SPL are ubiquitous in cars, planes, trains, and other complex systems, but SE spans far beyond the software component [186]. In so far as the variability aspects are concerned, they need to be adapted to the MBSE models and approach.
In order to design our systems of interest (SOI), which includes electronic, mechanical and software components, we follow a framework that provides guidance and rules for organizing system architectures. The framework provides different viewpoints that cover the scope of the system architecture and relies on a modeling language to provide representations for each of the viewpoints, which in our case is SysML/UML with custom profiles for different domains (e.g. software, systems etc.)
Usually, modeling languages are different depending on the domain, and the interest of

the system level is to provide a holistic, integrated view on the product architecture, bringing different subsystems or disciplines together. We performed a survey some years ago in respect to model based engineering methods regarding the "model-based power train control" [68]. The survey revealed that the scope varied - either to consider the power train control system, or a part of the physical power train, while the activities ranged from upstream design or control algorithms to final vehicle validation. What this survey has taught us is that every "métier" defines its modeling approach and activities, which need to fit in the overall development process. As for variability management, solutions are not always well integrated into the overall process as is the case for other model based activities. The challenge was to manage variability across different SE viewpoints and also to provide interfaces with other activities and domains, from marketing and vehicle design to components. Not all of these domains need to have the complete vehicle information about variability, but it is sufficient to use partial views on variability depending on the concerns of each stakeholder or domain.

3.3.1 SE and Model Based Approaches in the automotive context - the Renault example

Variability management in SE emerged in the first place as a need that complemented the decision to adopt SE at Renault. SE was introduced through two initiatives [58]: (i) by "filling the least populated place" in respect to already well-established processes and skills [91] and (ii) by preparing innovations in Research & Advanced Engineering (R&AE) for reuse by providing early well documented architectures.
Considering the background of the automotive industry, where mechanical engineering has played an important role, often processes as well as information systems are oriented towards "parts engineering". The vehicle parts are centralized in BOM databases, with variability relative to vehicle features and configuration information for plant manufacturing processes. Therefore, SE was initially introduced to link customer requirements to components, which are usually developed by Tier suppliers and delivered to OEM factories where they are assembled. So far, variability specification has been essential for product assembly and reuse oriented design of physical components, but configuration definitions related to vehicle features (or marketing definitions) were provided after the product design was completed. This has brought difficulties in managing architectures which are still in the process of development, where there was no direct means to introduce new variations, which is often the case for new innovative products. Furthermore, document based definitions of systems (or more recent model based) were difficult to manage and reuse. Different solutions have emerged in the context of Product Line Engineering, where many variability modeling techniques were proposed [123][163][199][177]. The implementation of such an approach in a large organization would face some challenges (also pointed out by Filho et al. [93]), such as methodological support, adaptation of processes and engineering practices, but also integration to the specific context of the organization.
In respect to the second initiative, where SE was introduced in Research & Advanced Engineering - variability can be leveraged to introduce flexibility for architecture spec-

ifications and provide opportunities for reuse [2]: (a) of problem definitions and newly introduced solution alternatives; (b) adapt existing solutions to new requirements and vehicle features and (c) of adaptations of existing improvements of existing architectures such as integration of updated or improved versions of components. Automotive industry is experiencing some major breakthroughs today: new architectures imposed by new concepts (e.g. the electric vehicle) or the design and management of vehicle fleet, as a system of systems. SE provides a generic framework that addresses the challenges of these new breakthroughs.
The three main viewpoints used in our MBSE analysis, based on Krob D. [129] are described in figure 3.5.

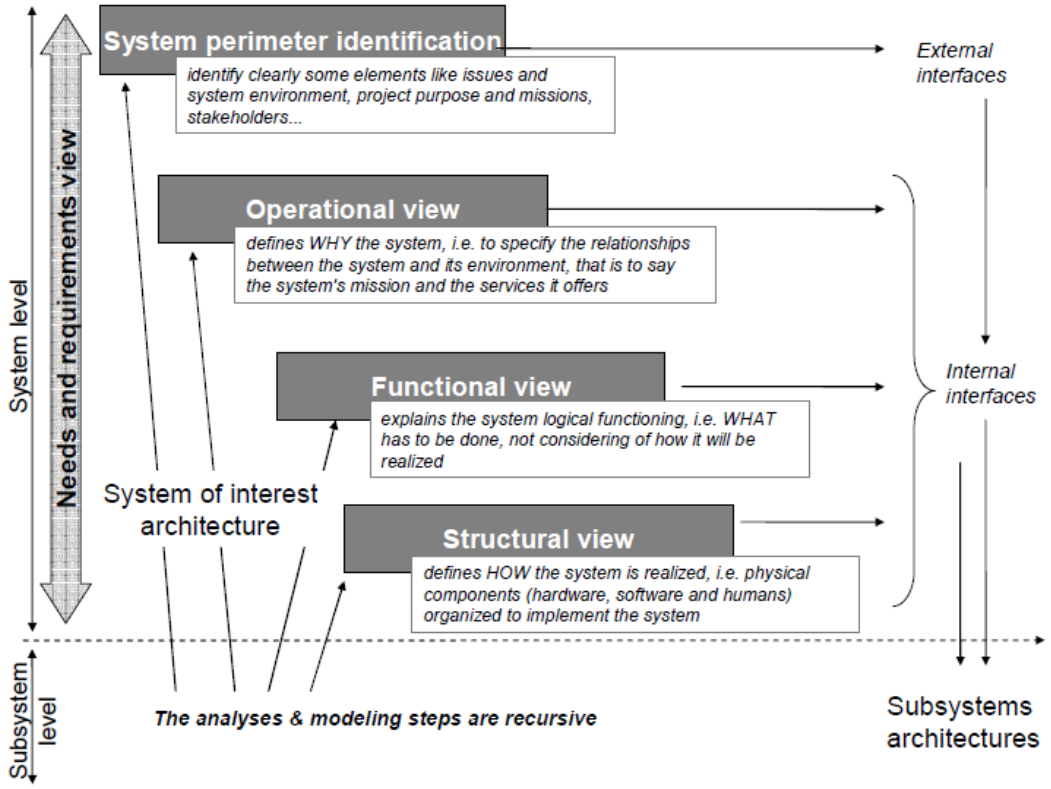


Figure 3.5: Summary of viewpoints in the proposed SE architecture framework [80]

(a) The operational viewpoint defines why the system is designed. It clarifies the mission, the services as well as the relationship of the system with its environment (actors, stakeholders, enabling systems etc.).

(b) The functional viewpoint explains how the system works, its functioning, what the system has to do to achieve its mission.

(c) The structural viewpoint defines how the system is organized, what is made of, how it is structured in respect to its components (hardware, software or human).

In the automotive industry, change (in generations of systems) and variability (in families of systems) are present on all levels, but in different proportions. On an operational and functional level variability is driven by different norms, regulations and customer profiles (requirements). However, changes in services are outpaced by changes in technology, and component supplier variety introduces much more variability on a component level. A set of representative diagrams were chosen from SysML, with complementary UML profiles for the representation of each of these viewpoints [58]. Product line derivation is performed in stages, by creating partial configurations for each of these viewpoints, as explained in section 4.7.

3.3.2 Variability in systems engineering

As is done for the deployment of the SE process, we distinguish between two types of products. These are different in terms of novelty, purpose, and the amount of reused items from previous experiences. Research SE projects (R&AE) propose new, improved designs and architectures for existing problem definitions, and aim at creating prototypes. Commercial vehicle SE projects involve system designs for commercial vehicles, where solution and problem definitions are usually well explored and documented in the *Domain Systems Engineering*, and thus provide opportunities of reuse.
The organization of family of systems models is presented in figure 3.6, corresponding to the company context.
A common repository would allow easy reuse of specifications from upstream research to downstream vehicle projects. Currently, support for variability for MBSE is only applied for research projects in order to provide valuable feedback for the application on a larger scale. By assessing the methodology and tools on a exploratory case studies (such as the EPB, the automatic lighting system) and validation case studies (the automatic parking assistant) we intend to understand the challenges of deployment on a larger scale.
A typical system design scenario (Systems and Research Systems Engineering) relies on the SE process and tools, with specific activities for the management of variability and reuse. There are three main types of activities: (a) searching and reusing items in the domain models (repository), (b) defining new system items and (c) updating domain models to reflect changes. Each of these types of activities has specific problems due to the complexity induced by the "spatial system family dimension". Developing the system domain by updating models (activity type c), provides a convenient way of capitalizing information for later reuse. However, because the items usually target only a few vehicle projects (upper level system configurations), there is a drawback concerning the reduced re-usability of items across the family of systems.
Domain Systems Engineering activities address the development and definition of reusable items: definition of reusable collections of requirements, functions and components; development of product platforms, which will provide the base for future applications;

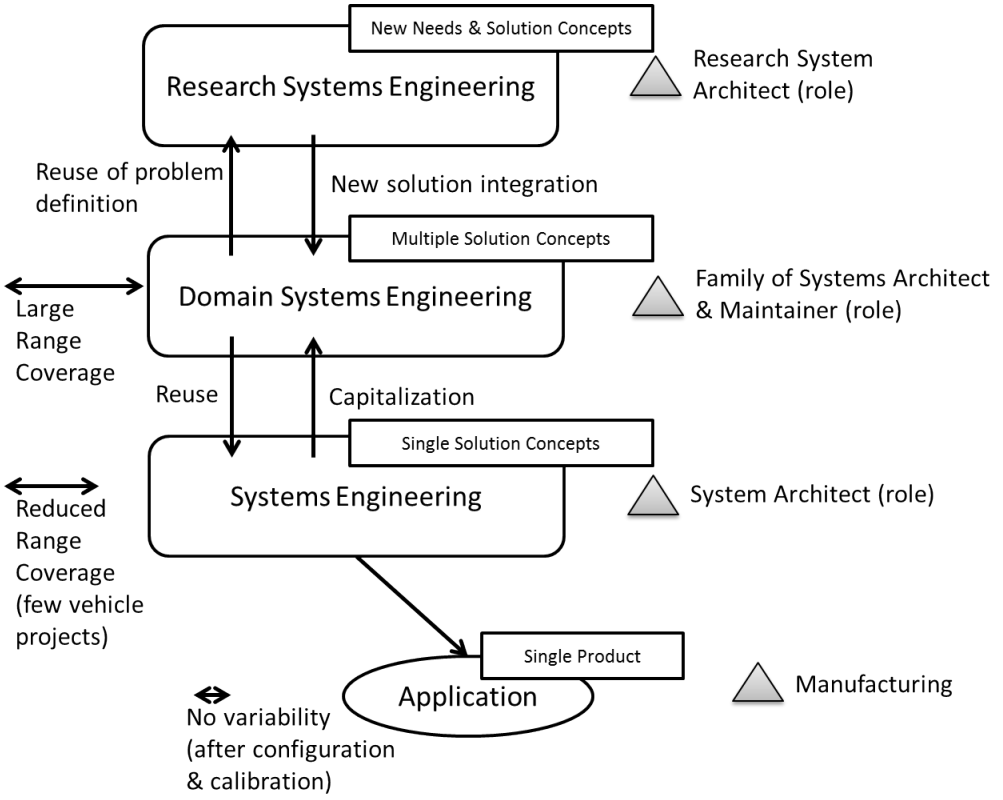


Figure 3.6: Proposed organization of product line models and processes for MBSE

development of reusable structures with functionality (modules). The development of such items requires an upfront investment [54][214], which does not address directly immediate industry issues and customer needs.

3.3.3 Constraint based representations for vehicle configuration

At the core of product line engineering practices is constraint programming [176][143][49][47], which enables both the representation and the analysis of product line models. At Renault, the Documentary Language [43] enables the description of variability on the vehicle level. Astesana et al.[43] describe the way variety (or "diversity") is expressed and exploited at Renault (and probably in many similar customization oriented applications): as constraint satisfaction problem variables. Different business activities occur at different times in the life-cycle of a vehicle range: modeling and documentation of the product range, design processes, management of the bill of material, online exploitation by customers (through the online configurator). All of these activities need to rely on the same constraint based description of the product line [43]. This is indeed the aspect addressed by all variability models, which is essential for the derivation of a

single system model. Furthermore, constraints are introduced through the commercial offer and stakeholder requirements, but also due to technical, architectural dependencies. Technical constraints are the result of dependencies to variable resources from within the system or from external enabling systems.
Not only do these technical constraints need to be taken into account during derivation, but they may, sometimes, need to be rendered visible for the commercial offer or on organization level. At Renault, features visible on an organization level are introduced under the authority of a group that manages the "product diversity". Enabling the system engineer to represent architectural constraints in relation to variability mechanisms becomes a necessity for dealing with the complexity introduced by variability.
The variability expressed in the Documentary Language follows the "version-option" model and partitions variables in two sets:

- *Major variables* define the main configurations of a particular vehicle model (referred to as "version").
- *Other variables* define vehicle features which depend on the particular model, and thus on the configurations described by major variables.

Constraints are then expressed in two manners:

- *Major constraints* are expressed as explicit configurations, usually as tables.
- *Option constraints* relate non-major variables to particular (partial) configurations defined by major constraints.

3.4 Engineering scenarios

The different engineering scenarios define general themes that the development of systems sharing items might address:

1. Deriving a single system from a system family
2. Develop a system from a partially derived system
3. Integrating of a single system into a product line
4. Synchronizing systems development
5. Merging two product lines

Deriving a single system from a system family – this is the nominal case where a single system can be derived from a collection of domain items.

Develop a system from a partially derived system – the derived system does not represent a completely defined product. This scenario requires that system engineering activities continue to complete the system asset collection.

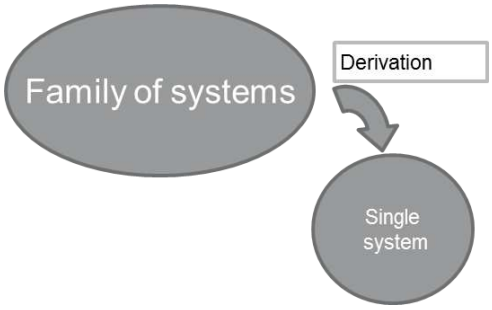


Figure 3.7: Deriving a single system from a system family

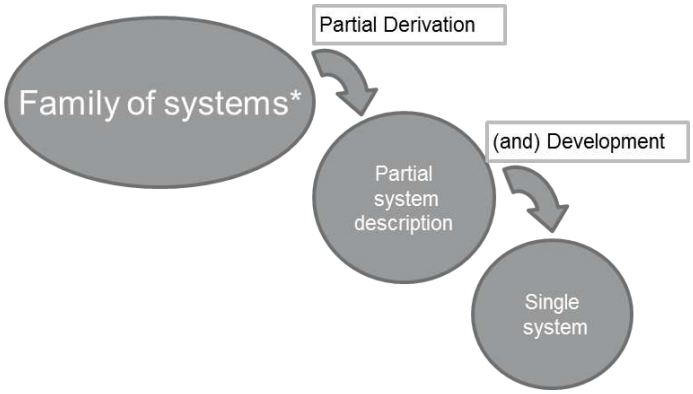


Figure 3.8: Developing a system from a partially derived system

Integrating of a single system into a product line – this is often the case of an innovation that enters mainstream development, a new system or option is proposed, but it is still possible to share system and life-cycle items with existing system (e.g. improvement of a braking system by adding ABS, ESP etc.). When the innovation is based on or represents an improvement of a member of the actual product line, the scenario corresponds to an incremental development of a product line, where items and options are constantly enriched.

Synchronizing systems development – several systems within the same product line are developed in parallel, while all reuse related to commonalities between systems is anticipated and planned. The purpose is to maximize reuse decisions among a set of projects or contribute to the common items.

Merging two product lines – this represents a middle case which can be useful when different systems are developed independently, initially, but common elements can be identified for future developments (e.g. development of similar products in different companies that eventually establish a partnership).

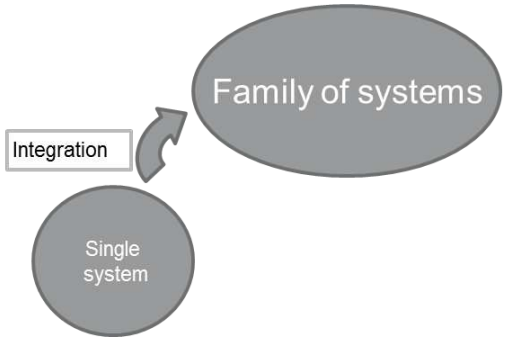


Figure 3.9: Integrating of a single system into a product line

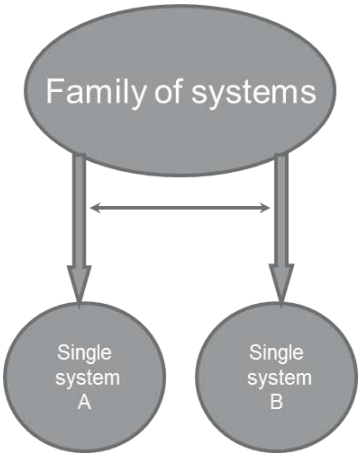


Figure 3.10: Synchronizing systems development

3.5 Use cases for variability management tool support

Engineering activities need to be supported by proper tools. In an environment where SE is confronted with product diversification, the tools need to address variability management activities, in relation to the specification of the vehicle range. At Renault, the vehicle variability is addressed at three levels:

- At the vehicle level, variability is described by the Documentary Language, by the representation of vehicle features
- At the system level, variability is represented using vehicle features, with little or no tool support. Requirement and architecture documents are created ad-hoc for specific vehicle configurations.
- At the component level, tools provide means of describing variability integrated to the bill of material (BOM) data model, stemming from customer offer, component solution alternatives and supplier alternatives.

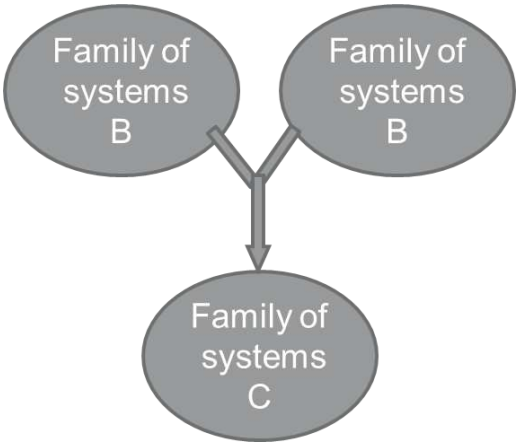


Figure 3.11: Merging two product lines

As discussed in Subsection 3.3, at Renault, vehicle features are directly mapped to components. However, current SE activities at Renault lack support for representing, managing and referencing variability on vehicle and component level.

Problem 1 *An intermediate level is needed to describe variability, so as to bridge the gap between customer requirements ("prestations") and vehicle physical components.*

In addition to engineering scenarios, which are related to the engineering process, we define use cases for variability management support in SE. The Use Cases for variability management support (at Renault), provide a more detailed description of what the functionality of tool support should be, compared to the engineering scenarios in Subsection 3.4.
Figure 3.12 presents the flow of requirements from vehicle families to components, as it is its present form, which should allow for a better understanding of each use case.

This is a complex scenario, where variability is present on every level. A generic component *Cmp1* can have multiple physical implementations and may be obtained from multiple suppliers. It also needs to satisfy requirements from multiple systems, and the higher level system also contains variability. Meanwhile, the higher level system is designed for a family of vehicles, based on a common platform. Once the engineer in charge of component *Cmp1* collects requirements issued by systems *SA1, SA2*, and *SA3*, they are able to choose a solution alternative. The selected component is associated to the vehicle configurations where it is used, by the definition of a logical expression called "diversity use case", described in more detail in Chapter 4.
The following subsections present the different use cases to be addressed by a SE tool with variability management support, at Renault. These use cases were mainly identified by direct observation of the way systems are developed at Renault, using the tool SRx[1], by identifying its shortcomings and ad-hoc practices not yet supported by tools.

[1]System requirements management tool at Renault

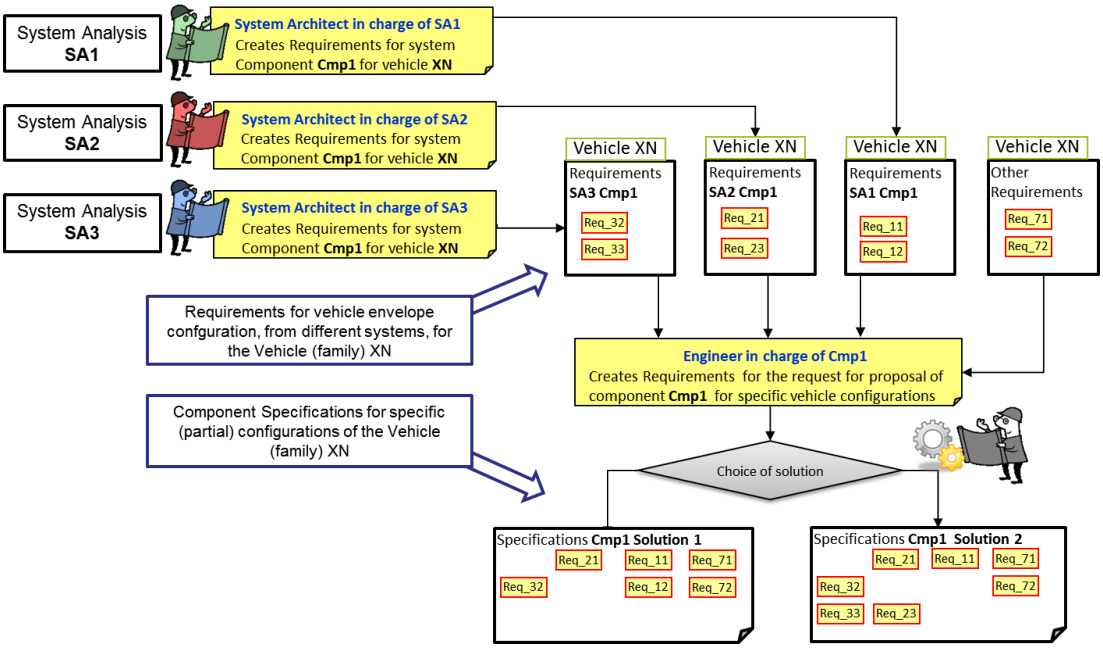


Figure 3.12: Requirements definition workflow through different levels: vehicle, systems, components

The main case study considered for the identification of variability management use cases was the Vehicle Climate Control system [81].

S1 : Description of System Engineering Artifacts and Variability In a model based approach, the first necessary step to manage variability is simply describing it. Systems engineers need to perform the following activities:

- Relate SE items (requirements, functions, flows, components etc.) to vehicle features defined by the Documentary Language.
- Capture fine grained variations of SE items, system design alternatives, and component alternatives.

S2 : Generation of "envelope" system and component specifications A system "envelope" refers to a partial definition of a system family. This means it contains all neded common items and possibly variable items, but it does not necessarily contain all needed details to generate complete product configurations. The architecture definition and requirements, corresponding to a given vehicle diversity, provide a base of existing reusable assets for new systems.
The system architect has to provide the requirements from the system's point of view, as shown in figure 3.12. In addition to requirements, the description needs to contain the following:

- The vehicles family code and vehicle configurations that the component needs to address. These are associated to the requirements and specification document.
- The system configurations that need to be addressed. These are associated to the requirements and specification document.
- Any technical or marketing constraints related to the configurations are included.
- For each SE item (e.g. requirements), the vehicle and system configurations addressed need to be specified, if they are different from the ones associated to the document.

The component requirements and specification document[1] is issued for a particular vehicle family project. For a new vehicle project, a new document needs to be generated.

S3: Generation of the diversity use cases for components Diversity use cases[2] define the logical condition for a component to be always present in a vehicle configuration. For instance, let's consider that component *Cmp1* from the example in Figure 3.12 is used on every vehicle derived from platforms *XM* and *XN*, except those vehicles with anti-lock braking system (ABS) and opening roof (TO). Then, this could be expressed as:

$$(XM \vee XN) \wedge \neg(ABS \wedge TO)$$

where *XM, XN, ABS* and TO are logical variables that represent the vehicle platforms and vehicle features.
These expressions are associated to each vehicle component, once the system is developed. However, the description of variability on system level, provides the necessary information to generate expressions automatically.

S4: Filter an "envelope" for selecting a subset of variants from the all possible variants The system engineer needs to provide partial descriptions of the system

- at different stages of the project,
- to reduce variability in case of a change in requirements,
- to reduce variability in the solution space, by removing solutions that are not retained.

S5: Identification of the number of necessary configurations for components Request for Quotation The differences between system configurations may introduce variability for a given component. Before initiating a request for quotation (RFQ) it is useful to have an estimation of the minimum number of component configurations needed.

[1] Cahier des charges
[2] "cas d'emploi" in the Renault terminology

S6: Brief description of variants in a family of systems This functionality refers to the generation of a brief description of variability present in a family of systems, with details about the origin of the variations, impact etc.

S7: Minimization of number of tests and of test support The purpose of this tool function is to obtain the configurations of test vehicles that cover all tests taking into account vehicle and environment variability. In order to obtain the configurations for test vehicles, the engineer should provide:

- System and vehicle variability, and system description (e.g. requirements, architecture, tests)
- Pertinent vehicle configurations for each quality attribute.
- List of quality attributes to be tested.

S8: Generation of partial system description as base for a new specific system analysis For each system analysis, a scope of the study is defined in terms of target vehicle configurations. Most often new developments rely on reuse of previous solutions. Thus, by taking into account the target vehicle configurations, a partial description of the system should be generated from the family of systems description. This partial description should retain all reusable elements with respect to the scope of the new study.

S9: Comparison between configurations from the same family of systems For the same system family, the tool should provide information about the differences between two configurations, globally, or in respect to a certain type of SE item. For two configurations *Cfg1* and *Cfg2*, the generated information includes:

- items common to the two configurations
- items in *Cfg1* and not in *Cfg2*
- items in *Cfg2* and not in *Cfg1*

For example, it is useful to compare configurations for different geographical areas. These may reveal differences in requirements, in particular due to variability in regulations, differences in physical components, due to choices of local suppliers, etc. Understanding these differences can contribute to reducing variability by changes in system design, where this is possible.

3.6 Requirements for a variability management framework in Systems Engineering

Since SE is a domain that requires a multidisciplinary approach, the research strategy depends on the specific context of the problem.

We identified our problem as a design problem, where the subject is a socio-technical system. We are required to provide the means (models) and processes that enable the successful realization of families of systems that satisfy the diversity of customer requirements.
Our research strategy is based on a set of development scenarios for families of systems that we propose, based on observations of development practices involving single systems and carry over practices. The need for tailoring processes to the development of families of systems is justified also by the current ad-hoc practices that leverage existing documents and specifications to adapt a given system for different vehicles or to create a new generation with improved characteristics.
At the same time we identified a list of requirements that a framework for the development of families of systems should satisfy. The requirements and process scenarios should allow us to reason about the solution.

3.7 Problems in current tool support and practices

While a model based system-engineering approach brings a better structured and finer description of development items, the product line paradigm is rarely addressed in this context, or it is based on solutions that are applicable to a context with a limited, reduced amount of diversity, or to a specific kind of product like software applications. From the perspective of Renault tools and methods, variability links customer preferences to vehicle parts, but with little or no support for SE items.
Typically, a single vehicle subsystem is designed for several vehicles from the same family. The vehicle subsystem may need software calibration or configuration for a specific vehicle configuration. Often solutions are based on reuse through : (i) carry-over - by adapting existing solutions from previous vehicles; or (ii) carry-across by adapting a solution from a vehicle family to other vehicles. As explained in subsection 3.2, variability milestones are related to vehicle project milestones, especially JDOC0/Orientation and JDOC1/Pre-contract, when a solution is already defined. However, the present tools and methods are oriented towards the documentation of components or vehicle parts. In respect to variability in SE, the following issues can be pointed out:

Problem 2 *Lack of tool support for the* ***configuration*** *of models and document based specifications for a single vehicle family. (a single vehicle project)*

System Engineers are often faced with the problem of producing document based specifications of a system which is to be used on multiple vehicle configurations from the same family. The large number of configurations makes it difficult to create a document for each configuration. Meanwhile, ad-hoc techniques have had limited success, especially when the number of vehicle configurations was important.

Problem 3 *Lack of tool support for the* ***reuse*** *of models and document based specifications over different families of vehicles. (multiple vehicle projects)*

This problem is different from the previous in respect to the system scope, which involves vehicle families developed as different projects. This means that configuration techniques are still necessary, but emphasis is on reuse and capitalization.
In addition to the current problems related to variant management in SE, some constraints are imposed due to existing practices, technical and organizational context, expressed as requirements. Table 3.2 provides the identified requirements, that the approach and modeling techniques must satisfy.

Id.	Type	Requirement
1	Modeling	The approach shall provide a representation for reuse, in extension to the system modeling paradigm (used at Renault)
2		The model shall capture design decisions that lead to different designs of interest.
3		The variability management model shall not represent information captured by system models (non-redundancy).
4		The approach shall be compatible/integrated to existing legacy tools and processes in respect to variability.
5		The variability model shall enable the user to interact with the tool during the derivation process to express his choices.
6		The number of choices available during the derivation process shall be less than the number of variable elements in the system model (minimize the number of choices).
7	General	The approach shall support current reuse strategies.
8		The approach shall be applicable to multiple levels of system decomposition and abstraction.
9		The approach must be applicable to all types of SE artifacts.
10		The approach must integrate all stakeholders from the organization, that are concerned by variability management.
11		The approach must support traceability of variability along the system's life-cycle.
12		The approach must support reuse of SE models across (vehicle) projects for systems belonging to the same family (e.g. brake systems, lighting systems)
13	Methodology	The System Architect (SA) shall be able to obtain partial configurations in respect to deliverables for different milestones of development process.
14		The System Architect (SA) shall receive the definition of the technical and requirements diversity of the system to be studied as input.
15		The SA shall be able to perform partial configurations in respect to specific (automotive) systems decomposition and abstraction levels.

Table 3.2: Requirements for variability management in our organizational context

3.8 Forms of variability in systems engineering

According to the Hall-ETH process [100] designing a system requires problem solving and searching for solution variants. They are classified according to three levels: variants of solution principles, variants of overall concepts, variants of detailed concepts. For each existing, previously defined problem statement (in the form of requirements), existing solutions should be taken into account before considering a novel design.
Variability appears at different levels of the system during development. The number of unresolved variants may decrease while the development advances, because at each step in the development process, a certain number of decisions and actions are taken. Variability in SE needs to take into account specific aspects characteristic to this domain. Reuse addresses not only the solution, through system architecture elements, but also the problem definition, and product line models need to trace existing requirements to candidate solutions. SE models enable the engineer to represent elements both form the problem and solution spaces of a system specification. Figure 3.13 illustrates the two dimensions present in the design of families of systems : (A) problem resolution and (B) reuse.

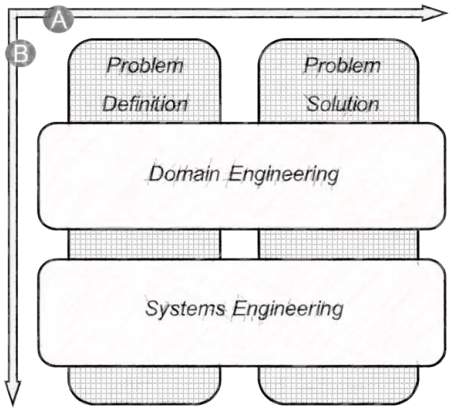


Figure 3.13: Reuse across problem and solution space

Product line engineering identifies two separate types of processes [63, 163]: domain engineering where reusable items are defined, and application engineering – where a specific application is derived from the existing elements. Depending on the degree of novelty of the new design and on the externally induced variations, the amount of reused elements may vary and new elements may need to be designed.
Several aspects that should be taken into account were identified, depending on the phase of development [2]:

- Solution vs. Problem definition
- Physical vs. Behavioral variability
- Dynamic vs. Static

- Single Stage vs. Multi-Staged configuration

Besides, several forms of variability were identified, depending on the way variability is expressed in system items description (from requirements to architecture).

- presence / absence
- number of instances
- choice in a list of alternatives
- configuration variables
- replacement
- amount of detail left unspecified for an asset (generic asset)

3.8.1 Sources of system variability in systems engineering

Variability is one of the main causes of complexity in the automotive industry. The product diversity is centered on the client needs and preferences, which are observed and followed, anticipated or even created before they would naturally occur. However, this is not the only source of variability in the design of a system and all stakeholders can potentially induce variations in the system design.
Most of the sources of variation correspond to different stakeholders or attributes of the system. These alternative requirements have to be satisfied in different configurations of the system and are treated in the conception phase of the development cycle. However, variability in high-level requirements impacts other elements in the development cycle [163], as suggested in Figure 3.14.

If the problem definition, given by system attributes and stakeholder requirements are one major source of variation for family of systems, the second source is given by the system context. We refer to the higher level system as the "technical context" of the system that is being developed. Thus for any vehicle subsystem, the technical environment represents the vehicle in all required configurations.
Figure 3.15 illustrates this phenomena through an example. It can be noticed that certain elements are variable expressed through different forms of variations, such as alternatives or optional elements. Decisions concerning these elements depend on the higher level system, in this case the vehicle configuration.

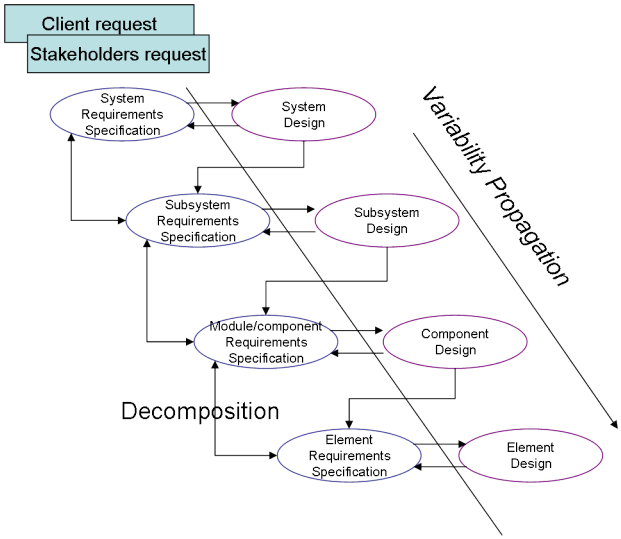


Figure 3.14: Propagation of variability through the SE development process

3.8.2 SE Standards and families of systems

Several standards and capability models for SE exist, with differences in scope, levels of of description detail, system life cycle span and phases.
ISO/EIA 632, for example defines processes for engineering a system in respect to the following themes: acquisition and supply, technical management, system design, product realization and technical evaluation. The system design phase requires on the one hand defining the problem, through requirements elicitation and derivation, and on the other hand formulating and performing trade-offs to obtain the best candidate for a solution, as Figure 3.16 shows it.

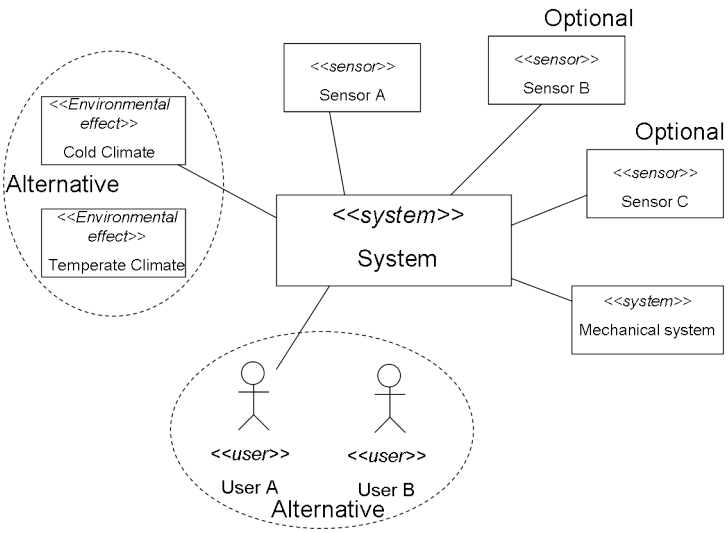


Figure 3.15: Context variability impacts the system solution

In the case of a family of systems, the two phases would require making reuse choices along the way, taking into account the different nature of items belonging to each phase. The way the choices are made during reuse-based system design depends a lot on the approach for developing the product line and deriving individual products. The methodology may involve, for example developing and evaluating alternative solutions or deferring design decisions, where tracing variabilities and reusing items can be of great help. Figure 3.16 illustrates different reuse choices that can be performed on the items corresponding to each development phase, when developing a product based on reuse.
SE standards address the subject of developing single systems, whereas they do not focus on the processes needed for developing systems sharing common items. Work is currently in progress to integrate product lines in standards [122]. Standard ISO/IEC 15288 specifies that process tailoring can be used to adapt the process to the organization's needs.

3.8.3 Models in systems engineering

While models may themselves represent a source of variability, our focus is on how to adopt a variability representation that is both able to capture and propagate variability from external sources, but also to provide a link to internal system representations.
Many representations and architectural frameworks [209] exist in SE – DoDAF/MoDAF [108], NAF [64], FEAF [208], TOGAF [107, 120], etc., analytical (executable) models which allow the engineers to perform simulations, descriptive models like OMG initiatives – UML or SysML [193], Object-Process-Methodology (OPM) [79], Functional Flow Block Diagram (FFBD) [133], IDEF0 [139], Universal System Language [105].

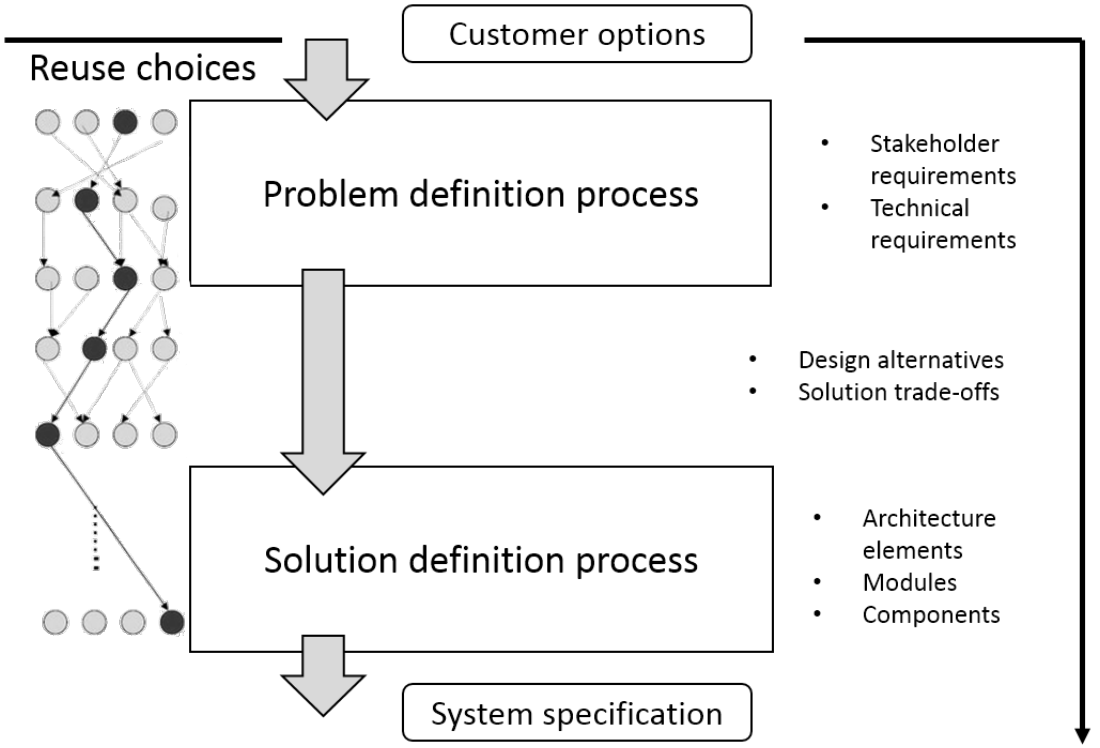


Figure 3.16: Reuse choices during system design

Some of the models have means of expressing forms of variability (e.g. inheritance mechanisms, abstraction, etc.), but none defines variability as an explicit characteristic of the system and documents it exhaustively and in a consistent manner. Most of the descriptions adhere to paradigms like abstractions, structural representations, integration of different viewpoints. Table 3.3 presents an overview of different forms of variability in product line models, and some SE models.

	Presence/ absence	*No. of instances*	*Alternatives*	*Replacement*	*Variable domain*	*Configuration guide*
FODA	+	-	+	-	-	+
Cardinality Based Features	+	+	+	-	-	+
Extended Feature Model	+	+	+	-	+	+
OVM	+	-	+	-	-	-
DOPLER	+	+	+	-	+	+
UML/SysML	-	+	Using constraints; disjoint inheritance	Using templates	Using templates	-
eFFBD -	-	-	-	Reference; shared function	-	-

Table 3.3: Forms of variability in some product line and generic models

Feature models are the most used among product lines models today, in both research and commercial tools. They are organized in a hierarchical structure, which represents product characteristics in relation to their form of variability. *Presence/absence* and *alternative* characteristics are among the core concepts of the FODA [123] models. In addition to variability, feature models also capture commonality through mandatory features, which are product characteristics appearing in all configurations. The hierarchical structure presents an important advantage for the configuration of products for guiding: the readability easy parsing of features. However, top-down selection of feature is not compulsory, and this can easily lead to invalid configurations created by the user. Cardinality based-feature models (CBFM) [66] and extended feature models can express in addition the number of instances of product characteristics and associated parameters.
Another representation is the orthogonal variability model (OVM) [163]. Like FODA representations, OVMs capture presence/absence or choice alternatives, but leave out common product line elements [65]. Unlike features, elements are not structured in hierarchies, which can make them more difficult to read in the case of large models. Configuration guidance, introduced in Table 3.3, refers to a form of assistance for the user, for the derivation of products from a product family. The "guidance" can be

addressed through tool automation and in the variability model. OVMs provide little aid for the user in respect to parsing and selecting elements.
As mentioned before, some system representations already include forms of variability, which leads to the following issues when using product line and system models together:

- information redundancy, when the same concept is found in both models (e.g. multiple instances in variability model vs. system model)
- consistency between the variability and system models, when the same concepts can be used differently in the two models
- modeling methodology clarity (and model readability), when there are multiple ways of representing the same concepts, in different places

For instance, UML based models support variability forms such as replacement, alternatives or cardinality. In UML/SysML, a *templateable element* is an element that can optionally be defined as a template or bound to other templates [207]. The template allows templateable UML elements (classifiers, collaborations, packages, operations) to be parametrized. This is a possible implementation for the replacement of model elements or of numerical parameters.
eFFBD diagrams have been used for a long time in SE. They too address reuse, by using function references and shared functions.
Orthogonal variability representations have the advantage of covering variability across a variety of engineering artifacts regardless of their representation – textual, graphical, model based. On the one hand, none of the variability models can be used as a standalone representation for systems containing variable assets. On the other hand, in the context of MBSE the issues mentioned previously remain unsolved.
From a representation point of view, the solution proposed by this project is to extend system models to include the different forms of variability. The aforementioned issues can thus be solved on the meta-model level, by avoiding redundancy and aligning the different concepts.
In SE, systems are often designed to unique mission and customer specifications. However, approaches to introduce variability in SysML models have been proposed before. Trujillo et al. [203] rely on feature based representation and mapping to variation points defined within the SysML models though stereotypes. While, this approach enhances SysML and system architecture expression support for variability, in our context, it does not address some aspects of variability orthogonal to system models, such as traceability, complex constraint modeling, integration with legacy enterprise information systems, shared features among families of systems. The SYSMOD [24] approach introduces *variations* and *variants* as stereotyped packages. The approach defines a "variation contains a set of variants that have a common discriminator" [17]. A variant groups a set of system elements, but a system element belongs to a single variant. The variability is thus almost completely separated from the system elements. In practice and according to our examples, this may lead to many one-to-one relationships between variants and system elements.

Our approach chooses a different solution to system variability: the forms of variability are represented where they occur. At the same time, variants refer to separate modeling items used to structure the marketing offer, technical solutions, or supplier alternatives in relation to the system architecture.
In the SYSMOD approach [24] configurations are modeled explicitly, by the user. In respect to the needs of our project, where the number of possible configurations is very large, this can make the derivation of products cumbersome and some automation, guidance or methodological support are necessary.
Most important, this project aims to address variability management at a system level, according to the current organization of systems at Renault, whereas many of the existing approaches in automotive deal with variability in software engineering (e.g. variability in software architectures [135], embedded systems [200]).

4

CO-OVM: variability in Model Based Systems Engineering

Still at an early stage, automotive MBSE needs to be adapted to the industry specific needs, in particular by implementing appropriate means of representing and operating with variability. We rely on existing modeling techniques as an opportunity to provide a description of variability adapted to a systems engineering model. However we also need to take into account requirements related to backwards compatibility with current practices, given the industry rich experience with mass customization. We propose to adopt the product line paradigm in MBSE by extending the orthogonal variability model [163], and adapting it to our specific needs, which brings us to an expression closer to a description of constraints (Co-OVM), related to both orthogonal variability and to system SysML system models. The approach is introduced through a discussion on the different aspects that need to be covered to express variability in systems engineering, guided by the observations made from our case study, and in relation to a list of requirements for variability management.

Product diversity impacts all organization levels. However, one of the areas that could benefit and improve in respect to the management of variability in the automotive sector, is MBSE.

Our purpose is to bridge the gap between vehicle features and component specifications, by introducing variability management on an intermediate level, and enable engineers to design systems for reuse, but also reuse specifications, requirements, documented system architectures.

Many variability modeling techniques exist [123][163] [199] [177], but the simple implementation of such an approach in a large organization would have some shortcomings and challenges, as pointed out by Filho et al. [93], and would have to be adapted to the specific context of the organization. We aim to draw on the rich experience provided

by software product line engineering and either adopt or adapt modeling techniques for both MBSE context and existing product variety practices at Renault.

4.1 Some motivating examples

The problem we are facing can be analyzed from multiple perspectives: from an information systems point of view - structure, storage, accessibility of data, compatibility with legacy systems; modeling techniques and extension of SE meta-models, SE process and specific practices. A research protocol which enables to gradually refine our research problem in respect to each of these aspects, was adopted, as explained in Section 1.3 (Chapter 1).
However, in order to provide some insight on the context and modeling issues, we present some simple variability modeling examples.
Issue 1: Need for flexibility and detail. The Constraint Satisfaction Problem (CSP) formalism is suited for a great variety of problems. For instance, one of the applications is product configuration and the representation of the catalog or range of products at Renault [43]. This representation provides an efficient mean of interacting with customers through online configurators in order to create custom vehicle configurations.
For example, let's consider the following Boolean variables:

1. *Door opening system*: SOPA, SOPB, SOPC (three types of door opening systems);

2. *Window lifts*: LVAVMA (front manual window lifts), LVAVEL (electric front lifts), LVAVAP (front anti-pinch lifts), LVAVIP (electric impulse lifts), LVCIPE (front electric impulse for driver side and electric for passenger side);

and the constraint:

$$SOPC \Rightarrow (LVAVEL \oplus LVAVIP \oplus LVCIPE) \tag{4.1}$$

Constraint 4.1 imposes that variant *SOPC* of the vehicle *door opening system* requires the use of one of the three variants of window lifts - electric front lifts, electric impulse front lifts, or front electric impulse for driver side and electric for passenger side window lifts.
Because in the case of Renault, the CSP formalism implemented through the organization's Documentary Language captures characteristics of already developed products, it is only possible to take into consideration some of these constraints once the product is fully developed, and the appropriate variables and constraints have been added to the lexicon (variables in the Documentary Language expressing vehicle characteristics). However, some of these constraints, as in the case of constraint 4.1, are technical in nature and are discovered only during system design. Furthermore, detailed requirements and design variability is not captured through the Documentary Language. In order to efficiently reuse existing desgins, systems engineers need to apply configuration techniques for the reuse of requirements and specifications on any level of detail.

Renault's product range is very broad [43], with up to 10^{21} possible configurations for the Traffic van, for example. On the one hand, further increasing the number of variables, to capture more detailed variability for each vehicle system, would only create even more complex problems for online configurators. On the other hand variability expression in MBSE needs to be compatible with legacy systems, although variability models could be local for each family of systems (vehicle subsystems).
Issue 2: Representation of "complex" constraints. One of the modeling techniques for variability, the orthogonal variability model (OVM), was introduced by Pohl et al. [163] to manage variability in the context of SPL. The formalism, presented in figure 4.1, enables the representation of dependencies such as "requires" and "excludes" between variants and variation points. To express constraint 4.1 using OVM dependencies, we need to introduce an auxiliary variation point - *AuxiliaryVP* - as shown in the diagram[1] from figure 4.1.

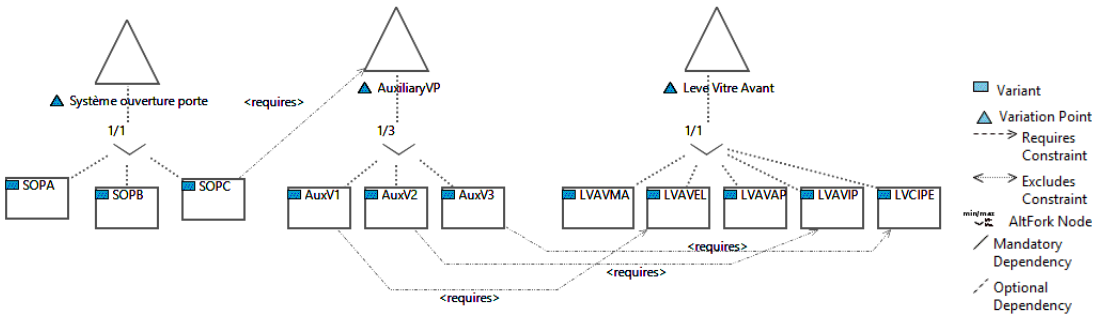


Figure 4.1: Using an auxiliary variation point to express constraints using OVM notation

Issue 3: Variability in system architectures. Feature models have become the most widely accepted modeling technique for variability management in the field of SPL [84]. They have the advantage of being easy to understand and independent of specific domains or notations. Figures 4.2 and 4.3 present two possible ways for modeling the functional decomposition for the electric parking brake system (EPB): (a) using the feature model hierarchy to represent common and optional functions[2] and (b) applying the stereotype << *variableElement* >> [199] to distinguish common and optional functions in SysML(UML).

[1]Diagram realized using the VARMOD Prime Tool (http://www.sse.uni-due.de/en/projects/varmod-prime)
[2]Diagram realized using FeatureIDE http://wwwiti.cs.uni-magdeburg.de/iti_db/research/featureide/

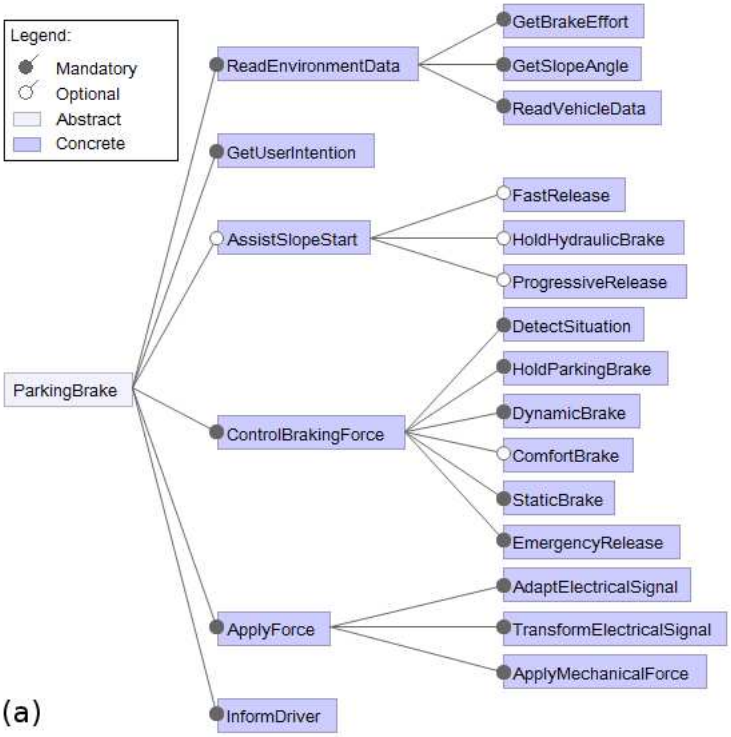


Figure 4.2: Electric Parking Brake system functional decomposition using the Feature Model notation

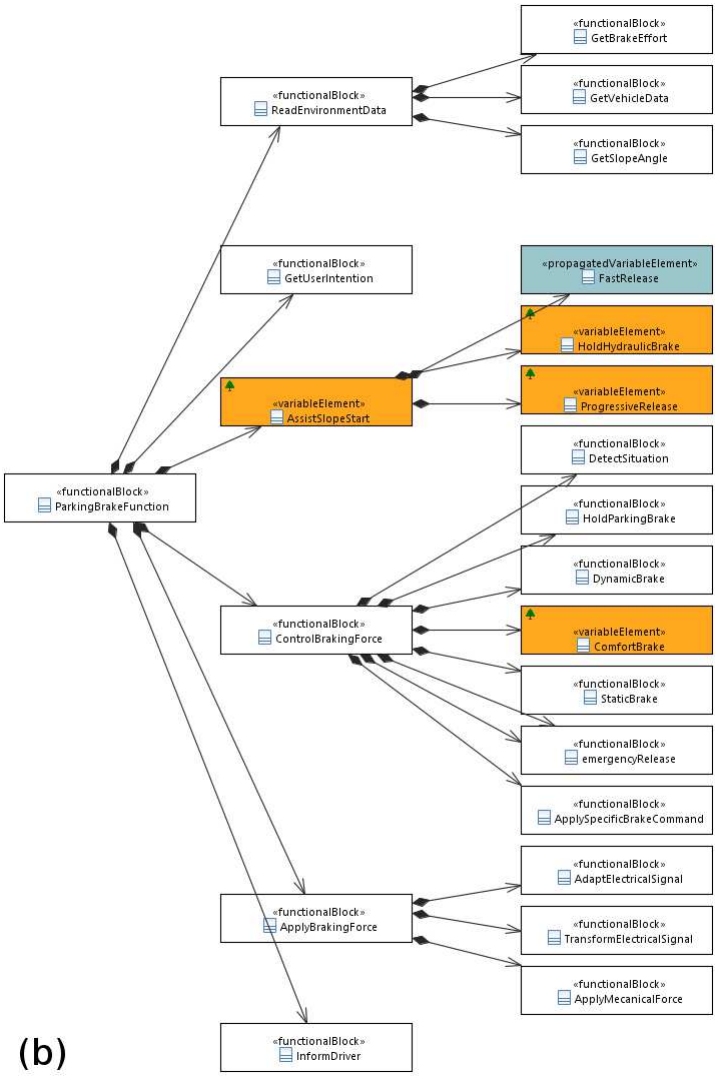


Figure 4.3: Electric Parking Brake system functional decomposition using stereotyped "optional" SysML block elements

In the case of system architectures, hierarchies are already introduced through decomposition, such as the functional or physical structures, which means two scenarios can be imagined to use feature models. In the first case, a partial feature structure may be deduced from variability in the system architecture along with the constraints imposed by the semantics of the SysML system model. However, characteristics defined in the Documentary Language are not structured in a hierarchical manner. In the second case, the systems engineer would create hierarchies for both variability and architectures, from different viewpoints, but this would result in redundant information and increase the model complexity.
We propose to adopt an extended orthogonal variability model Co-OVM (Constraint Oriented Orthogonal Variability Model) to cover specific needs for SE at Renault. From a practical point of view, CO-OVM is implemented as a UML profile in the SysML modeler Papyrus as an extension to Tessier's Sequoia plug-in [199]. Our purpose is to:

- Provide explicit representation of variability during system conception for reuse of requirements and model elements across families of systems.
- Support early documentation of configuration information for vehicle components in respect to the Documentary Language.
- Facilitate the discovery and modeling of technical constraints for introducing them later in the Documentary Language.

This chapter is structured as follows. Following the introduction, we present in Section 4.2 background information and a synthesis of existing approaches. In section 4.3 we develop the subject around the concepts related to variability, often with references to examples from the automotive industry. Section 4.4 explains the CO-OVM model by refering to groups of concepts form the complete metamodel, while the concrete syntax, based on UML Use Cases is described in Section 4.5. Section 4.6 includes a few examples based on the MBSE Renault variability management tool, and introduces for the first time the Electric Parking Brake system. Finally, Section 4.8 concludes this chapter.

4.2 Background and related work

The growing interest in variability modeling in the past years has led to a variety of meta-models and means of representation of variability, and many different approaches for the analysis, evaluation and classification of these models. This phenomenon of diversification of models could be interpreted as a real need to manage variability in multiple domains and types of applications. Until the recent approach of aligning these techniques in a single common language definition [192], variability models have spawn numerous dialects through extension of the base variability model or through application in general purpose modeling languages (e.g. extension of UML).
Products in the automotive industry are often driven by business cases, following customer trends and approaching competition through product variety. In this context,

reuse and capitalization in SE can reduce development time significantly, if proper tools and techniques are provided and integrated into the development process. We chose to draw on existing product line engineering modeling techniques to extend our MBSE framework to variability management, as the first challenge towards improved reuse support in systems engineering.
Although there is no single common vocabulary shared among variability models, even among the different analysis and classifications, a convergence among models appears both in the work of Czarnecki et al. [65] (variability present in feature and decision models) and Deelstra et al. [183] (choices as basic concept for variability).
By taking into account context information of each approach it is easy to point out that the variability models are of different nature and applied in different contexts, although they often fall under the generic name of "variability models". We draw on these works and propose the following classification: (a) conceptual models distinct of variability models, (b) embedding variability in existing modeling languages or artifacts and (c) enabling realization techniques [183]. A summary classification of these techniques is presented in Table 4.1. This does not cover aspect oriented variability modeling, as we considered this to be inadequate for SE.

VM technique class	Standalone variability models	Applied/ embedded variability	Enabling techniques
Feature Model based	FODA[123] , FeatuRSEB, FORE, FOPLE, GP, CBFM, Forfamel	UML Integration of features [164] ; RequiLine (requirement engineering) [138]	
Variation Point based	Orthogonal Variability Model (OVM)[163], Gomma and Weber, Quality Aware OVM [169], COVAMOF[183], VSL[46]	UML integration of OVM: Halmans and Pohl [104] ; von der Maβen and Lichter [138]	
Constraint Based		Sequoia (Tessier et al.)[199], Feature Modeling Constraint Language (FMCL) [132]	Salinesi et al.[177], Streitferdt et al. [187]

Table 4.1: Classification of variability modeling techniques

4.3 Identification of concepts for the representation of variability in Systems Engineering

System models alone are not able to completely ensure traceability from requirements to the implementation when variability is present. For example, variability traceability needs to be ensured in relation to the project or business context and system assets, answering to the questions: who specifies variability?, what is the rationale for increasing variety?, to which context does a particular expression of variability apply ? Based on our industrial experience, we can claim that variability is present not only on the client offer level, but also on the solution level through design alternatives and replaceable components.
The system model needs to be complemented with relevant information for capturing variability. As we already announced in the previous paragraph, several aspects are essential for the representation of variability in MBSE, which shall be discussed in this subsection.
Types of variability. Probably the central concept, present in all formalisms in one form or another is the concept of *choice*. From a SE point of view, the act of performing *choices* can be regarded as decision making, involving specific engineering domain knowledge, with impact on the final derived system model. We can thus distinguish three types of variabilies:

- *Diversity:* variability contributing to the stakeholder requirements (including the final customer offer), with alternatives in terms of system functional and non-functional properties, or optional customer features. At Renault, a part of the "Diversity" is defined in the Documentary Language.

- *Design decision:* Refining stakeholder requirements requires taking decisions in respect to the technical solution. The design alternatives start with high level concepts and are refined to a detailed system design [100, 218]. In single SE, only one technical solution is chosen, but in a family of systems this represents an opportunity for reuse. Multiple design alternatives can cover a variety of contexts (e.g. regulation, country, climate), but unimplemented alternatives can also be revisited at a later time, when technology availability or cost may change.

- *Component choice:* Typically there are multiple supplier for the same, or for similar vehicle components. Considering suppliers in the system analysis allows identifying the best alternatives early, as well as discover product line dependencies related to geographic availability of physical components.

Decisions are mentioned in the context of product lines, but their meaning is often related to product configuration. For instance, approaches such as DOPLER [166] refer to choices as decisions, and the "decision oriented" model represents a link between the domain of assets and the derived system. "Features" capture high level variability in requirements, exposed to the customer (or stakeholders) and enables system configuration from a higher perspective, without focusing on decisional impact.

In respect to the design space, a decision oriented derivation would be directed from the problem to the solution space, whereas a configuration oriented derivation would not take into account any particular workflow. We believe these approaches are useful at different phases in the lifecycle of the system. We can thus identify: *design* decisions with impact on the system model, characteristics that contribute to product *diversity* and replaceable system parts (like *components* from different suppliers, COTS etc.) One may wish to simply configure the architecture of a system or go through the decisions that lead to different designs by evaluating alternatives in respect to a particular context. Integrating variability models in MBSE enables early documentation during the system design phase of configurations for manufacturing in plant. Variability models have thus the role of describing configuration information of all engineering artifacts from conception to industrialization.
Sources of variability. Each phase of the system design requires to analyze the impact of existing variants on the system description or the creation of new variants. Variability is described orthogonally on the organization level through the Documentary Language, enabling the description of vehicle features and vehicle ranges. However, because of the amount of information present, it becomes difficult to manage all sources of variability on a single level, focused on customer needs. Figure 4.4 presents the way in which variability is introduced on different "visibility" perimeters: the Documentary Language is orthogonal across vehicles, but more detailed descriptions are introduced for each family of systems (e.g. temperature control system, engine) under the form of *variants* and *variation points*. In consequence we refer to the two levels of variability description as: vehicle and system (vehicle subsystem) level variability.

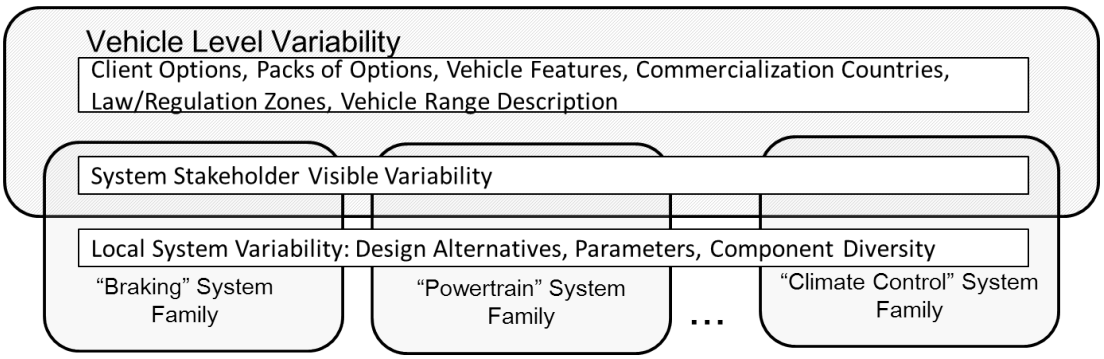


Figure 4.4: Variability visiblity perimeters : Documentary Language and local families of systems variability

Because existing vehicle level variants further impact the development each subsystem, it is important to trace their origin. For example variations defined on a vehicle level, for the gear box (automatic or manual), indirectly require the development of alternative behaviors for the electric parking brake (EPB) system. However, since not all vehicle models propose both automatic and manual gearboxes, it is important to

trace the origin of variability and the targeted configurations.
We consider the system levels of decomposition (system of systems, vehicle, vehicle system), abstraction level (behavior, physical) and specific (stakeholder) viewpoints as sufficiently fine grained information for tracing the source of variability. Furthermore, as suggested by Tessier [199], variability propagation rules can be applied in specific cases to reduce the work effort for binding variability to system models.
Legacy variability specification. Product variety is not a new concern in the in the automotive industry. As SE progressively takes into account variability, both the tools and existing knowledge for mass customization need to remain compatible and even provide a basis for new MBSE tools.
For instance, the Documentary Language is used at Renaut to describe variability on the vehicle level, as explained in figure 4.4. Astesana et al. [43] describe the way variability (or "diversity") is expressed and exploited at Renault for marketing purposes: a Boolean constraint satisfaction problem (CSP). In the case of the Documentary Language, there is no distinction between "mandatory" or "optional" variables and all defined variables need to be assigned a value. From a data representation point of view, variables are grouped by the corresponding variation subject (e.g. type of fuel : diesel, gasoline etc.), which brings us closer to the modeling techniques known in product line engineering. Other semantic properties (like commercial packs of options) are specific to marketing and commercial activities. They are neither specific to SE or product line engineering, nor crucial for compatibility.
Constraints, the core of product line engineering. Existing vehicle features (at Renault) are represented as Boolean expressions. At the core of product line engineering practices is constraints programming [176][143][49][47], which enables both the representation and the analysis of product line models. This is indeed the aspect addressed by all variability models, which is essential for the derivation of a single system model. Furthermore, constraints are introduced through the commercial offer and stakeholder requirements, but also due to technical, architectural dependencies. Technical constraints are the result of dependencies to variable resources from within the system or from external enabling systems. Not only do these technical constraints need to be taken into account during derivation, but they may also need to be rendered visible for the commercial offer. Enabling the system engineer to represent architectural constraints in relation to variability mechanisms becomes a necessity for dealing with the complexity introduced by variability.
Variability in system architecture models. Tseng and Jiao [205] define "variety generation" as a "way in which the distinctiveness of product features can be created". They [205] also define the basic methods in which variety can be generated: attaching (*presence*), swapping (*replacement*), and scaling (*parametrization*). More complex variety generation techniques can be created based on the repeated use of these basic methods. The purpose of these techniques is to introduce variety through components, by proposing different configuration items which satisfy system requirements, for the design of the system.
These concepts are also present in the Common Variability Language (CVL) [191], de-

scribed with slightly different vocabulary: existence, value assignment, substitution.
Multiple viewpoints. Variability can be represented as embedded in the system description or a separate orthogonal model. In the second case it is useful to take into account existing system viewpoints to navigate variability, and relate to system analysis. Since variability impacts all organization activities, the definition of additional viewpoints on variability can broaden the information taken into account system during development (e.g. taking into account supported variability for suppliers may influence decisions for component selection).
In our case, we also rely on the definition of viewpoints to enable partial derivations of the system model [3]. Partial derivation enables engineers to gradually reduce the solution space and generate specific deliverables for each project milestone.
System model variability visibility. System engineers use a range of descriptions in order to refine the description of the system. Decomposition is used for functional and physical specifications, which give an overview on the solution refinement. While variability has many benefits when considered as a separate model, from a system analysis point of view, it is important to understand and assess variability in the system architectures. By representing variability in hierarchies, we may reach a "kind of interpreted FODA" [56] description, which is useful for having an overview on system variability. While being able to visualize optional elements in architectures, it is equally important to understand which elements are impacted by variability (e.g. a function contains optional sub-functions; a system contains optional "subsystems"; a requirement is derived from an optional stakeholder requirement). By enabling the visualization of both *variable* and *impacted by variability* assets, one can navigate the model in a top-down or bottom-up manner to follow and trace variations in the system description.
Customer oriented variability. The customer is represented within the organization by the "product department", which manages, structures and proposes the commercial offer, by defining the vehicle *options*, *packs of options* and *levels of equipment*. However, there are multiple levels of decomposition, which can be considered: system of systems (vehicle fleet), vehicle, system (actually a vehicle subsystem), components. From a SE point of view we consider the commercial definition as a part of the stakeholder requirements and choose not to have any additional representations for this type of variability.

4.4 Modeling technique Co-OVM

The modeling tool which implements the system architecture framework is based on the Papyrus SysML modeler, while support for variability modeling relies on the Sequoia [199] plug-in. Sequoia enables the representation of constraints in relation with variability in UML/SysML models. As figure 4.5 suggests, both variability and system architecture are built on top of the Sequoia support for constraint modeling:

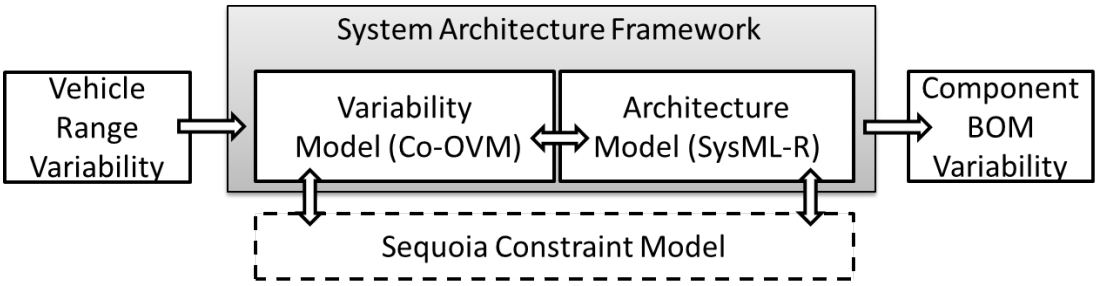


Figure 4.5: System Architecture with variability management modeling tools structure

- *The constraint oriented OVM* [1] *(Co-OVM) variability model* supports configuration activities and specification of variability in relation to the organization Documentary Language from vehicle to components. Here, Sequoia [199] constraints enable the representation of dependencies between variants, but also between variable system modeling artifacts.
- Architecture elements that are concerned by variability have an impact on other system elements through structural or functional dependencies. It is sometimes possible to infer which elements are optional based on the semantic relationship between elements, and existing optional elements. Some of the UML semantics for propagation of variability [196] are already supported by the Sequoia tool and we intend to add new inference rules specific to the Renault systems architecture model, but whenever automatic reasoning is not available, it is possible to introduce Sequoia constraints directly in the system architecture model.

The CO-OVM concepts are explained in the following paragraphs and the complete metamodel is presented in Appendix D. During the system analysis different types of constraints are applied on the family of systems artifacts, as suggested by the concepts in Figure 4.6. Users may wish to represent these on separate diagrams.

[1] Developed in the context of the MBSSE project, in collaboration with the CEA LIST, based on the UML modeler Papyrus - http://www.eclipse.org/papyrus

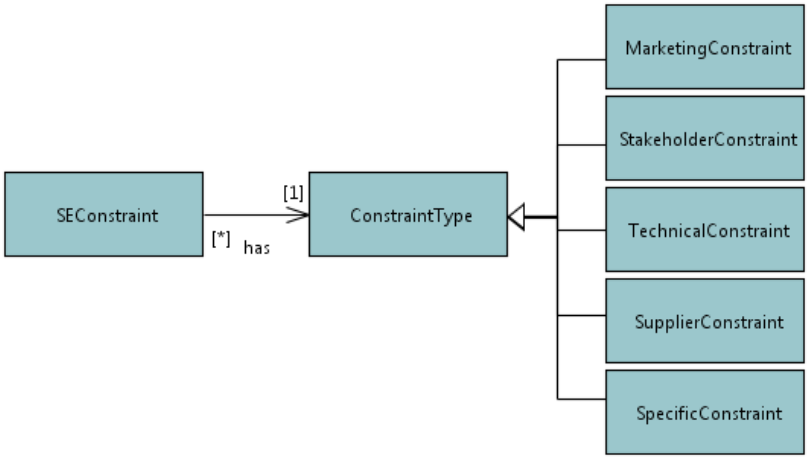


Figure 4.6: Different types of constraints

- *Marketing Constraints*: are defined by the product department in order to describe the commercial offer. These are imported into the work space at the beginning of the project from the Documentary Language (also called vehicle level variability, in figure 4.4).
- *Stakeholder Constraints*: refer to behavior of the system in interaction with its environment or to characteristics required by stakeholders.
- *Technical Constraints*: capture dependencies related to design alternatives. Sometimes these constraints have a global impact and need to be made visible in commercial offer. For example, the GPS navigation system requires a CD player, on certain vehicle models, which is a marketing constraint, that has its origin in the design of the system, where maps could be read from CD support.
- *Supplier Constraints*: the availability of certain components can depend on the country of commercialization, or differences in the characteristics of the supplied components may require other adjustments in the design of the system.
- *Vehicle project constraints* (specific) : describe constraints specific to each vehicle model for which the system shall be designed and deployed.

Choosing a variation point based model for the description of variability, means that the same formalism can be used for different types of variability (design decisions, system characteristics etc.). At the same time, the existing Documentary Language model which is in use in legacy systems, needs to be taken into account.
While the Documentary Language, covers many of the vehicle and context characteristics needed for vehicle configuration, a more detailed variability description needs to be refined during system design. The model implemented in the present system architecture framework is centered around the concepts of *variation point* and *variant*, as in the OVM model [163] (Figure 4.10). A variation point regroups several variants.

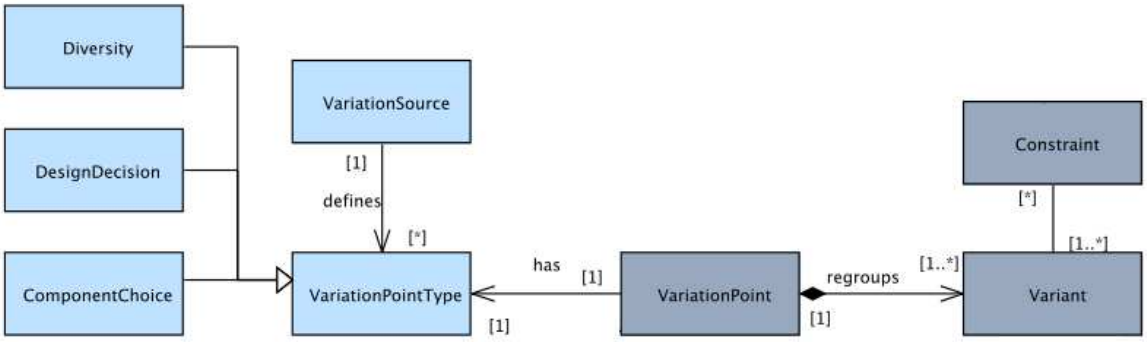


Figure 4.7: Core concepts based on OVM

- *Types of variability* distinguish between variation points that capture different types of information : diversity (stakeholder visible variability) , design (decisions), and components (replaceable COTS, different suppliers). By distinguishing among the different types of variability, we aim to prepare the extension of our method and tool with techniques to enable decision making and analysis for system design alternatives [80].

- *Variation source*, contributes to traceability by documenting what induced the variation initially, in the system analysis. For example variants imported at the beginning of the analysis are linked to the Documentary Language. It is also possible to specify the details of the cause by pointing to the concerned part of the model, or by documenting the rationale behind the existence of the variation. The term "traceability" has a broad use and possibly a wide range of semantics [36]: here the traditional mechanisms for traceability in MBSE are used and complemented in regard to variants. Because variability eventually impacts multiple levels of abstraction due to refinement relations in the system model, the purpose is to be able to trace the initial cause of that variability.

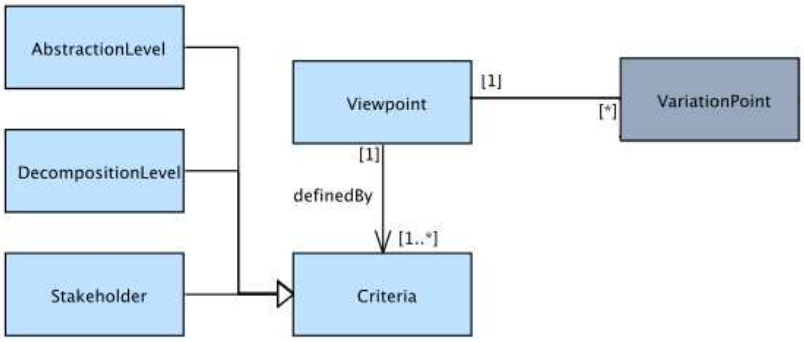


Figure 4.8: Viewpoints for variability

- *Variability viewpoints* allow for easier navigation of variability in complex system models and also play a methodological support role during configuration [3].A staged configuration is performed, that follows the different viewpoints allowing the user to gradually refine the system model. While the predefined list of viewpoints can be customized, we consider the following: Documentary Language (vehicle features), stakeholder requirements, system environment, operational scope, system technical context, architecture alternatives, functional variability, allocation alternatives, physical variability. The viewpoints can be categorized according to the following: (i) abstraction level - system description is partitioning is relative to the degree of refinement of the solution; (ii) decomposition level - system description is partitioning by decomposing the given problem is subproblems; (iii) stakeholder - information is described in respect to a particular domain or concern (e.g. regulation, safety).

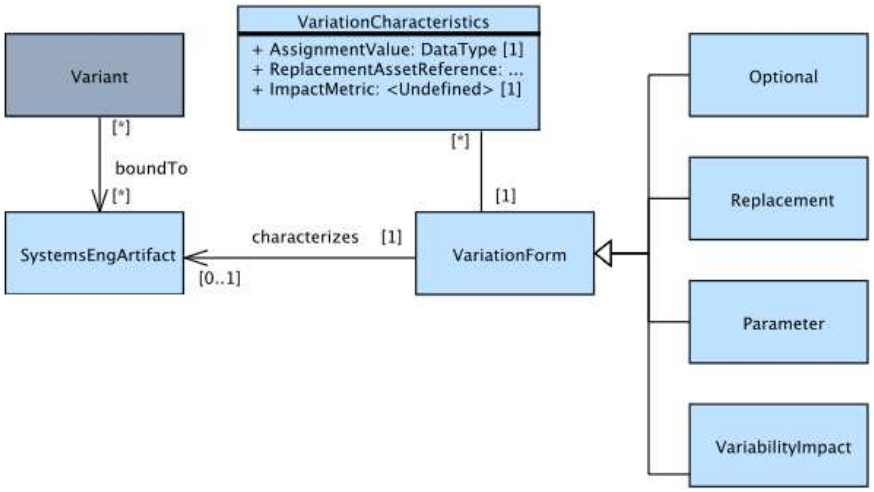


Figure 4.9: System architecture variability

- *Variation forms* characterize system elements to describe in which way they are variable: presence/absence, replacement, parameter, variability impact. The last represents a way to visually mark elements that are impacted in some way by other variable elements (e.g. a physical components that is decomposed in subcomponents, among which some elements are optional). In future work, we intend to refine the description of elements impacted by variability by calculating an "impact metric" (e.g. for decomposition, the ratio of optional to common parts within a decomposable element), which is now added for reference only, and not implemented in the current tool version.

- *Variability impact* is defined as a form of variability, where the concerned system item is related to at least another item characterized by one of the other three forms of variability: presence/absence (optional), replaceable, parameter. The

purpose is to render the variability more visible for system decompositions (e.g. an item containing at least one optional part) or requirements with parameters. For example, in the later case the requirement is only *impacted* by variability – thus visible to the user – while the numerical element is defined as *parameter*, with different possible values based on the selected bound variants.

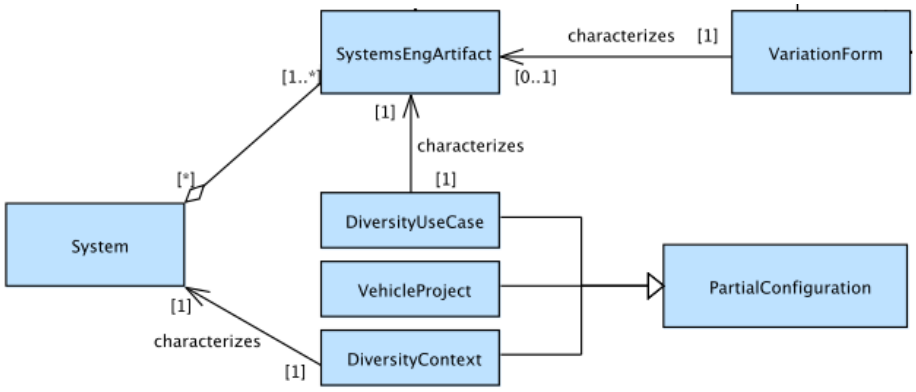


Figure 4.10: Concepts defined as partial configurations of the system family

- *A partial configuration* represents the variability and specific constraints reduced to a particular set of concerns, where the state of some of the variants is completely defined (e.g. selected, deselected). When a variant state is (intentionally) set as "undefined", it means that the next activity or subsystem needs to cover all possible cases, or further refine solution alternatives for each case. The concept is important from a process perspective, because variability impacts most of the activities of an organization. However, due to the wide range of artifacts impacted, a complete configuration cannot be specified in a single step.

- *The studied diversity* represents a partial configuration of a system that refers only to: the system environment, system technical context, and final customer requirements (commercial offer). The studied diversity is specified in the operational analysis phase of the system development. Alternatively (at Renault), it is received as input in a specific format (configuration tables).It represents the variability that needs to be studied and taken into account for the system analysis.

- *Vehicle Project* describes the constraints specific to the vehicle model which will integrate the system. A system (with variability) is designed for multiple vehicle models.

- The *Diversity Use Case* can be associated to any model element, but it is typically used for physical components, and inspired from the application of the Documentary Language [43] on physical components for manufacturing purposes. It consists of a logical constraint that restricts the set of possible system configurations, such that the system element (component) is always present.

Constraints allow to specify on the one hand the variant state - used in the derivation process, and dependencies on the other hand - in the variability and system model alike. The different types of constraints are introduced in figures 4.11 and 4.12.

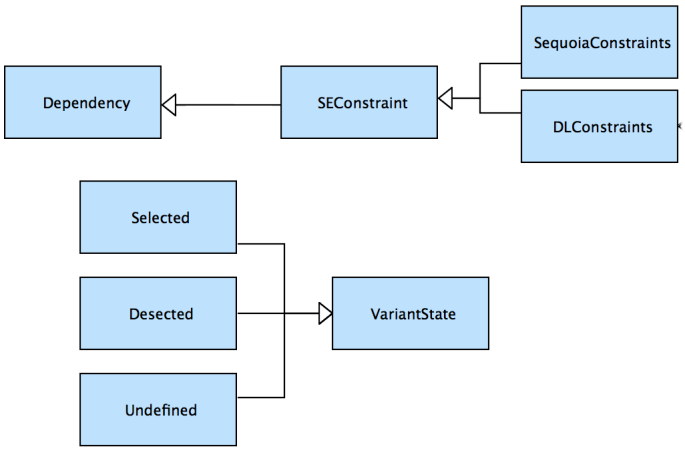


Figure 4.11: Dependencies and constraints for system family modeling and configuration

- *Documentary Language Constraints.* Boolean Constraints are present both in the orthogonal variability model and the optional system elements. The case study and various examples revealed that the initial set of available constrains defined in Sequoia were not sufficient to capture – on the one hand technical dependencies in SE introduced, and on the other hand existing documentary language expressions representing commercial variability. Obviously, two approaches are available in this case: a textual language to represent product line constraints and dependencies, or an extension to the existing concepts for the representation of logical constraints. Each of these approaches has its benefits, in usability, expressiveness, easy learning etc. However, the latter approach is sufficient for our needs and integrates easily in a model based engineering framework. The representation of these dependencies in graphical product line languages is limited, as explained in 4.1, but textual constraint editors fill this gap in some product line modeling tools.

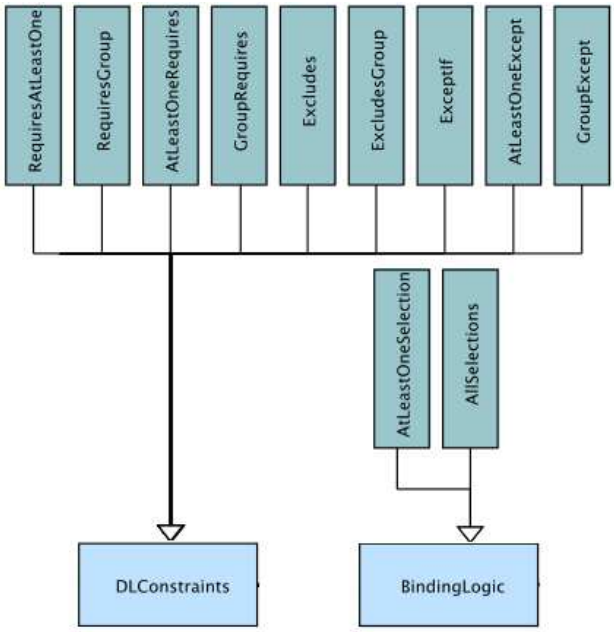


Figure 4.12: Detailed types of dependencies and constraints for system family modeling

- The *Binding Logic* enables customization of the relation between variants and SE artifacts. Since multiple variants can be associated to multiple SE artifacts, it is necessary to specify the logical condition for the presence or absence of these elements: *at least one selected variant* keeps the system element, or *all of the associated variants* need to be selected for the system artifact to be kept. By default the first behavior is activated, but the second is useful to factorize variants in respect to system items, for example when a single system component is selected based on multiple variable characteristics.

Once created, the different types of constraints are used to generate custom Sequoia constraints in normal conjunctive form (CNF). Code examples are included in Appendix F.
Figure 4.13 presents a simple theoretical example of an orthogonal variability model in relation to system model items.

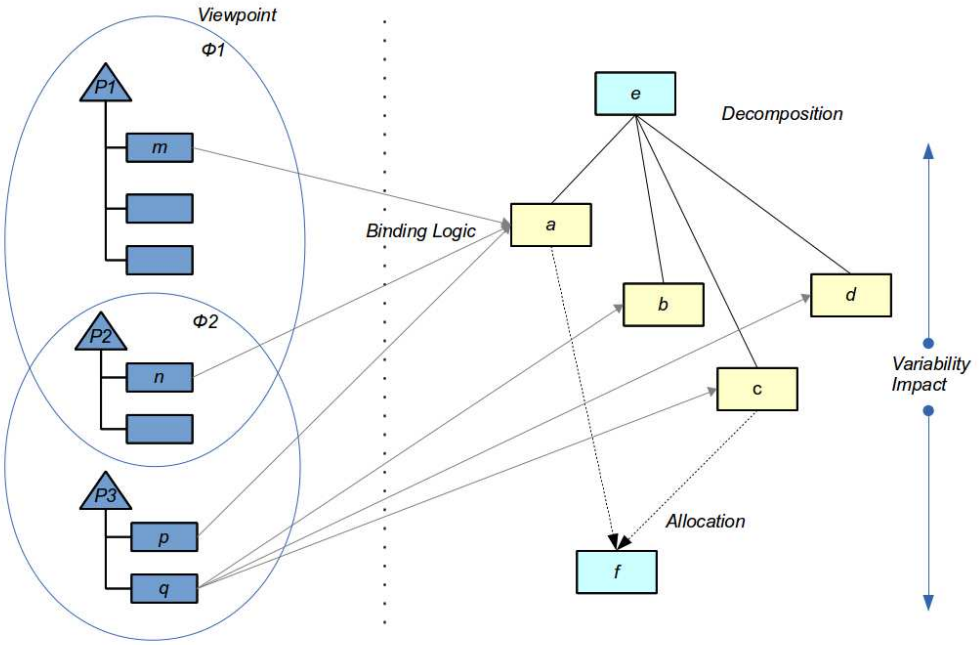


Figure 4.13: A simple theoretical example

Variants are visible according to different viewpoints to facilitate parsing the variability model when there is a large number of possible configurations, and during the derivation of product models according to the methodology described in the Chapter 4.7. Although the concrete syntax is different in the one in the developed tool, the example adheres for now to the concrete syntax of the OVM model. Variability is present in both variability and system models. Variants are bound to system items through *n-to-n* relationships., while system items are characterized by different forms of variability. Within the system model, variability can be propagated through allocation or decomposition relationships. In this context, *propagation* refers to the identification of new optional items and their variability constraints in respect to already defined optional items, according to the same principles described by Tessier et al. [199].
The proposed propagation rules for *allocation* are the following, based on the example from Figure 4.13:

- *Optionality:* System item f is *optional* if at least one of the items allocated to f (i.e. items a, c) is optional.
- *Presence:* System item f is optional and *present* if at least one of the items allocated to f (i.e. items a, c) is present in the current configuration.

- *Absence:* System item f is optional and *absent* if all of the items allocated to f (i.e. items a, c) are absent in the current configuration.

In this context, propagation is only applied in a top-down way, based on the presumption that the user may configure the system only from a physical point of view, with no impact to higher levels of abstraction (e.g. functional, requirements). In consequence, on the constraints programming level, these rules can be easily represented for the example in Figure 4.13 by the following constraints:

$$(a \vee c) \Rightarrow f$$

$$\neg(a \vee c) \Rightarrow \neg f$$

In a similar way, rules are defined for *decomposition*, based on the example from Figure 4.13:

- *Optionality:* System items a, b, c and d are *optional* if the parent element e is optional.
- *Absence:* System items a, b, c and d are *absent* if the parent element e is absent.

No information can be inferred about the presence of each of the items a, b, c, which is an engineering decision related to a more detailed physical level of the system. Thus, once item's e presence is determined, the selection of these items depends only on the variants selected by the user. On the constraints programming level, these rules can be easily represented for the example in Figure 4.13 by the following constraint:

$$\neg e \Rightarrow \neg(a \vee b \vee c \vee d)$$

Again, the reasoning is applied only in a top-down way, allowing partial configurations during the process of derivation, as explained in Chapter 5. If the propagation rules were to be applied in the opposite way too, the user could not select item e for a partial configuration, without selecting at least one of the items a, b, c, or d. However, this would not allow some partial configurations, still containing unresolved detailed variability.
In order to simplify the representation of variability, one variant is bound to one or multiple elements from the system model. When the variant is selected, the bound elements are kept (or replaced with a particular value or element). In some cases a system element may be selected based on multiple properties that characterize the element. One example is the *water boiler*, presented in the last part of this document 6: different water tanks can be selected based on their dimensions and mounting type. Allowing multiple variants to be bound to a single system element, simplifies the product line model. In this second type of binding the user needs to specify the behavior during configuration:

- *At least one selection:* This is the default behavior. The bound elements are present in the final configuration if at least one of the bound variants is selected.

This type of binding is useful in relation to the system context: e.g. element *ESP* is present if either variant *EU* or variant *US* is selected, where *US* and *EU* refer to the zone of commercialization (system context during usage).

- *All selections:* The user needs change the binding logic to activate this behavior. The system element is present in the final configuration if all bound variants are selected. This type of binding is useful when variants represent attributes of a system element (e.g. size, length, color for a physical component).

Contrary to the case of many product line models, there is no clear distinction between "mandatory" or "optional". It is the definition of the vehicle range, which establishes which features are optional, mandatory, or "by default" (if the customer expresses no preference) for each context or country. For example, the ABS (anti-lock braking) may be mandatory if the configuration of a vehicle is done for the EU, or may be optional for countries with less restrictive regulation.
From an engineering point of view, it is the variable character of the marketing vehicle features that needs to be taken into account for the design of the system to accommodate this variability, and thus the optional/mandatory character of variability is represented outside the scope of the system framework, as a constraint.

4.5 Concrete variability modeling syntax & implementation

Variability can be managed in two different Eclipse views.

- Model explorer view: allows performing most of the available operations on variability, like editing constraints and variants, validating the product line model in respect to constraints, and launching product derivation;
- RVU diagrams (based on UML use case diagrams and Sequoia constraints): allow visualizing and editing variability.

The concrete syntax includes stereotyped UML Use Cases for variants, as shown in Figure 4.14. Variants are regrouped by variation points, represented as stereotyped UML Packages on the diagrams. The palette proposes the creation of different types of constraints, as well as variants and variation points. A difference from Pohl's OVM [163], is the representation of constraints, which capture only dependencies between variants and include the following:

- Sequoia constraints [199][197]: *Equivalence, Alternative, Alternative-Empty, AtLeastOne, AllPossibilities, Implication*
- additional constraints: *RequiresAtLeastOne, RequiresGroup, AtLeastOneRequires, GroupRequires, Exclusion, ExcludesGroup, GroupExcept, ExceptGroup, Except*

The additional constraints are actually templates for the generation of more complex constraints, which usually require more than two operands. Some of these are often used in the Renault Documentary Language. For example *Except* is used to express the presence of a component in vehicle configurations (e.g. component use case is "(XA or XB) except (ABS and TO)", meaning that the component is present on all vehicles based on platforms XA or XB, except those which include both an ABS[1] system and on opening roof[2]).

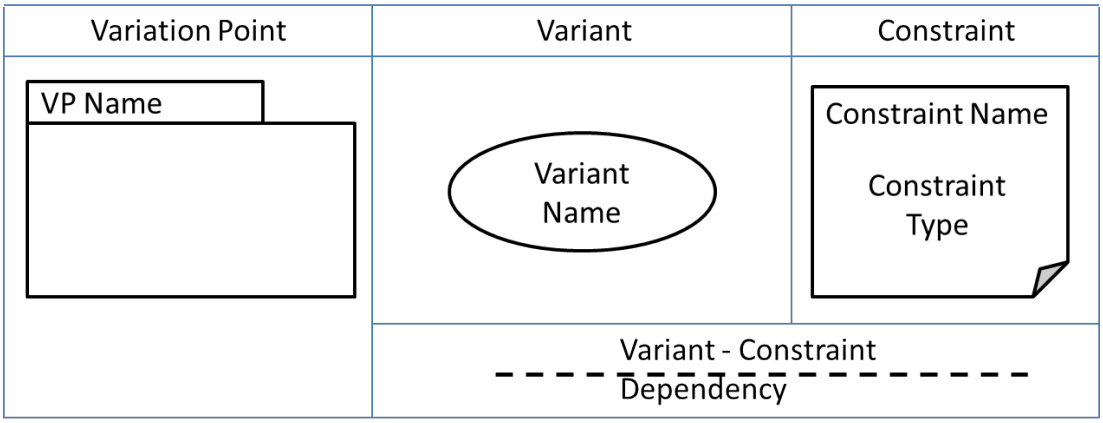


Figure 4.14: Co-OVM notation, based on UML Use Case Diagrams

The properties that specify the type of variation point, source, description, or rationale for variants are edited using the properties view of the editor Papyrus.
The implementation is based on the Papyrus[3] modeler and it extends the plug-in for product lines, Sequoia [199]. Sequoia already supports expression of variability by using constraints between variable SysML/UML model items. However we need to adapt it's functionalities in respect to the specific MBSE context at Renault.
Variability concepts were defined as a UML profile, that extends current notions present in SysML and in the SysML-R profiles [58, 194]. It is worth pointing out, that variants are implemented as stereotypes on use-cases, which was also proposed by Halmans et al. [104] and Maßen et al. [138]. This, along with the existing Sequoia profiles for modeling constraints, allowed us to adapt the use-case diagrams for modeling constraints between variants. We also take advantage of the existing Sequoia tool architecture based on the representation of constraints in a model based development environment to capture: constraints issued by the definition of the product line through the representation of variability dependencies in the variability model, but also constraints issued from the impact of variability on the system's architecture.

[1] ABS : antilock braking systems
[2] TO: toit ouvrant
[3] http://www.eclipse.org/papyrus/

4.6 Electric Parking Brake Example

We have used two simple systems to experiment our models: the electric parking brake (EPB) and the automatic lighting system. They were exploratory case studies [82] simple enough to provide us with valuable information by instantiation of the meta-model, but also to allow the integration of different components of the modeling software, regarding systems, software, safety etc.
The EPB model we considered, contains variations on all abstraction levels, and allowed us to exercise several scenarios [2] in respect to the sequence of choices that lead to different configurations (or designs) :

(a) reuse of alternative designs, for example based on a single DC motor and a puller cable, or caliper mounted electrical actuators;

(b) reuse of problem definition and introducing new solution alternatives, for example based on reduced power design, and relying on complementing braking force from the hydraulic brakes;

(c) separating or combining optional system functions through alternative allocations, for example providing of the "hill start assistant" function through the hydraulic or electric parking brakes.

The electric parking brake (EPB) application is a variation of the classical, purely mechanical, parking brake which ensures vehicle immobilization when the driver brings the vehicle to a full stop and leaves the vehicle. We have chosen this example for its simplicity in respect to other automotive systems and because it contains many variations from the requirements to solution design and component implementation. Table 4.2 presents the main variations which were identified in the conception of this system, relative to the viewpoints specific to the Renault SE process [58, 194].

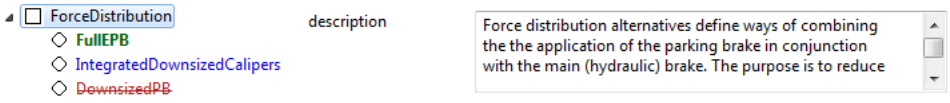
System ViewPoint	Occurrence place	Variation point	V_Id	Variants
Operational Analysis	High Level Customer Requirements	Brake Lock	BL1	Automatic
			BL2	Manual
		Brake Release	BR1	Automatic
			BR2	Manual
		Hill Start Assistance	HSA	Present
	System Context	GearBox	GB1	Automatic
			GB2	Manual
		Clutch Pedal	CP	Present
		Vehicle Trailer	VT	Present
	System Environement	Regulation	Reg1	EU
			Reg2	US
Functional Viewpoint	Functional Decomposition	Braking Strategy	BS1	Dynamic
			BS2	Comfort
			BS3	Static
		Effort Monitor On Engine Stop	EMon1	Permanent
			Emon2	Temporary
Engineering Decisions	Technical Solution	Main Design	PC	Puller Cable
		Alternative	CA	Caliper Actuators
Physical Viewpoint	Physical Decomposition	Electrical Action	DCM	DC motor
			ACT	Actuators

Table 4.2: Electric parking brake system main variations

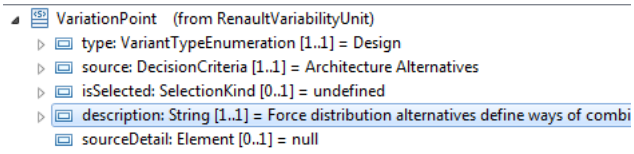
For the sake of simplicity we do not present all variations relative to the EPB system, but only the most representative ones. Other variations are present, such as specific behavior relative to the vehicle model or range, depending on the configuration of CAN bus messages, vehicle weight, etc. As an observation, we point out that all variations occurring in the *operational viewpoint* propagate towards the behavior and physical implementation of the system. At the same time new variations may appear due to alternative design solutions during the problem solving process.

Figure 4.15: The Electric Parking Brake system main design alternatives

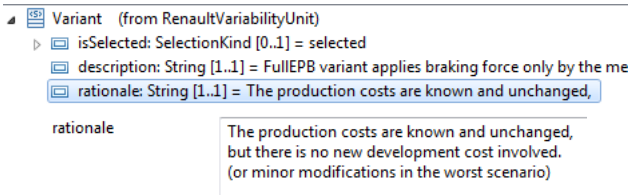
These variants induce alternative representations of the SysML system model. We used the modeler Papyrus with the Sequoia plug-in [196] in order to model the system and constraints related to variability (e.g. GB1 or GB2, BS2 requires CA).
Figures 4.16 and 4.17 introduce an example of a variation point and a product line constraint from the Electric Parking Brake system model. More detailed examples can be found in Appendix A.1.2.



(a) Variation point *ForceDistribution*, associated variants and description



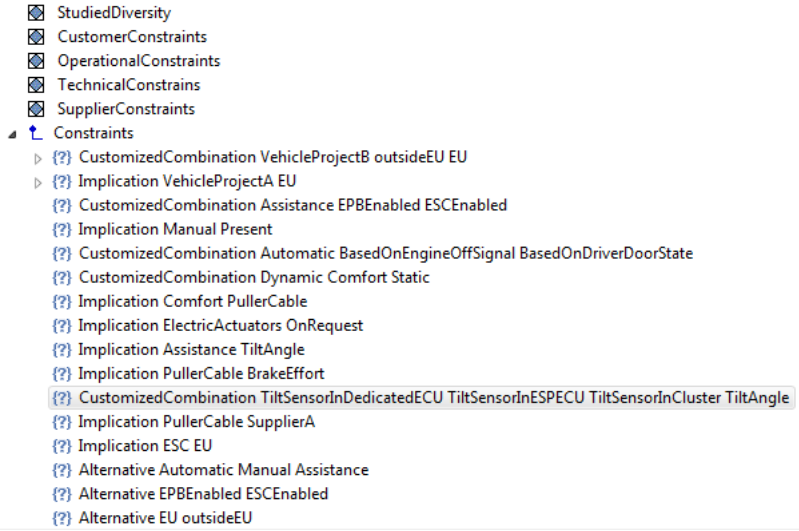
(b) Tagged values for *ForceDistribution* variation point



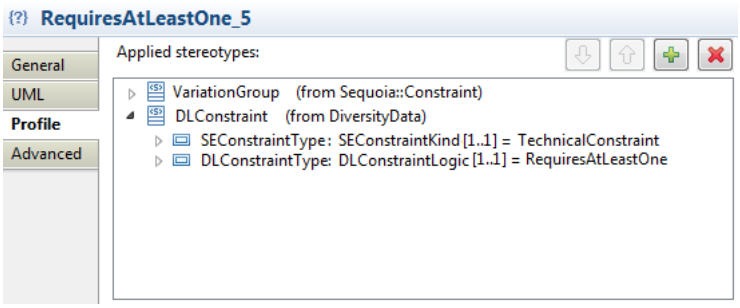
(c) Tagged values for the *FullEPB* variant

Figure 4.16: Example of variation point from the EPB system model

The variation point *ForceDistribution* captures system architecture alternatives (e.g. type is *Design*), which aim at redistributing the braking force between the main hydraulic brakes and the electrical parking brake. The purpose is to reduce manufacturing costs by reducing size and power of the parking brake system. It regroups three variants: *FullEPB*, *IntegratedDowsizedCalipers*, and *DownsizedPB*. The rationale for each variant explains the advantages and downsides for the available alternatives. As shown in the example (4.16c) variant *FullEPB* is currently selected.



(a) List of constraint diagrams and constraints from the EPB model



(b) Tagged values for the constraint *RequiresAtLeastOne_5*

Figure 4.17: Example of constraints view with selected constraint from the EPB model

The selected constraint in Figure 4.17a states that the variant *TiltAngle* requires the presence of at least one of the solution alternatives that specify where the tilt sensor is located. This is a technical constraint because it describes the dependency between the information required by the system to function properly and physical implementation alternatives. Because the constraint "RequiresAtLeastOne" was not defined in Sequoia, it is provided as an extension through another UML Profile (e.g. profile DiversityData, stereotype *DLConstraint*). However, the product line validation function is still achieved by using Sequoia *CustomizedCombination* and expressing the boolean logic directly in conjunctive normal form (according to SequoiaCNF language specification [197]).
The EPB system model contained 21 variation points with 46 related variants representing system requirements, design decisions and choices related to components. The model also contained 36 constraints related directly to variants, and 44 constraints issued from binding to the system model and from propagation of optional elements (represented and Sequoia VariationGroups). It is often the case that in practice both

system and constraint models are far more complex [43]. While in the EPB case the Co-OVM model satisfied our requirements, we still have to experiment with larger models. The details of this example are presented in Chapter 6.

4.7 A configuration process for SysML models

Product line derivation requires methodological and tool support, being an error prone and complex task. Derivation can also be regarded as a decision making activity [77], where the engineers take into account existing or new alternatives to reach a complete definition of the product. Furthermore, it is essential that the activities needed to perform product derivation integrate into the process and methodology of the organization.
We have focused our example on a common scenario for the automotive industry, where carry-over and carry-across techniques are used to reduce costs and accelerate the development cycle - development by reuse [3].
Figure 4.18 presents a few more details on the activities performed during derivation. The derivation is done in stages, by taking into consideration viewpoints, as presented in Chapter 5. Depending on the phase of the process, reuse choices may be of different nature - high level vehicle characteristics and environment interaction alternatives (diversity), engineering choices related alternative designs or specific component selection from suppliers.
The input of the process determines the perimeter of the diversity context : variants concerning the environment and technical context, that should be covered by the system solution. The output shall document diversity use-cases for components or subsystems (in which diversity context they can be used). The process can be repeated on each level of decomposition, each time the applicability of the system solution, or component (the configuration information) being expressed in respect to the upper level system.

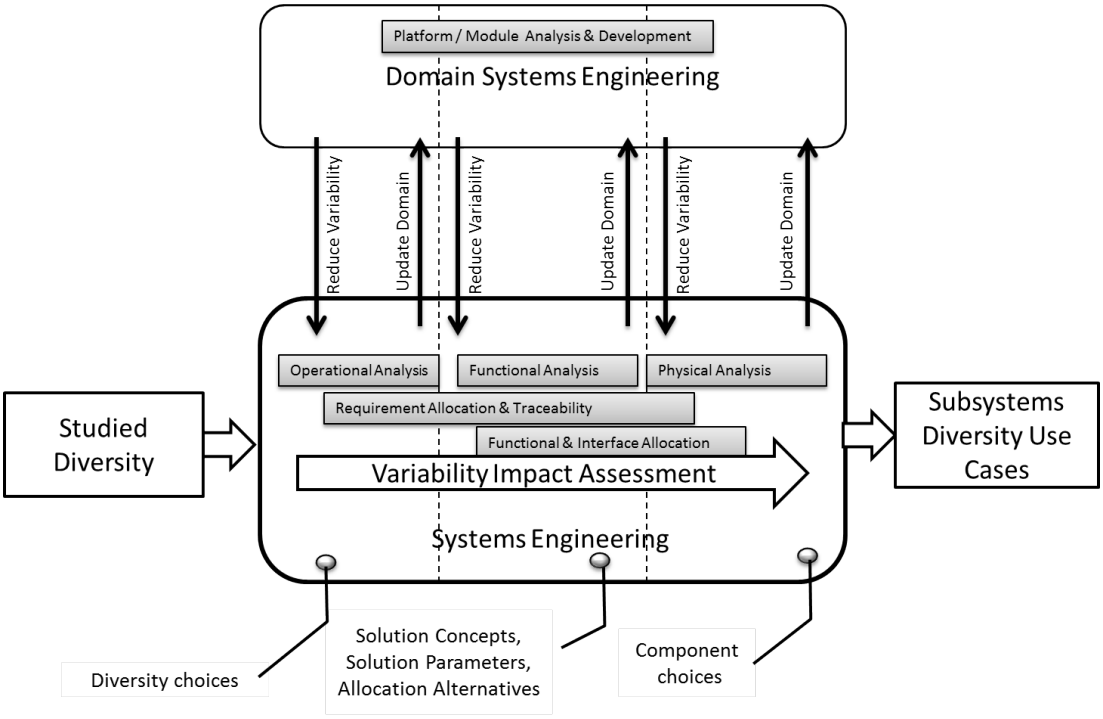


Figure 4.18: Derivation process viewed from a MBSE perspective

During the modeling activities, variability impact needs to be evaluated in requirements, function and component hierarchies, and through allocation. Dependencies between components stem from the ability of each component to provide the required functions in relation to the other system resources.
The type of project has a direct impact on the amount of reusable elements from the family of systems model - research projects focus on providing alternative solutions to existing contexts and needs (reuse of operational analysis elements), while typical commercial vehicle applications may benefit from existing elements during all the phases of the process.
Figure 4.19 presents the activities performed at each phase of the analysis, based on the assumption that each phase addresses a single level of abstraction or decomposition of the system.

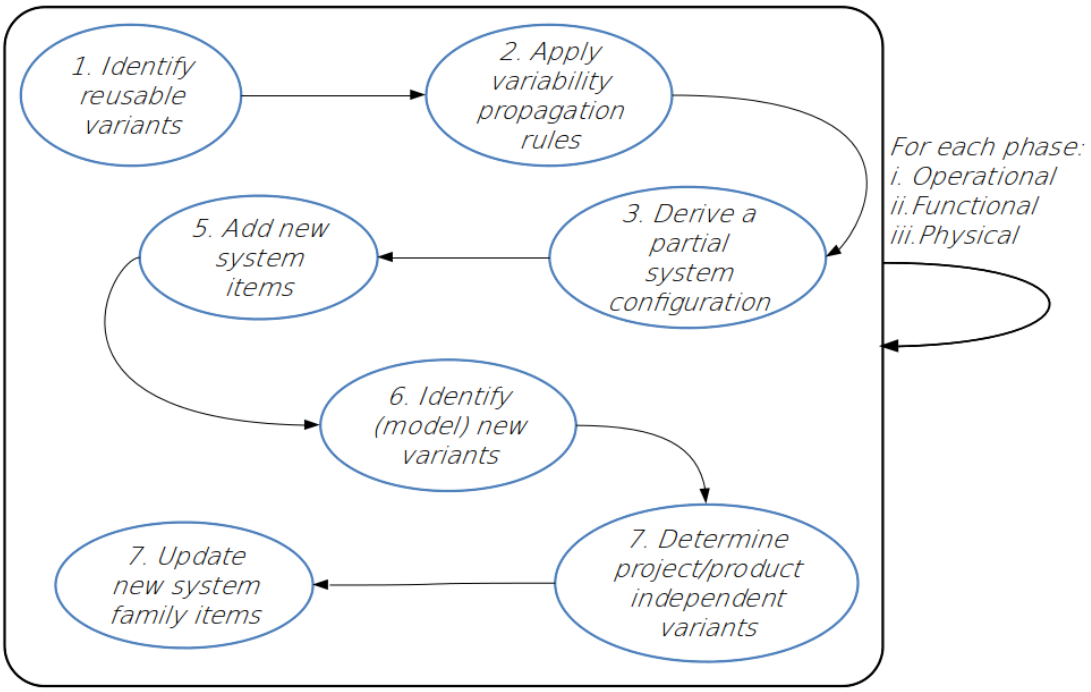


Figure 4.19: Reuse based system modeling activities for each analysis phase

They need to distinguish between two models: the system family model (domain model) and system model, which is created from the former through reuse and adaptation. The activities related to variability management for a single phase of system analysis are:

1. *Identify reusable variants* – the System Architect (SA) analyzes the existing variants in the system family model and selects the variants applicable to the current project.

2. *Apply variability propagation rules* – the SA selects the appropriate variability propagation rules to prepare the reuse for the next analysis phase. For example, the propagation of optional elements through allocation can prepare the model when going from system functional to physical architecture. Consequently, this activity is not applicable for the last analysis phase.

3. *Derive a partial system configuration* – the SA creates a partial configuration, which can be used to provide deliverables at corresponding project milestones. By creating the partial configuration, the model complexity is now reduced and the SA can focus on the system items necessary in the next phase.

4. *Add new system items* – since most projects contain some degree of novelty or change, the SA adapts the system model to satisfy new requirements, or provide improved solutions to existing needs.

5. *Identify and model new variants* – once the system model has been changed, the changes need to be mapped to new variants, unless they are common to the whole system family.

6. *Determine project/product independent variants* – the variants that are potentially reusable in other projects need to be identified, and proposed as candidates for updating the system family model. Because the development of a single system may address multiple vehicle configurations, the model itself can contains local variability, which may not necessarily be of interest for the whole system family. For example, the EPB system locks the parking brake automatically, when the engine is stopped, on most vehicle models. However, on a few vehicle models (91, 43, 47, 45) the condition is that the driver door is open, due to a specificity – the lack of a CAN (controller area network) message from the dashboard signaling the stop of the engine command. Assuming that future vehicles will not bear the same specificity, this latter behavior should not be merged into the system family model.

7. *Update new system family items* – the new variants and the related model items are accepted (by the domain authority, if different from the SA) and merged into the system family model.

4.8 Closure

System models represent an efficient tool for the support of SE activities, but variability management becomes a necessity for approaching product complexity, in large organizations which develop customized products. MBSE is just one of the activities which are impacted by product variety and need to support variability management for adoption in mass customization industries. This chapter proposed the adoption of a model for system variability in model based system engineering, based on concepts coming from the software product line literature and organization practices. Variability models are local for each family of systems, they complement system models and avoid redundant aspects. Meanwhile, they share a part of the elements with the organization "documentary" language, bridging the gap between vehicle characteristics and component configuration specifications.
The next chapter – 5, introduces the technique that enables the derivation of SysML system model to be performed as a succession of partial configurations, as required by the activities described in Figure 4.19, and in accordance to the SE process phases.

5

Product Derivation for Model Based Systems Engineering Processes

Systems engineering enables the successful realization of systems, focusing on defining customer needs early in the development cycle. However, when the development of systems needs to rely on legacy designs there is little methodological support. Furthermore, in the automotive domain, product diversity increases system complexity so much, that at some point reuse becomes difficult and time consuming to achieve. We believe a specific strategy must be adopted to prepare for reuse and to achieve SE by reuse. While product line derivation provides the means to obtain single products form a collection of assets, there is a lack of flexibility and support from a systems perspective. In this chapter we present an approach that takes into account SE methodological aspects in product line engineering and provides a means to develop new systems by reusing existing designs.

Whether developing a new innovative system or reusing existing assets, engineers often need to adapt existing systems to new customers needs or deployment environments. In the case of automotive systems, the family of systems assets are useful (though not sufficient) to derive the specification for a new vehicle system project. While existing designs are used, improvements or new functionalities are introduced with each new iteration, which means that systems frequently require redesigning or altering the products.
Product line engineering derivation brings us closer to reuse, but from the SE perspective we believe it needs to improve:

a) *flexibility* – to satisfy the need of altering existing models, derived from a family of systems

b) *methodological support* – to follow the traditional SE process from requirements

to detailed architectural design.

c) *guidance* – to support the configuration process in variability rich system models, which is typically the case of the automotive industry.

This chapter introduces two approaches, which aim at satisfying the three aspects presented above, on two different levels. The first one deals with methodological support and flexibility on the UML/SysML model level, by introducing new concepts that link the derivation and SE processes. The second approach deals with the guidance in large models, on constraint programming level, in order to improve the configuration process time and length. The two approaches can be applied effectively together, because they deal with configuration of MBSE models on different levels.
The chapter starts with a brief literature review on methodological support for product line derivation, in Section 5.1.1, and on recommendation-based configuration for product lines, in Section 5.1.2. Section 5.2 introduces the issue of methodological support and presents some industry and background information on PL derivation. The first contribution is introduced in Section 5.3. Section 5.4 presents the way the approach was applied to the Electric Parking Brake case study, following different derivation paths. These derivation scenarios were used for validation of the tool support. The second contribution, consisting in recommendation heuristics for product line configuration, is introduced in Section 5.5. Finally, Section 7 concludes the chapter with perspectives. Additional information on the application of the recommendation heuristics can be found in Appendix B, which presents a practical application on the Electric Parking Brake case study, in GnuProlog, and a synthesis of the advantages of the different heuristics.

5.1 Related work

5.1.1 Related work on methodological support for product derivation

As other publications have pointed out, derivation requires methodological and tool support. It is indeed an error prone and complex task consisting in performing complex configuration on a set of assets of different nature and with intricate inter-relations. Furthermore, derivation methodology needs to integrate with existing organization practices, in our case with the SE development process as explained by Chalé et al. [58]. In consequence the derivation of the product line needs to be regarded, as proposed by Djebbi [77], as a decision making activity, where the engineer might take into account existing alternatives or develop new assets to reach a completely defined product. However, another constraint is to allow the user to perform usual SE activities during derivation according to the framework proposed by Chalé [58], with a focus on flexibility, which is not an issue considered by the previously mentioned approach.
Deelstra points out [70] that the derivation "is a time-consuming and expensive activity" and provides a thorough investigation based on real industry case studies, that identify the sources of these problems. We are confronted with a series of similar issues,

such as managing the complexity of the derivation activity or evolving the family of systems by adding new assets during derivation, but in the context of a predefined framework for SysML models (Renault SysML profile).
Rabiser et al. [166] present an approach to support product derivation through adaptation or augmentation of variability models. This approach is supported by the DOPLER tool suite and it explicitly captures derivation information in a decision model. The solution that we propose relies heavily on complementary information contained in the system model, such as predefined stakeholders or specific points of view. We do not capture role information for example in the variability model, as in our context this kind of information can be provided by family of system models. At the same time the SE framework provides an asset collection structure which proves useful during derivation. The problem of finding the right assets during derivation is also pointed out by Hunt [112] who provides an evaluation of the impact of organizing the asset base in different ways.
Product diversity is the cause of an outstanding complexity increase in both automotive software and system domain, as suggested by Astesana [43] and Tischer [202]. Variability and complexity impact all aspects of the organization activity, from marketing, customer interaction (through online configurators), engineering (through the design of families of systems and capitalization) and eventually manufacturing in the plant and logistics.

5.1.2 Related work on recommendation-based configuration of product line models

An interactive product configurator is a tool that allows the user to specify a product according to his specific requirements and the constraints of the product line model to combine these needs. This process can be done interactively, i.e. in a step-wise fashion, and guided, i.e. proposing the resolution of certain requirements before others and automatically proposing a valid solution when there is only one possible choice in the solution space. To be useful in the e-commerce context, a configurator must be complete (i.e., to ensure that no solutions are lost), allow order-independent selection/retraction of decisions, give short response times and offer recommendation to maximize the possibilities to have one satisfactory configuration. Solution techniques applied to the interactive configuration problem have been compared by Hadzic and Andersen [101] and Hadzic et al. [102]. They mainly distinguish approaches based on propositional logic on the one hand and on constraint programming on the other hand. When using propositional logic based approaches, configuration problems are restricted to logic connectives and equality constraints [102, 189]. Arithmetic expressions are excluded because of the underlying solution methods. These approaches have two steps. First, the feature model is translated into a propositional formula. In the second step the formula is solved by appropriate solvers, in particular SAT solvers [148], and BDD-based solvers [102, 190]. BDD-based solvers translate the propositional formula into a compact representation, the BDD (Binary Decision Diagram). While many operations on BDDs can be implemented efficiently, the structure of the BDD is crucial as a bad

variable ordering may result in exponential size and, thus, in memory blow up. Even if several heuristics are used to improve the use of BDD solvers in the context of PLs [148] heuristics are not exploitable in industry; they are not even used in the author's web site[1] for heuristic's instability reasons.
Feature models can be naturally mapped into constraint systems in order to reason (e.g., configuration) on them, in particular into CSPs as presented by Subbarayan [190], Benavides et al. [48], and Deursen and Klint [71]; into Constraint Programs over finite domains CPs as presented by Mazo et al. [140], and into Constraint Logic Programs (CLPs) as presented by Mazo et al. [141].
There are several academic tools dealing with configuration of product line models (e.g., SPLOT, FaMa[2] , VariaMos [145], Feature Plug-in [37] but very few have looked at PL tools and their ability to answer industry needs [41].
Jiang et al. [118] propose a constraint-based recommendation technique that gives to stakeholders a minimal set of repair actions in situations where no solution can be found for their configuration choices. Their approach also takes into account the preferences of a customer community to include collaborative recommendation. Authors deal with the situation where no solution can be found for the customers' configuration as a constraint satisfaction problem. Authors use a well-known collaborative recommendation technique consisting in calculating the contribution of each configuration choice in terms of reliability, economy, and performance by means of a utility function (cf. Multi Attribute Utility Theory (MAUT) [215]). In that work the term utility denotes the degree of fit between a feature of the product and the given set of customer requirements.

5.2 Flexible product line derivation in MBSE and industry needs

Our approach for the configuration of a system from a family of systems, consists in several successive steps, that gradually reduce the model scope, but also enable the specification of new model items. As support for our approach we use the SysML modeler Papyrus with the Sequoia add-on for managing product lines.
In order to understand the industry needs, we have developed a set of SE scenarios involving the reuse of assets (e.g., requirements, solution designs, models): synchronized systems development, integration of a single system into a product line, merge of two product lines, derivation of a single system from a system family, development of a single system based on reuse (through derivation). We have chosen the latter as a nominal case, that we exploit in this chapter to define our goals and constraints, which are articulated around the following statements:

- Starting working context of the project:
    1. We assume that the family of systems (FoS) model is defined a priori;

[1] http://www.splot-research.org
[2] http://www.isa.us.es/fama/?FaMa_Framework

2. We assume that there is sufficient detail in the FoS description to derive a single complete system specification, as a consequence of previous development efforts
3. As process input, we assume that a variability definition of the system context (environment, technical context and client services) is "inherited" from the upper level system analysis (e.g. vehicle variability propagates to subsystems).

- In addition to the typical activities of the SE process, we need to take into account other activities related to reuse:

  1. The approach shall enable identification of *system scope reuse opportunities*: for that the engineer has to choose existing assets in relation to high level requirements, environment variability, technical context and commercial offer. These assets, related to the operational analysis viewpoint of the SE process, define and place the system context and its use cases in relation to previously developed systems and and orient subsequent activities towards reuse.
  2. The approach shall enable identification of *solution reuse opportunities*. The engineer is required to choose among potential solutions by performing concept synthesis/analysis and trade-off analysis. He may either decide to go for an existing solution, matching a similar "system scope" and use cases, either develop a new architecture, to satisfy new requirements or to provide more efficient solutions to existing problem definitions.
  3. The approach shall enable *identification of component and module reuse opportunities*. The engineer is required to match the system architecture with existing solutions and take advantage of existing components or modules. Thus, in a top-down approach, a common architecture can lead to reusable components. In the opposite case, a bottom up-approach may allow the engineer to match and combine existing components and modules to realize the required system functions.

- The output of the process consists of:

  1. Allocation of requirements to components for the selected configurations.
  2. Variability description for each component belonging to the selected configurations.

These statements are compatible with the general SE process described by Chalé et al. [58]. Figure 5.1 illustrates the relation between the activities presented above and system assets : with each reuse decision, variability contained in the description of the family of systems is solved, leading to an intermediate and partial configuration of the system.

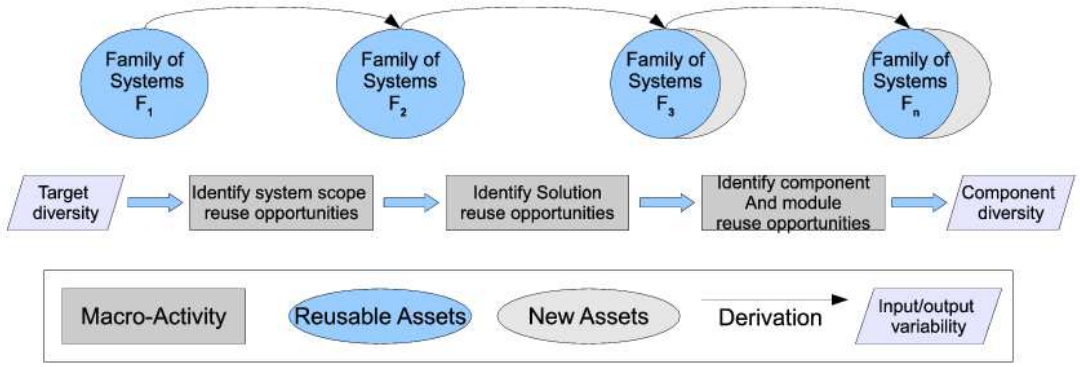


Figure 5.1: Reuse related activities in the development process.

The proposed approach allows to introduce new assets for each partial configuration. The *target diversity* is specified, as input to the process, using the Documentary Language [42]. The key activities where reuse opportunities should be evaluated correspond to transitions between the different viewpoints: operational, functional and physical [80], that correspond to our reference model-based SE architecture framework. At a more general level, these transitions correspond to the passage from higher to lower levels of abstraction (to a more detailed description of the problem) or to a subsystem or subcomponent (e.g. environment to vehicle or vehicle to subsystem). Of course, if the requirements offset is too important in respect to the family of systems and if new requirements cannot be satisfied by existing solutions, the activities will be oriented towards development, instead of reuse.
A *partial system configuration* represents a collection of assets that includes variability and is obtained through partial resolution of variability contained in the family of systems definition.
We define *target diversity* as a partial system configuration of a system that refers only to: the system environment, the system technical context and customer services (commercial offer). It represents the diversity of customer needs that the system shall cover and the diversity of environments a system shall be used in.
Even if most of the system and variability assets are already defined before starting the derivation, the target diversity may have a larger scope than what is already defined. In consequence variability creation and resolution activities may be performed even during the development of the same product.

5.3 Application of partial configurations in MBSE

We have imagined two scenarios for the derivation of a single system with the Sequoia tool, which are compatible with the context and process related elements presented in section 5.2.

Scenario A (*Derivation based on system viewpoints or stakeholders*): we consider the derivation process follows the development process and a certain sequence of choices performed to gradually reduce the model scope.

Scenario B (*Non-functional requirements as criteria for derivation*): we take into account a derivation process where the final product has to satisfy certain non functional properties (e.g. weight, cost)

We can achieve these scenarios by performing several partial configuration on the family of systems model. In consequence, each configuration step should take into account only a part of the variable elements, while the rest are to be kept in the resulting model. At the same time the engineer should be able to modify or define new variable or non variable model items between stages.

5.3.1 Partial system configuration

In order to perform a partial configuration we rely on the description of a CSP problem (classically defined as a triplet $P = (X, D, C)$, for any variable $x_i \in X$, $D(x_i)$ is the domain of x_i, while $c_{1..n} \in C$ a set of constraints related to $x_{1..k}$). The Sequoia tool allows the user to select among a list of predefined constraints: implication, equivalence, AtLeastOne, AllPosibilities, Alternative, or to define a custom expression. Each variable is related to a model item, which is stereotyped as *variable*, meaning the item can be present, absent or replaceable.
A viewpoint (on a system) is an abstraction that yields a specification of the whole system restricted to a particular set of concerns. Suppose γ denotes a viewpoint restricted to a set of concerns $\phi_\gamma = \{\varphi_1..\varphi_n\}$. We suppose there is a user defined rule of correspondence $f_\gamma : \phi_\gamma \rightarrow X$, which associates to each concern φ a set of variables x from X. An assignment g_V is a function that maps any $x_i \in X$ to a value in $D(x_i) \cup \{\#\}$, using the symbol "#" to denote that the variable has not yet been assigned a value from D. The activities that lead to a partially derived (configured) system model are:

0. Define family of system concerns γ for derivation.
1. Define viewpoint ϕ_γ.
2. Determine $V_d = f_\gamma(\phi_\gamma)$ the set of variables for partial derivation.
3. Assign values from $D(x_i) = \{0; 1\}$ for each variable x_i until g is a complete assignment on V_d, such as $\forall x_i \in V_d, x_i \neq \#$.
4. For each $x_i = 0$ remove model items.
5. Allow the user to define new model items.
6. Regenerate CSP problem : $P' = (X', D, C')$, where $X' = X \setminus V_d$ and C' the set of related constraints.

5.3.2 A UML profile to support viewpoint based derivation

The Sequoia profile was modified to support partial derivation, by introducing stereotypes that enable filtering constraints in respect to particular concerns defined by the user, as described in Figure 5.2. They implement the concepts defined in Chapter 4.

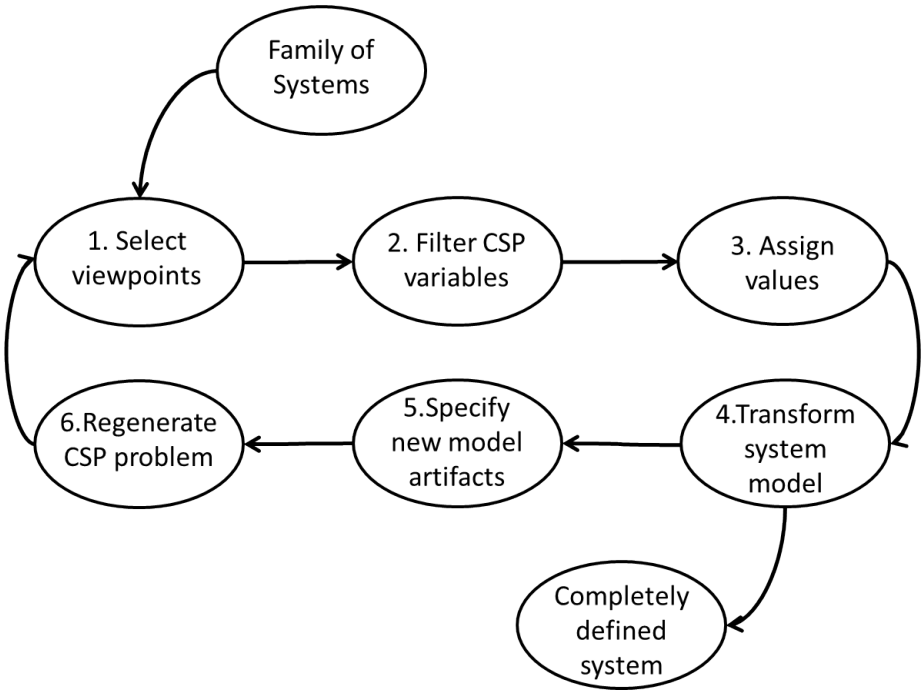


Figure 5.2: Partial derivation activity flow

The stereotypes that were added are *DecisionCriteria* and *ProductCriteria*. The concept of *DecisionCriteria* is used to define each viewpoint and associated variants and variable elements. They provide a way to map specific user concerns for derivation to configuration variables and constraints.
The viewpoints are generally defined depending on the SE development process phases and system stakeholders. For example, a series of *DecisionCriteria* can be defined which follow the different levels of analysis of a system, or *DecisionCriteria* which regroup choices concerning a certain development phase: requirement elicitation, technical requirement definition, solution alternatives, functional architecture definition physical architecture definition, etc.
One of the ways the system can be partially derived, is to associate the different stakeholders (stakeholder model element) included in the development process and in the system scope. A partial system model can be obtained through derivation, where we all customer or marketing variations are resolvedn, still containing variations for all detailed design decisions or parameters. Each variation point can be referenced by

multiple *DecisionCriteria*, which allows for different aspects to be taken into account in the creation of a configuration model for the system family.

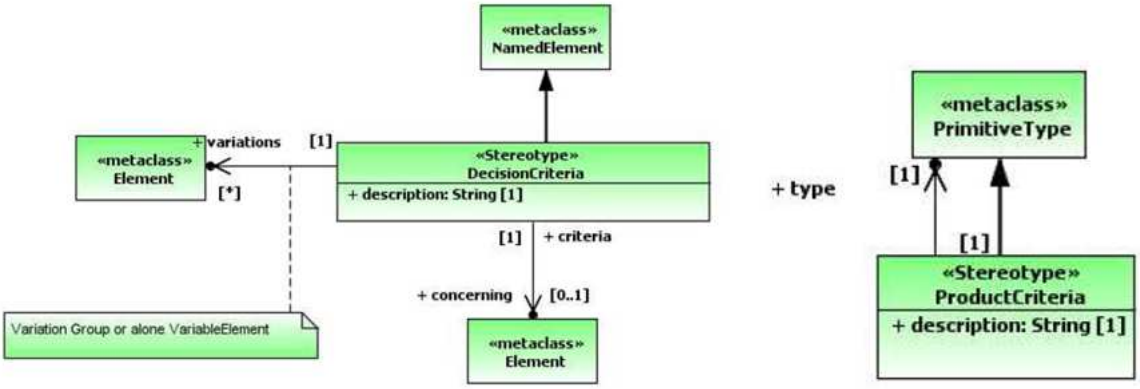


Figure 5.3: The stereotypes DecisionCriteria and ProductCriteria

The element *ProductCriteria* was added to express non-functional properties of the products. For example, it can contain an integer, a string or a boolean value which can be associated to the product.
The activities that need to be performed in the "Renault variability management" tool are presented in 5.4.

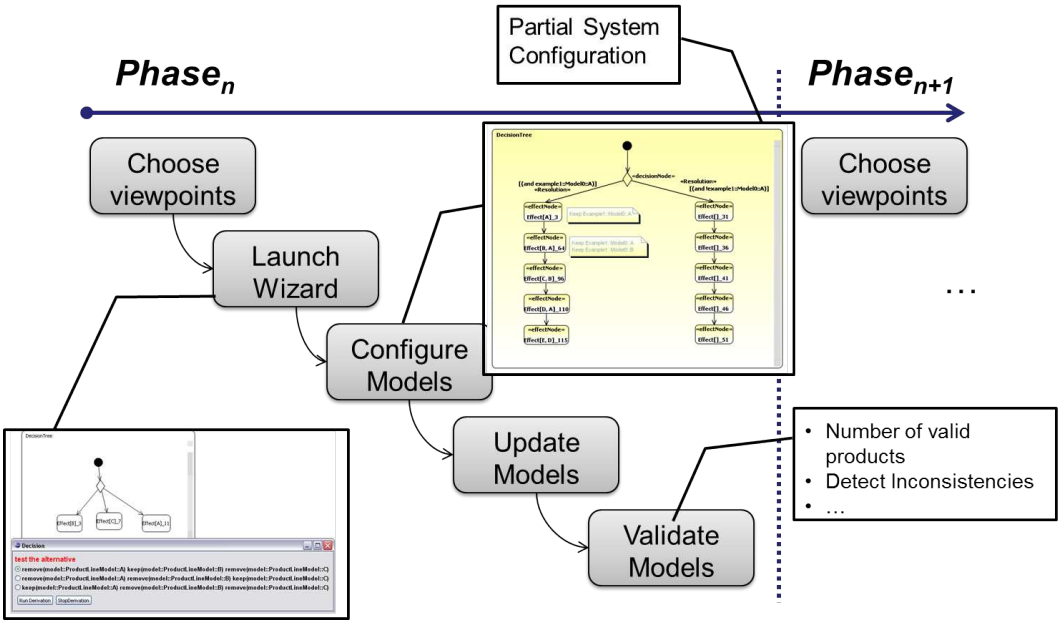


Figure 5.4: Activities performed in the tool for each partial derivation

Variants can be selected, deselected or left undefined. Only defined variants and constraints in the active viewpoints are taken into consideration during the configuration of SysML models, in the first step. In a second step, the user can choose to continue in the Sequoia [197, 199] interactive mode (launch wizard in Figure 5.4), in order to allow the user to select variants that were left "undefined", from the active viewpoint. In this mode, individual questions for variant selection are proposed interactively to the user, as in most online configurators.

5.4 Configuration alternatives for the Electric Parking Brake

We tested a derivation scenario, on the electric parking brake example, according to the implementation presented in section 5.3.2 in the Sequoia tool for managing model based product lines. In order to create the criteria sequence, the family of systems model is considered already available along with the required variability model.

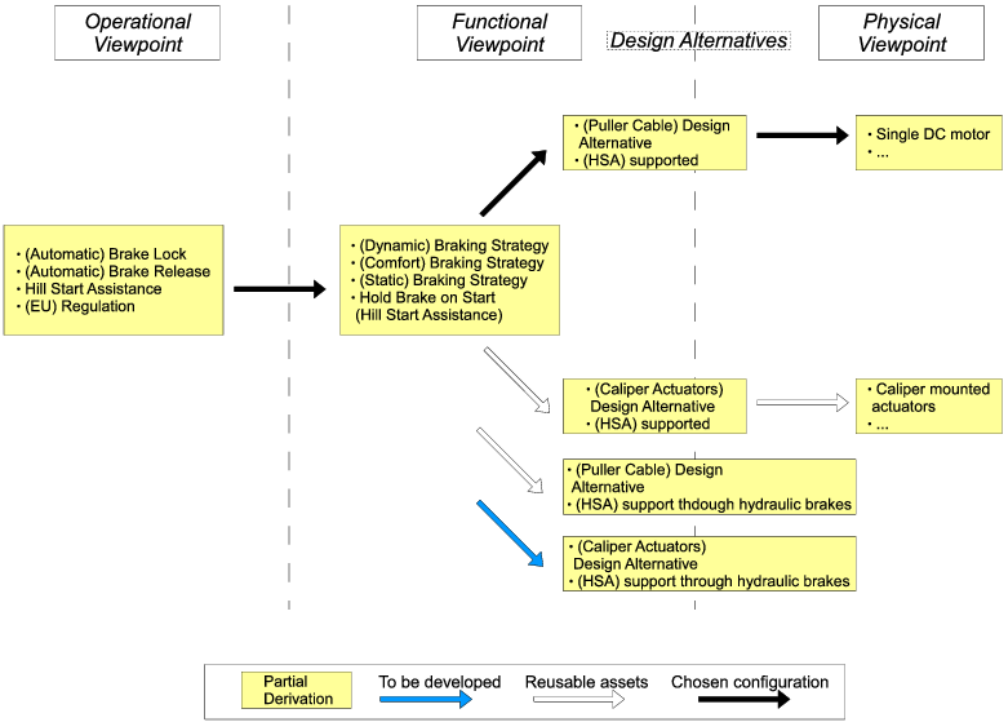


Figure 5.5: Derivation scenarios for the EPB system

The first step from here, is to model the derivation criteria within the FoS model. After modeling all needed derivation criteria, derivation process flow is customized by selection of the appropriate order and desired criteria through a specific model wizard. Figure 5.5 describes a sequence of choices that lead to the configuration of a single EPB system model. Two types of variability can be seen: one issued by stakeholders or cus-

tomer requirements (the customer being represented by the marketing representative), and the other belonging to the solution domain, relative to design alternatives. Thus it is possible to reach different designs on the solution level, depending on the selected path and on the initial configuration on problem level. However some of the explored alternatives may not yet be implemented (e.g. a new design leading to cost reduction, increased performance, based on component availability etc.), and reusing assets will only be possible for the operational viewpoints and stakeholder requirements in this case.
The stereotyped elements presented in Figure 5.6 enable the user to specify the kind of model generation, partial or complete, along with a sequence of criteria that will impact the order of proposed choices. The *decision criteria* defines which variants are associated to a SE engineering viewpoint, for configuration. By introducing these elements ir is also possible to customize of the flow of decisions, to match the SE process, and produce documents based on the partial configurations. In our application we used one *decision criteria* for each SE viewpoint, but it is possible to define sequences on a finer grained level. However, there should be no need to control the configuration flow to individual variants. The purpose is to enable support for the SE process and for the generation of deliverables along the processs, based on the assumption the the existing family of system models need to be revisited by the engineer, improved or extended. When the models are very large, this kind of methodological support provides of form of guidance for revisiting and extending the family of systems models. Form an implementation point of view, the way the user associates variants to criteria and selects the derivation sequence, based on the viewpoints, is presented in Appendix E. In the case where the decision criteria only correspond to a subset of all variation groups, after generating a decision model, only the linked choices will be proposed during the derivation phase. The result of a derivation based on a partial decision model will be a new family of systems model containing fewer variations as the previous one, but not the complete specification of the product.

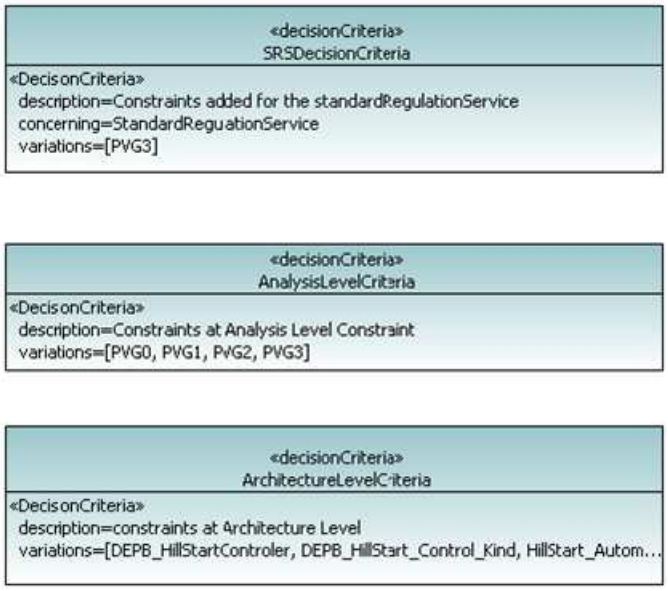


Figure 5.6: Elements stereotyped "decisionCriteria"

In the case of a complete derivation, the tool will search the variation groups which were not selected in the derivation sequence, and will propose them as decisions after the treatment of the variations linked to the selected criteria.
In this way, we are not only able to impose a desired sequence of choices, in accordance to the SE process, but also to create several iterations of derivation and modeling, where new assets can be introduced or removed. It allows at the same time to distribute the derivation process between several actors, which could have different preoccupations, without the need to know the details of the product line model.
Since the derivations are successive, there is no problem concerning conflicting decisions, because the constraint model is regenerated after each partial derivation. In the case where the derivations are made in parallel however, conflicts can appear, but this aspect is treated for now on the methodological level, meaning that co-configuration is not supported.
Figure 5.7 presents the tool menu that enables the inclusion of variants in each viewpoint.

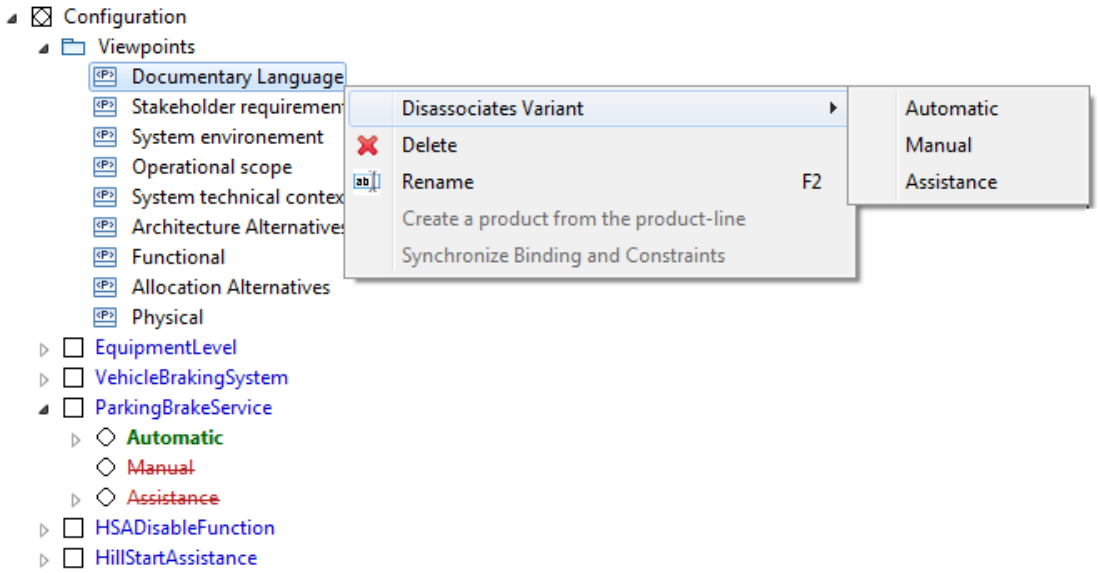


Figure 5.7: Association of variants and pre-defined viewpoints

In this example, variation point *ParkingBrakeService* belongs to both *Stakeholder Requirements* and the *Documentary Language* viewpoints. The variation point regroups three variants: *Automatic*, *Manual* parking brake, and *Assistance*. Currently, the user needs to define manually associations, except for the variants existing in the Documentary Language, for which associations are automatically defined on import.
Figure 5.8 presents the main concept introduced in this chapter. It is important to note that in other contexts where configurators are used (e.g. online product configurators), there is no need to follow a particular configuration methodology. While in the proposed approach the user free configuration order is maintained within each step (for each SE viewpoint), in online applications this aspect needs to be respected on the product level [157].

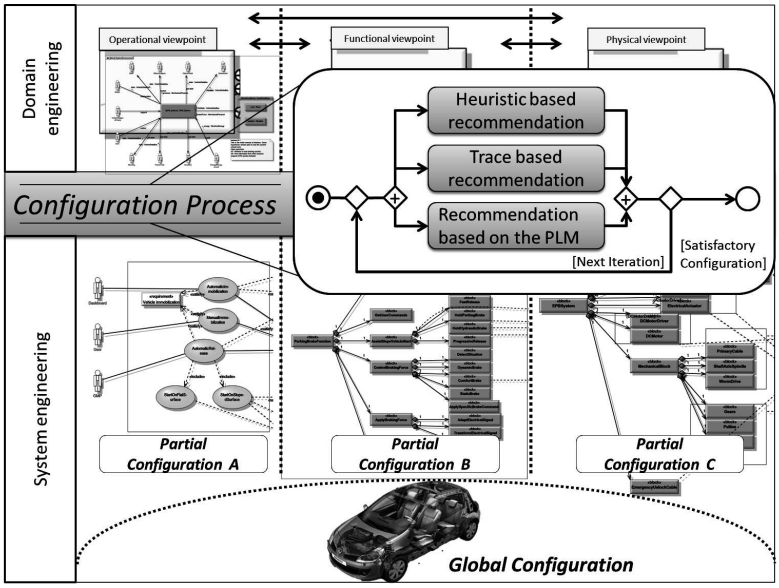


Figure 5.8: Overview of the derivation process with partial configurations over SE models

Another approach to support to the users, when they are faced with large product line models, is to provide recommendations regarding the order of configuration choices. While the first part of this chapter considered a relative order of choices based on methodological aspects, the next part deals with the improvement of the overall configuration process in respect to the order of choices. The purpose of this second approach is to improve the response time from the solver and shorten the number of steps of the configuration process. In the following section 5.5, we introduce three types of recommendation techniques applicable in product line engineering, and propose some recommendation heuristics, in the context of heuristics-based recommendation [8]. In the following section, UML/SysML modeling aspects are left aside, and the focus is on the constraints programming level, with a practical application in GnuProlog.

5.5 Recommendation heuristics in the product line configuration process

In an industrial context, letting the customers express their requirements that they want, while considering the constraints imposed by the product line model, entails a cumbersome and error prone configuration process. Evidence of these difficulties are multiple. For instance, in Business-to-Consumer commerce (B2C) in France, 60% of the shopping baskets in e-commerce are abandoned before purchase, because the customer does not find a satisfactory configuration. The conversion rate visitor/buyer

rarely exceeds 15%, according to FEVAD[1]. This difficulty is even more critical when the e-catalog contains a huge number of items, as is the case when the products to be sold are highly customized (e.g., computers or vehicles). One possible answer to this difficulty can be an interactive configuration process that recommends customers the best configuration alternatives to follow. When configuration is done interactively, the user specifies the characteristics of the product step-by-step according to his requirements, thus, gradually shrinking the search space of the configuration problem. This interactive configuration process is supported by a software tool (called configurator) that is intended to recommend to customers the best configuration alternatives that lead them to a satisfiable product in a minimum number of steps.
The purpose of te collection of heuristics, introduced in this section, is to: (a) help customers specify the characteristics of their products step-by-step according to their requirements, and (b) to avoid useless or inefficient decisions. The collection of heuristics were designed to improve the configurators' interactivity and thereafter successfully contribute to a faster and less error prone configuration process. A detailed description of the implementation of these heuristics is presented using Constraint Programming. This formalism was chosen because it can be implemented in a straight-forward way, and because it can be used to formalize any product line whatever notation was initially used to specify it [141, 142]. The application of the heuristics is demonstrated using our Electric Parking Brake Systems case.

5.5.1 Principle of heuristics-based configuration

The objective of the application of heuristics to the configuration process is to increase the chance of success and to reduce the configuration time: (i) either by reducing the number of configuration steps, or (ii) by optimizing the computation time required by the solver to propagate configuration decisions.
Configuration is an interactive and iterative process. The user makes choices by assigning values to configuration variables. A step in the configuration process consists of a user choice and propagation of configuration decisions by the solver, if the value is valid, or of going back to the previous configuration state otherwise. Each value assignment can affect other variables. As a result, the order of value assignments has an impact on the overall number of steps of the process. Finally, the computation time of each configuration step is influenced by the complexity and size of the product line model.
A generic configuration process iteration is presented in Figure 5.9, as a flow of activities. Activities (e.g. selection, transformation, decision making) are represented as triangles, while assets, used in the configuration process (e.g. product line model, configuration sequence), are presented as rectangles. The asset may be the input of an activity, if it is on the left of that activity, or the produced output if it is placed on the right. Finally, the arrows link the graphical elements and point to the direction of the activity flow. Each configuration iteration consists in the following activities:

[1] Fédération e-commerce et vente à distance – http://www.fevad.com

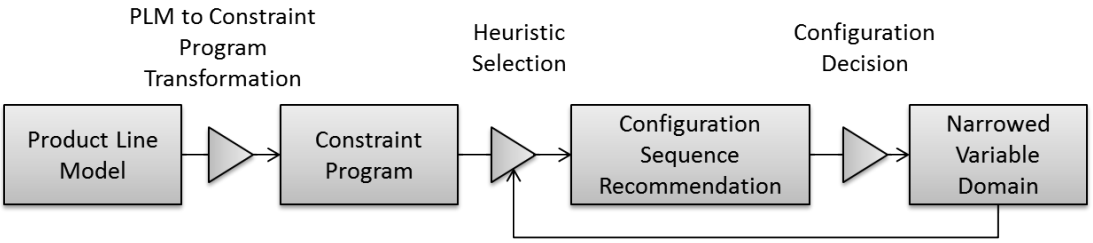


Figure 5.9: Configuration workflow for product line heuristics-based recommendation

1. *Product Line Model (PLM) transformation*: The product line model, described using OVM, is transformed to a constraint program. We use the GNU Prolog language [74] to represent the constraint program and the GNU Prolog solver [74] as the engine to solve it. The configurator consists in a front-end (e.g., online interface) and a solver. The solver propagates the configuration decisions and ensures they are valid with regard to the product line model. The transformation from OVM to GNU Prolog, for the Electric Parking.

2. *Heuristic selection*: The user can change the heuristics taken into consideration for future configuration steps, in respect to the desired objective. The appropriate context and advantages for each heuristic are discussed in Section B.2.

3. *Variable prioritization:* The configurator recommends variables to be configured, ordered in respect to the applied heuristics.

4. *User choices*: The user can follow the recommendations, or can continue to assign values to other variables of interest.

5.5.2 Configuration heuristics

Configuration of variability models enables the realization of consistent system specifications from requirements at the domain level. However, as diversity plays a major role in automotive industry competitiveness, it is often difficult to manage the large number of variations that a vehicle system includes.
While users can express their priorities in a free configuration order, the configurator should choose the order that best optimizes its computational task and complete the configuration of the product line models based on the users decisions. The configurator completeness should ensure that no solutions are lost. Besides backtrack-freeness [101] should guarantee that the configurator only offers decision alternatives for which solutions remain, and the configurator should guarantee a short response time compatible with an interactive usage (e.g., with a web-based configuration). By defining a set of heuristics for product line configuration we intend to address the issue of decision ordering in order to allow the user to have access to pertinent choices and optimize the number of steps required to reach a complete configuration and the response time of the solver. The heuristics presented here are:

- Heuristic 1: Variables with the smallest domain first
- Heuristic 2: The most constrained variables first
- Heuristic 3: Variables appearing in most products first
- Heuristic 4: Automatic completion when there is no choice
- Heuristic 5: Variables required by the latest configured variable first
- Heuristic 6: Variables that split the problem space in two first

It is worth noting that there is not a predefined order to use the heuristics. Indeed, the configuration heuristics that can be suitable to be used in a configuration time t can be different from the configurations heuristics that the user will want to use at *t+1*. There are several heuristics that the user can select together for one or several configuration steps. In the case that users select several configuration heuristics, the configurator will propose a collection of candidate variables to configure according to the selected heuristics. For instance, if the users want to configure first the variables with the smallest domain (Heuristic 1) and the variables with the largest variability factor (Heuristic 3), the configurator will recommend them to configure the variables with the smallest domain decreasingly sorted by the variability factor. These heuristics are discussed below.

Heuristic 1: Variables with the smallest domain first **Principle:** This heuristic recommends to choose first the variable with the smallest domain. The domain of a variable is the set of possible values that the variable can take according to its domain definition and the constraints in which the variable is involved. This strategy is known as "first fail principle" [55] and can be explained as "to succeed, try first where you are most likely to fail".
Rationale: This could be counter-intuitive since configuring first the variable with the biggest domain reduce the search space the most. However, this decreases the possibility of obtaining a valid product at the end because that constrains the solver's possibilities to choice a particular domain value that satisfies the set of constraints. Thus, even if setting first the variables with the largest domains reduces faster the solution space to be analyzed by the solver than setting first variables with small domains, it also decrease the possibility of having a valid product at the end of the configuration process. Of course, we prefer the latter choice. If all the variables have the same number of domain values (e.g., all variables are Boolean), the user can choose other heuristics to improve the configuration process.
Example: For instance, let us consider the variables:

$$V_1 = ForceDistributionDesignAlternatives, \text{ and}$$
$$V_2 = ElectricalActionComponents.$$

They have enumeration domains with different cardinalities, where $card(V_1) > card(V_2)$, thus the second variable has the smaller. The choices associated to the first variable decide about the way brake force generation should be shared between the electric parking brake electrical actuators and the classic vehicle hydraulic system. The second variable decides about the physical architecture: single DC motor or two electric actuators attached to the vehicle calipers. The solver should propose variable *ElectricalActionComponents* first, according to the proposed heuristic.
Advantages: This heuristic avoids unnecessary evaluations (e.g., when variable V_2 is set to a particular value of its domain, the solver does not need to check for other values of V_2) and increases the possibility to succeed the configuration process (i.e., configuring first the variables where the constraint program is most likely to fail, we increase the possibilities of the solver to succeed).

Heuristic 2: The most constrained variables first **Principle:** Another heuristic, that can be applied when all variables have the same number of values, is to choose the variable that participates in the largest number of constraints (in the absence of more specific information on which constraints are likely to be difficult to satisfy, for instance). This heuristic follows also the principle of dealing with hard cases first.
Rationale: In industrial languages like the constraint networks [41] where there are no root artifacts to guide the configuration process, this heuristic allows identifying the variables that mostly reduce the number of choices in a configuration process.
Example: In the EPB model, the *PullerCable* variant is one of the most constrained in this model, being linked to the way force is distributed among braking systems (*ForceDistributionDesignAlternatives*), internal system behavior (*ForceMonitorOnEngineStop*) and physical implementation (*DC_motor*). In consequence, it reduces other choices in the configuration process, directly linked to the initial variable.
Advantages By setting first the variable related with the largest number of other variables the solver would automatically propagate the user's choice to the largest number of other variables. In that way the space of the solution is considerably reduced after each choice, which reduces the number of configuration steps (user's choices) and the average configuration time (solver's inference time).

Heuristic 3: Variables appearing in most products first **Principle:** This heuristic proposes configuring first the variables that have an impact on all potential products. To avoid the generation of all products of the PL, which is usually impossible in very large models [45, 142], we propose two steps:(i) configure first the core variables, and (ii) configure the rest of variables once ordered according to their impact on the solution space. To implement the first step, we use the *computing the core elements* operation fully automated in the VariaMos tool [145]. To avoid generation of all solutions in order to calculate the core elements as proposed by Schneeweiss et al. [179], we evaluate if each variable can take the *nullValue* (e.g., the 0 value on Boolean variables) in at least one correct configuration. If a variable cannot take its *nullValue*, the variable does not become part of the core variables of the product line model be-

cause this can never be omitted from any product. To know if a variable can take the *nullValue* in a valid configuration takes only few milliseconds even in the largest product line models available in literature; Details of this algorithm, its improvements, implementation and evaluation can be found in Mazo et al. [141]. The second step is implemented thanks to the operation of computing the variability factor of a given variable, and fully automated in our tool VariaMos [145]. This variability factor of a given variable corresponds to the ratio between the number of products in which the variable is present and the number of products represented in the product line model.
Rationale: The heuristic can be applied when the user needs to reach a valid configuration with as little input as possible. Fixing the values of the core variables (usually approximately 2/3 of the variables of the product line model [63]) decreases the size of the problem space in the same proportion, avoiding unnecessary input from the user. This also makes feasible the calculation of the variability factor of the remaining variables. The variables with a higher variability factor reduce the path to a valid product and the need for user input.
Example: In products where the customer requirements do not include assistance ($ParkingBrakeService = Assistance$) from the system, other than for parking brake situations, the option for hill start assistance and for disabling the hill start assistant function are excluded. In consequence, the definition of these variables does not appear in all products, and according to this heuristic should not be proposed before core variables.
Advantages: This heuristic is useful because (i) it avoids wrong users' configurations in the sense that core variables cannot be configured with *nullValues*, and (ii) the order proposed in this technique decreases the solver's inference in finding satisfactory configurations.

Heuristic 4: Automatic completion when there is no choice **Principle:** This heuristic provides a mechanism to automatically complete the configuration of variables where only one value of their domain is possible.
Rationale: Due to the fact that the aim is to eliminate, in the domains of variables yet to be instantiated, the values that are not consistent with the current instantiation, this heuristic also works when a variable has several values on its domain but only one is valid. This heuristic is automatically provided by most constraint solvers available on the market, with at least, three algorithms that implement it: Forward Checking, Partial Look-Ahead, Full Look-Ahead.
Example We consider variable $V_1 = HillStartAssistance$, $V_2 = BrakeLock$ and $V_3 = ParkingBrakeService$. Setting the value for $V_1 = EPBEnabled$, $V_2 = OnSystemDecision$ excludes all values for V_3, except $V_3 = Automatic$, in which case this single possible value should be selected automatically.
Advantage: The automatic assignment of values to variables, once there are no more alternative solutions, avoids the possibility of invalid configurations due to wrong values requested by the user. Therefore, it contributes to the success of the configuration process.

Heuristic 5: Variables required by the latest configured variable first Principle: Another heuristic consists in choosing the variable that has the largest number of constraints with the past-configured variables.
Rationale: This heuristic has a particular application during configuration of SE artifacts, where choices often represent engineering decisions linked by causality relations. A choice may not be reached before a previous choice has been performed. In this particular context, following related variables (that serve for the reuse of SE artifacts) would mean advancing in the design space towards a more detailed description of the system being configured.
Example: In the case of the EPB system we can find logical implications which have their origin in technical constraints or problem solving logic. For example, the system needs to be able to disable the hill start assistant function (*HSADisableFunction*), if a trailer is used (when the function is not able to adapt to the change of weight of the vehicle). In this case it is convenient to propose variable *HSADisableFunction* once variable $VehicleTrailer = TrailerW1$ has been set, both for the user (logical flow of ideas), but also to avoid potential future conflicts in the configuration.
Advantages: This heuristic allows conflicts to be identified as soon as possible (with regards to the number of configuration steps) in a configuration process. This hence, increases the possibility to fix the configuration conflicts and the possibility to succeed in the configuration process.

Heuristic 6: Variables that split the problem space in two first Principle: Breaking a problem into sub-problems is a powerful tool in many domains. This heuristic consists in setting first the variables splitting the problem space in two. Due to the fact that these variables have most potential to reduce the solution space, the application of this heuristic naturally reduces the number of configuration choices.
Rationale: This is due to the fact that after one choice, the solver will have a reduced solution space to exam and the user will have fewer configurations to do. Several product line modeling languages (e.g., Feature models) are inspired by this principle and propose the use of a root element with two of more choices related by OR or XOR operators that divide the problem space in branches that can be easily removed from the solution space. To achieve this we need to search for variables that divide the problem space in roughly similar parts, searching the decision trees have almost the same depth for each of the values of the variable.
Example: In the case of the parking brake system, the choice concerning the presence of the hill start assistance function splits the decision tree in roughly similar parts.
Advantages: Setting first the variables that break the problem in two equal parts, reduces the computation time of future choices and increases the chances of reaching a completely defined product in a reduced number of steps.

5.6 Closure

We presented in this chapter an approach to integrate MBSE activities with product derivation and to render the configuration process more flexible, by enabling partial derivations and allowing the introduction of new model artifacts during the derivation process.
Because the development of systems in the automotive domain often relies on reuse, but requires modification or evolution of previous designs, the process of family of systems derivation needs to address flexibility. Being difficult and error prone, derivation methodological support reduces significantly the possible paths of configuration and simplifies the overall process. We also presented the implementation realized in order to support this approach through the tool Sequoia for managing product lines. The viewpoint-based derivation approach for SysML models, was developed in the context of the MBSSE project (Renault, CEA, Université Paris 1), whereas the heuristics were developed by the CRI team, Univerité Paris 1, and they are based on their previous contributions in constraint programming for product lines [146].
While the current approach allows the specification of new assets, other open questions remain for the framework such as evolution, maintenance and tracking changes for family of system model.
On the constraints programming level, the purpose of the heuristics [8] was to improve the configuration process by : (i) reducing the number of configuration steps or (ii) reducing the computation time required by the solver to test the validity of the product line. The configuration of variables is done interactively by the user. Thus the order depends on their preference. By prioritizing choices, and recommending certain configuration variables before others, it is possible to improve these two aspects of the configuration process.
However, other questions may be raised: Which heuristics are better to improve the quality and pertinence of the solutions? Against what other criteria can we use to compare the collection of heuristics presented in this chapter? How to classify the heuristics according to their pertinence in certain situations? What kind of systematic recommendation should be presented to a user during a product line configuration process? How to apply and even combine the collection of heuristics presented in this chapter, in an interactive and incremental configuration process? In conclusion, the application of these heuristics on other product line models, specified with different formalisms can be implemented using the same principles, but may need further evaluation.

6

Validation of Main Hypothesis and Requirements

The purpose of this chapter is to confirm the initial set of requirements regarding the approach, the modeling concepts and their application, and understand properties like easiness and scalability. This is achieved by drawing on the system modeling examples presented in Appendix A.1.2. It is possible to distinguish between two aspects when observing the results – the approach and its tool support. This discusses the modeling examples and the EPB case study considered in this project, in sections 6.1 and 6.2, distinguishing between approach and tool as follows: (i) the SE variability management approach is discussed in respect to the initial requirements and (ii) the tool features are compared to the set of tool use cases introduced in Chapter 4. Finally, the results of expert interviews, performed in the context of the Product Lines Working Group of the AFIS[1], are presented in section 6.3.

While the examples covered our needs in this study, a few aspects can represent biases. Many of the vehicle systems developed today are more complex and contain more variability than the Electric Parking Brake case, which could potentially lead to scalability challenges. However, this is not a straightforward issue that could be addressed here, since there are multiple contributing factors for evaluating the approach on a large scale and in real life situations, like personnel training, experience of users or tool ergonomy. A second aspect is related to the industrial context at Renault. Although, the study focused on a general SE framework [58, 128, 129], the little exposure to other sectors or industrial settings can be an important threat to transferability. Hopefully opportunities for improvements and extended use will exist, but future exposure is of course linked to internal Renault decisions.

[1] Association Française d'Ingénierie Système

6.1 Requirement satisfaction

Our main concern while developing our approach was to satisfy the needs of Renault SE, which are presented in Chapter 3. Table 6.1 looks at these requirements and at some of the tool use cases explained in the aforementioned chapter and relates them to the examples which were relevant for their validation. Requirements are marked by their Id number from Table 3.2 in Chapter 3, followed by the "+" sign - if the example was representative for the illustration of that particular requirement, or by the "-" sign - if the requirement was insufficiently represented.

Examples	Requirements	Tool Use Cases
Example 1	1,2,3,4,5,7,8-,9,10,11,13,14,15	S1,S2, S6,S8
Example 2	1,3,5,9,10,11	S1,S6
Example 3	1,2,3,5,6+,9	S1,S6

Table 6.1: Variability Management Requirement satisfaction for each example

Most of the requirements are covered at least by the EPB system example. A few are worth a more detailed discussion.

> Requirement 8: The approach shall be applicable to multiple levels of system decomposition and abstraction.

In order for the approach to be scalable (not from a tool support point of view), it needs to be applicable on subsystems. In a top-down approach the following steps are taken in respect to variability to define the context for the subsystem:

1. Define the *diversity use case* for the subsystem: logical constraint which restricts the system configurations only to those which contain the system. This is currently generated automatically in the tool based on variants bound to the subsystem (component) and on variability propagation traceability.
2. Identify variants that may have in impact on the conception of the subsystem related to the system or environment characteristics. This task relies on domain expert knowledge.
3. Define the new diversity context for the subsystem based on the two previous points.

This approach has to be applied for each level of decomposition where a new system analysis is launched. However, none of the examples consisted in a large system with multiple levels of system analysis, such as a system of systems.
As for multiple levels of abstraction (e.g. requirements, functional, physical), the examples showed that the approach takes this aspect into account effectively, as long as the constraints defined manually, or generated through propagation are consistent.

Requirement 12: The approach must support reuse of SE models across (vehicle) projects for systems belonging to the same family (e.g. brake systems, lighting systems)

Development in the automotive industry is often based on incremental innovation, and previous solutions are constantly reused (carryover) and improved. At the time of the implementation of the proposed approach, projects at Renault are structured per vehicle family. This aspect is explained in Figure, 3.4, Section 3.2.2. However, depending on the difference between families of vehicles, their subsystems can be similar, or solutions reused. Separation of the development per vehicle family, reduces complexity of solutions, but raises the following problems in respect to reuse:

- how to achieve reuse in concurrent engineering projects, as explained in the description of engineering scenarios, Section 3.4
- how to achieve reuse in sequential engineering projects for different families of vehicles

Product lines solve some of these issues by relying on a common description for the family of systems. Still, incremental improvement means that the problem of evolution of the family of systems needs to be addressed. In consequence, we suggested a process, which takes into account capitalization, by updating the domain models. However, the details regarding the implementation of the information system enabling the evolution of domain models remains a subject for future work, as it is not currently implemented in the present solution. This issue also corresponds to the engineering scenario of "merging a product and a product family". The purpose of the identified scenarios was to identify industry needs and set the ground for future work, thus not all of them were addressed for now.

6.2 Examples overview & discussion

Although we are not able to reach statistically significant conclusions based on return on experience or experiments within the available time frame and due to organization constraints, case studies can also provide important lessons. In particular, we believe multiple examples, each with focused specific characteristics can be valuable for testing, but also reveal weaknesses. According to Easterbrook et al. [83], case studies can be used to test existing hypothesis but also for refuting theories.
We adopted concepts from OVM [163], and extended them to differentiate between design decisions, stakeholder & requirements variability and component alternatives. In a SE process, all these choices link problem expression to a particular solution, while generating variability that provides reuse opportunities. We also adopted the concept

of viewpoint in relation to variants, which is a valuable methodological support during systems derivation, as explained in Chapter 5. Furthermore, variant properties are closely related to the configuration process: their properties reflect the state (e.g. selected), but not the fact they might be mandatory (this property in our opinion contextual). Adjustable binding of variants to system elements reduces model complexity and avoids manual creation of constraints. By extending Sequoia's variability propagation support to the SE meta-model, it is possible to avoid manual binding of variants to all optional elements. Finally, compatibility to the Renault Documentary Language is essential for the integration of SE activities to the organization information system. The main threats to the validity of this conceptual framework concern the specificity of the context, which may not cover: adoption to large complex systems (System of Systems) and adoption to the development of systems outside the automotive Renault context.
Several aspects regarding variability management were covered by the three examples:

- Engineering scenarios : the phenomena representative in the example related to the engineering scenarios described in Section 3.4 and 3.5.
- System model variability : modeling activities for the family of systems architecture and requirements.
- Binding : modeling dependencies between variants and system model items.
- Orthogonal variability : managing variants and support for derivation activities.

Of all examples, only the Electric Parking Brake system addresses process and methodology issues by illustration of some engineering scenarios. The comparison between the three examples is presented in Table 6.2.

Variability Management Coverage	Example 1	Example 2	Example 3
Engineering scenarios	++	-	-
System model variability	++	++	+
Binding	-	-	++
Orthogonal variability	++	-	+

Table 6.2: Advantages of each example in respect to different aspects of variability management in MBSE

The second example, the Automatic Lighting System, contains few variants, but traceability in the system model allowed testing of rules for propagation of optional elements. Finally, in the third example the semantic of the binding had to be extended to avoid duplication of elements in the variant and system model (e.g. create a variant for each water tank design).

6.3 Critical analysis of models: interviews on variability management

Product Lines in the MBSE context are a set of models that share a set of common elements and structural characteristics (e.g. similar physical or functional decomposition). The purpose of the interviews was to confront the concepts in the variability model with industry expert opinions, and with the real life engineering scenarios they meet in their professional environment. This is why the questions were divided in two parts. The first part addresses variability modeling aspects, seeking to understand if there is a dependency between the different domains of application of product line engineering and the way variability is treated in the automotive industry. In consequence, if there are significant differences it would make sense to adopt, or customize a base variability language for each application (e.g. though UML profiles). In the opposite case, a single language is the best choice for all applications, with the added advantages of easy information exchange and knowledge sharing. The second part briefly explores issues related to engineering processes based on reuse. The purpose is to understand if some aspects observed at Renault exist elsewhere, and thus shed some light on the question of transferability.
The questions are mainly addressed to a limited number of professionals working in SE being confronted to the problem of reuse, and to product line experts. They are asked to analyze the models presented below and validate or evaluate some requirements/ assertions.
It is important to note that the questions do not cover in detail concepts related to variability modeling existing in some models (e.g. CVL [191, 192]), but are focused on a limited set of questions related to the elements presented in the current report. Undoubtedly, an exhaustive survey on the problem of variability would be very useful in order to: (i) properly asses the current level of product line engineering maturity in large companies developing systems (not limited to software) and in respect to different phases of product development, (ii) understand the most common used or needed concepts for variability modeling, in respect to standards (e.g.ISO/IEC 26550:2013 [117]) and academic proposals. However the purpose of the set of questions presented here was different.

Common example. The example accompanying the questions is inspired from the AFIS "Product Lines" working group. It is also presented in Section A.3 of Chapter A.1.2, where some modeling examples are presented. The central concept of this example is the water boiler product line with four proposed capacities : 100l, 150l, 200l, 300l. The boiler can be fixed vertically, horizontally, mounted on a wall, or vertically on the ground. For each position or capacity a different water tank design is needed. The following subsections present the questions and a summary of the answers for each of them.

6.3.1 Theme 1: Modeling

[**Optional elements**] Which way to represent optional system elements do you find more efficient (from a modeling perspective)?

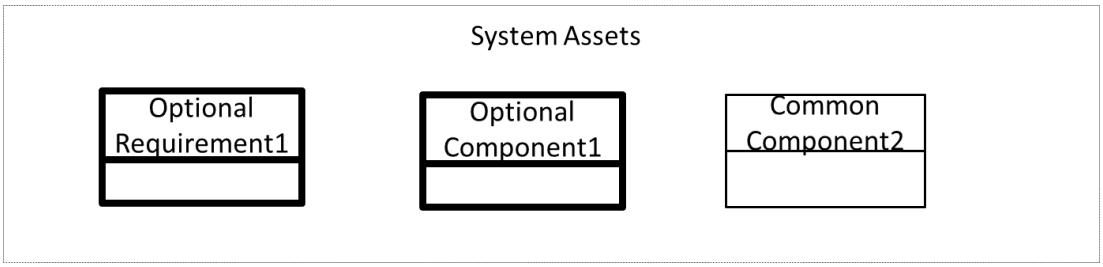


Figure 6.1: Optional elements, solution A

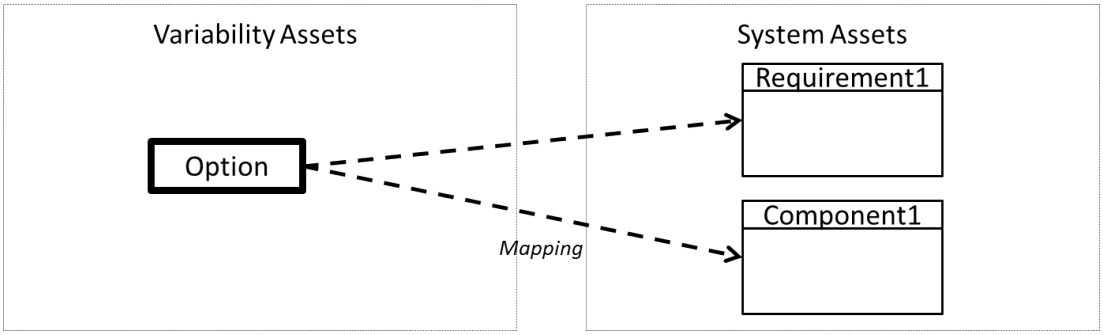


Figure 6.2: Optional elements, solution B

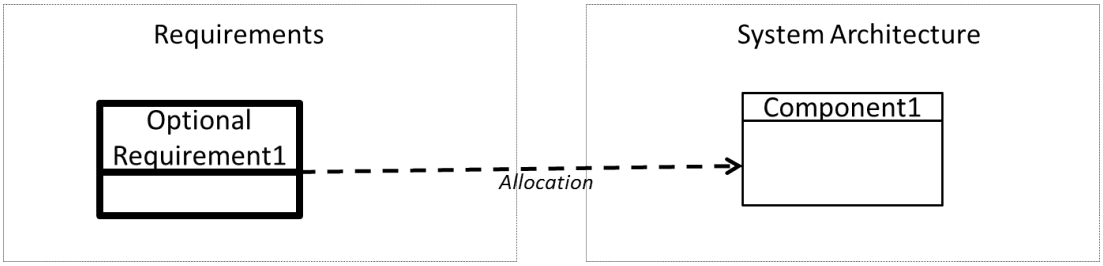


Figure 6.3: Optional elements, solution C

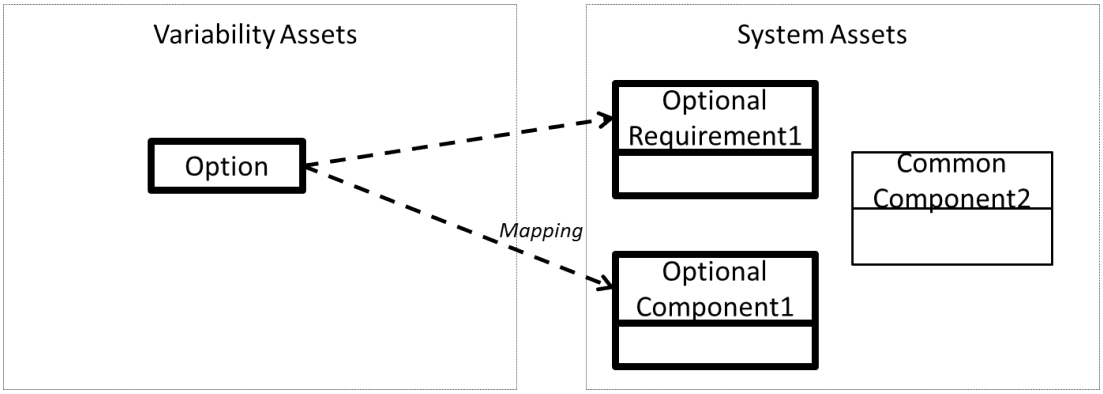


Figure 6.4: Optional elements, solution D

- A: Embedded (integrated) in the system model for all system elements.
- B: A separate variability model and mapping to system elements.
- C: Optional requirements allocated to system architecture elements.
- D: Both in a separate variability model and in the sytem model.
- Other :

A	B	C	D	Other
14%	14%	29%	0%	43%

Table 6.3: Optional elements question 1 answers summary

[Optional elements] Which way to represent optional system elements do you find more efficient? (from a software tool perspective)

1. A separate tool, dedicated for variability management.
2. In the system modeling tool.
3. In the requirements management tool.
4. Other :

1	2	3	Other
14%	43%	29%	14%

Table 6.4: Optional elements question 2 answers summary

[**Constraints and Dependencies**] Please analyze the following modeling alternatives for a Water Boiler system. Which modeling alternative would you adopt?

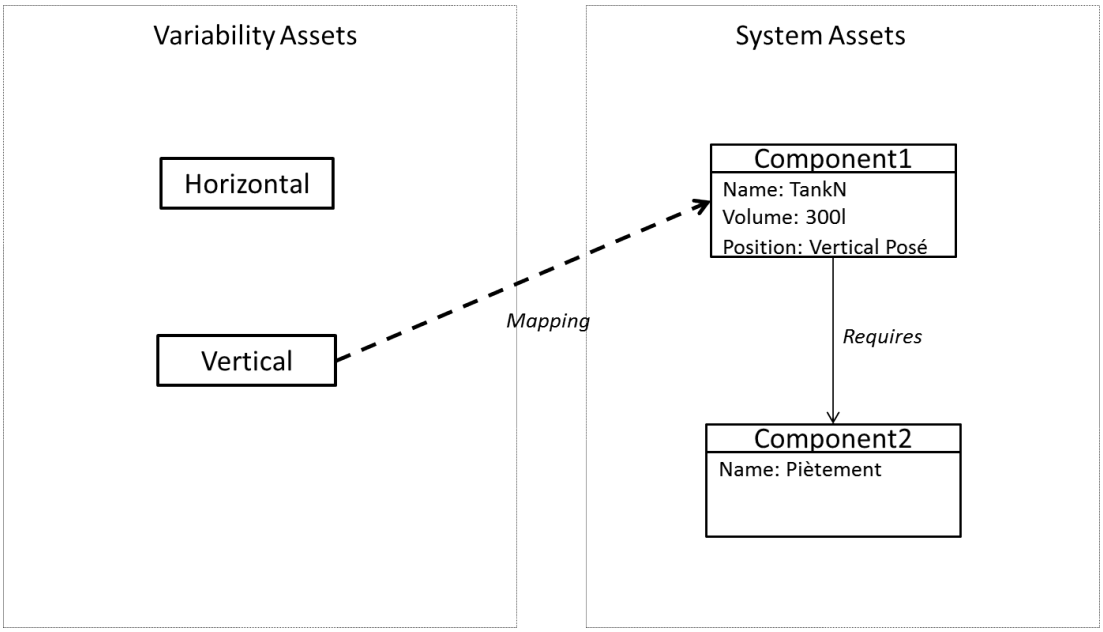


Figure 6.5: Constraints and dependencies, solution A

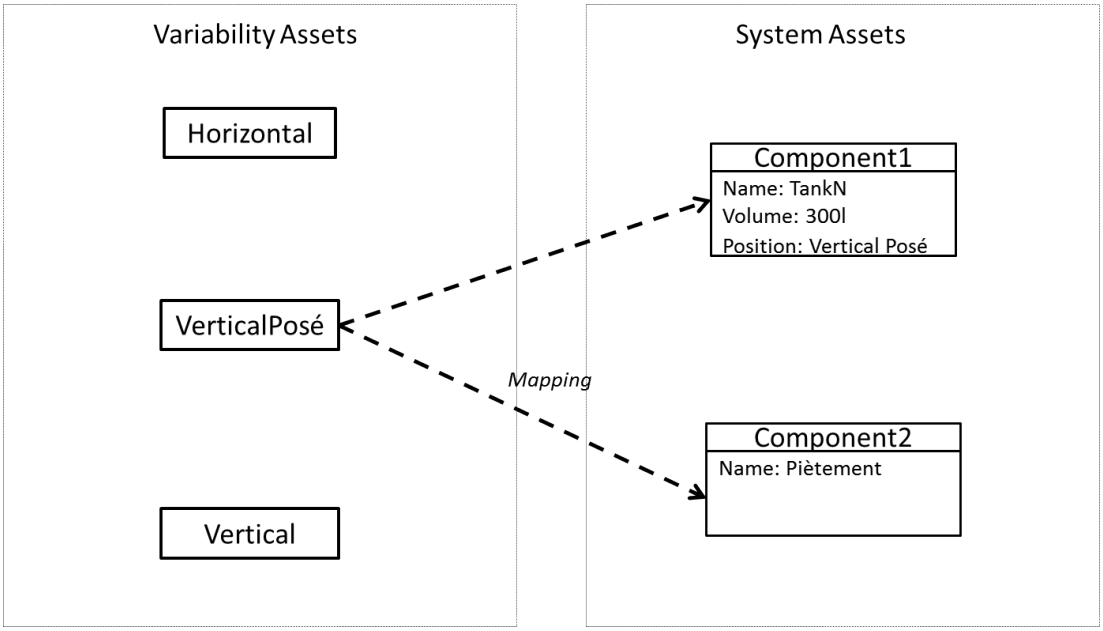


Figure 6.6: Constraints and dependencies, solution B

- A
- B
- Other :

A	B	Other
29%	43%	29%

Table 6.5: Constraints and dependencies answers summary

[**Mapping**] Please analyze the following modeling alternatives for a Water Boiler system. Which modeling alternative would you adopt?

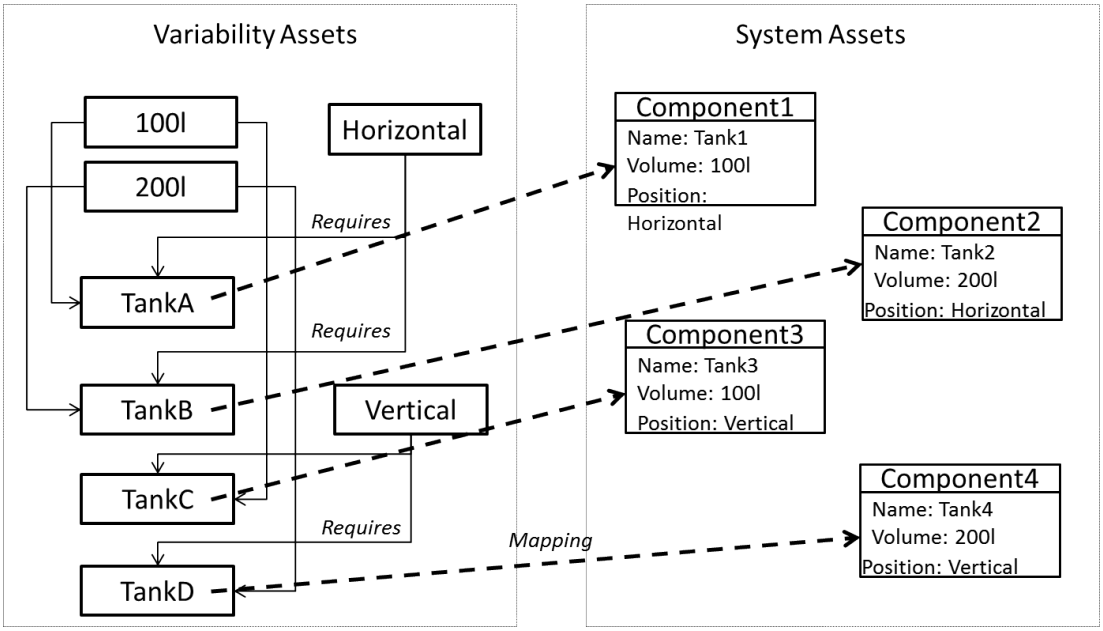


Figure 6.7: Mappings, solution A

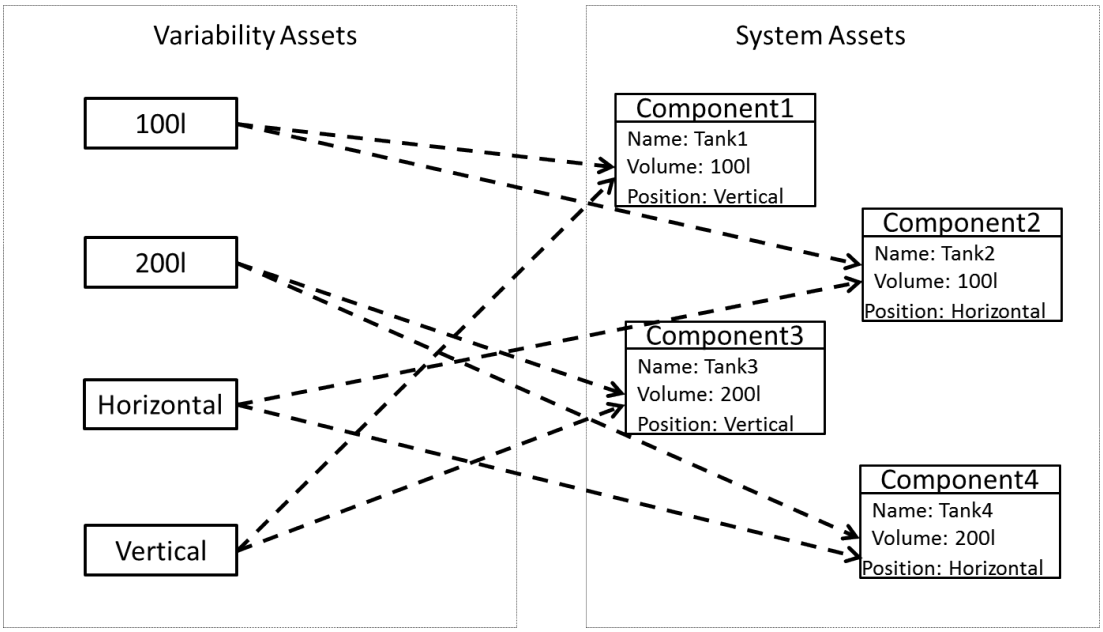


Figure 6.8: Mappings, solution B

Since each position and capacity needs a different tank design, for 100l and 200l we need 4 different tanks.

- A

- B

A	B	Other
29%	57%	14%

Table 6.6: Mappings answers summary

What do you think about an n to n mapping between the variability model and system model? (n variants or features to n system model elements)

1	2	3	4	5	Other
29%	0%	0%	29%	29%	14%

Table 6.7: Expressed interest for n to n mapping

[**Types of Variability**] What would you like to differentiate about variability (using different visual representations or different views)?[1]

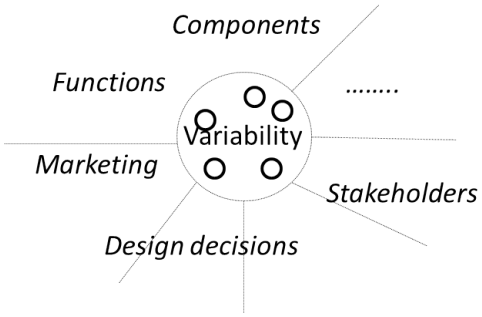


Figure 6.9: Viepoints on variability

1. Problem Space variability and Solution Space variability
2. Customer visible variability (External) variability and System (Internal) variability
3. Design decisions generating variety in the solution space
4. Allocation alternatives generating variety in the solution space (requirement or function allocation)

[1]Multiple answers possible

5. (Component) Supplier induced variety
6. Single view with no type differentiation among elements is sufficient or the preferable solution for me
7. Other :

1	2	3	4	5	6	7
58%	43%	29%	29%	29%	14%	14%

Table 6.8: Optional elements question 2 answers summary

6.3.2 Theme 2: Derivation Scenario

Consider the two scenarios given for the Boiler Example below. Regardless of the overall benefits, which one do you believe is more likely to happen in practice?

Scenario A

1. A market analysis reveals the need for Boiler capacities between 100 and 300 l, with sales estimates higher for lower capacity boilers.
2. An upfront investment is done to design architectures with volumes 100, 150, 200, 300.
3. Manufacturing capacity is adjusted on a quarterly basis given the sales and sales estimates per capacity.

Scenario B

1. A market analysis reveals the need for Boiler capacities between 100 and 300 l, with sales estimates higher for lower capacity boilers.
2. A limited upfront investment is done to design architectures with volumes 100 and 150l.
3. Sales and customer feedback provide information about the qualities and perception of the limited range of products.
4. Opportunities for extending the product range requires a second and third investment : existing designs are modified first for 200, and then for 300l

Figure 6.10: Engineering product line scenarios

- Scenario A
- Scenario B
- None

A	B	Other
14%	71%	14%

Table 6.9: Engineering scenarios answers summary

[Reuse vs. abstraction Level] Your products are usually:[1]

1. Products designed from scratch for each customer.
2. A new architecture design is needed per customer, but some constant requirements are reused.
3. A new physical architecture design, but at least some constant functions are reused.
4. Constant elements can be reused in requirements, functions and physical (or software) components.
5. The design is usually based on a physical (or software) product platform.
6. The system design is based on modular structures.
7. Other :

1	2	3	4	5	6	7
29%	0%	29%	29%	24%	29%	14%

Table 6.10: Reuse vs. abstraction level answers summary

[Reuse Method & Process] To reuse existing documents or models, which one of the proposed techniques or cases have you encountered in practice?[2] (check all that cases that apply)

1. Start with a generic document or model and customize it for a specific application
2. Start with a document or model of a developed system and adjust, add or remove elements for the new specific application.
3. Start with a partial model created through configuration, based on existing elements.

[1] Multiple answers possible
[2] Multiple answers possible

4. Always add new elements for each new application, as well as reuse parts of documents or models.
5. Manually copy existing reusable parts of documents or models from already developed systems.
6. Sometimes the new application model or document can be entirely created through configuration.
7. Other :

1	2	3	4	5	6	7
43%	43%	29%	14%	29%	14%	14%

Table 6.11: Reuse method process answers summary

Variability Impact assessment in architecture models (e.g. detect mappings to optional elements automatically, based on allocations). How do you find this functionality?

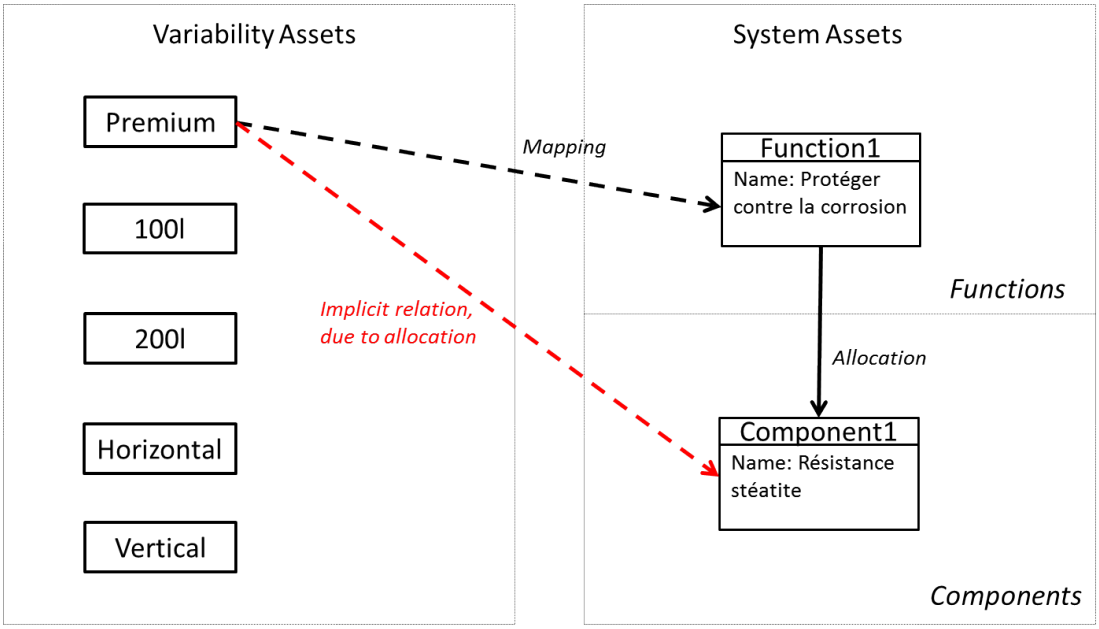


Figure 6.11: Variability impact in the system model

1	2	3	4	5
43%	0%	29%	14%	14%

Table 6.12: Expressed interest for variability impact assessment in the system model

The profile of the interviewed subjects is resumed as follows, in tables 6.13, 6.14, and 6.15.

Sector		**System Engineering**		**Product Lines**	
Academia	29%	Expert (Certified)	29%	1–3 years experience	71%
Defence & Aerospace industry	71%	Average & Basic	71%	≥4 years experience	29%

Table 6.13: Sector and level of expertise

UML		**SysML**		**Ontology**	
Expert	14%	Average	71%	Average	71%
Average or basic	86%	Basic	29%	Basic	29%
Feature Model		**Orthogonal Variability Model**		**Product Configurator**	
Expert or average	29%	Expert or average	29%	Expert or average	57%
Basic	71%	Basic	71%	Basic	43%

Table 6.14: Level of knowledge of various technologies, concepts

Product line engineering scope		**Product development activities[1] experience**	
Domain **OR** Application	57%	A single[2] product activity	43%
Domain **AND** Application	29%	Two[2] of the product activities	43%
Processes & tools	14%	All product activities	14%

Table 6.15: Experience in respect to engineering processes/activities

Although, in our opinion, the questions should be confronted with a much larger number of participants for any statistical significance, the answers presented above can be interpreted as follows.
Converging ideas:

C1 Multiple types of variability were recognized: external, internal, customers, but also supplier and design decision induced variability, which is also what was adopted in the proposed approach.

C2 Products almost always contain new elements. In consequence they are not created only through configuration, but are developed based on reuse of partial configurations.

C3 The idea that many practices exist, including ad-hoc reuse, feature-based, requirement driven etc.

Diverging ideas:

D1 A separate variability model is used or preferred, whereas in the proposed approach *forms of variability* characterize system elements, describing variability locally.

D2 Explicit mappings were preferred for each variable system item, over propagation of optional elements. However, in a variability rich model, a common automotive case, mapping all elements can become cumbersome.

Most of the participants come from a *Defence* or *Aerospace* industrial background, with few from the academic environment, but all posses average or advanced knowledge in product lines and UML/SysML modeling. While statements *C1, C2* and *C3* confirm the main themes of this project, the differences in statements *D1, D2* could be interpreted as a difference in needs, due to a different industrial sector, with less variability than in automotive. Furthermore, *C3* can represent the pretext for a variability model

[1] Analysis, Architecture, Design, Verification, Test, Implementation
[2] None of the these answers included Test and Verification

that is easy to customize and extend for the practices of each organization. Unfortunately, none of the participants comes from a different automotive manufacturer, which is one of the elements to address in future work.

6.4 Unresolved issues

While the implemented functionalities had the expected result, introducing variability in the system model remains a time consuming task for certain SysML diagrams. This is because binding for most elements has to be set manually. However this aspect can be improved with the addition of rules for the semantics of SysML for each diagram (e.g. remove combined fragments and their content at the same time, during derivation, for Sequence diagrams, with a single binding), while taking into account methodology aspects.
Validation of the variability constraints is supported by an external solver, which is configured with Sequoia. While this aspect is independent of modeling activities and outside the scope of this study, the examples reveal that computing the number of possible configurations and the derivation of products is also a time consuming activity. For the EPB system/ Example 1, we avoided computing the exact number of possible configurations ($> 10^4$), which took a significant amount of time. After the selection of variants for derivation, values are assigned to the corresponding boolean variables, and the number of possible configurations decreases.
Sequoia generates a "decision tree" before derivation, leading to all possible product configurations. We believe that for large SysML models, especially when no variants are selected, this can lead to very large, unmanageable "Sequoia decision trees". For example, when the maximum configured limit of configurations is exceeded, the tool suggests to select more variants.
However, because we rely on partial configurations at different stages of the SE process, as explained in Chapter 5, in large, variability rich SysML models (larger than the EPB example), selecting more variants might not be desirable. If the engineer wishes to generate a partial configuration from this large system family model, most of the variants are still left undefined (no value is assigned), possibly exceeding the maximum set limit of configurable products.

7 Conclusions

Systems Engineering is defined through a consensus of the INCOSE fellows as a "discipline whose responsibility is *creating and executing* an *interdisciplinary* process to ensure that the *customer and stakeholder's* needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle". This definition captures the essence of what SE is, as well as which aspects an effective management of variability should take into account: (i) problem solving approach (creating and executing), multiple viewpoints (interdisciplinary, stakeholder, customers), process phases (entire life cycle). While different ways to distinguish between variabilities were proposed (e.g. *internal* vs. *external* [163], problem variability vs. solution variability [72]), one of the main themes of this project is that many types of variability exist, based on the aforementioned aspects. For instance, one of the overlooked types of variability that should be distinguished in a problem solving approach, such as SE, are design decisions. They are distinct from customer visible variability, which in large organizations such as Renault is defined outside the scope of SE itself, but they do generate variety, indirectly (e.g. alternative allocations of functions on hardware, system design concepts). Furthermore, MBSE introduces multiple levels of abstraction that bridge problem and solution spaces. In this case, mapping variability from problem to solution space is not straightforward. Instead, variability needs to be gradually refined during the system analysis along with the other family of systems artifacts.

The aim of this project was to introduce an approach to manage variability in MBSE, integrated to the system models and consistent with the context at Renault. At Renault, commercial variability, defined outside the scope of SE, is mapped directly to vehicle components for configuration during manufacturing. In consequence, variability not captured in engineering design and development processes is not systematically

managed. This was achieved by adopting concepts from an existing variability model – OVM [163] – and adapting it to needs identified and defined through : (i) a set of engineering scenarios for system families; (ii) a set use cases for a SE tool, and (iii) a list of requirements to be satisfied by the SE variability management approach.
The OVM model presents in our opinion some advantages for easy adoption in SE, and at Renault:

- *System model compatibility* : it captures variability, but there is no redundancy in respect to other system concepts (e.g. hierarchies)
- *Method/Process compatibility* : while OVM can represent variability across all engineering items, there are no built-in methodological constraints related to how it should be configured and or created.
- *Simplicity* : it can be easily extended with other concepts (e.g types of variability) and embedded into general purpose languages (e.g. UML profiles)
- *Renault documentary language compatibility* : it has a very similar structure to the representation used at Renault for defining customer oriented variability.

7.1 Results and contribution

The results were applied, with the support of the tools developed by the CEA LIST[1], to case studies, revealing their utility.
First, the key issues and needs of the organization in respect to variability management were identified. The contributions address a part of these needs, also taking into account their priority:

- *Co-OVM* – a variant management metamodel for SysML models;
- *Viewpoint orietend product line derivation* – methodological support for configuration of family of system models;
- A set of heuristics for improving product line configuration processes;
- Identification of use cases for a tool for system variant management in the context of Renault engineering processes .

From the perspective of complex systems many subjects related to product lines deserve to be explored further. Indeed, introducing a spatial dimension in the management of complex systems opens up many new questions touching all aspects of SE, from requirements to design and validation. How can configuration of individual choices be complemented with numerical simulation and system optimization to optimize individual derived products? How can the impact of a single choice be assessed without the

[1] http://www-list.cea.fr/en

evaluation of complete configurations? How to best match design decisions in respect to context and component variability?...

7.2 A perspective on variability and system "ilities"

One way to interpret and explore issues related to system variability can be done in respect to the system "ilities". The following are in direct relation with the way variability is introduced and managed during the system's life-cycle:

- Flexibility - the ability of the system to support certain changes with relative ease is supported by documenting decisions that lead to different designs or coverage of stakeholder requirements. It reflects an ease in changing the system requirements and choosing alternative designs in order for the system to achieve a different set of functions, implemented in possibly different ways.
- Adaptability - according to Bencomo et al. [50] variability modeling can be applied to define how systems adapt at runtime. Variability modeling may be used as a way of managing the variety of ways in which a system can change internally to fit the occurring changes in the environment.
- Evolvability - can be defined as system flexibility over time. Variability management needs to take into account problems related to evolution of system assets and ensure consistency between models for changes in time of system families.
- Agility - represents "the ability of a system to be both flexible and" evolvable [212]
- Modularity - reuse of patterns of the system definition, with functional meaning, across a family of systems, which represents the very core purpose of product lines - achieving reuse on every level. However, traditionally modularity is strictly related to reuse of independent components. Identifying a potential reuse of different system assets, DeWeck et al. [212] define "modularity 2" as "the presence of functionally meaningful repetitive patterns or modules within a larger system"

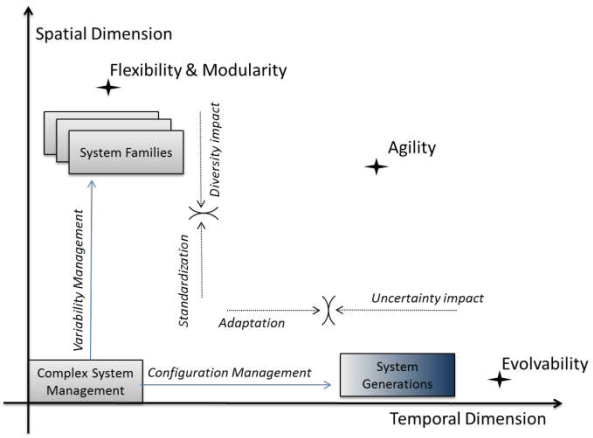


Figure 7.1: Family of systems dimensions and related system "ilities"

7.3 Perspectives

Product lines provide valuable theoretical foundations for managing variability in organizations and migrating towards a product line practice may be done for several reasons: overall cost reduction [214], market demand and competition, introducing flexibility for the design of new products. We were confronted with two types of challenges.

7.3.1 On variability modeling

The first of the challenges concerns the modeling activities, in particular the need to express more complex constraints which are often present in the definition of the vehicle range (e.g. $A \Rightarrow (B \vee C \vee D \vee \dots)$; $(A \wedge B \wedge C \wedge \dots) \Rightarrow D$; where variants A, B, C, D ... belong to different variation points). Graphical languages have the advantage of being intuitive for capturing and sharing information, but in large variability models, the management of complex constraints and dependencies can become difficult.
In large family models, users need to specify bindings for many inter-related artifacts. A better support for the assessment of the impact of variability, coupled with a well-documented system model, could support propagation of optional elements from requirements down to the physical components (through dependencies, allocation, traceability etc.). Many types of relations in SysML models are not yet taken into account in the variability propagation mechanism. One particular example are the sequence diagrams, where we need to manually specify "optional" elements contained inside an "optional" Combined Fragment.

7.3.2 On the adoption of product lines and organizational change

From a methodological point of view the challenge was reconciliate product line practices with the existing "diversity" culture already present in the organization: our purpose was to extend variability management to MBSE in order to reuse specifications, but also for early specification of vehicle component configurations. We believe this scenario was different from the typical "migration to product lines" (or adoption scenario), and we are faced with some challenges specific to large organizations, presented below. Among these, the last two also apply to MBSE practices in general, as well as product lines:

- Putting product lines into context, harmonizing SE practices with existing variability management related activities;
- Overcome "cultural" barriers to change. On the one hand, adopting new practices does not always allow preservation of previous organizational structures or activities, as responsibilities and tasks may overlap. On the other hand, it may require a shift in the practices of employees. This is sometimes met with a "change resistance" attitude. One solution can consist in courses on product lines, available for employees to ease the adoption of new concepts.
- Overcome "visibility" barriers to change. Rather than an organization culture/habit, visibility on the company activities is different for each employee, which may lead to a lack of understanding of the global strategies and their rationale.

7.3.3 On product line derivation and configuration complexity

With the introduction of our approach, we also increased the level of detail of documented variability. One of the most challenging problems encountered, when variability was documented down to a very detailed level, was inevitably the complexity of resulting configurations. One of the frequent constraints that appears in online configurators is a fast response time. Besides, configuring a system with different types of assets - requirements, decisions, supplier selection, etc. - the engineer is confronted with an astounding number of options.
The first step in ensuring conformance to the SE project, and reducing the complexity of the configuration task, was to adopt a "derivation methodology", as explained in Chapter 5. However, it is possible to further improve the support for derivation, in respect to process, system or user objectives, as suggested in Figure 7.2.

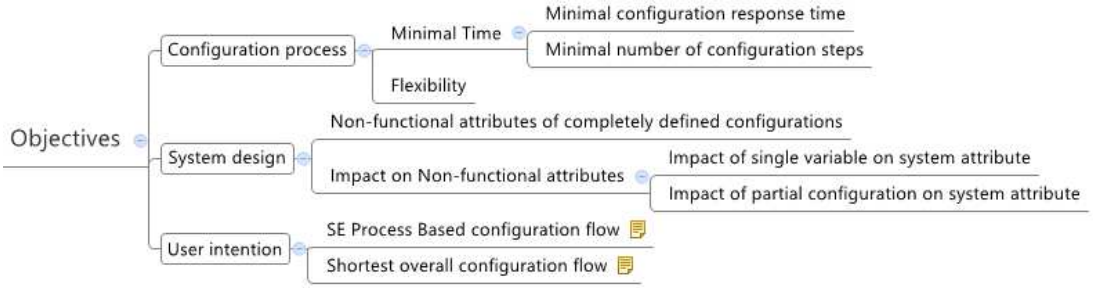


Figure 7.2: Objectives for system derivation improvement

Mazo et al. [8] proposed the adoption of heuristics based recommendation to improve the configuration process in respect to solver computation (constraint propagation) time and the number of necessary steps to reach a complete valid configuration. Relying on the methods proposed by Doufene et al. [80], we intend address the problem of determining choices that optimize different system properties, in future work.

7.4 Final words

In order to successfully adopt SE in mass customization industries, it is imperative that this includes activities for variability management. Product Line Engineering has delivered promising results in different industries, in the past years. However the path to adoption is not short: both methods and tools need to be tailored to the industry and organization context. This is especially true in SE where many different frameworks have been proposed or adopted. While the amount of adaptation for each setting can be different, this document can provide the main "ingredients" required for capturing variability in Systems Engineering.
Product Line Engineering has also become a subject of focus for the INCOSE [113] and for its French chapter AFIS (Association Fraçaise d'Ingénierie Système), with an active working group on product lines.
Systems Engineering models represent an efficient tool for tackling the complexity of large systems and organizations. However, complementing MBSE with variability modeling techniques becomes essential for dealing effectively with reuse, in any large organization which develops customized products.

Glossary

architecture framework A description of what it means to define and document the system of an architecture, its constituent parts, and how the parts fit together to fulfill a global purpose. x, 54, 85, 87, 110, 149

BOM Bill of Material. 34, 36, 149

carry-across A reuse strategy, present in the automotive industry, that consists in the transfer of a solution intended for a single or subset of products to a larger vehicle range; enlarging the scope of a product solution to a whole range of products. 14, 64, 101, 149

carry-over A reuse strategy present in the automotive industry, based on the transfer of solutions and knowledge from previous products and theior application or adaptation to new vehicle projects. 14, 64, 101, 149

CMMI Capability Maturity Model Integration. 149

completely defined product The requirements, design, specification of a system that cover a specific set of variants of a system. 22, 57, 106, 124, 149, 189, 192

configuration The requirements, design, specification of a system that cover a specific set of variants of a system. 149

configuration process The process of assignment of values to the variables describing the variability of a product line; the process of selection and rejection of variants (when boolean variables are used) of a product line with the purpose of obtaining a single product. 5, 18, 22, 25, 35, 40, 118, 119, 121–125, 129, 148, 149, 196–198, 201–203

CVL Common Variability Language. 28, 29, 36, 149

design decision A design decision refers to a choice taken during the conception and development of a system, that has a tangible impact on the technical solution

and provides the best trade-off for a particular context and set of stakeholder requirements, in respect to the system effectiveness (how well the system performs its mission).. 11, 12, 18, 27, 37, 41, 66, 70, 87, 100, 112, 128, 136, 143, 145, 149, 156, 174–176

diversity Synonym of variety at Renault. Represents the absence of uniformity, sameness, or monotony of a collection of existing, completely defined objects or products.(e.g. a vehicle range, components with similar features). 2, 3, 6, 8, 14, 22, 49, 64, 149

Documentary Language Language used at Renault to represent the variability of the vehicle range, for the purpose of manufacturing customized vehicles, based on individual customer orders defined in terms of vehicle features. xi, xiii, 9, 13, 16, 34, 56, 57, 59, 61, 76, 80, 82–84, 86–90, 96, 110, 117, 129, 149, 171, 172, 174, 178, 179, 222, 224, 227

domain A concept in software reuse, that refers to an area that promotes reuse by the description of commonalities and the adoption of product line practices. 2, 12, 18, 29, 31, 34, 67, 103, 104, 120, 141, 149

domain analysis An activity, part of the domain engineering processes, that defines the domain, collects information about the domain, and produces a domain model.[206]. 23, 149

ECE The United Nations Economic Commission for Europe. 149

ECU Electronic Control Unit. 149

EPB Electric Parking Brake. 149

feature A product or system characteristic reflected by the system stakeholder needs and requirements. 2, 4, 6, 17, 22, 75, 81–84, 95, 149

heuristic "A technique or value derived from experience, experimentation, or intuition from which a problem can be solved with no expectation of optimality."[167]. 5, 106, 108, 118–121, 125, 144, 148, 149, 196–198, 200–203

heuristics-based recommendation definition of heuristic based recommendation. xi, 118, 120, 149

KPI Key Performance Indicator. 149

MBSE Model Based Systems Engineering. 2, 3, 5–7, 10, 18, 19, 24, 26, 33, 34, 38, 46, 52, 73, 75, 76, 82, 88, 104, 125, 143, 147–149, 187

MBSSE Model Based Software and Systems Engineering. 149

MDE Model Driven Engineering. 36, 42, 149

MOE Measures of Effectiveness. 27, 35, 149

MSR Manufacturer Supplier Relationship. 149

OCL Object Constraint Language. 33, 37, 149

OEM Original Equipment Manufacturer. 149

OMG Object Management Group. 28, 149

optional artifact An artifact of the system definition that may not be present in all product configurations, depending on the final specification of product line constraints, and customer variability. 38, 149

OVM Orthogonal Variability Model. 24–26, 31–33, 41, 43, 149

partial configuration A partial description of a system form in terms of requirements, design, specification and realization, that covers a subset of the available variants in a family of systems, such that configuration choices can further be performed by different actors within the system life cycle. (e.g. in systems engineering reuse can be achieved partially in respect to specific concerns; in manufacturing choices like the selection of a specific component supplier can be delayed until all context details are known). vii, xi, 23, 40, 90, 94, 104, 110, 115, 118, 141, 142, 149, 178, 198–201, 203

PLE Product Line Engineering Engineering. 36, 149

recommendation "An information item estimated to be valuable in a given context."[167]. 106–108, 118, 120, 125, 149

SE Systems Engineering. x, xi, 1, 2, 5–12, 14, 15, 17–24, 26–29, 33, 35, 38, 39, 41, 43–47, 51–55, 59–67, 69–71, 73, 76, 80–82, 84, 85, 91, 97, 104–110, 112, 115–118, 126–130, 142–144, 147–149, 155

SE artifact A tangible work product created, used or reused during the development of a system or of a family of systems. (e.g. requirements, models, architecture descriptions). 2, 38, 66, 92, 124, 149

SoSE System of Systems Engineering. 149

SPL Software Product Lines. 1, 11, 39, 52, 77, 149

SysML A general purpose language for systems engineering, that covers the specification, analysis, design, verification and validation of systems. 4, 5, 14, 15, 26, 29, 35, 46, 52, 55, 70, 73, 75, 149

system complexity The characteristic of a system that renders it difficult to describe, understand, predict, manage and change. [134] While there are many types of system complexity, in the context of a family of systems it refers to the number of possible products that can be obtained from a family of systems definition through configuration (e.g. before any new artifacts are added). 105, 149

traceability The potential for traces to be established and used. The ability to follow an artifact or a collection of artifacts thorough their lifecycle. In the context of variants, described here, it establishes relationships to the entity that issued the creation of the variation point, and may trace the system lifecycle phases where the variant is applicable. (e.g. conception, development, manufacturing). 4, 18, 45, 66, 73, 82, 88, 127, 129, 146, 149

UML Unified Modeling Language. 24, 29–31, 33, 149

vehicle platform A shared set of common solution elements shared among different automobile models, which usually includes at least the chassis, and the placement and choice of engines. 3, 12, 40, 62, 149

view A view is a representation of a whole system from the perspective of a related set of concerns.[116]. 33, 53, 137, 149

viewpoint A systems engineering concept that describes the system with a restriction to a particular set of concerns, which enables particular issues to be treated separately. These concerns are typically related to the process, organization, specific areas of expertise. According to the OMG [11, 193]: "A viewpoint is a specification of the conventions and rules for constructing and using a view for the purpose of addressing a set of stakeholder concerns". x, 18, 33, 46, 52–55, 71, 80, 84, 85, 89, 93, 97, 101, 110, 111, 115, 129, 143, 149, 155, 174, 175, 188, 222, 223

VM variability model. 149

VMT variability modeling technique. 149

Appendices

A

Systems Modeling Examples

According to Easterbrook et al [83], case studies and real industry examples provide an in-depth mean of observation for understanding how certain phenomena occur, according to and can be used to test hypothesis. This chapter presents three real life examples, which served as confirmation for the satisfaction of the requirements described in Table 3.2, that the approach has to satisfy.
The concepts described Chapter 4 are implemented in the framework developed in the context of the project "Model Based Software and Systems Engineering", involving Renault and the CEA LIST[1]. The modeling environment is based on the SysML/UML modeler Papyrus[2] with additional functionalities for the support of the systems engineering methodology at Renault. The purpose of this chapter is to validate these concepts by case study or real examples from the industry. The main objective is to provide evidence of an improvement in respect to previous tools at Renault for capturing requirements and system architecture design. This mean of validation was chosen due to the opportunities in respect to industry examples, but also in respect to con-

[1] http://www-list.cea.fr/
[2] http://www.eclipse.org/papyrus/

textual constraints. While this points out the applicability in the context of Renault systems engineering, validity threats exist: lack coverage of some implicit domain concepts, non-obvious meta-model contradictions.
The description of each example or case is introduced through a general presentation of the system family, of the variability and related constraints. After examples of products, the each example ends with a discussion in respect to the relevant requirements that were covered.
Two of the examples represent real industry cases from Renault, which were chosen due to their reduced complexity and relevant content: the Electric Parking Brake (EPB) system and the Automatic Lighting System (SAFE[1]). The third example is inspired form the AFIS[2] working group for product lines - the water heating system. While the last of these is examples does not come from the automotive industry, its rich variations in customer offer in relation to components were more than suitable for testing some of the functionalities of the tool support.

A.1 Example 1: Electric Parking Brake System

The Electric Parking Brake is a recent variation of the classic handbrake, which offers improved functionality, allowing for more space in the vehicle and comfort of use. This type of system was first installed on the Renault Vel Satis in 2001, but today is widely used on vehicles.
The system served both as an exploratory case study and also to confirm some of the initial requirements. In a first we developed models with annotations for variability, for each systems analysis viewpoints : operational, functional and structural, as described by [128], [58]. These models, described in Appendix C, allowed for a better comprehension of the case study, as well as the identification of some phenomena related both to the SE process and the modeling techniques.

A.1.1 System description

The purpose of the EPB is to keep the vehicle immobilized when parked, but it also implements some complementary (opportunistic) functions, given its main function and hardware support. It supports the following features:

- Engage/release on driver request by activation of the manual control in any situation
- Release automatically during drive away
- Engage automatically when the engine is stopped or the driver leaves the vehicle

[1] Système d'allumage et d'extinction Automatique des Feux Extérieurs
[2] Association Française d'Ingénierie Système

- Braking force is applied in relation to the context: vehicle speed, temperature, brake effort, wheel motion etc.
- Anti-rolling (hill start) assistance during drive-away.

There are two variations available. One is the "puller cable" type, where an electric motor pulls the emergency brake instead of performing the action manually with a lever. The second is implemented with two electric actuators attached to the brake calipers of the rear wheels.
However, different implementations are possible, for example by taking advantage of the hydraulic brake and available sensors in the vehicle. Because one of the responsibilities of the system engineer is to choose among different possible architectures, in the present case study we considered several design alternatives, which are not necessarily implemented to this moment. The main design alternatives are enumerated in Appendix A, and they were considered for the SysML variant models for configuration and design decisions of the EPB family of systems.

A.1.2 Examples from the architecture model

This section presents some examples from the architecture model of the Electric Parking Brake system for each of the system analysis phases: operational, functional and constructional analysis. The SysML diagrams associated to each viewpoint follow the methodology proposed by Chalé et al. [98]. A few of the artifacts are characterized by different forms of variability – optional, replaceable and numeric parameters. However, because at this stage the modeler is in a prototype form, some of the its features are experimental or need manual intervention (e.g. setting colors for some stereotyped elements, setting some stereotypes). This is also due to the fact that a few functionalities were added after the MBSSE project, in particular the creation of custom constraints based on the Sequoia framework and extended binding of variants to system elements, which proved to be necessary for modeling the EPB system. The propagation of optional elements is well integrated, since it is based on an existing API of Sequioa and existing user interface menus. Some of the ideas remain to be addressed in future work, as they are not yet implemented in the current version (e.g. visualization of system elements containing variability). The current tool prototype provides the necessary support for system modeling and validates the concepts related to variability at Renault.
The following figures (A.1, A.2, A.3, A.4) present requirement diagrams of the EPB system.

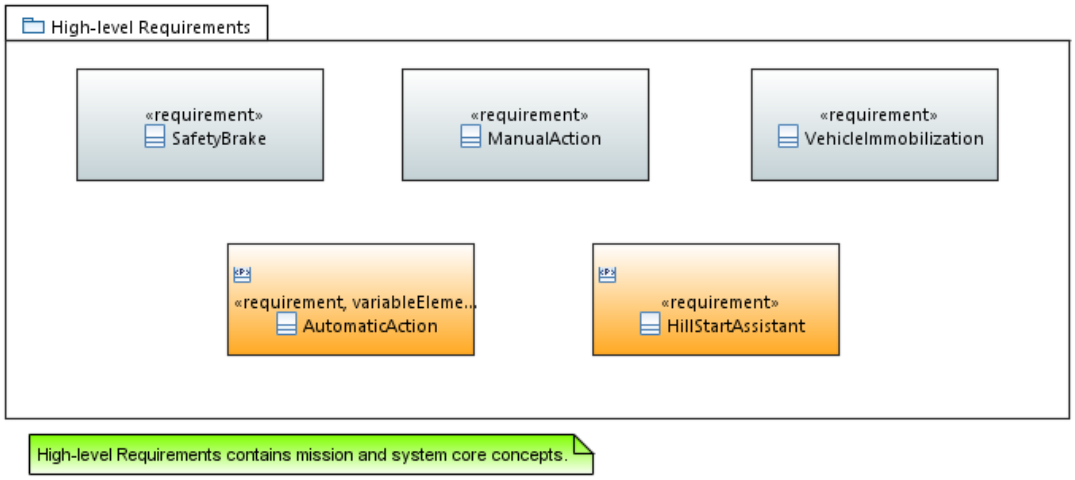


Figure A.1: High level requirements : system mission and core concepts

High level requirements include the *system mission* and the *system core concepts*. Requirements named *AutomaticAction* and *HillStartAssistant* are optional and bound to corresponding variants. They depend on the final vehicle configuration and customer preferences.

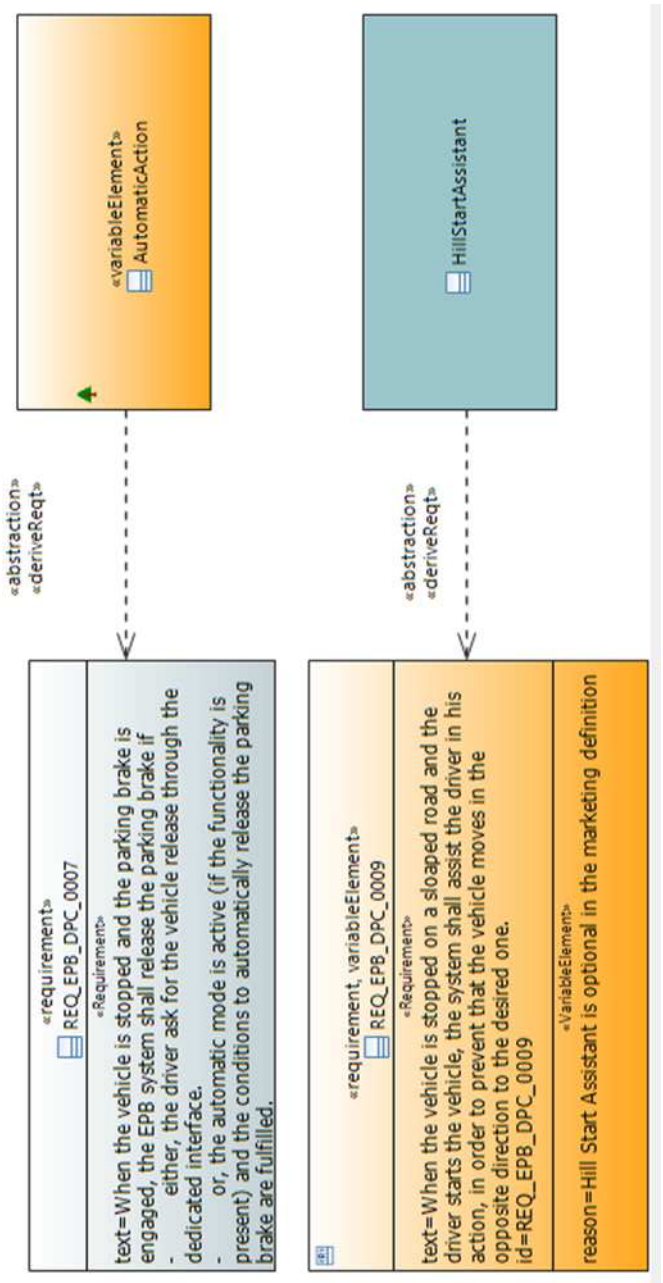


Figure A.2: Variable customer requirements leader (stakeholder) requirements

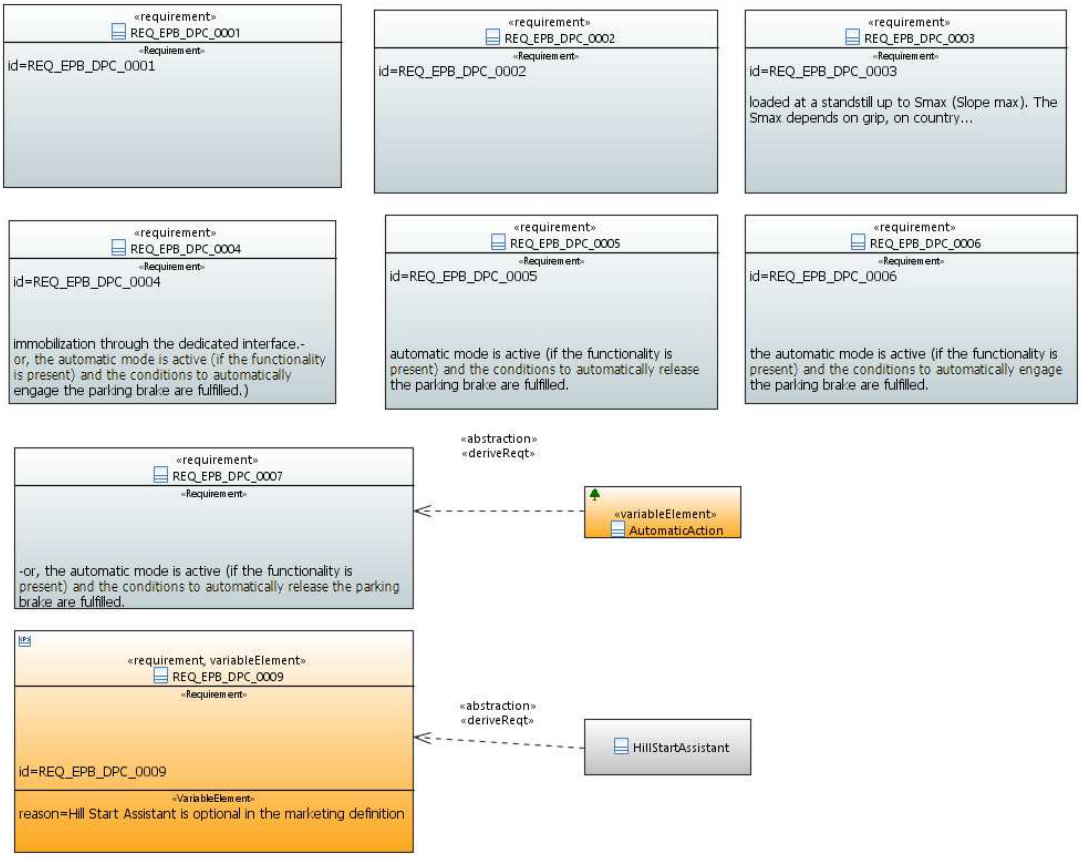


Figure A.3: Customer requirements leader (stakeholder) requirements diagram

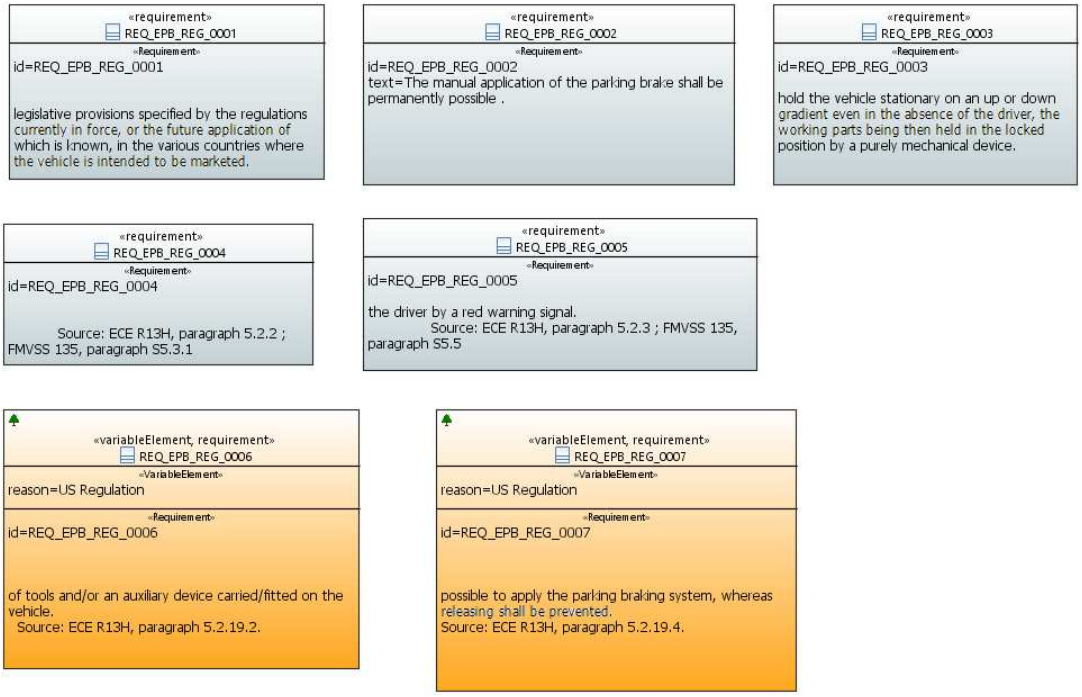


Figure A.4: Standards and regulation (stakeholder) requirements

REQ-EPB-DPC-0009: When the vehicle is stopped on a sloaped road and the driver starts the vehicle, the system shall assist the driver in his action, in order to prevent that the vehicle moves in the opposite direction to the desired one. [Source: *DPC*[1]]
REQ-EPB-SYS-0008: The system shall assist the driver (HSA functionality) in putting the vehicle in motion on a sloped road, by gradually and automatically releasing the parking brake at the right time (in correlation to the state of the vehicle). [Derived from: REQ-EPB-DPC-0009]
REQ-EPB-SGP-0003 The system shall either self adjust its behavior to be able to fullfil the hill start assistance operation when the vehicle has a towed trailer, or it will allow deactivation of the hill start assistant functionality by the driver. [Source: SGP[2]]
REQ-EPB-SYS-0013: The system shall allow the user to inhibit/deactivate the hill start assistant functionality by maintaining the parking brake command during the maneuver. [Derived from *REQ-EPB-SGP-0003*]
REQ-EPB-SYS-0014: The system shall be able to recalculate the total weight of the vehicle and trailer and adjust the performance during the hill start assistance operation. [Derived from *REQ-EPB-SGP-0003*]

[1]Direction Prestation Clients/Customer Requirements Leader
[2]General Product Safety

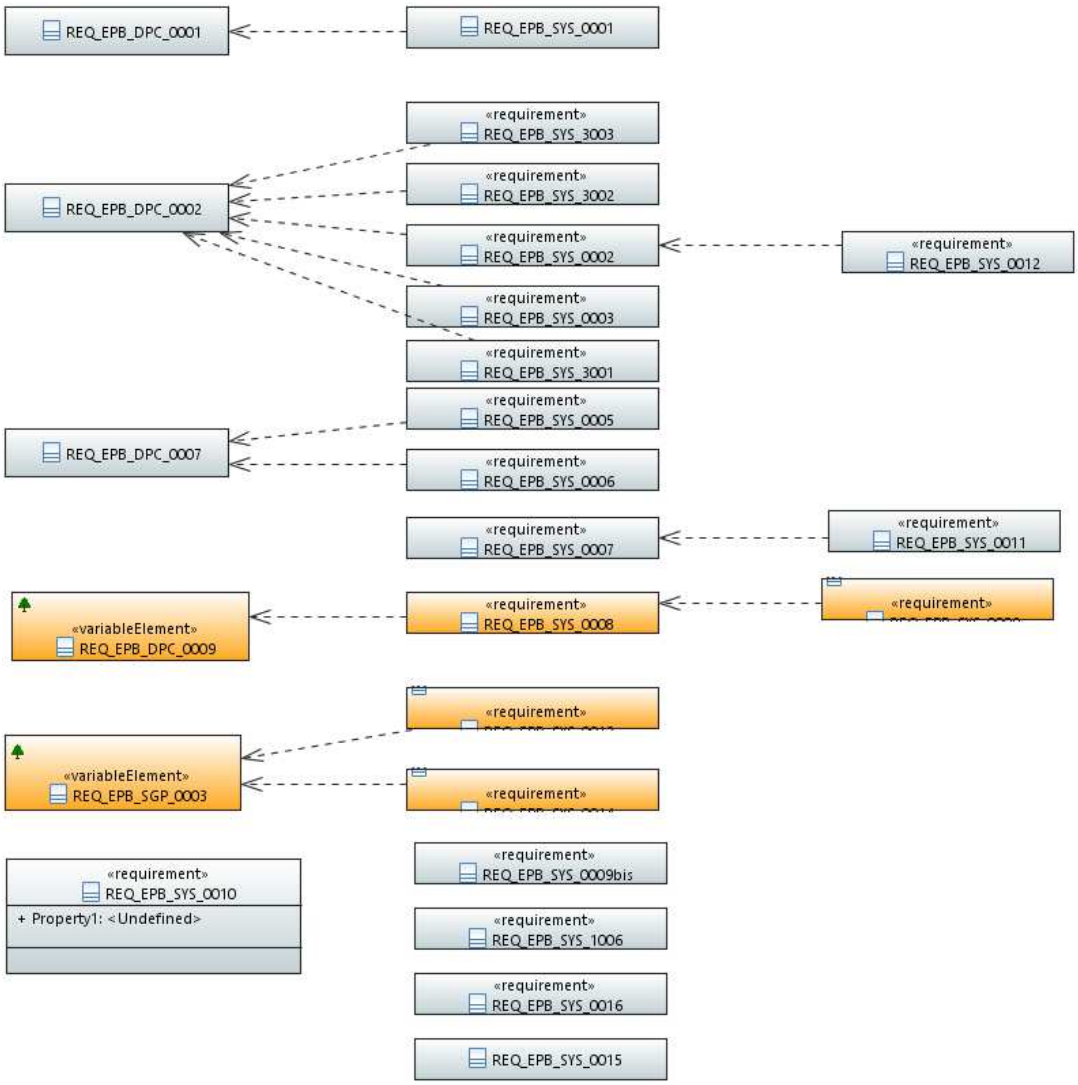


Figure A.5: Requirement derivation and propagation of optional requirements

REQ-EPB-SYS-4010: The system shall be compatible with the environment of the vehicle and the natural environment corresponding to the country where the vehicle is to be commercialized.

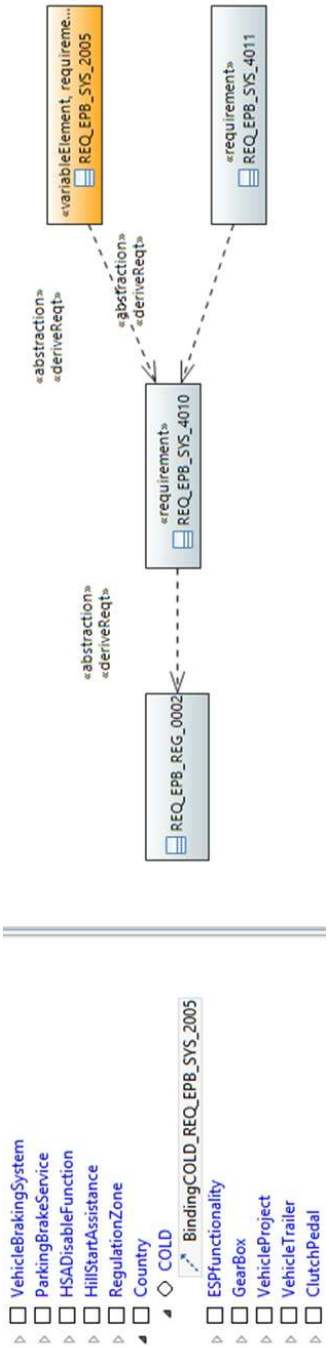


Figure A.6: Requirement derivation and optional technical requirement

REQ-EPB-SYS-2005: The system shall allow cancellation (inhibition) of the auto-

matic brake order application on engine stop, in factory. (for cars that are to be sold in Cold countries) [Derived from *REQ-EPB-SYS-4010*].

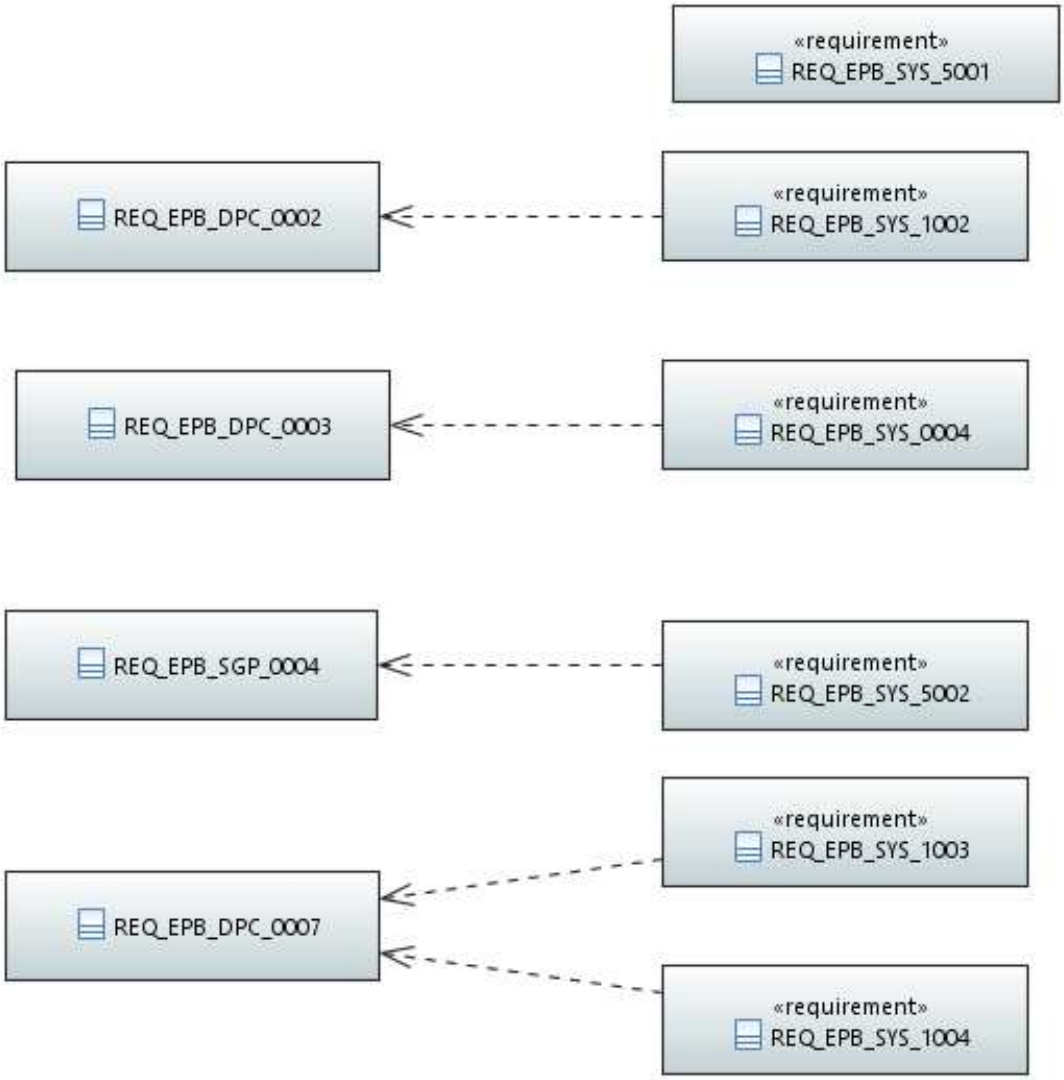


Figure A.7: Common (non-variable) requirements of the EPB system

REQ-EPB-DPC-0004: When the vehicle is parked and the parking brake is released, the EPB system shall engage the parking brake if

- either, the driver ask for the vehicle immobilization through the dedicated interface.

- or, the automatic mode is active (if the functionality is present) and the conditions to automatically engage the parking brake are fulfilled. [Source: DPC]

REQ-EPB-SYS-5002: The system shall generate a residual force when the brake is not applied, during the running phase (vehicle is in movement) that is ¡ CR.

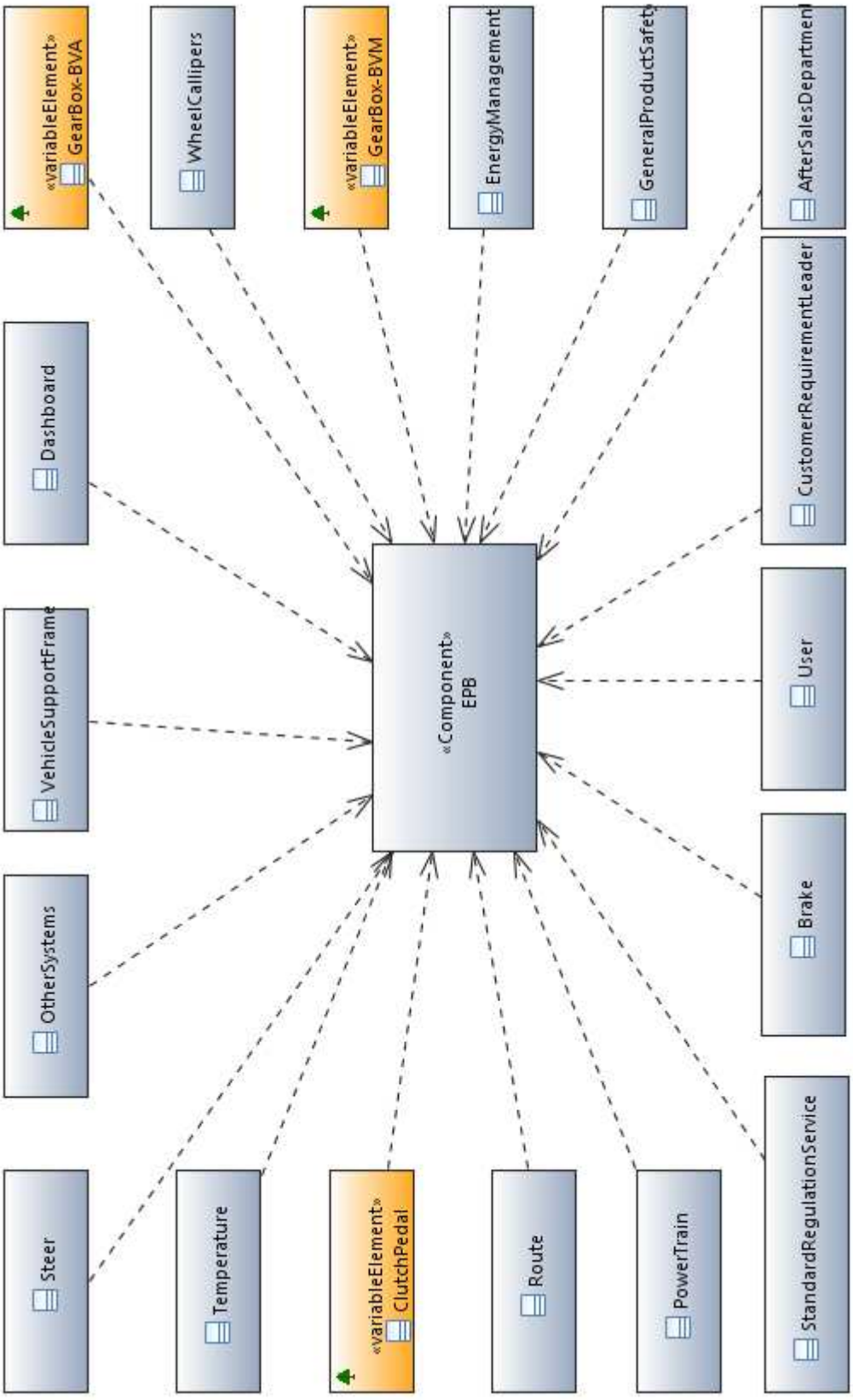


Figure A.8: EPB system context

Variability is also present in the interactions with external actors - the user or other systems. These are studied during the operational analysis and represented as sequence diagrams, among other representations [98]. Figure A.9 presents a sequence diagram with an optional UML Combined Fragment and two activities.

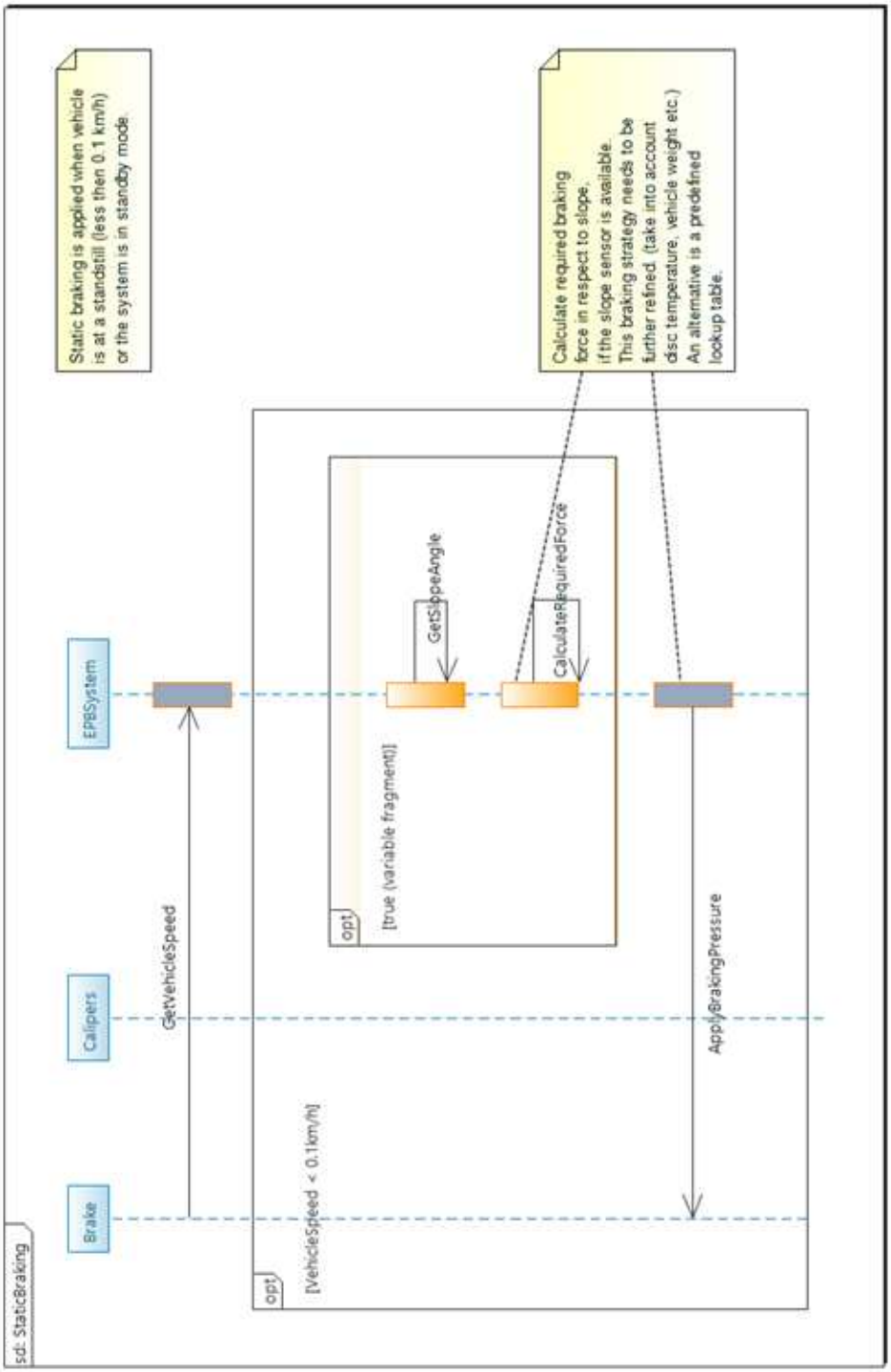


Figure A.9: Static braking strategy sequence diagram, with optional Combined Fragments and Activities

The optional activities are related to the presence of the sensor (or availability of the information) regarding the slope angle.

The functional analysis uses representations that illustrate the interaction and flows between functions, but also the function decomposition, introduced in figure A.10.

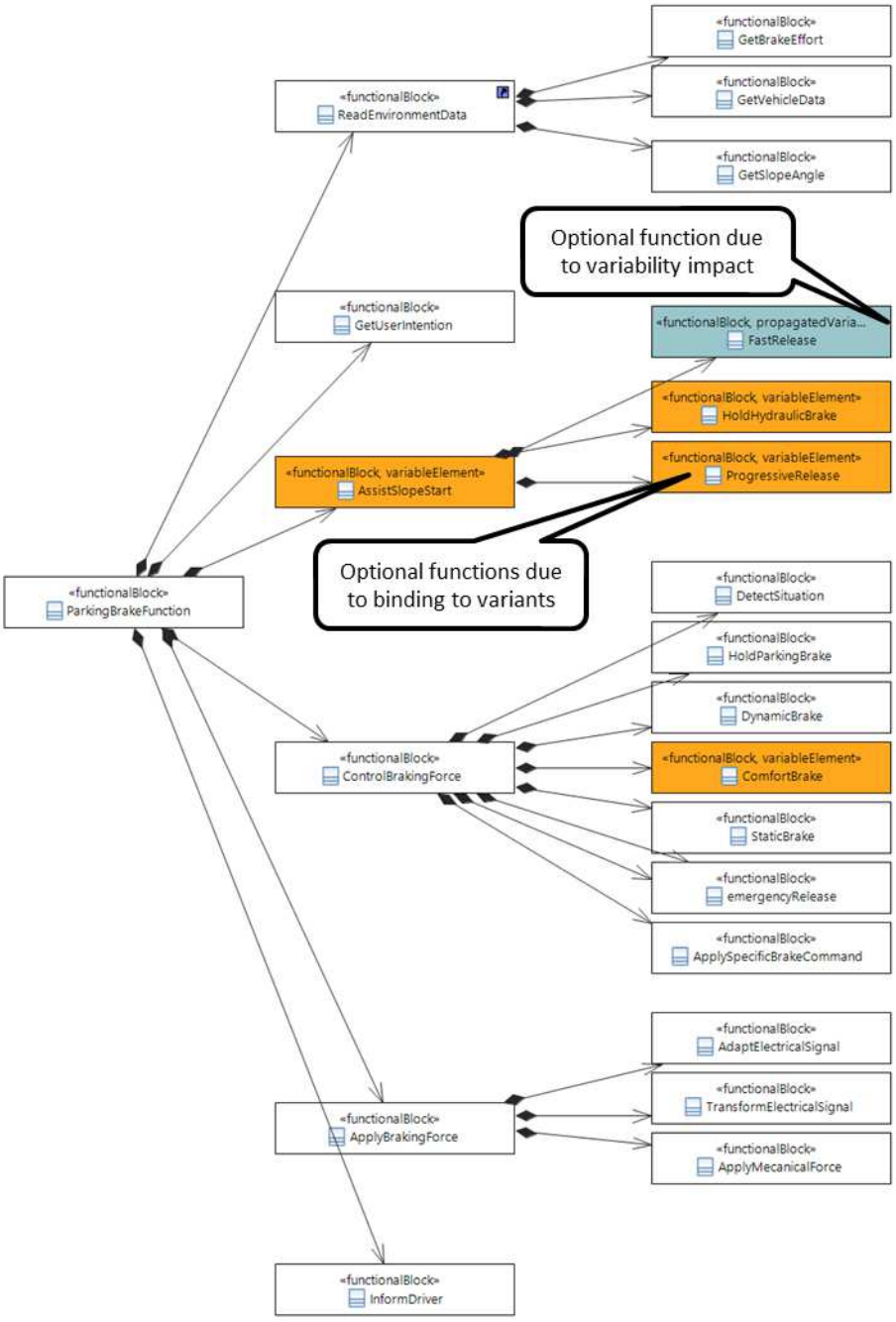


Figure A.10: Electric Parking Brake functional decomposition

The functional hierarchy, where some of the functions are mandatory, others optional, can be seen as "feature-like" diagram. This contains optional functions due to variants mapping: *AssistSlopeStart*, *HoldHydraulicBrake*, *ProgressiveRelease*, *ComfortBrake*. It also contains functions rendered optional by the context - *FastRelease*. For

instance, *HoldHydraulicBrake* is only considered if the assistance during hill starting relies on the hydraulic brakes. (functionality also known as "full" hill start assistant). The *FastRelease* function is available regardless of the braking system used, but is rendered optional by the higher level function. The automatic propagation can provide valuable support during modeling, minimizing the number of explicit associations of system elements to variants. We still lack support for many of the rules related to our systems engineering meta-model and modeling all binding of variants to the system elements is a time consuming task.
Figure A.11 describes the physical EPB system decomposition with the variants that are bound to optional components.

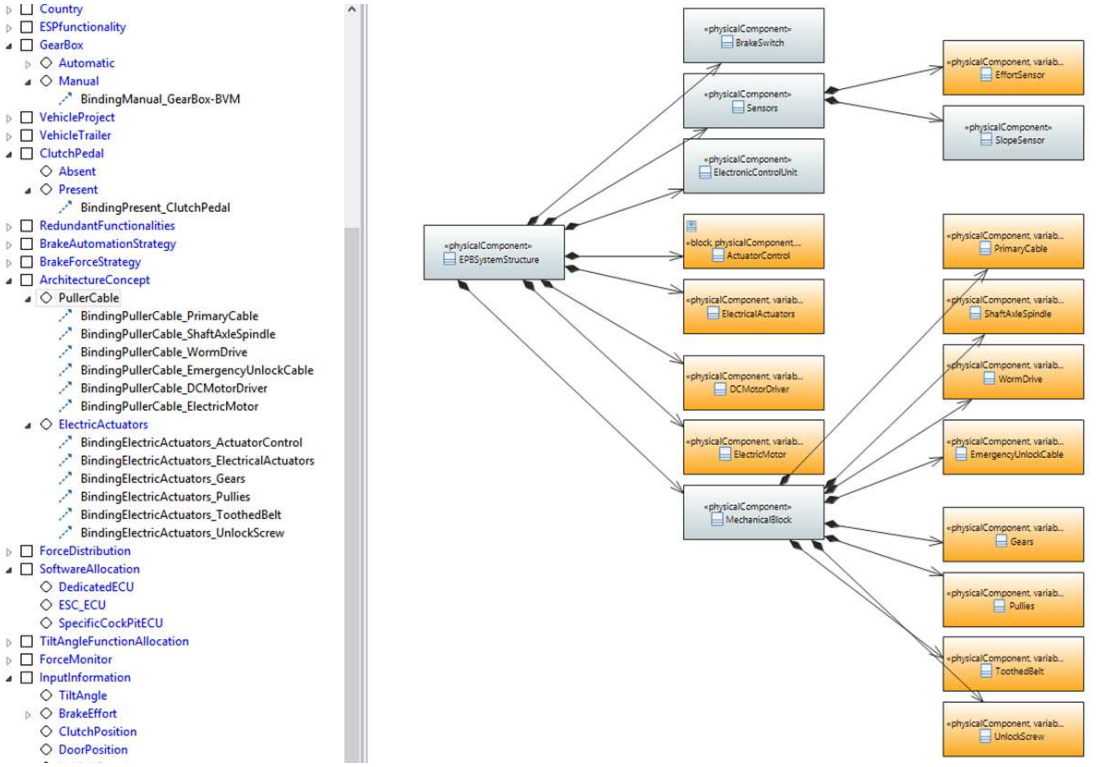


Figure A.11: EPB system physical decomposition and variant binding to components

	client	supplier	[Label]
1	GetBrakeEffort	EffortSensor	Allocate1
2	GetSlopeAngle	SlopeSensor	Allocate2
3	AssistSlopeStart, ControlBrakingForce, GetVehicleData	ElectronicControlUnit	Allocate3
4	ApplyMecanicalForce	MechanicalBlock	Allocate4
5	AdaptElectricalSignal	ElectricMotor, ElectricalActuators	Allocate5
6	GetUserIntention	BrakeSwitch	Allocate6
7	EPBSystemFunctionality	EPBSystemStructure	Allocate7

Figure A.12: EPB System function-component allocation

Figure A.13: EPB system variability propagation counter example

Variability imported from the Renault Documentary Language is in direct relation to the commercial offer of the vehicle range. The system is to be used on vehicles which propose an automatic parking brake as an option, as well as hill start assistance. The studied diversity captures the variable characteristics of the vehicle and environment that need to be taken into account in the study: vehicle projects (we refer to them as project A and B), gear box type, regulation which impacts the usage of the system (e.g. regulation requires that a manual release of needs to be provided). The diagram presented in figure A.14 is based on UML Use Case diagrams for variants and Sequoia [199] to represent constraints. Constraints which were not included in Sequoia were needed, for example to express the fact that *VehicleProjectB* targets both EU and non EU markets (*RequiresAtLeastOne(VehicleProjectB, (EU, outsideEU))*). There are similar approaches [44][97] which rely on Use Case diagrams to represent product line variability in UML models. However, the advantage of our approach comes from capability of Sequoia to be extended for complex, N to N constraints, which are also used between system architecture elements (i.e. functions, components). Some of the constraints depend on the specific domain of the models (e.g. the use of *except*[1] in the Renault Documentary Language). In conclusion, more complex constraints remain model items, and are not represented textually in external editors.

[1]SAUF

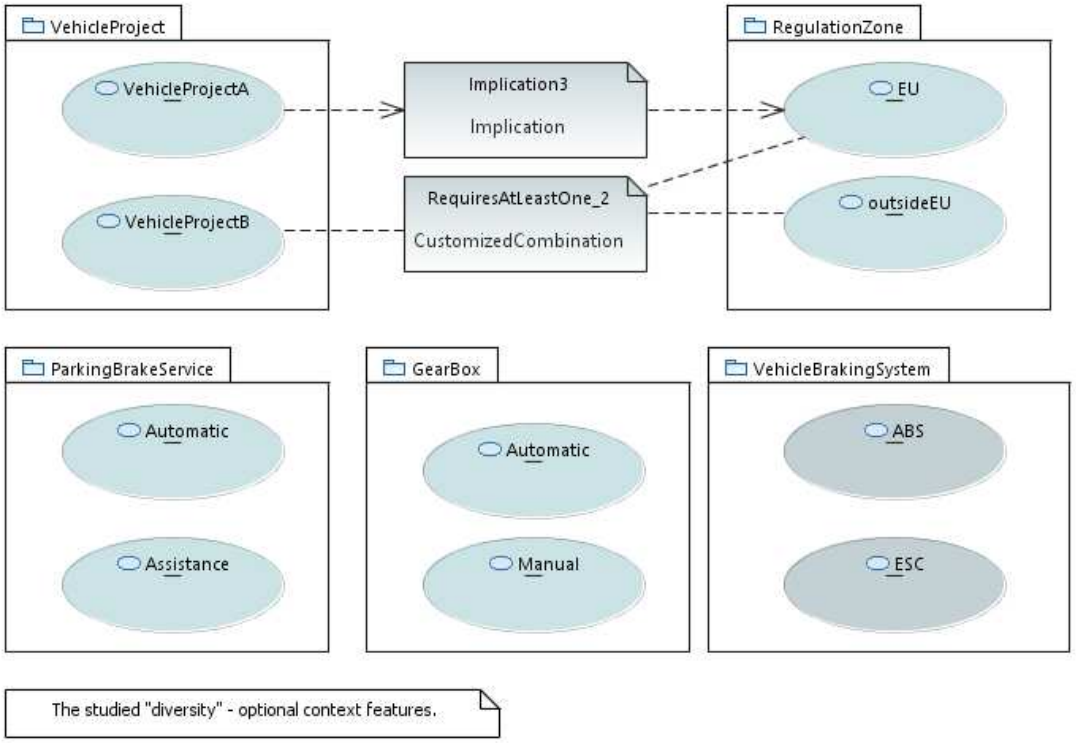


Figure A.14: Studied diversity - optional context features that define the scope of the study

These elements are limited to elements defined in the company "Documentary Language" (vehicle level variability). Variability in the environment and interactions is further refined during the operational analysis. Some of the constraints between these variants remain external to the model and need to be checked against the Documentary Language definitions. The separation of organization level variability from the variability of each system family gives more flexibility in modeling and reusing assets to the engineer, which is not limited to the Documentary Language representation.
Figure A.15 presents some constraints stemming from the system environment and system behavior in its environment.

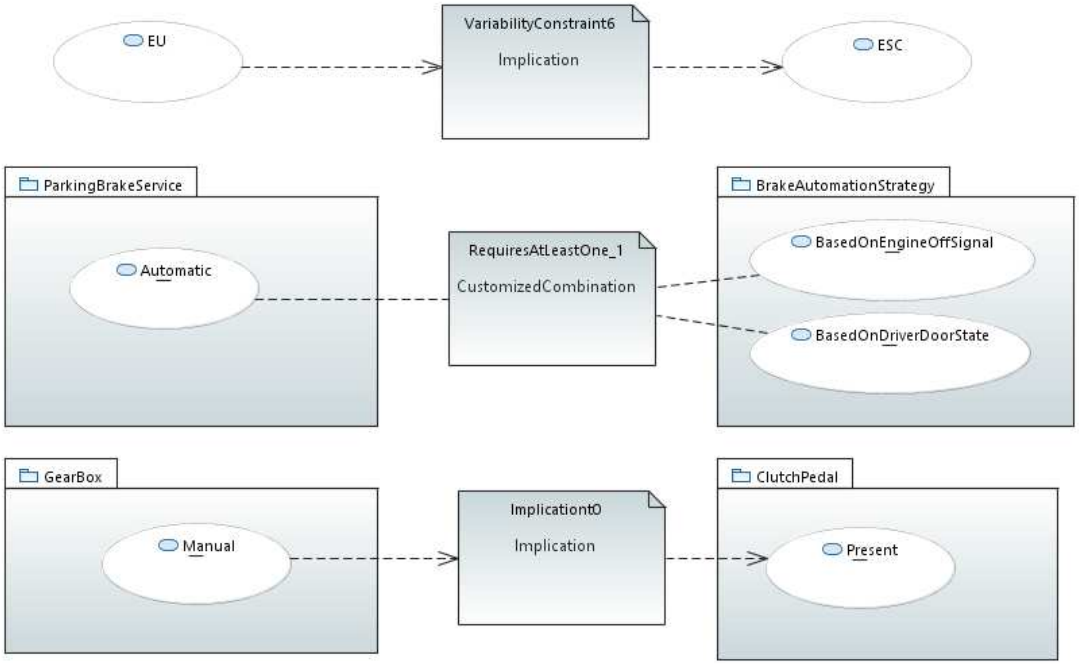


Figure A.15: Examples of variability constraints from the system operational analysis

According to these constraints, the vehicle must include the Electronic Stability Control (*ESC*) variant, if it is commercialized in Europe, due to the regulation in place. Furthermore, for the *Automatic* variant, there are two alternative system behaviors in regard to other vehicle system. The EPB system can lock the parking brake when (i) the engine is turned off (*BasedOnEngineOffSignal*), or (ii) when the driver door is open (*BasedOnDriverDoorState*). At least one of these behaviors needs to be implemented, which is an engineering design decision. Theoretically, these can also be implemented together at this phase of the analysis, which is why the constraint *RequiresAtLeastOne* was chosen ($Automatic \Rightarrow (BasedOnEngineOffSignal \vee BasedOnDriverDoorState)$).

A.1.3 Electric Parking Brake System Variability

The Electric Parking Brake (EPB) System is commonly used by automotive companies to replace or improve the functionality of the conventional parking brake system [185]. We have used this system in conjunction with the "hill start assistant" function of the vehicle, as an exploratory case study, for exploring development scenarios and identifying requirements for system variability management. Indeed, the complexity as well as the cases it reveals made this a suitable example. It contains variations on all levels: for example, the service provided and in the way it interacts with the user and it's environment, design alternatives taking into account force repartition between the

electric and hydraulic brakes, architecture or function allocation alternatives.
From the point of view of variability, the electric parking brake system is less complex[1] than other vehicle systems. However, it contains some interesting characteristics related to the systems engineering process, and variations on all levels of abstraction. The variability model can generate more than 10^4 valid configurations.
Figure A.16 presents the list of variability points in the EPB model.

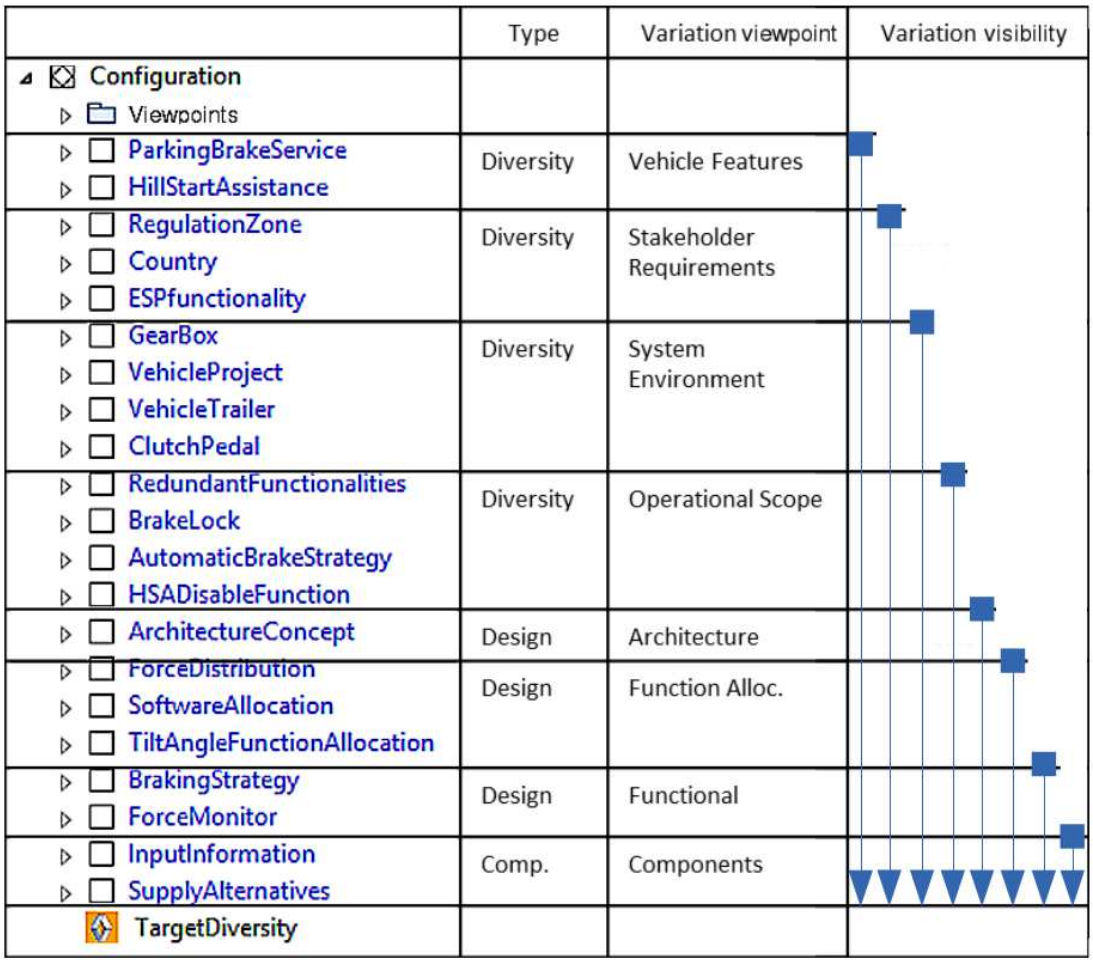


Figure A.16: Electric Parking Brake list of variation points (adapted screen capture from the Renault Variability Management Papyrus plug-in)

The viewpoints correspond to the system analysis viewpoints, but some additional were defined in respect to design and configuration decisions: vehicle features – represent vehicle variability defined by marketing, in the form of the Documentary Language at Renault; architecture concept – high level design decisions related to the main

[1] Combinatorial complexity induced by system variability

system concepts (e.g. choices regarding technologies, way of operation); (function) allocation alternatives – design decisions in respect to how functions are allocated to components. Due to the distributed nature of the electric/electronic architecture of the vehicles today, often there are multiple alternatives regarding data acquisition or software functions. These can be an important source of variability, depending on the technical differences of the available vehicle configurations. As explained in Figure A.16, variability can belong to multiple viewpoints. This functionality is useful in large variability models, and during the system analysis, when only the variation points of interest can be shown. Managing viewpoint associations and their use during derivation in the MBSSE tool, is presented in Appendix E. The Electric Parking Brake system variants and the related viewpoints are described below.

Customer visible variability corresponds to the variability stemming from the vehicle level – the vehicle features. This is defined by the product division. Three types of service are proposed: Manual, Automatic and Assisted [1]. The "manual" brake is controlled by the driver either through the classical lever or a switch. The "automatic" parking brake system variant may enable or disable the brake itself depending on the situation: for example when the driver leaves turns off the engine and leaves the vehicle, the parking brake is activated. The "assisted" brake brings extra functions that aid the driver in other situations : such as assistance when starting the car on a slope. In all operational scenarios, except for the manual variant, the system can decide to lock the parking brake. This is for instance the case when the driver exits the car, engine is stopped, the vehicle starts on a slope.

The operational viewpoint contains different facets of the system context, such as: system boundary variability, enabling systems and vehicle environment. The gear-box and the presence of certain types of trailers (*VehicleTrailer* variation point) and their characteristics have a direct impact on the internal behavior of the EPB system. The presence of a trailer, for example, may require that the hill start assistance functionality be disabled, or its behavior adapted to the new total weight conditions.

The *architecture alternatives* and *allocation alternatives* viewpoints specify design decisions that impact the whole or parts of the technical solution.

This includes: main solution alternatives (*ArchitectureConcept*), choices on how to distribute the effort among the EPB and the main hydraulic braking system (*ForceDistribution* variation point), decision on software allocation to hardware (*SoftwareAllocation* variation point) and the allocation of the slope angle detection function (*TiltAngleFunctionAllocation*). Allocation of this last function to a specific computer would obviously require that the computer (ECU) already exists.

The variability entailed by the system internal behavior viewpoint impacts the states and transitions of the system physical and software components.

In the EPB , the braking strategy can vary depending on the deriving conditions. Each strategy requires specific information: Comfort and Dynamic require vehicle speed information (*VSpeed*) and the specific strategy for hill start assistance requires that there

[1] The variability presented here does not necessarily use the same nomenclature and expose the same options as current online product catalogs.

is a tilt angle sensor. Braking pressure is monitored after the vehicle has stopped for a certain amount of time (*Temporary*) for the single DC motor, puller cable solution, and permanently monitored (Permanent) for the other solutions.
The variability entailed by the physical architecture specifies variability in component decomposition, through optional or replaceable components, as well as physical interfaces variability between components.
Physical variability of the EPB consists in the presence of different means of applying the brake force: electric actuators mounted on the calipers or single DC motor and puller cable much like the traditional mechanical parking brake. Also the type of sensors available may vary depending on the configuration and needs.
In addition to variation points and dependencies, variants attributes were associated to the different variants, in order to specify supplementary needed information regarding the impact of PL configuration on performance (Braking Force Dissymmetry, Response Time on Brake), reuse (Vehicle Range Coverage) or cost increase in respect to a reference configuration of the system (Extra Engineering Cost). These attributes are numerical variables, that serve during the derivation process and help the engineers make the right choices, by assessing the impact of their choices on the system configuration, and as the basis for supplementary constraints.
The product on the right side of Figure A.17 corresponds to a "puller cable" technical solution (*ArchitectureConcept - PullerCable*), while the instance on the left corresponds to the solution based on "electric actuators" (*ArchitectureConcept - ElectricActuators*). Figure A.17 presents two examples of physical configurations of the EPB system.

Figure A.17: The Electric Parking Brake system main design alternatives

These two configurations are typically found on passenger Renault vehicles and most likely vehicles from other manufacturers. Some of the variability is linked to design decisions, while others remain unresolved choices until the vehicle complete configuration is known at the manufacturing time. An orthogonal variability model [163] of the EPB system is presented in Figure A.18. The coloring marks which variants are selected in each of the two main configurations (a. b. c.): (a) the gray pattern marks variants present in both configurations, (b) blue variants are present in the *puller cable* configuration, and (c) green variants are present in the *electric actuators* configuration.

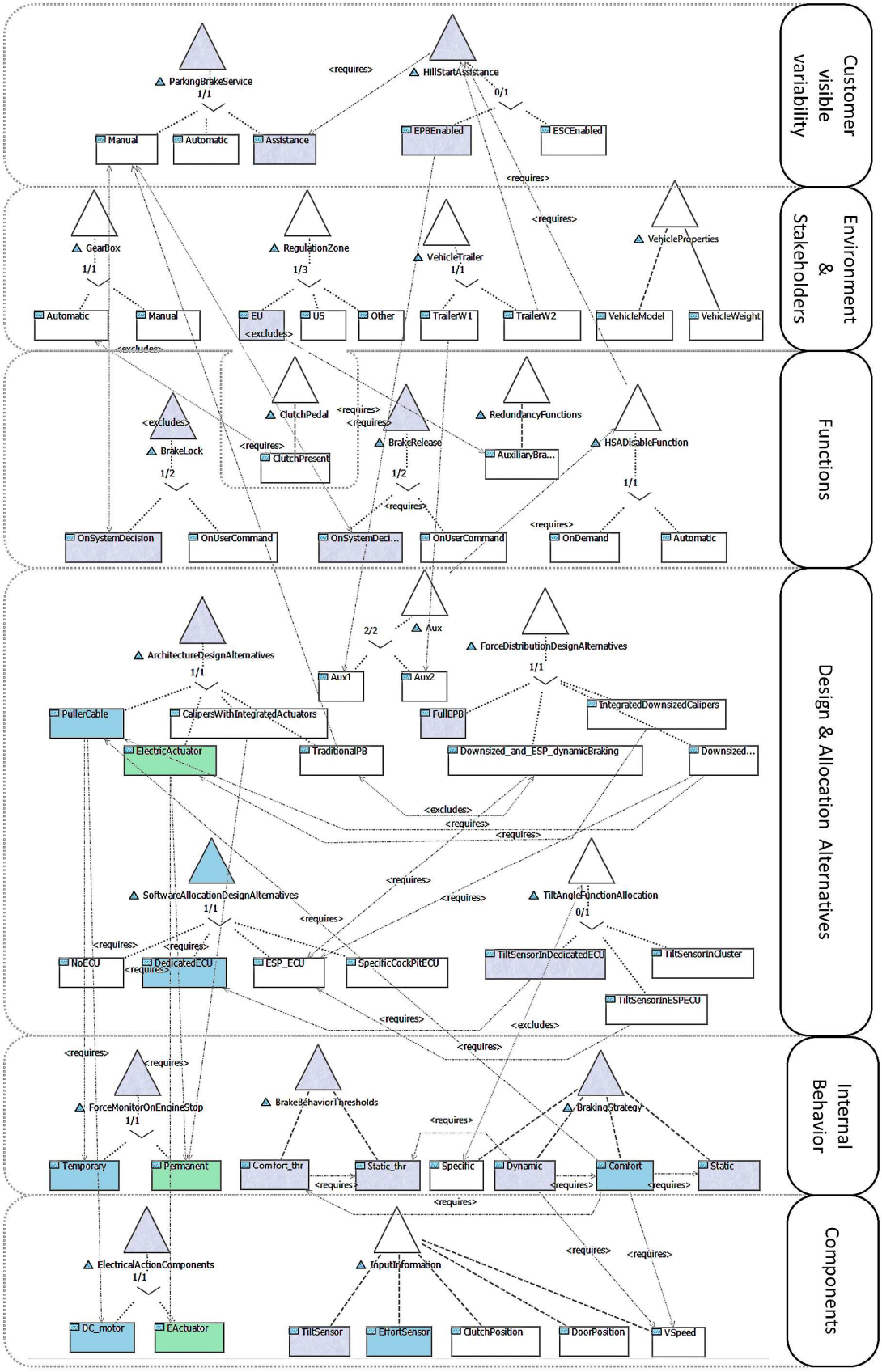


Figure A.18: OVM model of the EPB system with colored selected variants

Because many of the variants regarding the system environment are resolved later in the development process, the colored OVM model reflecting the two main technical solutions, illustrates only partial configurations. The type of gearbox and some of the vehicle properties are yet to be determined, they will eventually impact the system behavior (e.g. the EPB system disengages the parking brake on different conditions whether the vehicle has an automatic or manual gearbox). Usually, by the time these decisions are made, the physical product implementation already exists and the final system behavior is determined through software calibration, in plant. However, the way the product is designed to accommodate these delayed decisions, in software or physical structure, is up to the system engineer/architect.

A.1.4 System family constraints

Given the reduced complexity of the EPB model, we have identified 36 constraints between variants in the final SysML model of the system, as presented in figure A.19. The notion of (diversity) constraint is consistent with Renault Documentary Language vocabulary and marketing constraints are also defined in the Documentary Language. In addition, 44 constraints are issued on the system model side, for binding variants to system model items and impact of variability. For example the "comfort braking" strat-

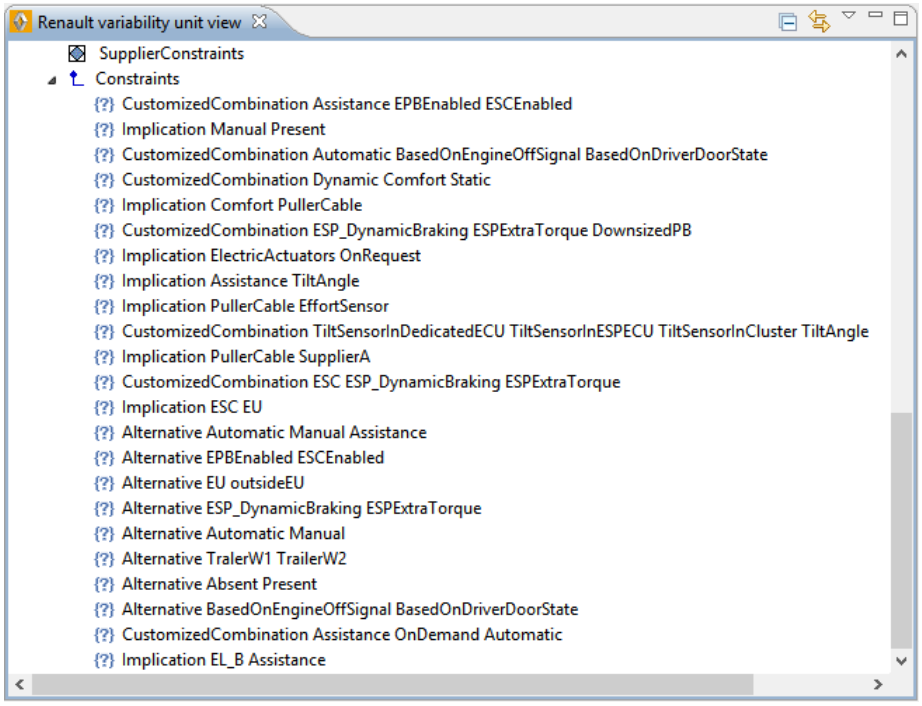


Figure A.19: Constraints between variants stemming from commercial definitions and system design

egy can only be implemented on the solution with two electric actuators mounted on

the wheel calipers. These allow sufficient control over the brake pressure to implement this strategy, thus $Comfort \Rightarrow ElectricActuators$ .These are completed by constraints generated by dependencies between system architecture elements and variants (binding), and others generated by the Sequoia [199] variability propagation functionality. The nature of the constraints is different. We distinguish the following type of constraints, without taking into account the ones between variants linked to the same variation points (e.g. alternative, at least one etc.)

Customer Constraints:

$$EquipmentLevel\_B \Rightarrow Assistance \tag{A.1}$$

$$Assistance \Rightarrow (HSA\_EPBEnabled \oplus HSA\_ESCEnabled) \tag{A.2}$$

$$Assistance \Rightarrow (HSADisable\_OnDemand \oplus HSA\_DisableAutomatic) \tag{A.3}$$

Customer constraints are related to the commercial definition, which is usually described with the Documentary Language. The equipment level is used to define options which are grouped together in packs. For example constraint (A.1) expresses the fact that the hill start assistance option is included in a pack of options, thus choosing *EquipmentLevelB* requires the presence of the *Assistance* feature. Furthermore, two types of hill start assistance are possible, as specified by constraint (A.2): one uses the parking brake, the other the hydraulic brakes (also called "full hill start assistance"). In a situation where, the weight of the vehicle increases over a certain limit, the hill start assistance (HSA) feature should be disabled for safety reasons, either manually or automatically - (A.3). One example where the HSA should be disabled is the attachment of a trailer.

Operational Constraints:

$$EU \Rightarrow ESC \tag{A.4}$$

$$GearBox\_Manual \Rightarrow ClutchPedal\_Present \tag{A.5}$$

$$Automatic \Rightarrow (BasedOnEngineOffSignal \oplus BasedOnDriverDoorState) \tag{A.6}$$

Operational constraints are related to the environment where the system will be used. Since November 2011 all vehicles sold in the European Union must be fitted with Electronic Stability Control (ESC, or ESP) (expressed by constraint (A.4)). Information from the environment important for the functioning of the system also needs to be considered. For example, the position of the clutch pedal and that of the gearbox - (A.5), (A.6) - are needed for the automatic release or engage of the parking brake.

Technical Constraints:

$$(ESP\_DynamicBraking \oplus ESPExtraTorque) \Rightarrow ESC \tag{A.7}$$

$$DownsizedPB \Rightarrow (ESPDynamicBraking \oplus ESPExtratorque) \quad \text{(A.8)}$$

$$Dynamic \Rightarrow Static \quad \text{(A.9)}$$

$$Comfort \Rightarrow Static \quad \text{(A.10)}$$

$$Comfort \Rightarrow PullerCable \quad \text{(A.11)}$$

$$PullerCable \Rightarrow ForceMonitor\_Temporary \quad \text{(A.12)}$$

$$ElectricActuators \Rightarrow ForceMonitor\_OnRequest \quad \text{(A.13)}$$

$$Assistance \Rightarrow TiltAngle \quad \text{(A.14)}$$

$$PullerCable \Rightarrow EffortSensor \quad \text{(A.15)}$$

$$TiltAngle \Rightarrow (InDedicatedECU \oplus InESPECU \oplus InCluster) \quad \text{(A.16)}$$

Technical constraints stem from the system design or from dependencies to enabling systems, from the technical context. An enabling system is an autonomous system, but which is required for the functioning of other systems. For example constraint and (A.7) refers to features of enabling systems. The ESC system is able to complement the braking force of the parking brake, if needed, depending on the design of the EPB system.
The rest of the technical constraints relate to design choices. For instance, only the *PullerCable* variant allows implementation of the *Comfort* braking strategy, defined in constraint (A.11). The hill start assistance function needs to take into account the angle of the slope, specified by constraint (A.14), in order to calculate the necessary braking force. The architecture based on the "puller cable" concept needs effort information for the control of the braking force, specified by constraint (A.15).
Supplier Constraints:

$$SupplierA \Rightarrow PullerCable \quad \text{(A.17)}$$

Finally, once the physical architecture has been defined, component suppliers need to be considered. Supplier constraints take into account the commercial offer of suppliers, geographical, volume availability, performance or quality impact on the system etc. In our example we considered a supplier that delivers only the solution based on the "puller cable" concept, which we called *SupplierA*.

A.2 Example 2: Lighting System Family

The automatic lighting system manages the on-off state of the vehicle lights with the purpose of improving the capacity of the driver to see the zone situated in front of the vehicle, and to improve vehicle visibility for the other persons in the close environment. This is done by turning on or off the vehicle front and rear lights in respect to the external light levels. The system also reduces energy consumption and is subjected to local regulations.

A.2.1 System description

Two main variations are considered, the *manual* variant and the automatic variant of the lighting system. In addition, a "fully automatic" variant can be considered, which does not allow the driver to disable the automatic function. The manual commands always have priority over the system automatic behavior.
The system reacts to changes in the environment light levels, captured by a photoelectric sensor. Parameters *Param_high* and *Param_low*, presented in Figure A.20, represent the limits for switching between high beam and low beam headlight positions. The switch depends on the previous state of the headlights, and the system switches to the next state only after the environment light levels cross the transition range.

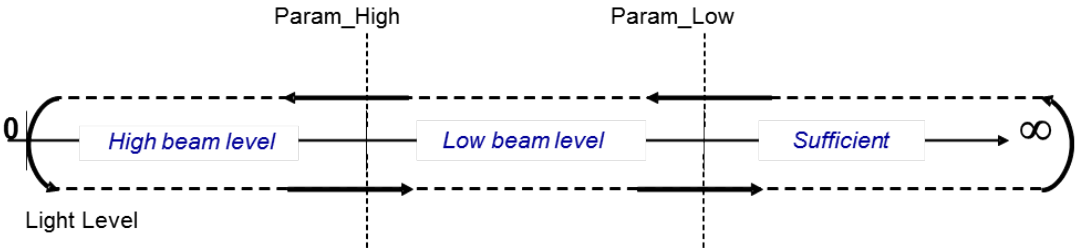


Figure A.20: Automatic light switching levels

The light levels need to take into consideration the regulation of the country where the vehicle will be commercialized. As illustrated in Figure A.20, the system adapts the headlamps level in respect to the ambient light: (a) headlamps are off when the ambient light level is sufficient ($> Param\_Low$); (b) headlamps are in low beam mode (dipped beam, passing beam, meeting beam) whenever other vehicles are present ahead ($Param\_High < Ambient\_light\_level < Param\_Low$); headlamps are in high beam level (main beam, driving beam, full beam) when the ambient light level is very low, corresponding to natural night light, or absence of light ($< Param\_High$). The switching around the two thresholds is done through hysteresis loops to avoid frequent switching.
The are some differences between the ECE[1] and US regulation in regard to lighting

[1]Economic Commission for Europe

behavior, while in some countries it is required to have the headlamps always on, including daytime (daytime running lights). The *low beam* mode provides a distribution of light designed to provide adequate forward and lateral illumination with limits on light directed towards the eyes of other drivers, to control glare. The international ECE Regulations for filament headlamps and for high-intensity discharge headlamps specify a beam with a sharp, asymmetric cutoff preventing significant amounts of light from being cast into the eyes of drivers of approaching ahead. Control of glare is less strict in the North American SAE[1] beam standard [27]. The *high beam* provides a bright, with a centered distribution of light, and without any control of light directed towards other vehicles. It is suitable for use when alone on the road, as the produced glare would disturb other drivers. ECE Regulation allows higher intensity high-beam headlamps compared to the North American regulations [27].

A.2.2 Examples from the architecture model

Most of these diagrams for the SAFE system were developed by the MBSSE team as a joint effort. They are presented in the figures below.

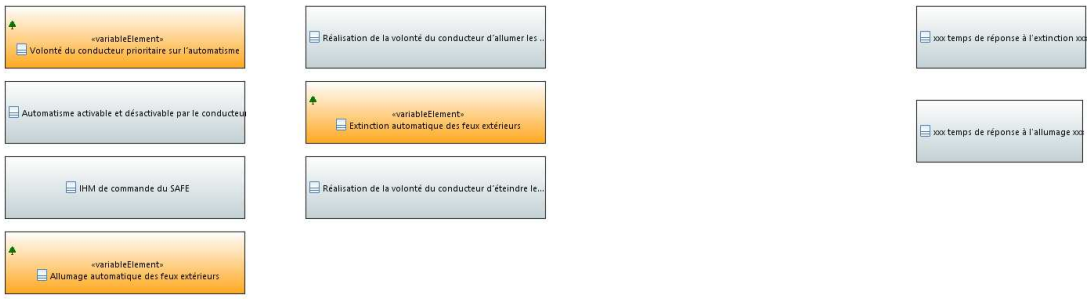


Figure A.21: SAFE system high level requirements

[1] SAE International, formerly known as the Society of Automotive Engineers

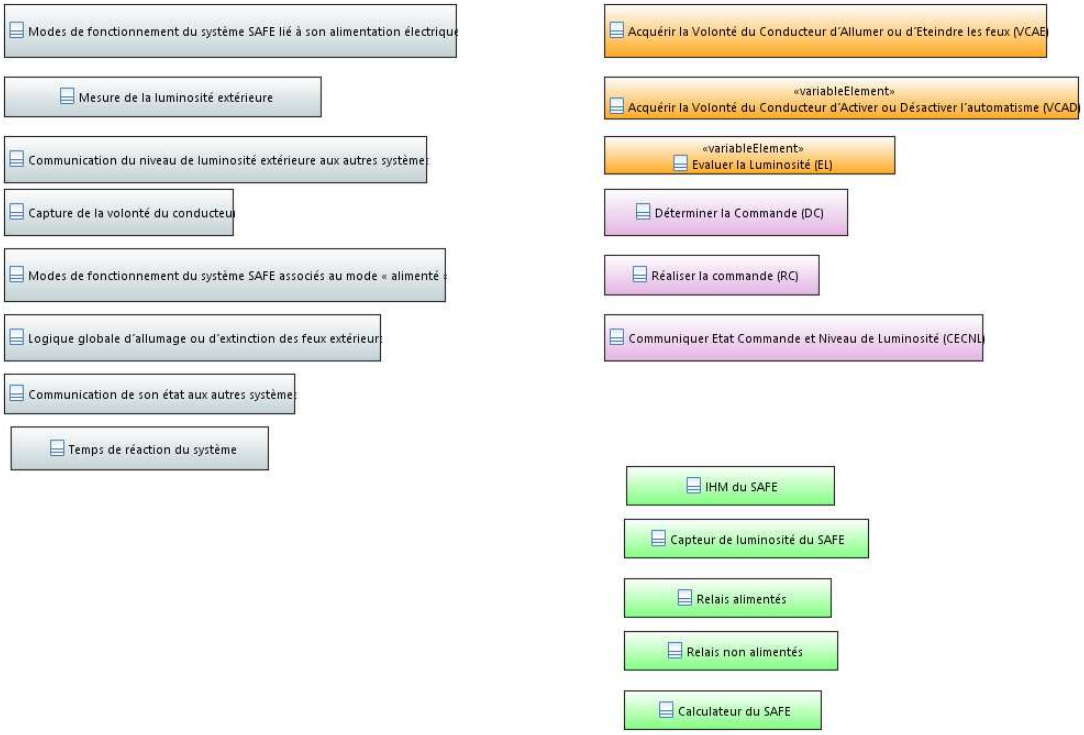


Figure A.22: Some SAFE system technical requirements

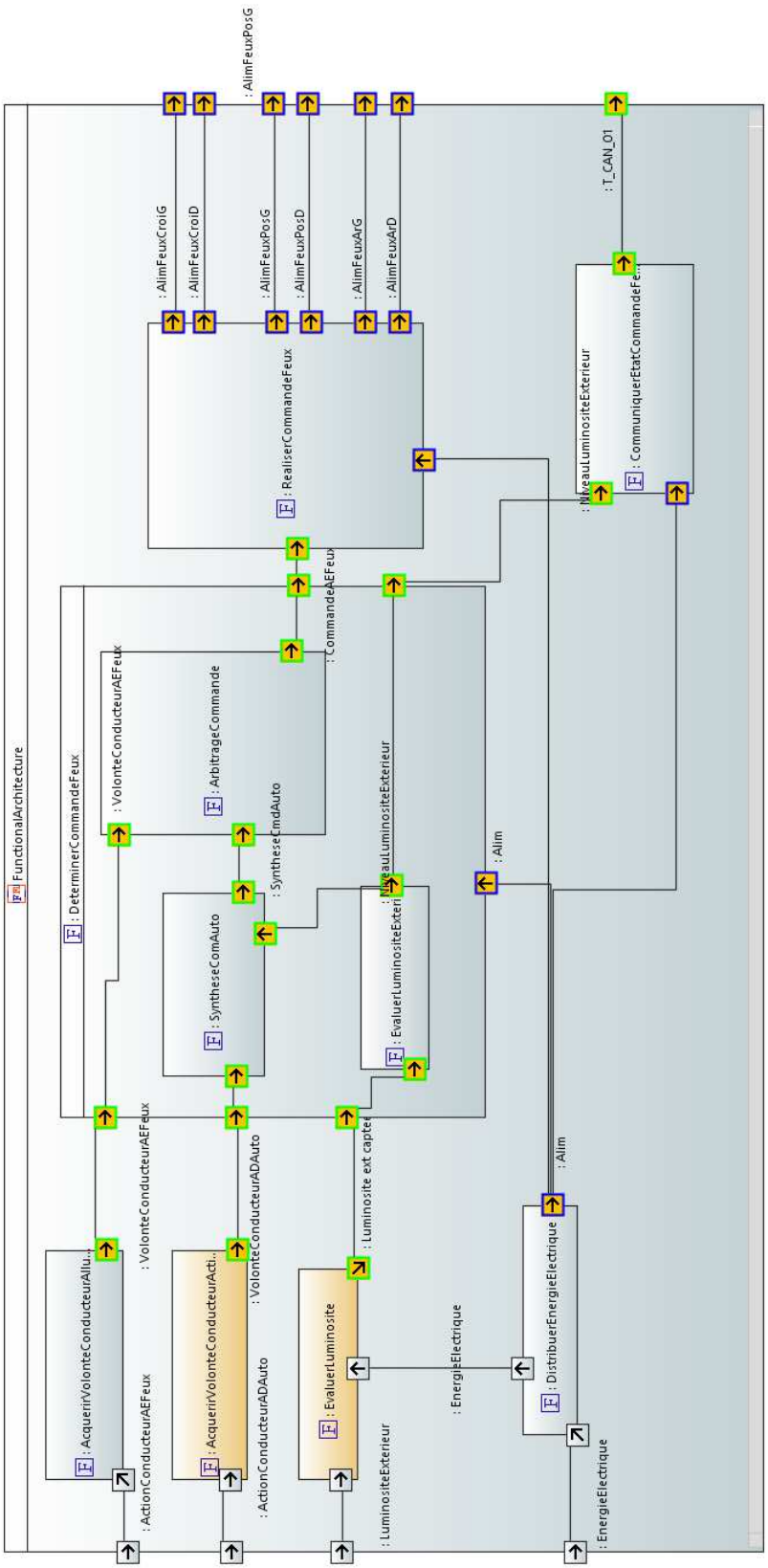


Figure A.23: SAFE system functional architecture with optional functions

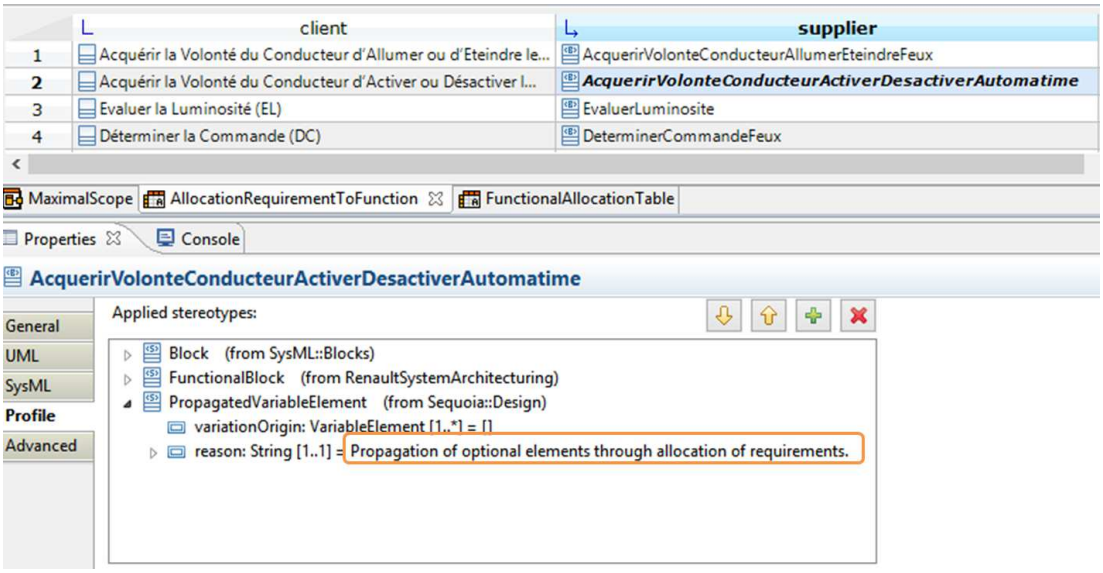


Figure A.24: SAFE system requirement allocation with optional function

Figure A.25 describes the physical architecture of the SAFE system. The "Light Sensor"[1] to enable the optional function "Evaluate light intensity"[2].

[1]Capteur de luminosité
[2]Evaluer luminosité

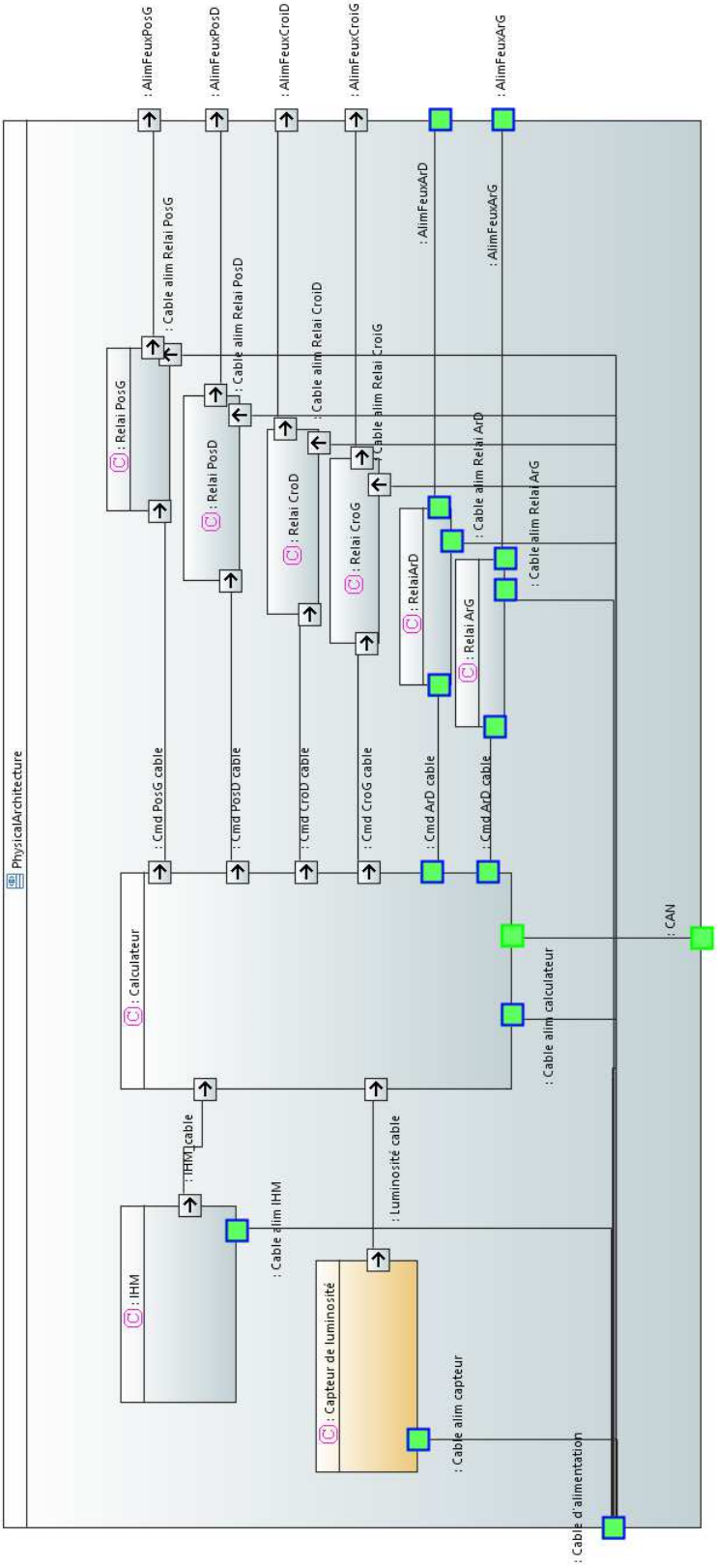


Figure A.25: SAFE system physical architecture with optional component

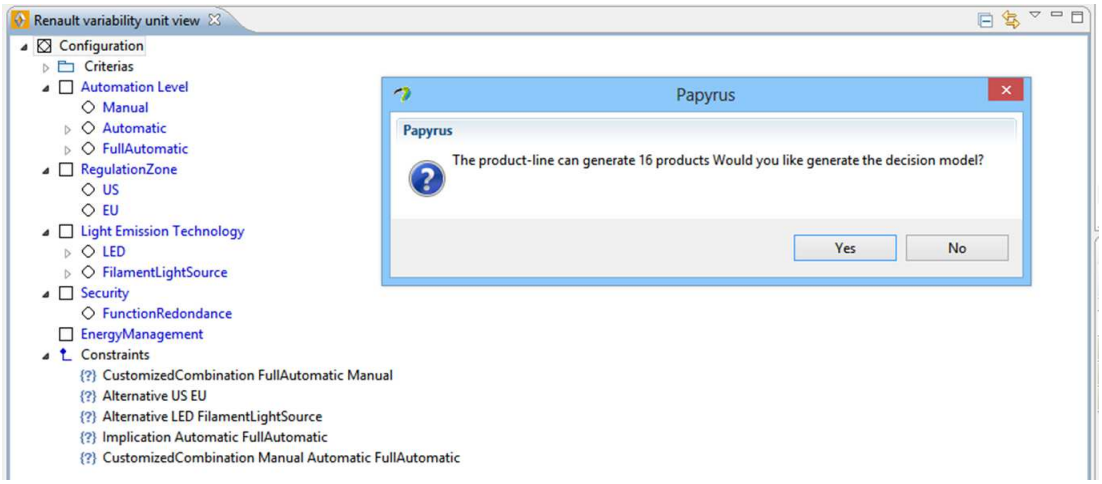


Figure A.26: Launching product derivation for the SAFE system

A.2.3 Automatic Lighting System Variability

The system contains four variation points, with 8 variants, which can generate 16 configurations, taking into consideration the additional constraints. While most of the elements of the system model are common for all configurations, it is possible to use dependencies between elements to identify impact of variants on the system model.
The variant *Automatic* is related to the following high level requirements:

- *REQ_SAFE_M1* Automatic switch on: When the automatic lighting function is active, the system shall switch on the (a) low beam or (b) high beam positions, given the external light level is: (a) lower than the *low beam threshold* or (b) lower than the *high beam threshold*.

- *REQ_SAFE_M2* Automatic switch off: When the automatic lighting function is active, the system shall switch off the (a) low beam or (b) high beam positions, given the external light level is : (a) higher than the *low beam threshold* or (b) higher than the *high beam threshold*.

These requirements are modeled as optional elements bound to the variant *Automatic* (variation point *Automation level*). The functions to which these requirements are allocated also become optional due to the impact of requirements. The identification of optional functions or components based on allocation extends the tool proposed by Tessier et al., which implements his approach "variation point propagation" [198]. The identification of optional elements depends on the domain metamodel, which means rules need to be implemented for each specific application, in this case MBSE.
Figure A.27 illustrates the allocation of requirements to functions and the application of the rule *"Propagation of optional elements through allocation of requirements"*. Each

variation point is visible in one or more viewpoints. The visibility is also managed automatically by variability propagation rules. After the application of the rule mentioned above, the variation point *Automation level* has the following characteristics:

- Variation Point Source: *Requirements*
- Visibility of variation points/Viewpoints: *Requirements* and *Functional Architecture*

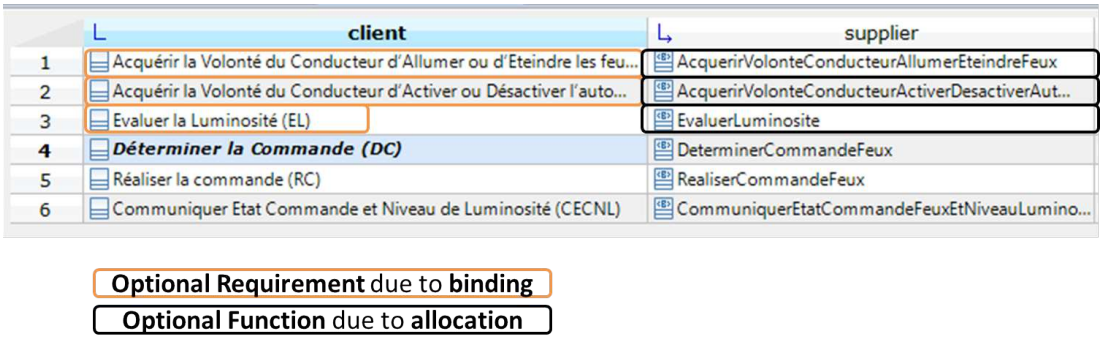


Figure A.27: Requirement to function allocation and propagation of optional elements

Furthermore, it is possible to identify optional elements among components, by applying the rule *"Propagation of optional elements through allocation of functions"*, as shown in figure A.28.

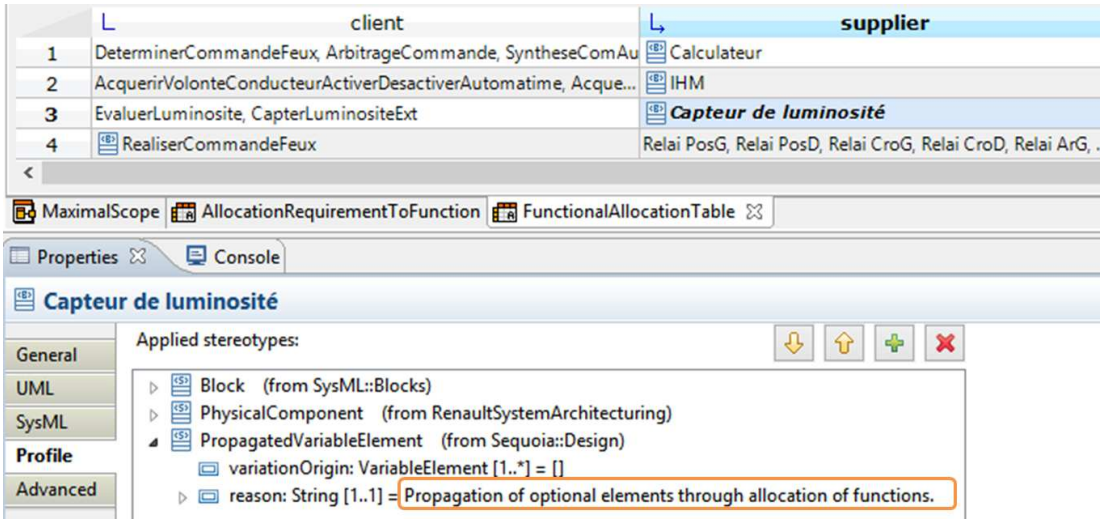


Figure A.28: Allocation of functions to components and propagation of optional elements

Other variants include the geographical area where the vehicle is commercialized (*EU, US*), which impact threshold parameters for light switching and requirements

related to vehicle lighting regulation. Design alternatives take into account the *Light emission technology*, which includes LED lighting or incandescent light bulbs. This variation point impacts only the physical components of the system, but has no impact on functions or customer visible features, in this example.

A.2.4 System Constraints

The system model contains 4 constraints related to variants and 32 constraints generated by binding to system elements and through impact of variability. These can generate 8 completely defined productcompletely defined system configurations. The dependency constraints between variants include the following:

$$FullAutomatic \oplus Manual \tag{A.18}$$

$$Automatic \Rightarrow FullAutomatic \tag{A.19}$$

$$EU \oplus US \tag{A.20}$$

$$LED \oplus Filament \tag{A.21}$$

Expressions (A.18), (A.20), (A.21) specify alternatives between variants related to the system design, while (A.19) specifies that fact that system elements related to variant *FullAutomatic* are included when variant *Automatic* is chosen. Indeed, the only difference between the two is the possibility of disabling the automatic lighting function for variant *FullAutomatic*. There are of course multiple ways the constraints and relation to the system model elements can be expressed, resulting in the same system configurations. (e.g. binding both *Automatic* and *FullAutomatic* variants to the same model elements, except those representing the automation disabling function, and adding constraint: $Automatic \oplus FullAutomatic$)
Variants and constraints between variants are orthogonal to models from the same family of systems. Meanwhile, binding and propagation are also expressed through Sequoia constraints, but the information is contained in the system model. Some of the constraints generated through propagation of optional elements based on allocation are:

$$(\neg EvaluerLuminosite \wedge \neg CapterLuminositeExt) \Rightarrow \neg CapteurDeLuminosite \tag{A.22}$$

$$(EvaluerLuminosite \vee CapterLuminositeExt) \Rightarrow CapteurDeLuminosite \tag{A.23}$$

Function allocation allows to identify requirements that each system component has to satisfy. If both functions *EvaluerLuminosite*[1] and *CapterLuminositeExt*[2] are absent,

[1] "Evaluate Light Level" system function
[2] "Capture External Light" system function

then the component supporting these functions is also absent, as specified by (A.22). However, if at least one of the functions is present, so is the component, as specified by (A.23). These constraints are based on the principle that a component can also contain variability and be configurable. Thus, it would be impossible to know which of the functions are realized only by selecting the component.
Binding dependencies also rely on the description of constraints, for derivation. The Automatic Lighting System only contains "1 variant to N system elements" binding dependencies. In this case each binding dependency is associated to an implication constraint:

$$Automatic \Rightarrow IHM\_Automatisme \tag{A.24}$$

where the *Automatic* variant is mapped to component *IHM_Automatisme*.

A.3 Example 3: Water Heating System

The Water Heating System[1] is a system of reduced complexity which contains a significant amount of customer variability with impact on the physical architecture. The example contains stakeholder and system requirements and the functional and physical decompositions.

A.3.1 System Description

The water heater product family is designed for families of up to 5 persons. It can be mounted either on a wall or standing, but technical constraints need to be taken into account. In order to cover this range of use, four capacities are proposed: 100l, 150l, 200l and 300l. The variants are illustrated in figure A.29, as well as selected and deselected variants.

[1] This SysML model with variability is based on the water heater example defined in the project group "product lines" of the technical committee "Global Processes" of AFIS (Association Française d'Ingénierie Systéme)

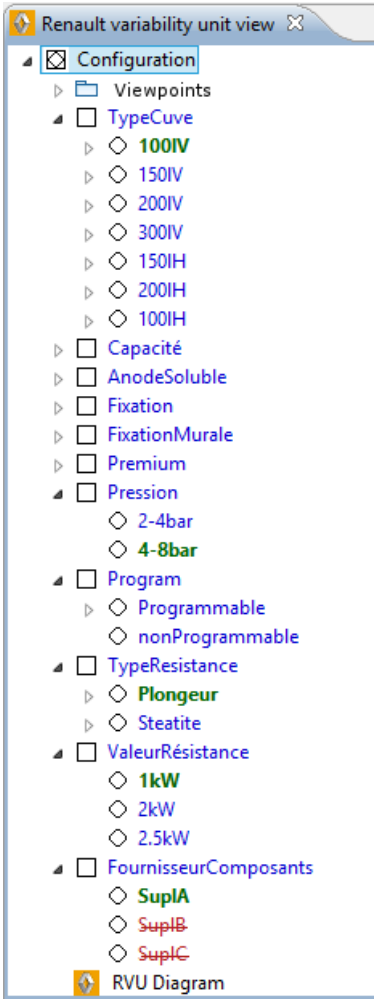


Figure A.29: Water Heater variability with selected variants

The feature *Programmable* is available to customers, which allows a decrease in the consumption of energy. The feature *Premium* requires the use of a soapstone-coated resistor for heating, and should not available for the 100l capacity heater.
The main functions of the system are presented in figure A.30. Variation points are presented on the right side of the figure.

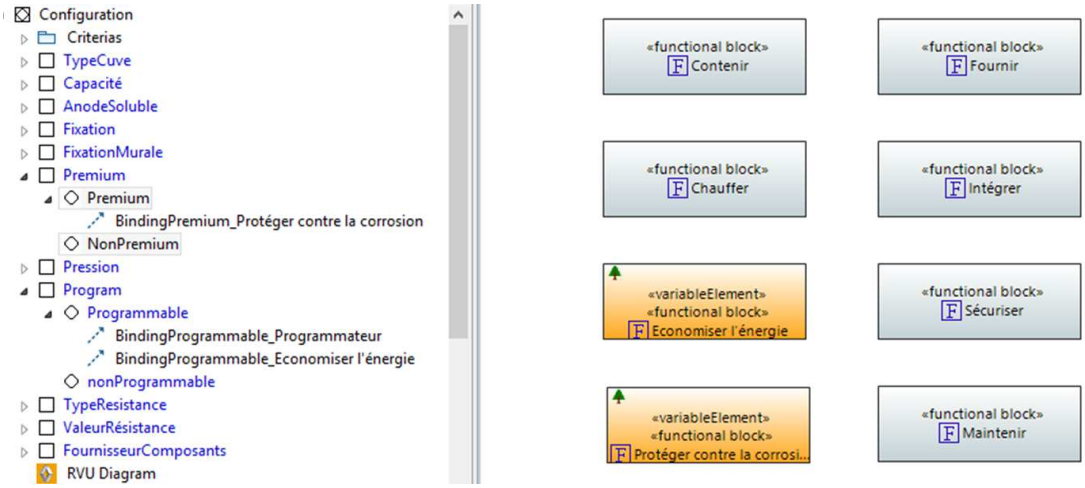


Figure A.30: Water Heater variation points (left side) and list of main functions

The variation points *TypeResistance*[1], *ValeurResistance*[2], *AnodeSoluble*, *Pression*[3] are design choices, marked with the attribute "Design". The variation points which define the commercial offer and are visible to customers are, marked with the attribute "Diversity": *TypeCuve*[4], *Capacité*[5], *Fixation*[6], *Progam*. A variation point providing alternatives for component suppliers was also considered in the example.

A.3.2 Water Heater System Variability

The product line can generate 624 configurations of completely defined products, as shown in Figure A.31, taking into account technical and marketing constraints described is Subsection A.3.3.

[1]Resistance Type
[2]Resistor Values
[3]Pressure
[4]Water tank type
[5]Tank volume
[6]Mounting position

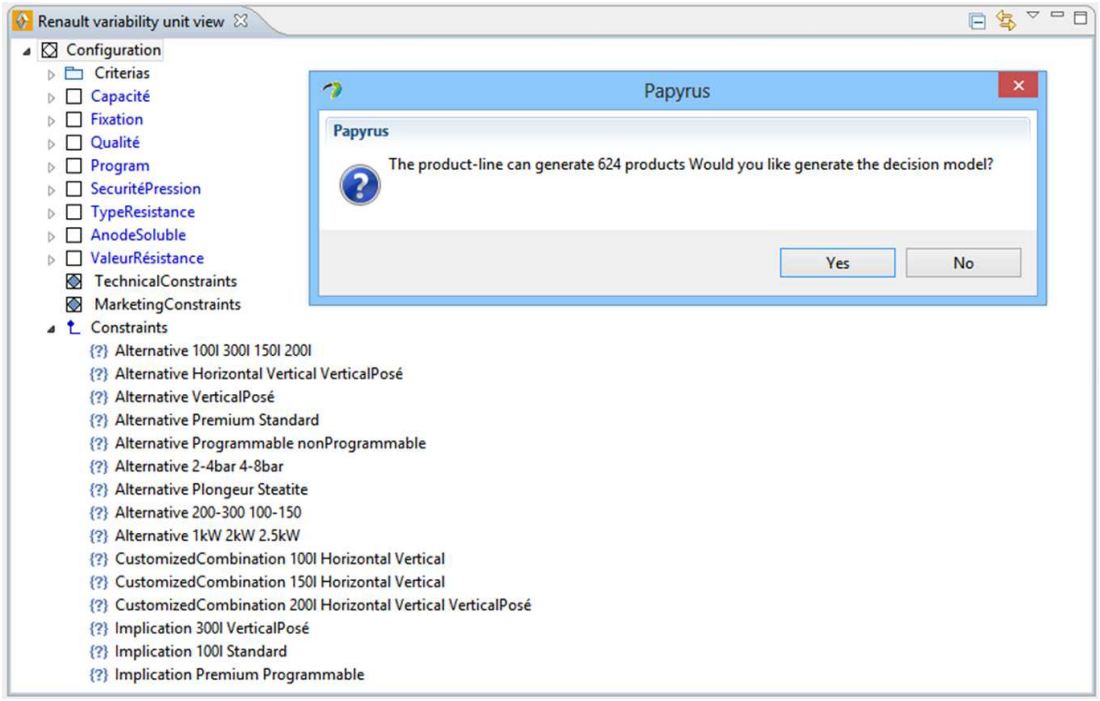


Figure A.31: Water Heater system family variability and launch of product derivation in the MBSSE tool

This example is different from the other two in respect to the way binding of variants to system elements is done. By default, an optional element is kept after derivation if at least one of the bound variants is selected[1] (*BindingLogic* property is set to *AtLeastOneSelection*). In the case of elements which are selected in respect to multiple characteristics, which are not represented in detail in the system architecture, a different way of modeling is needed to avoid exponential multiplication of variants.
For example, variation point *Capacite* provides 4 alternatives, ranging from 100l volume to 300l. The position - *Horizontal* or *Vertical* - also has an impact on the physical design of the water tank. In consequence, 7 water tank designs are possible, because the *300l* variant is always *Vertical*. The selection of the component *Water Tank* is done during derivation, based on which of the alternatives for position and volume are kept. Two variants are bound to each *water tank* component. In this case the component is kept if all bound variants are selected (*BindingLogic* property set to *AllSelections*).

A.3.3 System Constraints

The marketing constraints provide rules which structure the commercial offer. These constraints are presented in Figure A.32, along with the implementation in variability management tool, while figure A.33 introduces the design constraints.

[1]Solution implemented in the final deliverable for project MBSSE 2012

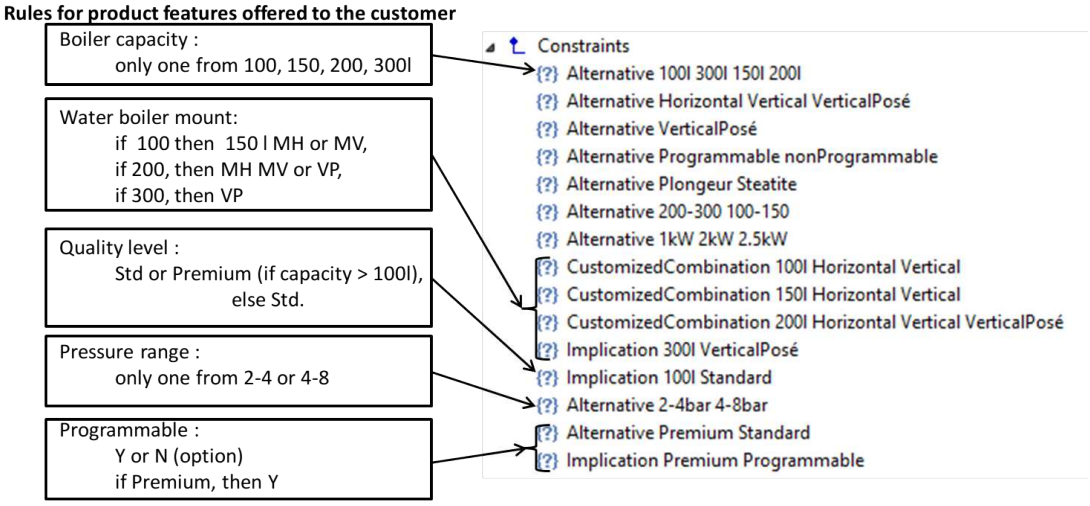


Figure A.32: Water heater customer variability constraints

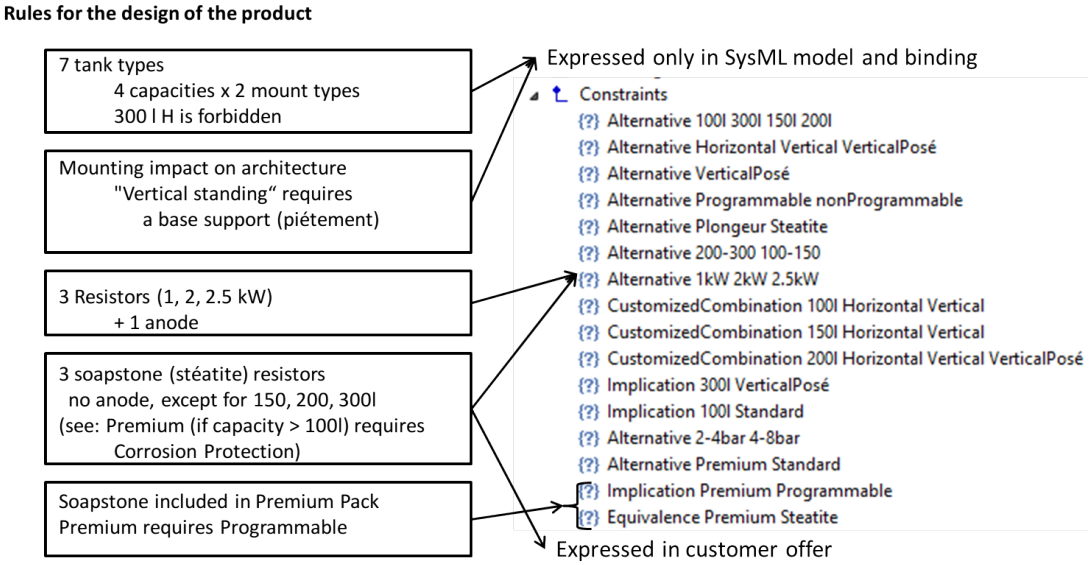


Figure A.33: Water heater design variability constraints

In addition, the system model contains generated constraints, for binding as propagation of optional elements, as in the other examples.

B

Improving the Configuration Process

RECOMMENDATION-based configuration improves the configuration process by employing recommendation techniques, such as heuristics. The contribution consists in collection of heuristics introduced in Chapter 5. This appendix brings some additional elements: an application of these heuristics to the Electric Parking Brake case study is presented in Section B.1. Section B.2 discusses the advantages of each of the heuristics and presents a synthesis of the assessment of the heuristics on the EPB case study.

B.1 Application to the configuration of a parking brake system

One particular question that can be raised about the configuration heuristics is *are they useful*? Although only long-term experience will provide a definitive answer to this question, one might be interested in looking for its implementation and its application

in a real case. To do that, we have (i) implemented our collection of heuristics in the constraint logic programming solver GNU Prolog [74], and (ii) applied these heuristics in typical configuration processes of our industrial case. For each heuristic, we measure the time required by the solver to generate N products; then, we compare these results with the ones obtained when we do not use any heuristics or use a contra-sense approach in the same configuration process. Our Electric Parking Brake Systems (EPB) model (c.f. appendices A.1.2 and C) is represented by means of the OVM notation [162] and is composed of 19 variation points, 46 variants, 16 alternative choices, 1 mandatory and 7 optional variability dependencies, and 28 additional dependencies (21 requires-type, 5 excludes-type, and 2 non-classified dependencies [142]). To present the feasibility of our approach and conclude what heuristics should be recommended for use during a product line configuration process, we first transform our EPB model into an automatically exploitable language, then we explain how to implement each heuristic and we compare the results gathered from their application to our industrial case.
Once our EPB model is represented as a GNU Prolog constraint program, we can apply our collection of heuristics to configure different Electric Parking Brake systems and measure the results obtained each time and compare them with the results obtained when no heuristic is used.

Heuristic 1: Variables with the smallest domain first The GNU Prolog predicate $fd\_labeling(Vars, Options)$ assigns a value to each variable of the list Vars according to the list of labeling options given by Options. Vars can be also a single finite domain variable. This predicate is re-executable on backtracking. *Options* is an optional list of labeling options that specifies the heuristic to select the variable to enumerate. When no option is specified, the solver selects the leftmost variable in the list of variables (Vars) to enumerate in the configuration process.
This heuristic is already implemented in GNU Prolog solver and can be used in a configuration process by means of the following predicate:

$$fd\_labeling(Vars, [variable\_method(ff)]),$$

Where $Vars$ is the list of variables of the constraint program (variation points and variants of our model) and $variable\_method(ff)$ is the GNU Prolog predicate offered by the solver to call the *ff* (first fail) heuristic in the current configuration process (i.e., in the current $fd\_labeling$).

Heuristic 2: The most constrained variables first To implement Heuristics 2 in GNU Prolog we just need to sort the list of variables (Vars), where the first variables in the list are those that are more constrained, and then use the predicate $fd\_labeling(Vars)$ in our configuration process. In our industrial case, *Vars = [PullerCable, ESPECU, PBSManual, ElectricActuator, HillStartAssistance, Static, TrailerW1, Permanent, DownsizedAndESPDynamicBraking, VSpeed, HSADisableFunction, TraditionalPB, DCMotor, DownsizedAndExtratorqueESP, AuxiliaryBrakeRelease,*

EU, BROnSystemDecision, BROnSystemDecision, TiltSensorIn1, TiltSensorIn2, TrailerW2, EPBEnabled, Comfort, CalipserWithIntegr, EActuator, PBSAssistance, FullEPB, Temporary, ClutchPres, DedicatedECU, Dynamic, BLOnSystemDecision, IntegratedDownsizedCalipers, GBAutomatic, ...] all the other variables appear one time in the constraints of the model.

Heuristic 3: Variables appearing in most products first Considering the explanation given, to implement Heuristics 3 in GNU Prolog we just need to calculate the variability factor of each variable of the model (this function is fully implemented in our tool VariaMos [145]) and sort the list of variables (Vars) according to the variability factor, being the first on the list those variables that have largest variability factor. Then, we use the predicate $fd\_labeling(Vars)$ in our configuration process to constraint the solver to use first the variables of Vars with the largest variability factor in the configuration process. The 10 variables with largest variability factor in our industrial case are: *ClutchPosition, DoorPosition,InputInformation, Static, ElectricalActionComponents, DCMotor, BrakingStrategy, VehicleTrailer, TrailerW2 and ForceMonitorOnEngineStop.*

Heuristic 4: Automatic completion when there is no choice Partial look-ahead [210] is about propagation on the min and max values of the variables' domains. Partial look-ahead is configured by default in GNU Prolog. For instance, given the following constraint expressed in GNU Prolog: $X\ \#= 2 * Y + 3,\ X\# < 10$ (where "#" before each constraint symbol forces the solver to apply a partial look-ahead propagation technique in the corresponding constraint and "," means a logic AND) the solver will define the following domain for both variables involved:

$$X = (3..9)$$
$$Y = (0..3)$$

Indeed, when the solver uses the partial look-ahead propagation technique, it only considers the border values to define the new domain of each variable after propagation. On the contrary, full look-ahead [210] allows operations about the whole domain in order to also propagate the "holes". Thus, if we use this technique in our constraint, i.e. $X\ \#= 2 * Y + 3,\ X\ \# < 10$ the solver will define a more precise domain:

$$X = (3 : 5 : 7 : 9)$$
$$Y = (0..3)$$

It is worth noting that to use full look-ahead in GNU Prolog, we just put "#" at the end of operations that will use full propagation.

Heuristic 5: Variables required by the latest configured variable first Because of this heuristic helps identify configuration conflicts as soon as possible in an interactive configuration process, it cannot be implemented by means of static list of variables sorted in a certain order as we did for the other ones. Conflicts that can

be identified with this heuristic look like "a configuration with *TraditionalBP* and an *DownsizedAndESPDynamicBraking* in a same product is not possible". This kind of configuration conflicts can, in certain situations, be avoided if people who configure the variant *DownsizedAndESPDynamicBraking* follow the requirements of this variant (e.g., the variant ESP_ECU) and not just the intuition or hazard to define the next variables to configure. Thus, the use of this heuristic is highly recommended to use in interactive and guided configuration environments.

Heuristic 6: Variables that split the problem space in two first This heuristic can be implemented as follows. First, it is necessary to find variables in which a configuration dichotomy must be done, i.e., assuming that one variable, belonging to the collection of variables to be configured in a certain configuration stage has a domain value of 10, the dichotomy consists of computing the number of results when the variable is less than 5, computing the number of results when the variable is greater than or equal to 5, compare both results and classify this variable according to its ability to divide the solution space. The difficulty while implementing this heuristic is to build this list of variables at each configuration step. A good choice in the context of product line models which are represented by graph like or tree like formalisms (e.g., Feature Models), is to begin the configuration process by the root feature and then navigate the tree structure to define what are the variables that most divide the solution space of the product line being configured. Since our industrial case is modeled in the OVM notation, where the notion of a single root does not exist. We will then need to consider all the variation points of our EPB model as roots. Thus, to implement this heuristic when the EPB model is represented as a GNU Prolog constraint program, the list L of variables corresponding to the constraint program is constituted in the following way: $L = [VP_1, VP_2, \ldots, VP_n, V_1, V_2, \ldots, V_n]$ where the collection of variables $VP_1, VP_2, \ldots, VP_n$ corresponding to the variation points of the EPB model are placed at the beginning of the list, and the collection of variables $V_1, V_2, \ldots, V_n$ corresponding to the variations of the model are placed at the end of the list. One or several of these decision points can be configured from the beginning of the configuration process by means of a partial configuration and the rest will be left to be configured by the solver (in the order defined by the list L) through the mechanism of propagation. In our industrial case, the configuration list L should looks like *L = [ParkingBrakeService, BrakeLock, BrakeRelease, HillStartAssistance, HSADisableFunction, RegulationZone, GearBox, VehicleTrailer, ArchitectureDesignAlternatives, ForceDistributionDesignAlternatives, SoftwareAllocationDesignAlternatives, TiltAngleFunctionAllocation, ElectricalActionComponents, BrakingStrategy, ForceMonitorOnEngineStop, InputInformation, ...]*. The rest of L is composed of variables corresponding to the variants of our OVM industrial case.

B.2 Discussion on the application of recommendation heuristics

We used two partial configurations to test our approach. The first partial configuration gives a value to the variables with the largest domain: *DCMotor, Static and FullEPB (DCMotor #= 10, Static #= 50, FullEPB #= 1000)*. The second partial configuration used to test our approach is typical for vehicles equipped with an automatic parking brake (*ParkingBrakeService #= 1, BrakeRelease #= 1, ForceDistributionDesignAlternatives #= 1, HSAAutomatic #= 0, GBAutomatic #= 1, DCMotor #= 10, Comfort #= 90, FullEPB #= 1000*).
The results of the assessment of heuristics are presented in tables B.1 and B.2.

	Time in ms, per each situation and each X solutions requested	100 solutions	1000 solutions	10000 solutions	100000 solutions	900000 solutions
H0	Without initial configuration	1	2	11	98	850
	With the first partial configuration of reference	0	2	11	90	870
	With the second partial configuration of reference	0	2	13	94	837
H1	Without initial configuration	0	1	6	48	440
	With the first partial configuration of reference	0	0	5	50	449
H2	Without initial configuration	1	3	16	125	1081
	With the second partial configuration of reference	1	3	15	118	916
	With a configuration in which all the most constrained variables are set-up and heuristics 1.	0	1	7	56	445
H3	Without initial configuration.	0	3	17	135	1142
	With a configuration in which the 9 variables with largest variability factor are set-up.	0	2	12	87	776
	With a configuration in which the 4 variables with largest variability factor are set-up and heuristics 1.	1	1	6	34	237

Table B.1: Results of the assessment of the proposed heuristics.

	Time in ms, per each situation and each X solutions requested	100 solutions	1000 solutions	10000 solutions	100000 solutions	900000 solutions
H4	Propagation with the Partial Look-Ahead algorithm (by default) and the second partial configuration of reference	0	2	12	93	831
	Propagation with Full Look-Ahead and the second partial configuration of reference	1	3	20	152	1323
H5	This heuristics helps identify configuration conflicts as soon as possible and not to reduce time spent in configuration. This heuristics has sense when is applied in an interactive configuration process. By these reasons this heuristics is not measured in the same way that the other ones.	N/A	N/A	N/A	N/A	N/A
H6	Without initial configuration of variation points	0	1	6	88	795
	With initial configuration of 8 (of 16) variation points	0	2	14	105	826
	With initial configuration of 8 (of 16) variation points and heuristics 1	0	1	9	52	434

Table B.2: Results of the assessment of the proposed heuristics.

We used the industrial case (the Electric Parking Brake System) to test our approach and get some insight on the improvements brought to the configuration process, by using the heuristics presented in this chapter. In order to do that, we used a typical configuration in automotive industry, where variables *ParkingBrakeService*, *BrakeRelease*, *ForceDistribution* and *GBAutomatic* are set to 1 to indicate that the reusable components corresponding to these variables are present into the cars that we intend to configure. In addition, the variable *HSAAutomatic* is set to 0 to indicate that the design alternative for disabling the hill start assistance will be not present in the product(s) that we intend to configure. Also, variable *DCMotor* is set to 10 to indicate the maximum power for the electric motor (design decisions), *Comfort* is set to 10 to indicate that when the system should switch to this braking strategy while the vehicle is in motion (design decisions regarding system behavior), and *FullEPB* is set to 1000 for the maximum allowed braking force.
We learn from the tests that Heuristics 1 (i.e., First the variables with the smallest domain) and 3 (i.e., First set the variables appearing in most products) are very useful when users need to get a fast feedback from the solver (this is the case, for instance, of online product configurators). In particular, we recommend to use Heuristic 1 because, in our case, it allows to reduce the time spend by the solver to propagate the configuration choices (feedback time). Also, we recommend using Heuristic 3 because, in our case, it allowed to reduce by 10%, on average, the feedback time.
Combining Heuristics 1 and 3 reduces even more the feedback time; which improves a lot the configuration process of large product line models. To be precise, this combination reduced, in our industrial case, by fourth the feedback time compared with the case when no heuristic is used in the configuration process. Nevertheless, we also recommend using Heuristic 2 combined with Heuristic 1 to reduce the computation time of the solver by half (on average, in our running case) the time spend by the solver to propagate our configuration preferences compared with the case where no heuristic is chosen.
The use of Heuristic 4 with application of the Full Look-Ahead algorithm (used by default in GNU Prolog to implement Heuristic 4) takes much more time than all the other tests; however, the configurations proposed by the solver after propagation of the configuration choices on the product line model are more accurate. Thus, using this algorithm, the solver never proposes an option that in reality the user cannot select later. This characteristic is very important in an iterative product line configuration process where the idea is to prevent false expectations about configurations that will in reality be impossible. Thus, we recommend the use of Heuristic 4 in configuration process where prevent the user mistake, his frustration and the subsequent abandon of the process is a more important issue compared with the long time spend by the solver to effectuate propagation and give a feedback.
From a usability point of view, the experiment shows that Heuristics 5 is useful in two cases: (i) to guide the users to continue the configuration process when they do not know how to continue the configuration process (i.e., they do not know which variables to consider next in the configuration process), and (ii) to test configurations at design-

time when engineers are calibrating the product line model. Heuristic 5 recommend setting first the variables required by the latest configured variable and in that way it helps identifying configuration conflicts as soon as possible in a configuration process (and not to reduce time spent in configuration as heuristics 1, 2 and 3). Heuristic 5 has sense when it is applied in an interactive configuration process to recommend users the next variables to configure without loosing the configuration sequence.
Our experiment also shows us that Heuristic 6 (i.e., First set the variables that split the problem space in two) only reduces the feedback time by 8% on average. This heuristics is implicit in the tree-like configuration processes like the one used by people that guide the configuration process by a feature-like product line model [124]. Even if this heuristics allows structuring the product line configuration processes by means of a predefined order, this is not always the best strategy (in terms of time and accuracy) to guide the configuration processes of industrial (often very large) product line models.
In our particular industrial case, we recommend to use Heuristic 3 combined with Heuristic 1 in order to reduce the computation time of the solver in the configuration process because this combination reduces by fourth the computation time compared with the case when no heuristics are used. Application of Heuristic 1 alone is also a good recommendation to improve computing time in our industrial product line configuration process. In that regards, we also recommend to use Heuristic 2 combined with Heuristic 1 to reduce the computation time of the solver because it is reduced by half when these two heuristics are applied with an initial configuration of the most restrictive variables. It is worth noting that the use of Heuristic 4 with application of the Full Look-Ahead algorithm has taken much more time than all the other tests; however, partial configurations proposed by the solver are more accurate, i.e., using this algorithm, the solver never proposes an option that the user cannot select later. This characteristic is very important in an iterative product line configuration process where the idea is to prevent false expectations about impossible configurations and thus prevent user mistakes, frustration and the subsequent abandon of the process. The other heuristics should be further evaluated to determine in which configuration situations and in which kind of models they should be recommended to use.

C

Automatic Parking Brake: System Stakeholder Requirements

C.1 Introduction

Presentation

This document describes all the requirements for the Automatic Parking Brake system and all its variants. In the context of the present document the Automatic Parking Brake shall be called EPB system (in English, Système FPE in French).

Referenced and applicable documents

- ECE R 13H (EU)

- FMVSS 135 (US)

Terminology and definitions

EPB, APB, FPA : different acronyms for the FPA system (detailed below)
EPB : Electric Parking Brake
APB : Automatic Parking Brake
FPA : Frein Parking Assistée
ABS : Anti Blocage de Roues. Anti Blocking System
AFU / EBA : Aide au Freinage d'Urgence. Emergency Brake Assistant
BFD : Répartition de freinage intelligente. Brake Force Distribution
ECU : Unité de contrôle électronique. Electronic Control Unit ESP : Contrôle dynamique de la trajectoire. Electronic Stability System
FdB/BB : Freinage de Base. Base Brake
IHM : Interface Home-Machine
APB : Automatic Parking Brake
ESC : Electronic Stability Control
MMAC : Maximum Authorized Mass in Circulation for the vehicle (Masse Maximale Autorise en Circulation)
MPB : Manual Parking Brake
MC : Master Cylinder
VODM : Empty and ready-to-use (Vide en Ordre De Marche)

C.2 Presentation of the system

C.2.1 Intended use of the system

System mission

MIS-EPB-1 Vehicle immobilization: The system shall ensure the immobilization of the vehicle during vehicle parking
MIS-EPB-2 – Safety brake: The system shall act as a safety brake during the vehicle movement
MIS-EPB-3 Hill start assistant: The system shall assist the driver when starting the vehicle on a sloped surface.

Variability requirements:
DIV-EPB-1 : The EPB system shall satisfy at least MIS-EPB-1 and MIS-EPB-2, regardless the variant.
DIV-EPB-2 : The EPB system shall offer a variant that assists the driver to start

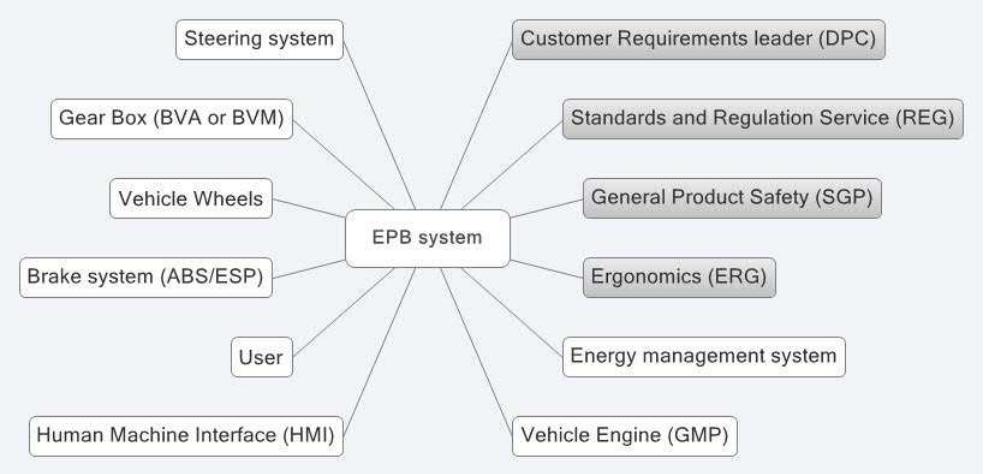


Figure C.1: Electric Parking Brake System environment

the vehicle on a sloaped road. This feature is named Hill Start Assistant (HSA)

System core concepts

CON-EPB-1 Automatic action: The system shall automatically engage parking brake when the vehicle is in parking state and release it when the vehicle leaves parking state.
CON-EPB-2 - Manual action: The system shall engage or release the parking brake on user request.

Variability requirements:
DIV-EPB-3 The EPB system shall satisfy at least requirement CON-EPB-2, regardless the variant.
DIV-EPB-4 The EPB system shall satisfy the requirement CON-EPB-1. DIV-EPB-4 requires satisfaction of DIV-EPB-3. In this configuration the system allows either manual action from the user in order to engage or disengage the parking brake, either does this action automatically without direct user intervention.

System environment

The physical and functional views of the system environment are presented in figures C.2 and C.3.

Position in the higher level system

The EPB system is a subsystem of the vehicle.

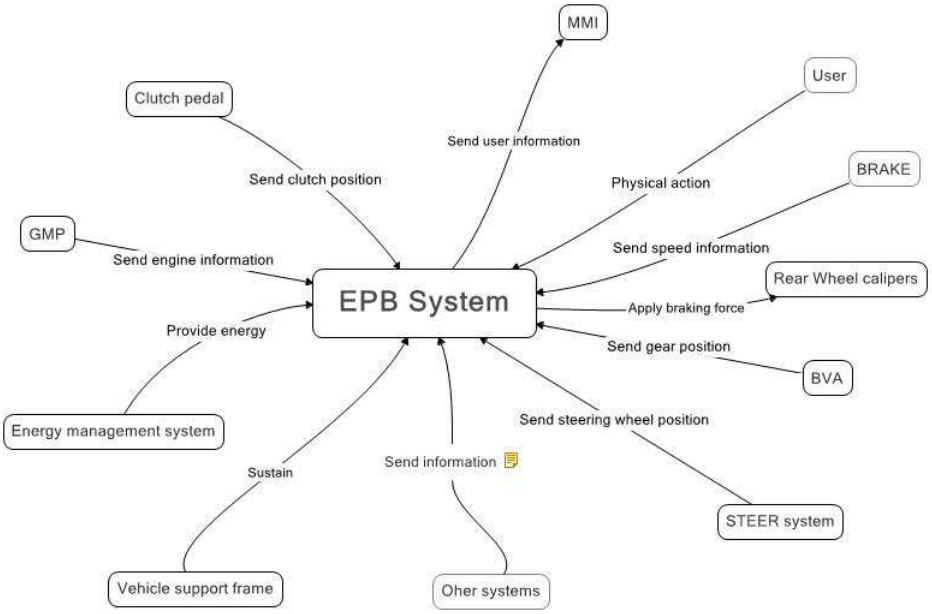


Figure C.2: Electric Parking Brake System functional boundary

Variability requirements: **DIV-EPB-boundaries** : Depending on the vehicle where the system shall be used the following elements are variable (there is a choice)

- Automatic Gear Box (BVA) or Manual gear gear box (BVM)
- BVM implies the existence of the Clutch pedal, BVA implies the absence of the clutch pedal

C.2.2 Life cycle stages (contexts, situations)

Vehicle behavior relative to the EPB system in presented in the diagram below:

C.2.3 Stakeholders

DPC Customer requirements leader (PPC)
REG Standards and regulation service
SGP General product safety
ERG Ergonomics

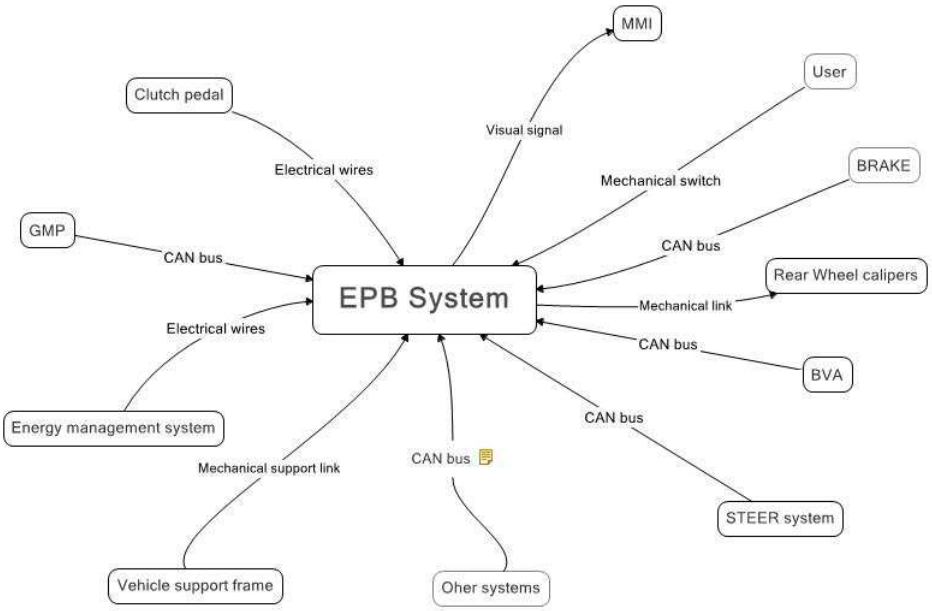


Figure C.3: Electric Parking Brake System physical boundary

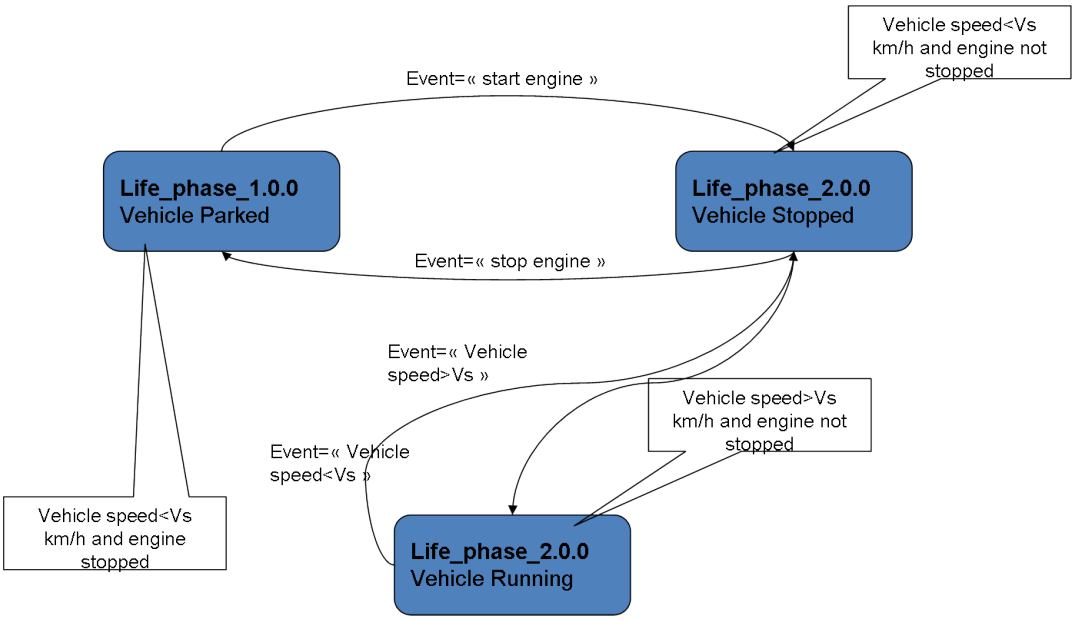


Figure C.4: States and Functioning mode graphs external point of view

Stage	Description
Life-phase-1.0.0	Vehicule parked (Vehicle speed ¡ Vs km/h and engine stopped)
Life-phase-1.1.0	Vehicule parked flat road
Life-phase-1.2.0	Vehicule parked sloped road
Life-phase-2.0.0	Vehicule stopped (Vehicle speed ¡ Vs km/h and engine not stopped)
Life-phase-2.1.0	Vehicule stopped flat road
Life-phase-2.2.0	Vehicule stopped sloped road
Life-phase-3.0.0	Vehicule running (Vehicle speed ¿= Vs Km/h)
Life-phase-4.0.0	Vehicle starting (Vehicle speed ¡ Vs km/h and engine starting)
Life-phase-4.0.1	Vehicle starting flat road
Life-phase-4.0.2	Vehicle starting sloaped road

Table C.1: Electric Parking Brake System life cycle stages

C.2.4 EPB Use cases

The behavior of the parking brake system is related to the states the of the vehicle, introduced in Figure C.5. Part of the operational analysis, the use cases help reveal the behavior of the system during usage. The existing specification were used as input to define the use cases, system stakeholder requirements, as well as the *variability* related to these items. The variability was initially described textually, which helped prepare the case study, referenced by *DIV–UC–EPB–x* for variability related to use cases, and *DIV–EPB-x* for requirements variability. While the use cases allow for a better understanding of the EPB case study, and are presented below, the EPB system requirements are described in Appendix C.

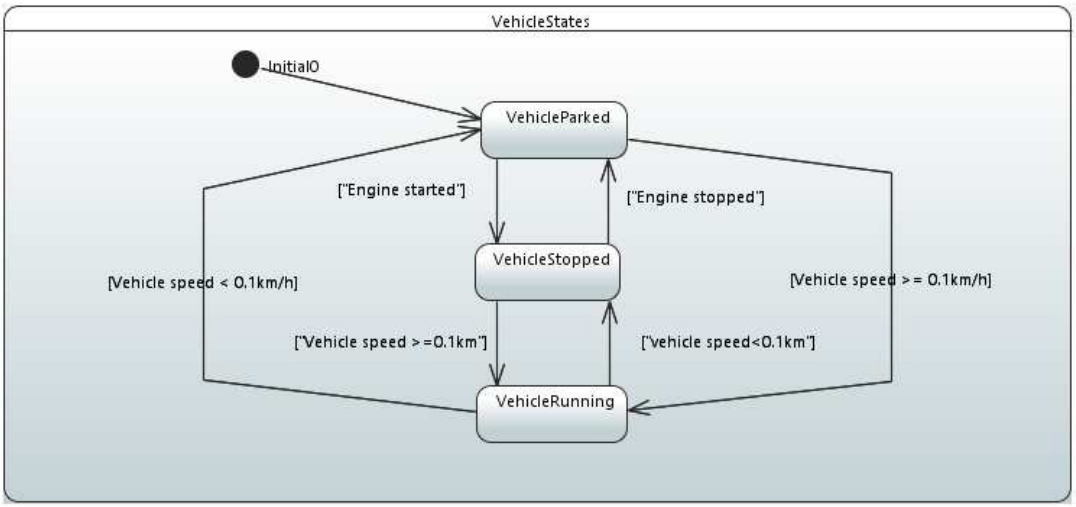


Figure C.5: Vehicle states related to the functioning of the EPB system

The behavior of the EPB system is analyzed in respect to three contexts: (i) the vehicle is parked - engine is stopped and the vehicle speed is 0 ($V_{vehicle} < 0.1km/h$); (ii) the vehicle is stopped - engine is running and vehicle speed is 0; (iii) the vehicle is running - $V - vehicle > 0.1km/h$ and the engine is running. The expected behavior of the EPB system is described in the following use cases.

Vehicle parked

UC-EPB-1 : The system holds the vehicle locked into position, on the parking area.
Initial context : The parking brake is engaged, the vehicle wheels locked and the EPB system is monitoring the brake force (alternatively the position of the wheels or the temperature of the actuator).
Trigger : The vehicle moves or the braking force reduced (due to heat dissipation effects, contraction of components)
Actions :

1. The system monitors, verifies if the force applied on the calipers has changed.
2. Alternatively, the system can verify if the vehicle wheel has moved.
3. Alternatively, the system can anticipate component contraction effects due to heat dissipation and verify if a change of temperature from high to low has occurred, since the brake has been applied (and command/ force calculated).
4. If the pressure on the calipers has reduced or the wheel has moved, the system adjusts the braking force accordingly.
5. If needed the actions are iterated again from 1 and continued as long as needed. (exit from the iteration may be imposed by technology or technical solution)

Final context : The parking brake is engaged, the vehicle wheels are locked.

Vehicle stopped

UC-EPB-2 : The system engages the parking brake on the driver's command (Manual engage)
Initial context : The system holds the parking brake disengaged/in released position. ($V_s < 0.1km/h$)
Trigger : The driver applies the command to engage the parking brake.
Actions :

1. The system detects drivers intention to apply the parking brake.
2. The system verifies the speed is below Vs threshold.

3. The system applies the required force on the calipers according the external conditions (slope, vehicle constraints)

Final context : The system holds the parking brake engaged.

UC-EPB-3 : The system disengages the parking brake on the drivers command (Manual release)
Initial context : The system holds the parking brake engaged. ($V_stop < 0.1km/h$)
Trigger : The driver applies the command to disengage the parking brake.
Actions :

1. The system detects driver's intention to release the parking brake : the EPB switch (HMI for operating the system) in the required position and brake pedal pressed.
2. The system verifies the speed is below Vs threshold.
3. The system applies the required force to release the force applied on the calipers.

Final context : The system holds the parking brake disengaged/in released position.

UC-EPB-4 : The system gradually disengages the parking brake (Hill start assistance)
Initial context : The system holds the parking brake engaged.
Trigger : The driver presses the acceleration pedal above a specified threshold (A_{thr}), while the gear is engaged
Note: The user action is prioritary. A new command while the scenario takes place, will interrupt the current scenario.
Actions :

1. The system detects the driver's intention to put the vehicle into motion.
2. The system gradually releases the braking force in order to avoid movement in the opposite direction to the users commands.
3. The system fully releases the braking force once the vehicle speed is above a certain threshold V-release . Final context : The vehicle is running, the system holds the parking brake released.

DIV-UC-EPB-4 : The variation applies to the trigger context of UC-EPB-4. In addition to the given conditions, the clutch torque has to be above C_{thr} threshold if the vehicle hosting the EPB system is equipped with a manual gear box or the estimated power train wheel torque has to be above the specified wheel torque threshold (WT_{thr}) if the vehicle hosting the EPB system is equipped with an automatic gear box.

Vehicle running

UC-EPB-5 : The system gradually decreases the vehicle speed up to a full stop. (Comfort braking and static braking scenario 1)
Initial context : The system holds the parking brake disengaged. Trigger : The driver applies the command to engage the parking brake and maintains the command. (pulls the switch and maintains its position)
Actions :

1. The system gradually applies the pressure for braking on the calipers, the vehicle speed decreases.
2. If the vehicle speed decrease until it reaches a threshold V_{stop} ($V_{stop} = 0.1km/h$), when the system applies the full pressure on the calipers in order to immobilize the vehicle (static braking).(Below the V stop threshold, the life phase actually changes, the vehicle being considered as stopped, engine running)

Final context: The vehicle is stopped , the system holds the parking brake engaged.

UC-EPB-6: The EPB system gradually decreases the vehicle speed, and continues running (Comfort braking scenario 2)
Initial-context: The system gradually brakes in order to decrease the vehicle speed, while the driver holds the parking brake command on.
Trigger : The user releases the parking brake command.
Actions:

1. The system detects that the vehicle speed is still above V_{stop} threshold ($V_{stop} = 0.1km/h$)
2. The system releases the braking force.

Final context: The vehicle keeps running.

DIV-UC-EPB-6 The use cases UC-EPB-5 and UC-EPB-6 are available for any EPB system variant.

Vehicle stopping, flat road

UC-EPB-7 : the system automatically engages the parking brake.
Initial context : the vehicle is stopped (not in motion, but engine running), the system holds the parking brake disengaged.
Trigger : the driver applies the command to stop the engine.
Actions :

1. The system detects the driver's intention to stop the engine.

2. The system applies the parking brake.

Final context : the system holds the parking brake engaged.
DIV-UC- EPB-7: Satisfying the behavior described by UC-EBP-7 for any EPB variant requires CON-EPB-1 (Automatic action concept).

UC-EPB-8 : the system automatically engages the parking brake (Degraded mode: different trigger conditions & context)
Initial context : the system holds the parking brake disengaged, vehicle may be running at a speed below Vs (Vs¡km/h)
Trigger : the driver opens the driver side door.
Actions :

1. The system detects the fact that the driver side door is open and the vehicle speed is below $V_{stop}(V_{stop} < km/h)$.

2. The system applies the parking brake.

Final context : The system holds the parking brake engaged.

DIV-UC-EPB-8: Satisfying the behavior described by UC-EBP-7 for any EPB variant requires CON-EPB-1 (Automatic action concept).

Vehicle stopping, sloped road

UC-EPB-9 : the system automatically releases the parking brake at the right moment, on order to prevent backward movement, when the driver wishes to start the vehicle on a slope.
Initial context : The vehicle is stopped on a sloped road (not in motion, but engine running), the system holds the parking brake engaged.
Trigger : The driver puts the gear in the uphill direction and presses the accelerating pedal.
Actions :

1. The system detects the driver's intention to start on a sloped road (steeper than 3%)

2. The system holds the parking brake engaged as long as the engine torque is lower than the necessary torque to put the vehicle in uphill motion.

3. If the torque is enough to start the vehicle uphill, the system releases the parking brake Final context : The system holds the parking brake in released position and the vehicle in uphill motion.

DIV-UC-EP-8: Satisfying the behavior described by UC-EBP-9 for any EPB variant implies CON-EPB-1 (Automatic action concept).

UC-EPB-10 : the system automatically releases the parking brake at the right moment, on order to prevent backward movement, when the driver wishes to start the vehicle on a slope, using the BB (basic brake).
Initial context : the vehicle is stopped on a sloped road (not in motion, but engine running), parking brake is disengaged (automatically or manually released) , the user holds the brake pedal pressed.
Trigger : the driver puts the gear in the uphill direction and releases the break pedal.
Actions :

1. The system detects the driver's intention to start on a sloped road (steeper than 3%)
2. The system holds the parking brake engaged as long as the engine torque is lower than the necessary torque to put the vehicle in uphill motion and a given maximum time has not passed (2 seconds)
3. If the wheel torque is enough to start the vehicle uphill, the system releases the parking brake.
4. If the maximum time passes (2 seconds), the system will progressively release the brakes (BB).

Final context : the system holds the parking brake released , BB (basic brakes) are released, vehicle is in uphill movement.

DIV-UC-EP-8: satisfying the behavior described by UC-EBP-10 for any EPB variant implies CON-EPB-1 (Automatic action concept).

UC-EPB-11 : the system automatically releases the parking brake at the right moment, on order to prevent backward movement, when the driver wishes to start the vehicle on a slope, using the BB (basic brake).
Initial context : the vehicle is stopped on a sloped road (not in motion, but engine running), parking brake is disengaged (automatically or manually released), the user holds the brake pedal pressed.
Trigger : the driver puts the gear in the uphill direction and releases the break pedal.
Actions :

1. The system detects the driver's intention to start on a sloped road (steeper than 3
2. The system holds the parking brake engaged as long as the engine torque is lower than the necessary torque to put the vehicle in uphill motion and a given maximum time has not passed (2 seconds)

3. If the system detects that the driver no longer intends to start the vehicle, the system releases the brakes.

4. If the system detects that wheels are slipping/ skidding, the system releases the brakes.

Final context : the vehicle is stopped or parked (not in motion).
DIV-UC-EP-8: Satisfying the behavior described by UC-EBP-10 for any EPB variant implies CON-EPB-1 (Automatic action concept).

UC-EPB-8 also applies for this life phase.

C.3 Stakeholder Requirements

Customer Requirements Leader (DPC)

REQ-EPB-DPC-0001 : The EPB system shall be able to capture the user intention (to engage or release the parking brake) through a dedicated interface. Source : DPC
REQ-EPB-DPC-0002 : The EPB system shall apply the user intention depending on the vehicle state (speed, engine state, steering wheel state). Source : DPC
REQ-EPB-DPC-0003 : The EPB system shall maintain the vehicle fully loaded at a standstill up to Smax (Slope max). The Smax depends on grip, on country... Source : DPC
REQ-EPB-DPC-0004 : (Use case) When the vehicle is parked and the parking brake is released, the EPB system shall engage the parking brake if

- either, the driver ask for the vehicle immobilization through the dedicated interface.
- or, the automatic mode is active (if the functionality is present) and the conditions to automatically engage the parking brake are fulfilled.

Source : DPC
REQ-EPB-DPC-0005 : (Use case) When the vehicle is parked and the parking brake is engaged, the EPB system shall release the parking brake if

- either, the driver asks for the vehicle release through the dedicated interface
- or, the automatic mode is active (if the functionality is present) and the conditions to automatically release the parking brake are fulfilled.

Source : DPC
REQ-EPB-DPC-0006 : (Use case) When the vehicle is stopped and the parking brake is released, the EPB system shall engage the parking brake if

- either, the driver ask for the vehicle immobilization through the dedicated interface.
- or, the automatic mode is active (if the functionality is present) and the conditions to automatically engage the parking brake are fulfilled.

Source : DPC
REQ-EPB-DPC-0007 : (Use case) When the vehicle is stopped and the parking brake is engaged, the EPB system shall release the parking brake if

- either, the driver ask for the vehicle release through the dedicated interface.
- or, the automatic mode is active (if the functionality is present) and the conditions to automatically release the parking brake are fulfilled.

Source : DPC
REQ-EPB-DPC-0008 : (Use case) When the vehicle is running and the parking brake is released, the EPB system shall brake the wheels if the driver ask for the vehicle deceleration through the dedicated interface. Source : DPC
REQ-EPB-DPC-0009 : (Use case) When the vehicle is stopped on a sloped road and the driver starts the vehicle, the system shall assist the driver in his action, in order to prevent that the vehicle moves in the opposite direction to the desired one. Source : DPC

Variability requirements:
DIV-EPB-5 The system shall satisfy requirements REQ-EPB-DPC-1 to REQ-EPB-DPC-8. **DIV-EPB-51** The system that includes the optional feature HSA (that assists the driver to take off on a sloaped road) shall satisfy REQ-EPB-DPC-0009.

Standards and Regulation Service (REG)

REQ-EPB-REG-0001 : The APB system shall be dimensioned and defined to meet the requirements of the legislative provisions specified by the regulations currently in force, or the future application of which is known, in the various countries where the vehicle is intended to be marketed. Source : REG
REQ-EPB-REG-0002 : The manual application of the parking brake shall be permanently possible . Source: ECE R13H, paragraph 5.2.2 ; FMVSS 135, paragraph S5.3.1
REQ-EPB-REG-0003 : The EPB system must make it possible to hold the vehicle stationary on an up or down gradient even in the absence of the driver, the working parts being then held in the locked position by a purely mechanical device. Source : ECE R13H, paragraph 5.1.2.3 ; FMVSS 135, paragraph S5.2.
REQ-EPB-REG-0004 : The control of the EPB system shall be independent of the service brake control. Source: ECE R13H, paragraph 5.2.2 ; FMVSS 135, paragraph S5.3.1

REQ-EPB-REG-0005 : The application of the parking brake shall be indicated to the driver by a red warning signal. Source: ECE R13H, paragraph 5.2.3 ; FMVSS 135, paragraph S5.5
REQ-EPB-REG-0006 : In the case of a failure, which prevents the release of the parking brake by use of the control from the drivers seat, at least release shall be possible by the use of tools and/or an auxiliary device carried/fitted on the vehicle. Source: ECE R13H, paragraph 5.2.19.2.
REQ-EPB-REG-0007 : After the ignition/start switch which controls the electrical energy for the braking equipment has been switched off and/or the key removed, it shall remain possible to apply the parking braking system, whereas releasing shall be prevented. Source: ECE R13H, paragraph 5.2.19.4.

Variability requirements:
DIV-EPB-6 The system shall satisfy requirements REQ-EPB-REG-0001 to REQ-EPB-REG-0007, when the vehicle integrating the system is to be sold in US and all countries that apply regulation FMVSS 135.
DIV-EPB-7 The system shall satisfy requirements REQ-EPB-REG-0001 to REQ-EPB-REG-0005 when the vehicle integrating the system is to be sold in EU and all countries that apply requlation ECE R13H.

General Product Safety (SGP)

REQ-EPB-SGP-0001 : The EPB system shall have an auxiliary control for emergency release. Source : SGP
REQ-EPB-SGP-0002 : The system shall offer the possiblity to the user to explicitly request a maximum parking brake application. (may be required in situations such as towing, transport, trailer not braked) Source : SGP
REQ-EPB-SGP-0003 : The system shall either self adjust its behavior to be able to fullfil the hill start assistance operation when the vehicle has a towed trailer, or it will allow deactivation of the hill start assistant functionality by the driver. Source : SGP
REQ-EPB-SGP-0004 : The system shall check that no residual braking force is applied. Source : SGP

Ergonomics

REQ-EPB-ERG-0001 : The system utilisation shall be intuitive, and the APB system operation shall be natural with respect to usual driving habits. Source : ERG
REQ-EPB-ERG-0002 : Shall have an electrical control which is easily accessible from the driver's place for brake application and release. Source : ERG

Solution alternatives/hypothesis

- **H0** : EPB architecture including clutch sensor, palette, tilt sensor;
- **H1** : Puller cable architecture with integrated ECU;
- **H2** : Calipers with integrated actuators and separate specific ECU;
- **H3** : Puller cable with software allocated to ESP ECU;
- **H4** : Calipers with integrated actuators and software allocated to ESP ECU.
- **H5** : Downsized puller cable architecture with extra-torque function implemented by the ESP.
- **H6** : Downsized puller cable architecture, and dynamic braking function implemented by the ESP.
- **H7** : Downsized architecture based on calipers with integrated actuators, and dynamic braking function implemented by the ESP.
- **H8** : EPB architecture without tilt sensor.
- **H9** : EPB architecture using tilt sensor information from the ESP (in ESP cluster).
- **H10** : EPB architecture using tilt sensor information from the ESP (in ESP ECU).

D

Variability Concepts for Families of Systems

The complete metamodel explained in Chapter 4 is presented in Figure D.1.

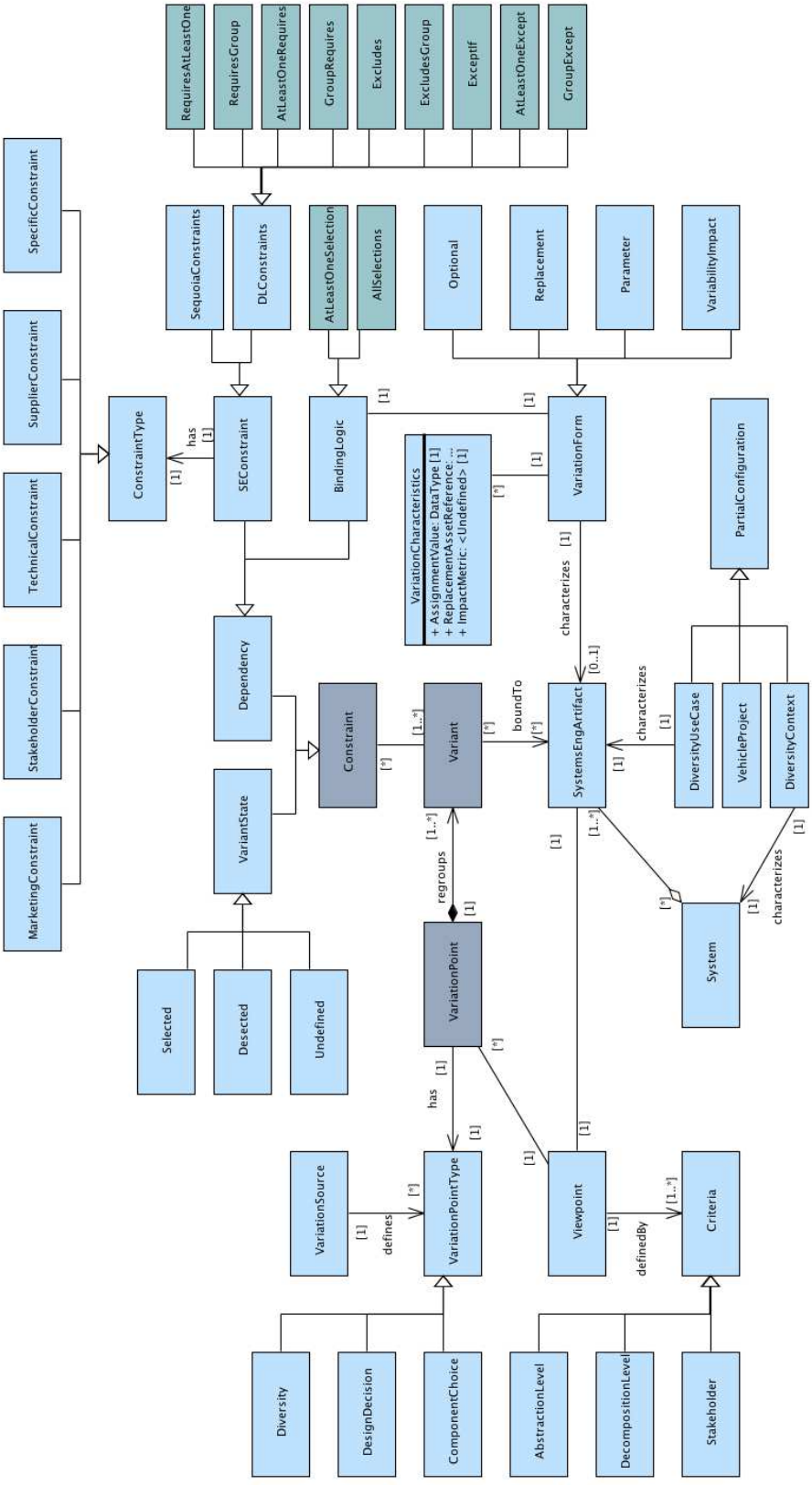


Figure D.1: Variability concepts for families of systems

E

Some examples from the MBSSE tool implementation

The examples presented here reflect the result if the MBSSE project and include the contribution and implementation of the CEA LIST [197].

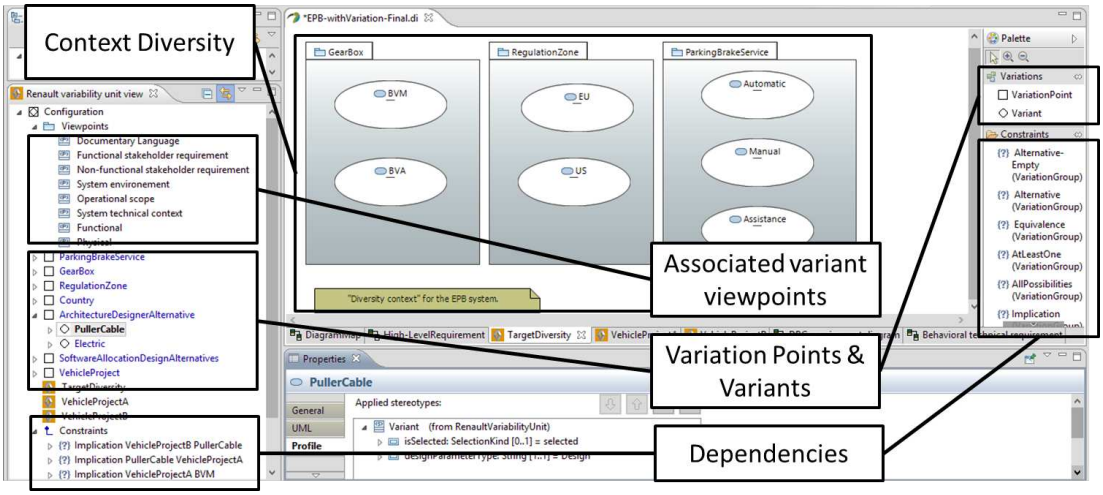


Figure E.1: Papyrus modeler including extensions for variability modeling Co-OVM

The system viewpoints play an important role in the configuration methodology, where the relative order of choices can be configured, through the menu presented in figure E.2.

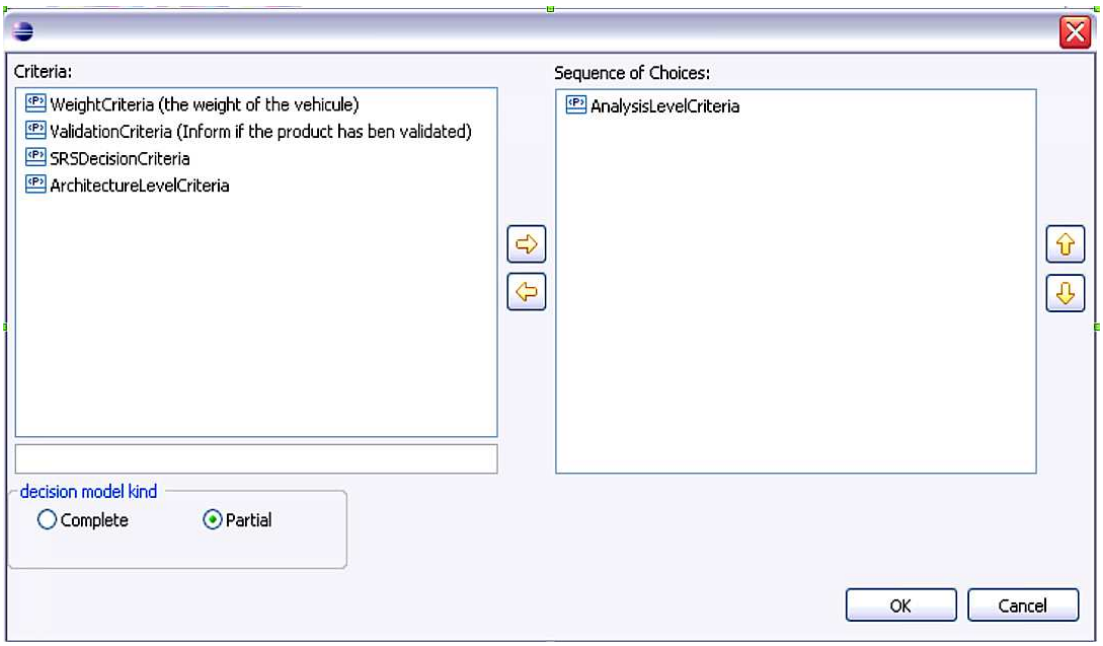


Figure E.2: Configuration of the partial derivation steps

In the current implementation variants are associated manually with the viewpoints where they are visible, except on import from the Renault Documentary Language. On import variants are automatically associated with the “Vehicle Features” viewpoint.

The *source* of the variation point designates the view where the variation originates from. Figure E.3 shows how associations are done between variants and viewpoints. Future work should address automation for these associations (e.g. based on bindings to system items and their viewpoints).

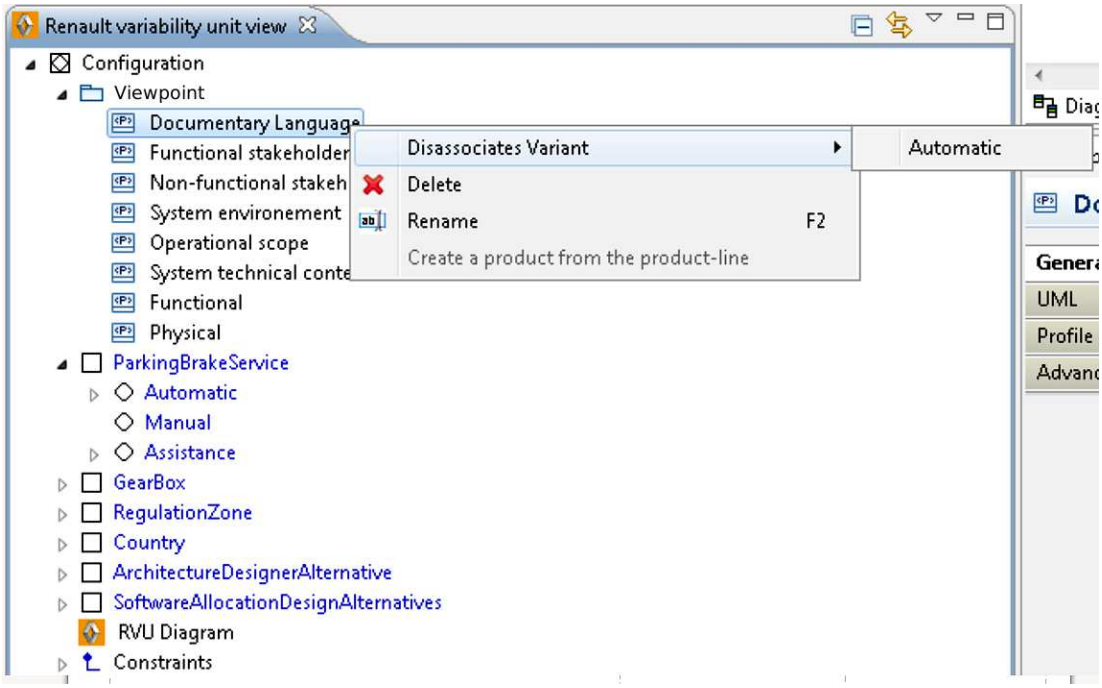


Figure E.3: Dissociating variants from viewpoints

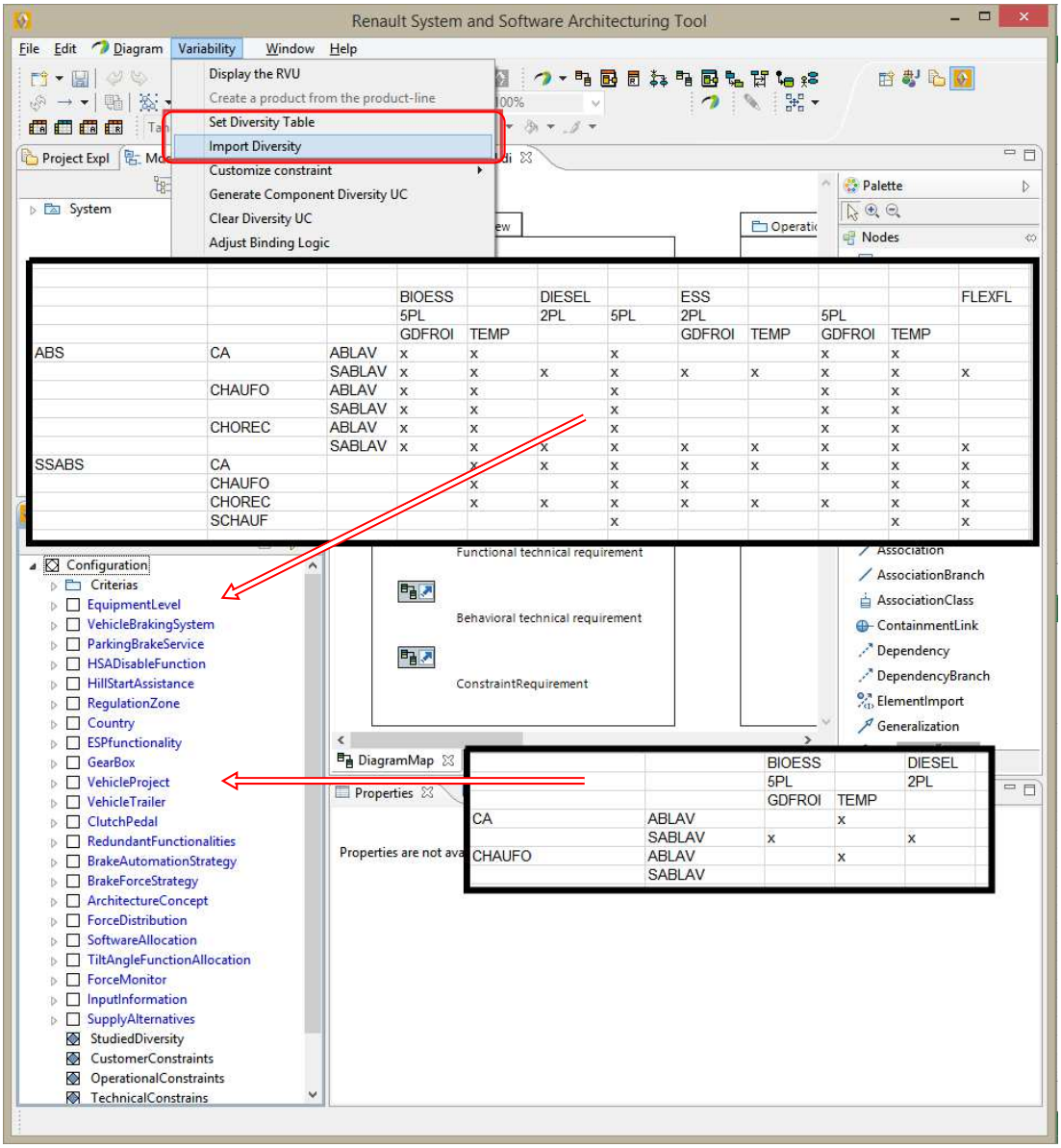


Figure E.4: Associating a Documentary Language configuration table to the variability model

F

Papyrus implementation examples

Some implementation examples for extended variability functionalities are presented here. They are provided here as reference for future extensions that shall be brought to the variability management framework at Renault. Variability concepts were implemented as a collection of UML profiles in Papyrus, most of them as part of the MBSSE project. Figure F.2 presents an additional UML Profile, which extends the binding of variants to system items, according to the metamodel explained in Chapter 4. The rules for binding are implemented based on the set of custom Sequoia constraints.

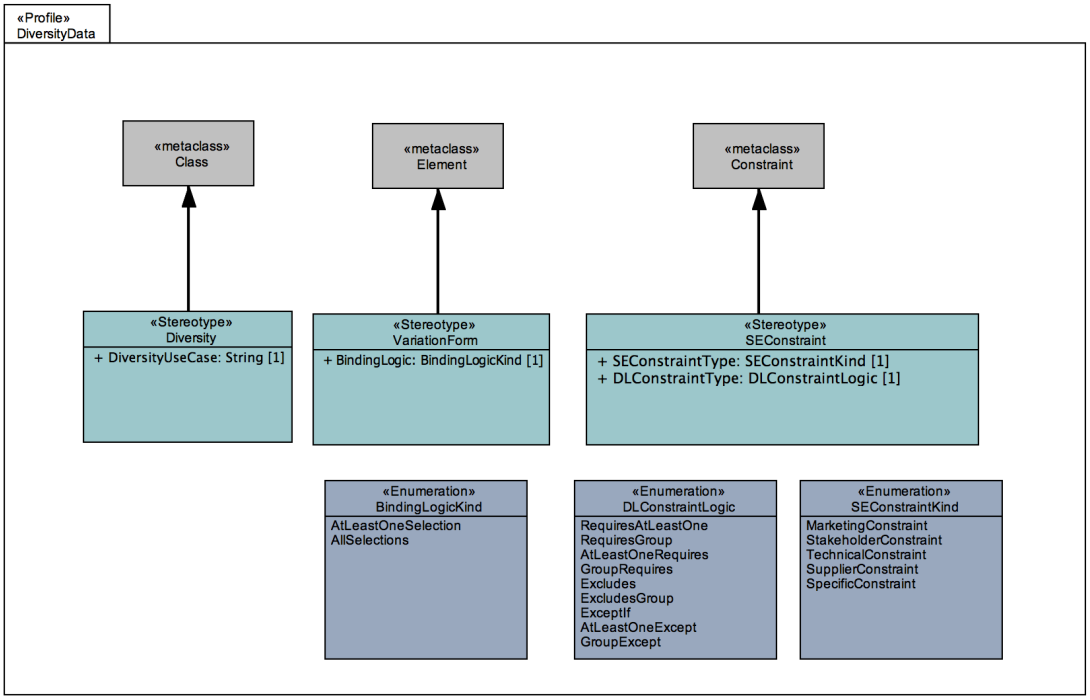


Figure F.2

Figure F.3 presents the Renault variability management Profile (the main UML Profile).

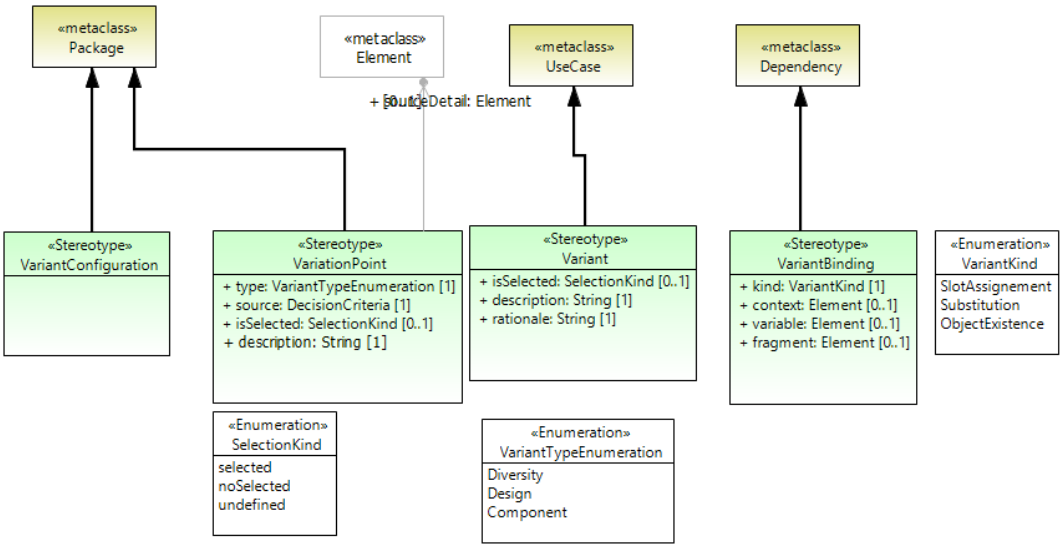


Figure F.3: MBSSE variability management UML Profile

A few modifications were brought to the initial version from the MBSSE project, as defined by the metamodel in Chapter 4 [197]:

- Variation point have the three types – Diversity, Design, Component.
- Variation point are traced to their *source*. When imported from the Renault Documentary Language, this property is set automatically.
- Variants have an optional field, the *rationale*. This can be used in conjunction with *Design* or *Component* Variants, which require engineering domain knowledge.
- Description fields for both variants and variation points, for any additional information the user might want to include.

Apart from binding, the custom Sequoia constraints are also used for system modeling purposes. Their implementation is described in the code Listing F.1.

Listing F.1: Customization of logical constraints based on Sequoia

```
package com.renault.diversitytable.impact;


import java.util.ArrayList;


import org.eclipse.emf.transaction.TransactionalEditingDomain;
import org.eclipse.papyrus.infra.core.utils.ServiceUtilsForActionHandlers
    ;
import org.eclipse.uml2.uml.Constraint;
import org.eclipse.uml2.uml.NamedElement;
import org.eclipse.uml2.uml.Stereotype;


import com.cea.sequoia.ISequoia_Stereotypes;
import com.renault.diversitytable.commands.CustomizePropagatedVGCommand;


/**
 * Customizes propagated constraints
 * The customized constraints are checked for the stereotype
     PropagatedVariableElement
 * @author dcosmin
```

```
 *
 */
public class CustomConstraintEditor {

  public static final String D_AtLeastOneRequires = "AtLeastOneRequires";
  public static final String D_ExcludesGroup = "ExcludesGroup";
  public static final String D_Exclusion = "Exclusion";
  public static final String D_GroupRequires = "GroupRequires";
  public static final String D_RequiresAtLeastOne = "RequiresAtLeastOne";
  public static final String D_RequiresGroup = "RequiresGroup";
  public static final String D_AllMissingExcludes = "AllMissingExcludes";
       //custom constraint for allocation propagation
  public static final String D_ConstraintName1 = "VariabilityConstraint";
  public static final String D_ConstraintName2 = "VariationGroup";

  private TransactionalEditingDomain transactionalEditingDomain;
  private String constraint_type;
  private NamedElement UMLelement;

  public CustomConstraintEditor(String constraint_type, NamedElement
      UMLelement) {
    this.constraint_type = constraint_type;
    this.UMLelement = UMLelement;
  }

  public Boolean customizeConstraint(){
    String constraint_expression = "";
    NamedElement[] constrained_elements;
    Stereotype st = null;

    if (UMLelement instanceof Constraint) {
      st = UMLelement.getAppliedStereotype(ISequoia_Stereotypes.
```

```
    PROPAGATEDVARIATIONGROUP_STEREOTYPE);
if (st != null){
  //get list of constrained elements
  Object elements = UMLelement.getValue(st, "variableElement");
  if (elements instanceof org.eclipse.emf.ecore.util.
      EObjectResolvingEList) {
  org.eclipse.emf.ecore.util.EObjectResolvingEList<Object>
      constrained_elements__ = (org.eclipse.emf.ecore.util.
      EObjectResolvingEList<Object>) elements;
  constrained_elements = (NamedElement[]) constrained_elements__.
      toArray();
  } else {
    return false;//failure
  }
  if (constraint_type.contentEquals(D_AtLeastOneRequires)){
    /*
     * (B or C or D or ...) => A
     * CNF: (non B or A) and (non C or A) and (non D or A) and ...
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and";
    for (int i=0; i < constrained_elements.length-1; i++){
      constraint_expression = constraint_expression + " (or \"" +
          constrained_elements[constrained_elements.length-1].
          getQualifiedName() + "\" !\"" + constrained_elements[i].
          getQualifiedName() + "\")";
    }
    constraint_expression = constraint_expression + ")";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
        UMLelement, constraint_expression, D_AtLeastOneRequires);
```

```
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_ExcludesGroup)){
    /*
     *  A => non (B and C and D and ...)
     *  CNF: non A or non B or non C or non D or ...
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and (or";
    for (int i=0; i<constrained_elements.length-1;i++){
      constraint_expression = constraint_expression + " !\"" +
          constrained_elements[i].getQualifiedName() + "\"";
    }
    constraint_expression = constraint_expression + " !\"" +
        constrained_elements[constrained_elements.length-1].
        getQualifiedName()+ "\"))";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
        UMLelement, constraint_expression, D_ExcludesGroup);
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_Exclusion)){
    /*
     *  A => non B
     *  CNF: non A or non B
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and (or !\"" + constrained_elements
        [1].getQualifiedName() +"\" !\"" + constrained_elements[0].
        getQualifiedName() +"\"))";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
```

```
        UMLelement, constraint_expression, D_Exclusion);
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_GroupRequires)) {
    /*
     * (B and C and D ...) => A
     * CNF: non B or non C or non D or ... or A
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and (or";
    for (int i=0; i<constrained_elements.length-1;i++){
      constraint_expression = constraint_expression + " !\"" +
          constrained_elements[i].getQualifiedName() + "\"";
    }
    constraint_expression = constraint_expression + " \"" +
        constrained_elements[constrained_elements.length-1].
        getQualifiedName()+ "\"))";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
        UMLelement, constraint_expression, D_GroupRequires);
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_RequiresAtLeastOne)){
    /*
     *  A => (B or C or D ...)
     *  CNF: non A or B or C or ...
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and (or";
    for (int i=1; i<constrained_elements.length;i++){
      constraint_expression = constraint_expression + " \"" +
          constrained_elements[i].getQualifiedName() + "\"";
```

```
    }
    constraint_expression = constraint_expression + " !\"" +
        constrained_elements[0].getQualifiedName()+ "\"))";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
        UMLelement, constraint_expression, D_RequiresAtLeastOne);
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_RequiresGroup)){
    /*
     * A => (B and C and D)
     * CNF: (non A or B) and (non A or C) and (non A or D) and ...
     */
    setTransactionalEditDomain();
    //build expression
    constraint_expression = "(and";
    for (int i=1; i < constrained_elements.length; i++){
      constraint_expression = constraint_expression + " (or !\"" +
          constrained_elements[0].getQualifiedName() + "\" \"" +
          constrained_elements[i].getQualifiedName() + "\")";
    }
    constraint_expression = constraint_expression + ")";
    CustomizePropagatedVGCommand cmd = new
        CustomizePropagatedVGCommand(transactionalEditingDomain,
        UMLelement, constraint_expression, D_RequiresGroup);
    transactionalEditingDomain.getCommandStack().execute(cmd);
  } else if (constraint_type.contentEquals(D_AllMissingExcludes)){
    /*
     * (non A and non B and non C and ..) => non D
     * CNF: non ( ... ) or non D
     *    A or B or C ... or non D
     */
    setTransactionalEditDomain();
```

```
        //build expression
        constraint_expression = "(and (or";
        for (int i=0; i<constrained_elements.length-1;i++){
          constraint_expression = constraint_expression + " \"" +
              constrained_elements[i].getQualifiedName() + "\"";
        }
        constraint_expression = constraint_expression + " !\"" +
            constrained_elements[constrained_elements.length-1].
            getQualifiedName()+ "\"))";
        CustomizePropagatedVGCommand cmd = new
            CustomizePropagatedVGCommand(transactionalEditingDomain,
            UMLelement, constraint_expression, D_AllMissingExcludes);
        transactionalEditingDomain.getCommandStack().execute(cmd);
      }

    }
  }
  Object cstr = null;
  if (st != null) {
    cstr = UMLelement.getValue(st, "constraint");
  }
  if (cstr!=null && constraint_expression.contentEquals((CharSequence)
      cstr)) {
    return true;//Success
  }
  return false;

}

/**
 * Get the type of constraint, whether it is a Sequoia or custom
     constraint
```

```
     * @return type of constraint as string, or null if failure to find
         type
     */
    public String getConstraintType(){

      return null;
    }

    /**
     * Get Constrained elements for the given constraint.
     * @return Ordered array of elements
     */
    public NamedElement[] getConstrainedElements(){
      return null;

    }

    private void setTransactionalEditDomain(){
      ServiceUtilsForActionHandlers util = new
          ServiceUtilsForActionHandlers();
      try {
        this.transactionalEditingDomain =util.getTransactionalEditingDomain
            ();
      } catch (Exception e) {
        System.err.println("impossible to get the Transactional Editing
            Domain "+e);
      }
    }

  }
```

Code Listing F.2 implements the propagation of optional system elements through allocation of functions to components. The methods are called by the variability prop-

agation functionality present in Sequoia.

Listing F.2: Propagation based on allocation of functions to components

```
package com.renault.diversitytable.impact;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;

import org.eclipse.uml2.uml.Abstraction;
import org.eclipse.uml2.uml.Element;
import org.eclipse.uml2.uml.NamedElement;
import org.eclipse.uml2.uml.Package;
import org.eclipse.uml2.uml.Stereotype;
import org.eclipse.core.internal.utils.Convert;
import org.eclipse.emf.common.util.EList;
import com.cea.sequoia.ISequoia_Stereotypes;
import com.renault.diversitytable.impact.SystemModelAnalysis;

import com.cea.sequoia.propagationtool.IPropagationRule;

public class FunctionAllocationRule implements IPropagationRule {

  public static final String RenaultComponent_Sterotype = "
      RenaultSystemArchitecturing::PhysicalComponent";
  public static final String RenaultFunction_Sterotype = "
      RenaultSystemArchitecturing::FunctionalBlock";
  public static final String Allocation_Stereotype = "SysML::Allocations
      ::Allocate";
  public static Integer element_cnt = 0;

  private String pluginName;
```

```
public FunctionAllocationRule() {
  pluginName = "FunctionAllocationRule";
}

@Override
public String getName() {
  return "System Function Allocation Rule";
}

@Override
public String getDescription() {
  return "Propagation of optional elements through allocation of
      functions.";
}


//rule: if all functions are optional, set components as optional
//normally there should be a single component (n functions to one)
//however, if several components are present, all shall be set to
    optional
@Override
public ArrayList<Element> execute(Package themodel, Package
    targetedPVGPackage) {
  ArrayList<Element> newPropagatedElement = new ArrayList<Element>();
  HashMap<NamedElement, Boolean> validate_propagation = new HashMap<
      NamedElement, Boolean>();//keep a record per component
  HashMap<NamedElement, ArrayList<NamedElement>>
      constrained_element_groups = new HashMap<NamedElement, ArrayList<
      NamedElement>>();

  Element[] allocations, target;
  //the model = model root element
```

```
//targetPVGpackage = the domain for rule application
allocations = SystemModelAnalysis.getSterotypedElements(themodel,
    Allocation_Stereotype);
for (int i=0; i<allocations.length;i++){
  Stereotype st = allocations[i].getAppliedStereotype(
      Allocation_Stereotype);
  //EObjectResolvingEList<Element> client = (EObjectResolvingEList<
      Element>) allocations[i].getValue(st, "client");
  if (!(allocations[i] instanceof Abstraction) && st!=null) {
    return null;
  }
  EList<NamedElement> clients = ((Abstraction) allocations[i]).
      getClients(); //functions
  EList<NamedElement> suppliers = ((Abstraction) allocations[i]).
      getSuppliers(); //allocation to components

  Iterator<NamedElement> suppliers_it = suppliers.iterator();
  while (suppliers_it.hasNext()){
    NamedElement component = suppliers_it.next();
    Stereotype component_st = component.getAppliedStereotype(
        RenaultComponent_Sterotype);
    Stereotype optional = component.getAppliedStereotype(
        ISequoia_Stereotypes.VARIABLEELEMENT_STEREOTYPE);
    Stereotype propagated = component.getAppliedStereotype(
        ISequoia_Stereotypes.PROPAGATEDVARIABLEELEMENT_STEREOTYPE);
    if (component_st != null && optional == null && propagated ==
        null){
      Iterator<NamedElement> clients_it = clients.iterator();
      while (clients_it.hasNext()){
        //if any of the functions for the current component is not
            optional invalidate propagation
        NamedElement function = clients_it.next();
```

```
            Stereotype func_st = function.getAppliedStereotype(
                RenaultFunction_Sterotype);
            Stereotype optional_fun = function.getAppliedStereotype(
                ISequoia_Stereotypes.VARIABLEELEMENT_STEREOTYPE);
            Stereotype propagated_fun = function.getAppliedStereotype(
                ISequoia_Stereotypes.PROPAGATEDVARIABLEELEMENT_STEREOTYPE
                );
            if (func_st==null  || (optional_fun==null && propagated_fun==
                null)) {
              //if allocated elements are not functions or they are not
                 optional invalidate propagation
              validate_propagation.put(component, false);
            } else if (optional_fun!=null || propagated_fun!=null){
              //if function is not null and it is either optional or
                 propagated optional then collect constraint info
              validate_propagation.put(component, true);
              ArrayList<NamedElement> fcn = constrained_element_groups.
                 get(component);
              if (fcn==null){
                fcn = new ArrayList<NamedElement>();
                fcn.add(function);
                constrained_element_groups.put(component, fcn);
              } else {
                fcn.add(function);
              }
            }
          }

        }
      }

    }
```

```
//check propagation per component
target = SystemModelAnalysis.getSterotypedElements(themodel,
    RenaultComponent_Sterotype);
for (int i = 0; i<target.length;i++){
  NamedElement cmp = (NamedElement) target[i];
  Boolean validate_element = validate_propagation.get(cmp);
  if (validate_element != null && validate_element){
    newPropagatedElement.add(target[i]);
    validate_propagation.put(cmp, false);
  }
}
Object[] component_list = newPropagatedElement.toArray();
//create array of constrained elements.

if (newPropagatedElement.size()>0){
  //perform transformations
  SystemModelAnalysis.applyStereotpes(component_list,
      ISequoia_Stereotypes.PROPAGATEDVARIABLEELEMENT_STEREOTYPE);
  SystemModelAnalysis.setStereotypeProperty(component_list,
      ISequoia_Stereotypes.PROPAGATEDVARIABLEELEMENT_STEREOTYPE, "
      reason", getDescription());

  //for each component, add propagation constraint : function
      requires components
  for (int i=0; i<component_list.length;i++){
    ArrayList<NamedElement> constrained_elm =
        constrained_element_groups.get(component_list[i]);
    if (constrained_elm != null){
      NamedElement[] constrained_elm_array = new NamedElement[
          constrained_elm.size()+1];
      Object[] constrained_elm_buf = constrained_elm.toArray();
```

```
    constrained_elm_array[constrained_elm_array.length-1] = (
        NamedElement) component_list[i];
    for (int k=0; k< constrained_elm_array.length-1; k++){
      constrained_elm_array[k]=(NamedElement) constrained_elm_buf[k
          ];
    }
    NamedElement nm = (NamedElement) SystemModelAnalysis.
        createVariationGroup("PR"+(element_cnt++).toString(),
        targetedPVGPackage, constrained_elm_array, getDescription()
        );
    //Constraint: if all functions are missing, component should be
         missing
    CustomConstraintEditor cs = new CustomConstraintEditor(
        CustomConstraintEditor.D_AllMissingExcludes, nm);
    cs.customizeConstraint();
  }
}

//for each component, add propagation constraint : components
    require functions
for (int i=0; i<component_list.length;i++){
  ArrayList<NamedElement> constrained_elm =
      constrained_element_groups.get(component_list[i]);
  if (constrained_elm != null){
    NamedElement[] constrained_elm_array = new NamedElement[
        constrained_elm.size()+1];
    Object[] constrained_elm_buf = constrained_elm.toArray();
    constrained_elm_array[constrained_elm_array.length-1] = (
        NamedElement) component_list[i];
    for (int k=0; k< constrained_elm_array.length-1; k++){
      constrained_elm_array[k]=(NamedElement) constrained_elm_buf[k
          ];
```

```
            }
            NamedElement nm = (NamedElement) SystemModelAnalysis.
                createVariationGroup("PR"+(element_cnt++).toString(),
                targetedPVGPackage, constrained_elm_array, getDescription()
                );
            //Constraint: if at least one component is present, component
                should be present
            CustomConstraintEditor cs = new CustomConstraintEditor(
                CustomConstraintEditor.D_AtLeastOneRequires, nm);
            cs.customizeConstraint();
          }
        }

      }
      return newPropagatedElement;
    }

    @Override
    public String getPluginName() {
      // TODO Auto-generated method stub
      return pluginName;
    }

    @Override
    public void setPluginName(String pluginName) {
      // TODO Auto-generated method stub
      this.pluginName = pluginName;
    }

  }
```

Publications

[1] Cosmin Dumitrescu, Alain Dauron, and Camille Salinesi. Formalization and exploitation of the coupling between systems engineering methods and product lines. *INCOSE Insight*, 14(4):12–13, December 2011.

[2] Cosmin Dumitrescu, Alain Dauron, and Camille Salinesi. Towards a framework for variability management and integration in systems engineering. Rome, 2012.

[3] Cosmin Dumitrescu, Patrick Tessier, Camille Salinesi, Sebastien Gérard, and Alain Dauron. Flexible product line derivation applied to a model based systems engineering process. Paris, December 2012.

[4] Cosmin Dumitrescu, Alain Dauron, Camille Salinesi, and Raul Mazo. La réutilisation en ingénierie de systèmes à base de modèles: Processus et activités IS pour l'Adoption d'une approche lignes de produits. *Génie Logiciel*, 105:52–59, June 2013.

[5] Cosmin Dumitrescu, Raul Mazo, Camille Salinesi, and Alain Dauron. Bridging the gap between product lines and systems engineering: an experience in variability management for automotive model based systems engineering. In *Proceedings of the 17th International Software Product Line Conference*, SPLC '13, pages 254–263, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1968-3. doi: 10.1145/2491627.2491655. URL http://doi.acm.org/10.1145/2491627.2491655.

[6] Cosmin Dumitrescu, Camille Salinesi, and Alain Dauron. Managing variability in an automotive company: Bridging the gap between PLE and MBSE. In *19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013)*, volume 56, Essen, Germany, March 2013. Institut für Informatik und Wirtschaftsinformatik (ICB). URL http://www.icb.uni-due.de/researchreports_en/.

[7] Cosmin Dumitrescu, Patrick Tessier, Camille Salinesi, Sebastien Gérard, Alain Dauron, and Raul Mazo. Capturing variability in model based systems engineering. In *Complex Systems Design & Management*, pages 125–139. Springer, 2014.

[8] R. Mazo, C. Dumitrescu, C. Salinesi, and D. Diaz. Recommendation heuristics for improving product line conguration processes. In M. Robillard, W. Maalej, R. Walker, and T. Zimmermann, editors, *To appear in: Recommendation Systems in Software Engineering*. Springer, 2014. (page 125).

[9] Raul Mazo, Gloria Lucia Giraldo, Léon Jaramillo, Camille Salinesi, and Cosmin Dumitrescu. A Bayesian-based approach to estimate material procurement in companies that use a product line approach. In *25th International Conference on SOFTWARE & SYSTEMS ENGINEERING and their APPLICATIONS*, Paris, France, November 2013.

[10] Sara Sadvandi, Hycham Aboutaleb, and Cosmin Dumitrescu. Negotiation process from a systems perspective. In Omar Hammami, Daniel Krob, and Jean-Luc Voirin, editors, *Complex Systems Design & Management*, pages 293–304. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-25202-0. doi: 10.1007/978-3-642-25203-7_21. URL http://dx.doi.org/10.1007/978-3-642-25203-7_21.

Internet Resources

[11] SEBoK authors. Viewpoint (glossary) — guide to the systems engineering body of knowledge (sebok), version 1.2, 2013. URL http://www.sebokwiki.org/w/index.php?title=Viewpoint_(glossary)&oldid=45712. [Online; accessed 3-January-2014]. (page 152).

[12] P. Clements and Linda M. Northrop. A framework for software product line practice, version 5.0, 2007. URL http://www.sei.cmu.edu/productlines/frame_report/introduction.htm.

[13] Common Variability Language Wiki. Common variability language wiki. URL http://www.omgwiki.org/variability/doku.php. [Online; accessed 3-February-2014]. (page 28).

[14] Customizing UML. Customizing uml. URL http://www.eclipse.org/modeling/mdt/uml2/docs/articles/Customizing_UML2_Which_Technique_is_Right_For_You/article.html. [Online; accessed 3-January-2014]. (page 29).

[15] INCOSE. A consensus of the incose fellows. http://www.incose.org/practice/fellowsconsensus.aspx. Accessed: 2013-11-10.

[16] INCOSE APE Project. The magicdraw web publisher 2.0 – ape project. URL http://mbse.gfse.de/extdocs/ape.html. [Online; accessed 10-February-2014]. (page 35).

[17] MBSEBlogWebsite. Model based systems engineering. URL http://model-based-systems-engineering.com.

[18] MilSTDWebsite. MIL-STD-499 military standard system engineering management, 1969. URL http://www.everyspec.com/MIL-STD/MIL-STD-0300-0499/MIL-STD-499_10376/.

[19] NissanWebsite. NISSAN | COMMON MODULE FAMILY (CMF): a NEW APPROACH TO ENGINEERING FOR THE RENAULT-NISSAN ALLIANCE. URL http://www.nissan-global.com/EN/NEWS/2013/_STORY/130619-01-e.html.

[20] PaultanWebsite. 2013 nissan terrano unveiled in mumbai - rebadged dacia duster. URL http://paultan.org/2013/08/22/nissan-terrano-2013-unveiled/.

[21] Renault ABS ESP. Document renault: Innovation – securité. URL http://www.renault.com/fr/innovation/au-service-de-la-securite/documents_without_moderation/pdf%20securite/integrale-securite.pdf. [Online; accessed 10-February-2014]. (page 43).

[22] RenaultNissanWebsite. Beyond platform sharing: The alliance's new approach to commonization. URL http://blog.alliance-renault-nissan.com/blog/beyond-platform-sharing-alliances-new-approach-commonization.

[23] SysMLWebsite. SysML 1.3. URL http://www.omg.org/spec/SysML/1.3/.

[24] SysMODWebsite. SYSMOD - the systems modeling process. URL http://sysmod.system-modeling.com/.

[25] SystemsWikipediaWebsite. Systems engineering, July 2013. URL http://en.wikipedia.org/w/index.php?title=Systems_engineering&oldid=564708864. Page Version ID: 564708864.

[26] Dorothee Tschampa and Mathieu Rosemain. Daimler to add renault-nissan fuso vans to global cooperation. URL http://www.bloomberg.com/news/2013-09-11/daimler-to-add-renault-nissan-fuso-vans-to-global-cooperation.html.

[27] Wikipedia. Headlamp, November 2013. URL http://en.wikipedia.org/w/index.php?title=Headlamp&oldid=574111336. Page Version ID: 574111336.

[28] ZigwheelsWebsite. Nissan's duster clone named terrano. URL http://www.zigwheels.com/upcoming-launches/nissans-duster-clone-named-terrano/16602/.

References

[29] *Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21-24, 2006, Proceedings*, 2006. IEEE Computer Society. ISBN 0-7695-2599-7. (pages 253, 254).

[30] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. Separation of concerns in feature modeling: support and applications. In Robert Hirschfeld, Éric Tanter, Kevin J. Sullivan, and Richard P. Gabriel, editors, *AOSD*, pages 1–12. ACM, 2012. ISBN 978-1-4503-1092-5. (page 33).

[31] Hwi Ahn and Sungwon Kang. Analysis of software product line architecture representation mechanisms. pages 219–226. IEEE, August 2011. ISBN 978-1-4577-1028-5. doi: 10.1109/SERA.2011.22. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6065644. (page 20).

[32] Christopher Alexander. *Notes on the Synthesis of Form*. Harvard University Press, January 1964. ISBN 9780674627512. (page 42).

[33] Anas Alfaris, Davor Svetinovic, Afreen Siddiqi, Charbel Rizk, and Olivier de Weck. Hierarchical decomposition and multidomain formulation for the design of complex sustainable systems. *Journal of Mechanical Design*, 132(9):091003, 2010. (page 42).

[34] Mauricio Alférez, João Santos, Ana Moreira, Alessandro Garcia, Uirá Kulesza, João Araújo, and Vasco Amaral. Multi-view composition language for software product line requirements. In Mark Brand, Dragan Gaevi, and Jeff Gray, editors, *Software Language Engineering*, volume 5969 of *Lecture Notes in Computer Science*, pages 103–122. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-12106-7. doi: 10.1007/978-3-642-12107-4_8. URL http://dx.doi.org/10.1007/978-3-642-12107-4_8. (page 33).

[35] Fabrice Alizon, Steven B. Shooter, and Timothy W. Simpson. Reuse of manufacturing knowledge to facilitate platform-based product realization. *Journal of Computing and Information Science in Engineering*, 6(2):170, 2006. ISSN 15309827. doi: 10.1115/1.2202135. URL http://ComputingEngineering.asmedigitalcollection.asme.org/article.aspx?articleid=1400497. (page 3).

[36] N. Anquetil, B. Grammel, I. Galvao Lourenco da Silva, J. A. R. Noppen, S. Shakil Khan, H. Arboleda, A. Rashid, and A. Garcia. Traceability for model driven, software product line engineering. In *ECMDA Traceability Workshop Proceedings, Berlin, Germany*, pages 77–86, Norway, June 2008. SINTEF. (page 88).

[37] Michal Antkiewicz and Krzysztof Czarnecki. FeaturePlugin: feature modeling plug-in for eclipse. In *Proceedings of the 2004 OOPSLA workshop on eclipse technology eXchange*, eclipse '04, pages 67–72, New York, NY, USA, 2004. ACM. doi: 10.1145/1066129.1066143. (page 108).

[38] Hugo Arboleda. *FieSta: An approach for Fine-Grained Scope Definition, Configuration and Derivation of Model-Driven Software Product Lines*. PhD thesis, Université de Nantes, 2009. (page 34).

[39] Hugo Arboleda, Rubby Casallas, Jean-Claude Royer, et al. Implementing an mda approach for managing variability in product line construction using the gmf and gme frameworks. In *Proceedings of the 5th Nordic Workshop on Model Driven Software Engineering (NW-MoDE'07)*, pages 67–82, 2007. (page 34).

[40] Hugo Arboleda, Rubby Casallas, and Jean-Claude Royer. Dealing with fine-grained configurations in model-driven spls. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 1–10, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. URL http://dl.acm.org/citation.cfm?id=1753235.1753237. (page 34).

[41] J.-M. Astesana, L. Cosserat, and H. Fargier. Constraint-based vehicle configuration: A case study. In *2010 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, volume 1, pages 68 –75, October 2010. doi: 10.1109/ICTAI.2010.19. (pages 21, 108, 122).

[42] Jeam-Marc Astesana, Laurent Cosserat, and Hélène Fargier. Modélisation par contraintes et exploitation d'une gamme automobile : nouveaux problèmes, nouvelles requtes, nouveaux besoins en programmation par contraintes. In *JFPC 2010 - Sixièmes Journées Francophones de Programmation par Contraintes*, pages 33–42, Caen, France, June 2010. URL http://hal.inria.fr/inria-00520306. (pages 14, 34, 110).

[43] J.M. Astesana, L. Cosserat, and H. Fargier. Constraint-based modeling and exploitation of a vehicle range at renaults: Requirement analysis and complexity study. In *Workshop on Configuration*, page 33, 2010. (pages 1, 34, 56, 76, 77, 84, 90, 101, 107).

[44] Sofia Azevedo, Ricardo J. Machado, Alexandre Bragança, and Hugo Ribeiro. Support for variability in use case modeling with refinement. In *Proceedings of the 7th International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, MOMPES '10, pages 1–8, New York, NY, USA,

2010. ACM. ISBN 978-1-4503-0123-7. doi: 10.1145/1865875.1865876. URL http://doi.acm.org/10.1145/1865875.1865876. (page 171).

[45] D. Batory, D. Benavides, and A. Ruiz-Cortes. Automated analysis of feature models: challenges ahead. *Communications of the ACM*, 49(12):45–47, 2006. (page 122).

[46] M. Becker. Towards a general model of variability in product families. In *Workshop on Software Variability Management. Editors Jilles van Gurp and Jan Bosch. Groningen, The Netherlands. http://www. cs. rug. nl/Research/SE/svm/proceedingsSVM2003Groningen. pdf*, pages 19–27, 2003. URL http://publications.jillesvangurp.com/svm2003/svm2003-proceedings.pdf#page=21. (page 81).

[47] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636, 2010. URL http://www.sciencedirect.com/science/article/pii/S0306437910000025. (pages 56, 84).

[48] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In *Advanced Information Systems Engineering*, number 3520 in Lecture Notes in Computer Science, pages 491 –503. Springer Berlin Heidelberg, January 2005. (page 108).

[49] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. Automated reasoning on feature models. In Oscar Pastor and Joo Falco e Cunha, editors, *Advanced Information Systems Engineering*, number 3520 in Lecture Notes in Computer Science, pages 491–503. Springer Berlin Heidelberg, January 2005. ISBN 978-3-540-26095-0, 978-3-540-32127-9. URL http://link.springer.com/chapter/10.1007/11431855_34. (pages 36, 56, 84).

[50] N. Bencomo, P. Sawyer, G. Blair, and P. Grace. Dynamically adaptive systems are product lines too: Using model-driven techniques to capture dynamic variability of adaptive systems. In *2nd International Workshop on Dynamic Software Product Lines (DSPL 2008), Limerick, Ireland*, volume 38, page 40, 2008. URL http://staffwww.dcs.shef.ac.uk/people/A.Simons/remodel/papers/BencomoDynAdapt.pdf. (page 145).

[51] Danilo Beuche. Modeling and building software product lines with pure:: variants. In *Software Product Line Conference, 2008. SPLC'08. 12th International*, pages 358–358. IEEE, 2008. (page 29).

[52] T. Blecker, N. Abdelkafi, B. Kaluza, and G. Kreutler. Mass customization vs. complexity: A gordian knot? 2004. (page 47).

[53] Ryan Boas, Bruce G. Cameron, and Edward F. Crawley. Divergence and lifecycle offsets in product families with commonality. *Systems Engineering*, 16(2):175–192, 2013. (pages x, 37).

[54] G. Bockle, P. Clements, J. D McGregor, D. Muthig, and K. Schmid. Calculating ROI for software product lines. *IEEE Software*, 21(3):23–31, June 2004. ISSN 0740–7459. doi: 10.1109/MS.2004.1293069. (pages 11, 56).

[55] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *ECAI*, volume 16, page 146, 2004. (page 121).

[56] Manfred Broy. The role of requirements and specification in product line engineering, August 2011. URL http://splc2011.net/program-1/keynotes/keynote-broy/index.html. (page 85).

[57] AlbertoJ. Caas, Roger Carff, Greg Hill, Marco Carvalho, Marco Arguedas, ThomasC. Eskridge, James Lott, and Rodrigo Carvajal. Concept maps: Integrating knowledge and information visualization. In Sigmar-Olaf Tergan and Tanja Keller, editors, *Knowledge and Information Visualization*, volume 3426 of *Lecture Notes in Computer Science*, pages 205–219. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26921-2. doi: 10.1007/11510154_11. URL http://dx.doi.org/10.1007/11510154_11. (page 16).

[58] Hugo G. Chalé Góngora, Alain Dauron, and Thierry Gaudré. A commonsense-driven architecture framework. part 1: A car manufacturers (nave) take on mbse. In *INCOSE 2012*, 2012. (pages x, 4, 7, 14, 50, 53, 55, 96, 97, 106, 109, 126, 155).

[59] Lianping Chen and Muhammad Ali Babar. A systematic review of evaluation of variability management approaches in software product lines. 53(4):344–362. ISSN 0950-5849. doi: 10.1016/j.infsof.2010.12.006. URL http://www.sciencedirect.com/science/article/pii/S0950584910002223. (page 20).

[60] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of TVL. *Science of Computer Programming, Special Issue on Software Evolution, Adaptability and Variability*, 76(12):1130–1143, 2011. doi: 10.1016/j.scico.2010.10.005. URL http://dx.doi.org/10.1016/j.scico.2010.10.005. (page 23).

[61] Matthias Clauß. Generic modeling using uml extensions for variability. In *Workshop on Domain Specific Visual Languages at OOPSLA*, volume 2001, 2001. (page 30).

[62] P. Clements and Linda M. Northrop. A framework for software product line practice, version 5.0. http://www.sei.cmu.edu/productlines/frame_report/introduction.htm, 2007. URL http://www.sei.cmu.edu/productlines/frame_report/introduction.htm. (page 21).

[63] Paul Clements and Linda M. Northrop. *Software Product Lines: Practices and Patterns*. Prentice Hall, 2002. ISBN 9780201703320. (pages 2, 11, 21, 67, 123).

[64] North Atlantic Council. Nato architecture framework (naf) ver 3. *North Atlantic Council*, 2007. (page 70).

[65] K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, pages 173–182, 2012. URL http://dl.acm.org/citation.cfm?id=2110167. (pages 20, 72, 81).

[66] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. (page 72).

[67] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005. (pages 23, 41).

[68] A. Dauron. Model-based powertrain control: Many uses, no abuse. *Oil & Gas Science and Technology – Rev. IFP Oil & Gas Science and Technology – Rev. IFP*, 62(62):427–435, 2007. (page 53).

[69] Eduardo Santana de Almeida, Tomoji Kishi, Christa Schwanninger, Isabel John, and Klaus Schmid, editors. *Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22-26, 2011*, 2011. IEEE. ISBN 978-1-4577-1029-2. (page 263).

[70] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Experiences in software product families: Problems and issues during product derivation. In RobertL. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 165–182. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22918-6. doi: 10.1007/978-3-540-28630-1_10. URL http://dx.doi.org/10.1007/978-3-540-28630-1_10. (page 106).

[71] Arie van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology*, 10, 2001. (page 108).

[72] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The DOPLER metatool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, 18(1):77–114, November 2010. ISSN 0928-8910, 1573-7535. (page 143).

[73] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The DOPLER metatool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering*, 18(1):77–114, November 2010. ISSN 0928-8910, 1573-7535. doi: 10.1007/s10515-010-0076-6. URL http://www.springerlink.com/index/10.1007/s10515-010-0076-6. (page 26).

[74] D. Diaz and P. Codognet. Design and implementation of the gnu prolog system. *Journal of Functional and Logic Programming*, 6(2001):542, 2001. (pages 120, 196).

[75] O. Djebbi and C. Salinesi. Criteria for comparing requirements variability modeling notations for product lines. In *Comparative Evaluation in Requirements Engineering, 2006. CERE'06. Fourth International Workshop on*, pages 20–35, 2006. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4019690. (page 20).

[76] Olfa Djebbi and Camille Salinesi. Red-pl, a method for deriving product requirements from a product line requirements model. In John Krogstie, Andreas L. Opdahl, and Guttorm Sindre, editors, *CAiSE*, volume 4495 of *Lecture Notes in Computer Science*, pages 279–293. Springer, 2007. ISBN 978-3-540-72987-7. (page 22).

[77] Olfa Djebbi, Camille Salinesi, and Daniel Diaz. Deriving product line requirements: the red-pl guidance approach. In *APSEC*, pages 494–501. IEEE Computer Society, 2007. (pages 101, 106).

[78] Dov Dori. *Object Process Methodology: A Holistic Systems Paradigm; with CD-ROM*. Springer, 2002. (pages 4, 42).

[79] Dov Dori, Iris Reinhartz-Berger, and Arnon Sturm. Developing complex systems with object-process methodology using opcat. In *Conceptual Modeling-ER 2003*, pages 570–572. Springer, 2003. (page 70).

[80] Abdelkrim Doufene, Hugo Chalé Góngora, and Daniel Krob. Complex systems architecture framework. extension to multi-objective optimization. Paris, December 2012. (pages x, 54, 88, 110, 148).

[81] Cosmin Dumitrescu. Study of the implementation of a system engineering tool in the context of product lines at renault. Master "Ingénierie des systèmes industriels complexes", Ecole Polytechnique, Palaiseau, October 2009. (page 61).

[82] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In Forrest Shull, Janice Singer, and DagI.K. Sjberg, editors, *Guide to Advanced Empirical Software Engineering*, pages 285–311. Springer London, 2008. ISBN 978-1-84800-043-8. doi: 10.1007/978-1-84800-044-5_11. URL http://dx.doi.org/10.1007/978-1-84800-044-5_11. (page 97).

[83] Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*, pages 285–311. Springer, 2008. URL http://link.springer.com/chapter/10.1007/978-1-84800-044-5_11. (pages 8, 128, 154).

[84] eclipse.org EMF Feature Model. eclipse.org EMF feature model. http://www.eclipse.org/proposals/feature-model/. URL http://www.eclipse.org/proposals/feature-model/. (page 77).

[85] Kathleen M. Eisenhardt. Building theories from case study research. *The Academy of Management Review*, 14(4):532–550, October 1989. ISSN 0363–7425. doi:

10.2307/258557. URL http://www.jstor.org/stable/258557. ArticleType: research-article / Full publication date: Oct., 1989 / Copyright 1989 Academy of Management. (page 8).

[86] EPBWikipediaWebsite. Parking brake, January 2013. URL http://en.wikipedia.org/wiki/Parking_brake#Electric_parking_brake. Page Version ID: 524112611. (page 8).

[87] Magnus Eriksson, Jürgen Börstler, and Kjell Borg. The pluss approach: Domain modeling with features, use cases and use case realizations. In *Proceedings of the 9th International Conference on Software Product Lines*, SPLC'05, pages 33–44, Berlin, Heidelberg, 2005. Springer-Verlag. ISBN 3-540-28936-4, 978-3-540-28936-4. doi: 10.1007/11554844_5. URL http://dx.doi.org/10.1007/11554844_5. (pages 28, 29).

[88] Huáscar Espinoza, Hubert Dubois, Sbastien Grard, Julio Medina, DorinaC. Petriu, and Murray Woodside. Annotating uml models with non-functional properties for quantitative analysis. In Jean-Michel Bruel, editor, *Satellite Events at the MoDELS 2005 Conference*, volume 3844 of *Lecture Notes in Computer Science*, pages 79–90. Springer Berlin Heidelberg, 2006. ISBN 978-3-540-31780-7. doi: 10.1007/11663430_9. URL http://dx.doi.org/10.1007/11663430_9. (page 24).

[89] Jeff A Estefan et al. Survey of model-based systems engineering (mbse) methodologies. *Incose MBSE Focus Group*, 25, 2007. (page 8).

[90] Evalueserve. Platform strategy will shape future of OEMs. flexibility to drive growth. Technical report, Evalueserve, January 2012. URL http://sandhill.com/wp-content/files_mf/evalueservewhitepaperplatformstrategywillshapefutureofoems.pdf. (page 3).

[91] R. Faure and A. Dauron. Ingénierie renault et ingénierie des systèmes. In *Journée Thématique AFIS "Stratégies d'Entreprises en Ingénierie Système"*, Paris, June 2009. (page 53).

[92] John Favaro and Silvia Mazzini. Extending FeatuRSEB with concepts from systems engineering. In Stephen H. Edwards and Gregory Kulczycki, editors, *Formal Foundations of Reuse and Domain Engineering*, number 5791 in Lecture Notes in Computer Science, pages 41–50. Springer Berlin Heidelberg. ISBN 978-3-642-04210-2, 978-3-642-04211-9. URL http://link.springer.com/chapter/10.1007/978-3-642-04211-9_5. (page 27).

[93] J.B.F. Filho, O. Barais, B. Baudry, and J. Le Noir. Leveraging variability modeling for multi-dimensional model-driven software product lines. In *2012 3rd International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, pages 5 –8, June 2012. doi: 10.1109/PLEASE.2012.6229774. (pages 53, 75).

[94] Paul. Frenz, Garry Roedler, Donald J. Gantzer, and Peter Baxter. Engineering measurement primer: A basic introduction to measurement concepts and use for systems engineering, version 2.0. (INCOSE—TP—2010—005—02), 2009. (page 24).

[95] Hassan Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004. ISBN 0201775956. (pages 29, 31).

[96] Hassan Gomaa. *Designing Software Product Lines With Uml: From Use Cases to Pattern-Based Software Architectures*. ADDISON WESLEY Publishing Company Incorporated, 2005. ISBN 9780201775952. (page 20).

[97] Hassan Gomaa. Designing software product lines with uml 2.0: From use cases to pattern-based software architectures. In *SPLC* DBL [29], page 218. ISBN 0-7695-2599-7. (page 171).

[98] Hugo G. Chalé Góngora, Thierry Gaudré, and Sara Tucci-Piergiovanni. Towards an architectural design framework for automotive systems development. In Marc Aiguier, Yves Caseau, Daniel Krob, and Antoine Rauzy, editors, *Complex Systems Design & Management*, pages 241–258. Springer Berlin Heidelberg, January 2013. ISBN 978-3-642-34403-9, 978-3-642-34404-6. URL http://link.springer.com/chapter/10.1007/978-3-642-34404-6_16. (pages 156, 166).

[99] M. L. Griss, J. Favaro, and M. d' Alessandro. Integrating feature modeling with the rseb. In *Proceedings of the 5th International Conference on Software Reuse*, ICSR '98, pages 76–, Washington, DC, USA, 1998. IEEE Computer Society. ISBN 0-8186-8377-5. URL http://dl.acm.org/citation.cfm?id=551789.853486. (pages 27, 29).

[100] Reinhard Haberfellner and Olivier de Weck. Agile systems engineering versus agile systems engineering. In *INCOSE 2005 Symposium*, pages 10–15, 2005. (pages 67, 82).

[101] T. Hadzic and H. R. Andersen. An introduction to solving interactive configuration problems. Technical report, Technical Report TR-2004-49, The IT University of Copenhagen, 2004. (pages 107, 120).

[102] T. Hadzic, S. Subbarayan, R. M. Jensen, H. R. Andersen, J. Mller, and H. Hulgaard. Fast backtrack-free product configuration using a precompiled solution space representation. *small*, 10(1):3, 2004. (page 107).

[103] AD Hall. *A Methodology for Systems Engineering*. Van Nostrand, 1962. (page 11).

[104] Günter Halmans and Klaus Pohl. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, 2(1):15–36, 2003. ISSN 1619–1366. doi: 10.1007/s10270-003-0019-9. URL http://www.springerlink.com/content/8r9fflq08gclme25/abstract/. (pages x, 29, 32, 81, 96).

[105] Margaret H Hamilton and William R Hackler. Universal systems language: lessons learned from apollo. *Computer*, 41(12):34–43, 2008. (page 70).

[106] Military Handbook. Configuration management guidance. Technical report, MIL-HDBK-61A (SE) Department of Defense–United States of America, 2001. (page 22).

[107] David Harrison and Lou Varveris. Togaf: Establishing itself as the definitive method for building enterprise architectures in the commercial world, 2004. (page 70).

[108] M. Hause. The unified profile for dodaf/modaf (updm) enabling systems of systems on many levels. In *Systems Conference, 2010 4th Annual IEEE*, pages 426–431. IEEE, 2010. doi: 10.1109/SYSTEMS.2010.5482450. (page 70).

[109] W. Hesse. Ontologies in the software engineering process. *EAI*, 2005. URL http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-141/paper1.pdf. (page 14).

[110] José R. Hilera, Coral Calero, Francisco Ruiz, and Mario Piattini. Using ontologies in software engineering and technology. In *Ontologies for Software Engineering and Software Technology*. URL http://www.springer.com/computer/swe/book/978-3-540-34517-6. (page 14).

[111] Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens, Dirk Deridder, and EbrahimKhalil Abbasi. Supporting multiple perspectives in feature-based configuration. *Software & Systems Modeling*, 12(3):641–663, 2013. ISSN 1619-1366. doi: 10.1007/s10270-011-0220-1. URL http://dx.doi.org/10.1007/s10270-011-0220-1. (page 33).

[112] John M. Hunt. Organizing the asset base for product derivation. In *SPLC* DBL [29], pages 65–74. ISBN 0-7695-2599-7. (page 107).

[113] INCOSE. Focus on product line engineering. *INCOSE Chicagoland Chapter Newsletter*, 1(2):2, 2011. (page 148).

[114] INCOSE MBSE Challenge Team SEˆ2. SysML for Telescope System Modeling - Variant Modeling -, December 2010. URL http://mbse.gfse.de/documents/IW11_Presentation_Telescope_Modeling.pdf. (page 35).

[115] KPMG International. Product diversity: Not for profit? Technical report, KPMG International, 2009. URL http://www.kpmg.de/docs/Product_Diversity.pdf. (pages 2, 47).

[116] ISO 14443. Systems and software engineering Architecture description, the latest edition of the original IEEE Std 1471:2000, Recommended Practice for Architectural Description of Software-intensive Systems., 2000. (page 152).

[117] ISO/IEC 26550:2013. Software and systems engineering – Reference model for product line engineering and management, 2013. (page 130).

[118] Zhenhui Jiang, Weiquan Wang, and Izak Benbasat. Multimedia-based interactive advising technology for online consumer decision support. *Commun. ACM*, 48(9): 92–98, September 2005. ISSN 0001–0782. doi: 10.1145/1081992.1081995. URL http://doi.acm.org/10.1145/1081992.1081995. (page 108).

[119] Isabel John and Dirk Muthig. Muthig d.: Product line modeling with generic use cases, splc-2 workshop on techniques for exploiting commonality through variability. In *Management, Second Software Product Line Conference*, 2002. (pages 29, 30).

[120] Andrew Josey. *TOGAF Version 9: A Pocket Guide*. Van Haren Publishing, 2009. (page 70).

[121] Jean-Marc Jzquel. Model-driven engineering for software product lines. *ISRN Software Engineering*, 2012:1–24, 2012. ISSN 2090-7680. doi: 10.5402/2012/670803. URL http://www.hindawi.com/isrn/software.engineering/2012/670803/. (pages 20, 27).

[122] T. Kakola. Standards initiatives for software product line engineering and management within the international organization for standardization. In *2010 43rd Hawaii International Conference on System Sciences (HICSS)*, pages 1–10. IEEE, January 2010. ISBN 978-1-4244-5509-6. doi: 10.1109/HICSS.2010.348. (page 70).

[123] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990. (pages 4, 20, 23, 32, 42, 53, 72, 75, 81).

[124] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990. (pages 21, 203).

[125] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Euiseob Shin, and Moonhang Huh. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. *Ann. Softw. Eng.*, 5:143–168, 1998. ISSN 1022-7091. (page 24).

[126] Kyo Chul Kang. Foda: Twenty years of perspective on feature modeling, 1 2010. URL http://www.sse.uni-due.de/vamos/2010/files/VaMoS2010_Keynote_Kang_FODA.pdf. Keynote by Kyo C. Kang at VaMoS - Fourth International Workshop on Variability Modelling of Software-intensive Systems - "Celebrating 20 Years of Feature Models" [Accessed: 2014 01 20]. (page 20).

[127] Robert Karban. Model-based systems engineering telescope modelling challenge team, January 2011. URL http://mbse.gfse.de/documents/IW11_Presentation_Telescope_Modeling.pdf. (page 35).

[128] Daniel Krob. Eléments d'architecture des systèmes complexes. In Alain Appriou, editor, *Gestion de la complexité et de l'information dans les grands systèmes critiques*, pages 179–207. CNRS, 2009. ISBN 978-2-271-06828-6. (pages 126, 155).

[129] Daniel Krob. Eléments d'architecture des systèmes complexes. In Alain Appriou, editor, *Gestion de la complexité et de l'information dans les grands systèmes critiques*, pages 179–207. CNRS, 2009. ISBN 978-2-271-06828-6. (pages 4, 42, 54, 126).

[130] Charles W Krueger. Biglever software gears and the 3-tiered spl methodology. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 844–845. ACM, 2007. (page 29).

[131] Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software product lines in action: the best industrial practice in product line engineering*. Springer, September 2007. ISBN 9783540714361. (page 11).

[132] Abel Gómez Llana and Isidro Ramos Salavert. *Automatic Tool Support for Cardinality-Based Feature Modeling with Model Constraints for Information Systems Development*. 2010. (page 81).

[133] Jim Long. Relationships between common graphical representations in system engineering. *Vitech white paper, Vitech Corporation, Vienna, VA*, 2002. (page 70).

[134] C.L. Magee and O.L. de Weck. Complex system clasification. In *Fourteenth Annual International Symposium of the International Council On Systems Engineering (INCOSE)*, 2004. URL http://strategic.mit.edu/docs/3_36_INCOSE2004_complexsys.pdf. (page 152).

[135] Stefan Mann and Georg Rock. Dealing with variability in architecture descriptions to support automotive product lines. 29:111–120, 2009. (page 74).

[136] Mike Mannion, Juha Savolainen, and Timo Asikainen. Viewpoint-oriented variability modeling. In Sheikh Iqbal Ahamed, Elisa Bertino, Carl K. Chang, Vladimir Getov, Lin Liu, Hua Ming, and Rajesh Subramanyan, editors, *COMPSAC (1)*, pages 67–72. IEEE Computer Society, 2009. (page 33).

[137] Marc H. Meyer and Alvin P. Lehnerd. *The power of product platforms: building value and cost leadership*. Free Pr, 1997. ISBN 9780684825809. (pages 4, 11).

[138] Thomas von der Massen and Horst Lichter. RequiLine: a requirements engineering tool for software product lines. In Frank J. van der Linden, editor, *Software Product-Family Engineering*, number 3014 in Lecture Notes in Computer Science, pages 168–180. Springer Berlin Heidelberg, January 2004. ISBN 978-3-540-21941-5, 978-3-540-24667-1. URL http://link.springer.com/chapter/10.1007/978-3-540-24667-1_13. (pages 81, 96).

[139] RJ Mayer. Idef0 function modelinga reconstruction of the original air force wright aeronautical laboratory technical report afwal-tr-81-4023 (the idef0 yellow book). *Knowledge Based Systems, Inc.: College Station, Texas*, 1992. (page 70).

[140] R. Mazo, R. E. Lopez-Herrejon, C. Salinesi, D. Diaz, and A. Egyed. Conformance checking with constraint logic programming: The case of feature models. In *Computer Software and Applications Conference (COMPSAC), 2011 IEEE 35th Annual*, pages 456–465, 2011. (pages 23, 108).

[141] R. Mazo, C. Salinesi, D. Diaz, and A. Lora-Michiels. Transforming attribute and clone-enabled feature models into constraint programs over finite domains. In *6th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, 2011. (pages 108, 119, 123).

[142] R. Mazo, C. Salinesi, D. Diaz, O. Djebbi, and A. Lora-Michiels. Constraints: The heart of domain and application engineering in the product lines engineering strategy. *International Journal of Information System Modeling and Design (IJISMD)*, 3:33–68, 2012. (pages 119, 122, 196).

[143] R. Mazo, C. Salinesi, D. Diaz, O. Djebbi, and A. Lora-Michiels. Constraints: The heart of domain and application engineering in the product lines engineering strategy. *International Journal of Information System Modeling and Design (IJISMD)*, 3(2):33–68, 2012. URL http://www.igi-global.com/article/constraints-heart-domain-application-engineering/65561. (pages 56, 84).

[144] Raul Mazo, Paul Grünbacher, Wolfgang Heider, Rick Rabiser, Camille Salinesi, and Daniel Diaz. Using constraint programming to verify DOPLER variability models. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 97–103, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0570-9. doi: 10.1145/1944892.1944904. URL http://doi.acm.org/10.1145/1944892.1944904. (page 23).

[145] Ral Mazo, Camille Salinesi, and Daniel Diaz. VariaMos: a tool for product line driven systems engineering with a constraint based approach. In *24th International Conference on Advanced Information Systems Engineering (CAiSE Forum'12)*, Gdansk, Pologne, June 2012. (pages 108, 122, 123, 197).

[146] Ral Mazo, Camille Salinesi, and Daniel Diaz. VariaMos: a tool for product line driven systems engineering with a constraint based approach. In *24th International Conference on Advanced Information Systems Engineering (CAiSE Forum'12)*, Gdansk, Pologne, June 2012. URL http://hal.archives-ouvertes.fr/hal-00707551. (pages 36, 125).

[147] Marcilio Mendonca, Moises Branco, and Donald Cowan. S.p.l.o.t.: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*,

OOPSLA '09, pages 761–762, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-768-4. doi: 10.1145/1639950.1640002. URL http://doi.acm.org/10.1145/1639950.1640002. (page 36).

[148] Marcilio Mendonca, Andrzej Wasowski, and Krzysztof Czarnecki. Sat-based analysis of feature models is easy. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 231–240, Pittsburgh, PA, USA, 2009. Carnegie Mellon University. URL http://dl.acm.org/citation.cfm?id=1753235.1753267. (pages 23, 107, 108).

[149] Raphaël Michel, Arnaud Hubaux, Vijay Ganesh, and Patrick Heymans. An smt-based approach to automated configuration. In Pascal Fontaine and Amit Goel, editors, *SMT@IJCAR*, volume 20 of *EPiC Series*, pages 109–119. EasyChair, 2012. (page 36).

[150] Mikyeong Moon, Keunhyuk Yeom, and Heung Seok Chae. An approach to developing domain requirements as a core asset based on commonality and variability analysis in a product line. *IEEE Trans. Softw. Eng.*, 31(7):551–569, July 2005. ISSN 0098-5589. doi: 10.1109/TSE.2005.76. URL http://dx.doi.org/10.1109/TSE.2005.76. (page 31).

[151] Divya Karunakaran Nair. Variability-modelling practices in industrial software product lines: A qualitative study. 2013. URL https://uwspace.uwaterloo.ca/handle/10012/7550. (page 43).

[152] Nan Niu, Juha Savolainen, and Yijun Yu. Variability modeling for product line viewpoints integration. In Sheikh Iqbal Ahamed, Doo-Hwan Bae, Sung Deok Cha, Carl K. Chang, Rajesh Subramanyan, Eric Wong, and Hen-I Yang, editors, *COMPSAC*, pages 337–346. IEEE Computer Society, 2010. ISBN 978-0-7695-4085-6. (page 33).

[153] Joseph D Novak and Alberto J Cañas. The theory underlying concept maps and how to construct and use them. *Florida Institute for Human and Machine Cognition Pensacola Fl, www. ihmc. us.[http://cmap. ihmc. us/Publications/ResearchPapers/T heoryCmaps/TheoryUnderlyingConceptMaps. htm]*, 284, 2008. (page 16).

[154] Pádraig O'Leary, Fergal Mc Caffery, Ita Richardson, and Steffen Thiel. Towards agile product derivation in software product line engineering. 2009. (page 37).

[155] Pádraig O'Leary, Fergal McCaffery, Steffen Thiel, and Ita Richardson. An agile process model for product derivation in software product line engineering. *Journal of Software: Evolution and Process*, 24(5):561–571, 2012. (page 37).

[156] Sebastian Oster, Florian Markert, Andy Schürr, and Werner Müller. Integrated modeling of software product lines with feature models and classification trees.

In *Proceedings of the 13th International Software Product Line Conference (SPLC 2009). MAPLE 2009 Workshop Proceedings. Springer, Heidelberg*, 2009. (page 43).

[157] B. Pargamin. Vehicle sales configuration: the cluster tree approach. In *ECAI 2002 Configuration Workshop*, pages 35–40, 2002. (page 117).

[158] D.L. Parnas. On the design and development of program families. *Software Engineering, IEEE Transactions on*, SE-2(1):1–9, 1976. ISSN 0098–5589. doi: 10.1109/TSE.1976.233797. (pages 11, 23).

[159] Gregory S. Parnell, Patrick J. Driscoll, and Dale L. Henderson. *Decision Making in Systems Engineering and Management*. John Wiley & Sons, February 2008. ISBN 9780470165706. (page 27).

[160] Gilles Perrouin, Jacques Klein, Nicolas Guelfi, and Jean-Marc Jézéquel. Reconciling automation and flexibility in product derivation. In *SPLC*, pages 339–348. IEEE Computer Society, 2008. ISBN 978-0-7695-3303-2. (page 37).

[161] JD Piques and E Andrianarison. Sysml for embedded automotive systems: lessons learned. *Embedded Real Time Software and Systems*, 3:3b, 2012. (page 19).

[162] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. (pages 16, 21, 22, 196).

[163] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 3540243720. (pages 4, 12, 15, 22, 23, 39, 53, 67, 68, 72, 75, 77, 81, 87, 95, 128, 143, 144, 176).

[164] T. Possompès, C. Dony, M. Huchard, H. Rey, C. Tibermacine, and X. Vasques. A UML profile for feature diagrams: Initiating a model driven engineering approach for software product lines. In *Journée Lignes de Produits*, pages 59–70, 2010. URL http://hal.archives-ouvertes.fr/lirmm-00542800/. (pages 31, 81).

[165] Thibaut Possompes, Christophe Dony, Marianne Huchard, and Chouki Tibermacine. Design of a uml profile for feature diagrams and its tooling implementation. In *SEKE*, pages 693–698. Knowledge Systems Institute Graduate School, 2011. ISBN 1-891706-29-2. (page 31).

[166] Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. Supporting product derivation by adapting and augmenting variability models. In *SPLC*, pages 141–150. IEEE Computer Society, 2007. (pages 82, 107).

[167] M.P. Robillard, W. Maalej, R.J. Walker, and Th. Zimmermann, editors. *Recommendation Systems in Software Engineering*, 2014. Springer. ISBN 978-3-642-45134-8. (pages 150, 151).

[168] Colin Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers*. Wiley, March 2002. ISBN 9780631213055. (page 9).

[169] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, August 2011. ISSN 0963-9314, 1573-1367. doi: 10.1007/s11219-011-9156-5. URL http://www.springerlink.com/index/10.1007/s11219-011-9156-5. (pages 25, 81).

[170] Fabricia Roos-Frantz, David Benavides, Antonio Ruiz-Cortés, André Heuer, and Kim Lauenroth. Quality-aware analysis in product line engineering with the orthogonal variability model. *Software Quality Journal*, 20(3-4):519–565, 2012. ISSN 0963-9314. doi: 10.1007/s11219-011-9156-5. URL http://dx.doi.org/10.1007/s11219-011-9156-5. (page 25).

[171] Jean-Claude Royer and Hugo Arboleda. *Model-Driven and Software Product Line Engineering*. John Wiley & Sons, March 2013. ISBN 9781118569795. (page 29).

[172] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, December 2008. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-008-9102-8. URL http://link.springer.com/10.1007/s10664-008-9102-8. (pages 8, 9).

[173] Jessica Ryan, Shahram Sarkani, and Thomas Mazzuchi. Leveraging variability modeling techniques for architecture trade studies and analysis. *Systems Engineering*, 17(1):10–25, 2014. ISSN 1520-6858. doi: 10.1002/sys.21247. URL http://dx.doi.org/10.1002/sys.21247. (page 35).

[174] C. Salinesi and R. Mazo. Defects in product line models and how to identify them. *Software Product Line-Advanced Topic*, 2012. (page 23).

[175] C. Salinesi, R. Mazo, D. Diaz, and O. Djebbi. Using integer constraint solving in reuse based requirements engineering. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 243 –251, October 2010. doi: 10.1109/RE.2010.36. (page 23).

[176] C. Salinesi, R. Mazo, D. Diaz, and O. Djebbi. Using integer constraint solving in reuse based requirements engineering. In *Requirements Engineering Conference (RE), 2010 18th IEEE International*, pages 243 –251, October 2010. doi: 10.1109/RE.2010.36. (pages 56, 84).

[177] C. Salinesi, R. Mazo, O. Djebbi, D. Diaz, and A. Lora-Michiels. Constraints: The core of product line engineering. In *2011 Fifth International Conference on Research Challenges in Information Science (RCIS)*, pages 1 –10, May 2011. doi: 10.1109/RCIS.2011.6006825. (pages 53, 75, 81).

[178] Kenneth J. Schlager. Systems engineering-key to modern development. *Engineering Management, IRE Transactions on*, EM-3(3):64–66, 1956. ISSN 0096-2252. doi: 10.1109/IRET-EM.1956.5007383. (page 11).

[179] Denny Schneeweiss and Petra Hofstedt. FdConfig: a constraint-based interactive product configurator. *arXiv:1108.5586*, August 2011. URL http://arxiv.org/abs/1108.5586. (page 122).

[180] Andy Schürr, Sebastian Oster, and Florian Markert. Model-driven software product line testing: An integrated approach. In Jan van Leeuwen, Anca Muscholl, David Peleg, Jaroslav Pokorný, and Bernhard Rumpe, editors, *SOFSEM*, volume 5901 of *Lecture Notes in Computer Science*, pages 112–131. Springer, 2010. ISBN 978-3-642-11265-2. (page 43).

[181] Zahed Siddique, David W Rosen, and Nanxin Wang. On the applicability of product variety design concepts to automotive platform commonality. In *ASME design engineering technical conference, DETC1998/DTM*, volume 5661, 1998. (page 40).

[182] Timothy W. Simpson, Zahed Siddique, and Jianxin Jiao. *Product Platform And Product Family Design: Methods And Applications*. Birkhuser, 2006. ISBN 9780387257211. (page 4).

[183] Marco Sinnema and Sybren Deelstra. Classifying variability modeling techniques. *Information and Software Technology*, 49(7):717–739, July 2007. ISSN 09505849. doi: 10.1016/j.infsof.2006.08.001. URL http://linkinghub.elsevier.com/retrieve/pii/S0950584906001042. (pages 20, 81).

[184] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In RobertL. Nord, editor, *Software Product Lines*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22918-6. doi: 10.1007/978-3-540-28630-1_12. URL http://dx.doi.org/10.1007/978-3-540-28630-1_12. (page 25).

[185] K. Slósarczyk, J. G. Linden, K. J. Burnham, K. Cockings, and R. Capolongo. Implementation of an electronic park brake feature with limited data availability, August 2008. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4616646. (page 173).

[186] Ian Sommerville. Systems engineering for software engineers. *Annals of Software Engineering*, 6(1):111–129, 1998. URL http://www.springerlink.com/index/k47239805u20j012.pdf. (page 52).

[187] D. Streitferdt, M. Riebisch, and K. Philippow. Details of formalized relations in feature models using OCL. In *Engineering of Computer-Based Systems, 2003. Proceedings. 10th IEEE International Conference and Workshop on the*, pages

297–304, 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1194811. (page 81).

[188] Maria E. Stropky and Deborah Laforme. An automated mechanism for effectively applying domain engineering in reuse activities. In *Proceedings of the conference on TRI-Ada '95: Ada's role in global markets: solutions for a changing complex world*, November 1995. (page 21).

[189] S. Subbarayan, R. M. Jensen, T. Hadzic, H. R. Andersen, H. Hulgaard, and J. Mller. Comparing two implementations of a complete and backtrack-free interactive configurator. In *Proceedings of the CP-04 Workshop on CSP Techniques with Immediate Application*, pages 97–111, 2004. (page 107).

[190] Sathiamoorthy Subbarayan. Integrating CSP decomposition techniques and BDDs for compiling configuration problems. In *In Proceedings of the CP-AI-OR. Springer LNCS*, 2005. (pages 107, 108).

[191] A. Svendsen, O. Haugen, and B. Moller-Pedersen. Using variability models to reduce verification effort of train station models. In *Software Engineering Conference (APSEC), 2011 18th Asia Pacific*, pages 348–356, December 2011. doi: 10.1109/APSEC.2011.21. (pages 84, 130).

[192] Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, ystein Haugen, Birger Mller-Pedersen, and Gran K. Olsen. Developing a software product line for train control: A case study of CVL. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Jan Bosch, and Jaejoon Lee, editors, *Software Product Lines: Going Beyond*, volume 6287, pages 106–120. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-15578-9, 978-3-642-15579-6. URL http://www.sintef.no/home/Publications/Publication/?pubid=SINTEF+S16631. (pages 46, 80, 130).

[193] OMG SysML. Omg systems modeling language (omg sysml) version 1.1. *The Object Management Group, November*, 2008. (pages 42, 70, 152).

[194] O. Taofifenua, H. Chale, T. Gaudr, A. Topa, N. Levy, and J. Boulanger. Reducing the Gap Between Formal and Informal Worlds in Automotive Safety-Critical Systems. In *21th annual INCOSE International Symposium*, Denver, USA, June 2011. Presented also at IEEE 5th Annual International System Conference, Montreal, April 2011. (pages 7, 96, 97).

[195] SysML Merge Team. Systems modeling language (SysML) specification. Technical report, 2006. URL http://www.sysml.org/docs/specs/SysML-v1-Draft-06-03-01.pdf. (page 4).

[196] P. Tessier, S. Gérard, F. Terrier, and J.M. Geib. Using variation propagation for model-driven management of a system family. *Software Product Lines*, pages 222–233, 2005. (pages 33, 86, 99).

[197] Patrick Tessier. Description de l'outil lot 4 pour la géstion des variantes. Technical Report LIST/DILS/12-0282/PT, Direction de la Recherche Thechnologique. Laboratoire d'Intégration de Systèmes et des technologies (LIST). Département Ingénierie logiciels et systèmes, November 2012. (pages 95, 100, 114, 221, 227).

[198] Patrick Tessier, Sébastien Gérard, François Terrier, and Jean-Marc Geib. Using variation propagation for model-driven management of a system family. In Henk Obbink and Klaus Pohl, editors, *Software Product Lines*, volume 3714 of *Lecture Notes in Computer Science*, pages 222–233. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-28936-4. doi: 10.1007/11554844_24. URL http://dx.doi.org/10.1007/11554844_24. (page 187).

[199] Patrick Tessier, David Servat, and Sébastien Gérard. Variability management on behavioral models. In Patrick Heymans, Kyo Chul Kang, Andreas Metzger, and Klaus Pohl, editors, *VaMoS*, ICB Research Report, pages 121–130, 2008. (pages 53, 75, 77, 80, 81, 84, 85, 86, 93, 95, 96, 114, 171, 179).

[200] Steffen Thiel and Andreas Hein. Modelling and using product line variability in automotive systems. *Software, IEEE*, 19(4):66–72, 2002. (page 74).

[201] Thomas Thüm, Christian Kästner, Sebastian Erdweg, and Norbert Siegmund. Abstract features in feature modeling. In de Almeida et al. [69], pages 191–200. ISBN 978-1-4577-1029-2. (pages 24, 43).

[202] Christian Tischer, Andreas Müller, Thomas Mandl, and Ralph Krause. Experiences from a large scale software product line merger in the automotive domain. In de Almeida et al. [69], pages 267–276. ISBN 978-1-4577-1029-2. (page 107).

[203] Salvador Trujillo, Jose Miguel Garate, Roberto Erick Lopez-Herrejon, Xabier Mendialdua, Albert Rosado, Alexander Egyed, Charles W. Krueger, and Josune de Sosa. Coping with variability in model-based systems engineering: An experience in green energy. In *Proceedings of the 6th European Conference on Modelling Foundations and Applications*, ECMFA'10, pages 293–304, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-13594-3, 978-3-642-13594-1. doi: 10.1007/978-3-642-13595-8_23. URL http://dx.doi.org/10.1007/978-3-642-13595-8_23. (pages 4, 73).

[204] M. M Tseng and J. Jiao. Mass customization. *Handbook of Industrial Engineering, 3rd edition, New York: Wiley*, pages 684–709, 2001. (pages 4, 5).

[205] Mitchell M. Tseng and Jianxin Jiao. Mass customization. In Gavriel Salvendy, editor, *Handbook of Industrial Engineering*, pages 684–710. Wiley–Interscience In

cooperation with Institute ofIndustrial Engineering, third edition edition, 2001. ISBN 0470241829. (pages x, 36, 84).

[206] Krysztof Czarnecki Ulrich W. Eisenecker, editor. *Generative Programming : Method Tools and Applications*. Addison-Wesley Professional, June 2000. (page 150).

[207] OMG Uml. 2.0 superstructure specification, 2004. (pages 29, 42, 73).

[208] Lise Urbaczewski and Stevan Mrdalj. A comparison of enterprise architecture frameworks. *Issues in Information Systems*, 7(2):18–23, 2006. (page 70).

[209] Lise Urbaczewski and Stevan Mrdalj. A comparison of enterprise architecture frameworks. *Issues in Information Systems*, 7(2):18–23, 2006. (page 70).

[210] Pascal Van Hentenryck. *Constraint satisfaction in logic programming*. MIT Press, Cambridge, MA, USA, 1989. ISBN 0-262-08181-4. (page 197).

[211] Thomas von der Maßen and Horst Lichter. Modellierung von variabilität mit uml use cases. *Softwaretechnik-Trends*, 23(1), 2003. (pages x, 30).

[212] Olivier L. De Weck, Daniel Roos, and Christopher L. Magee. *Engineering Systems: Meeting Human Needs in a Complex Technological World*. MIT Press, October 2011. ISBN 9780262016704. (page 145).

[213] David M. Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0-201-69438-7. (pages 21, 22, 23).

[214] David M. Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999. ISBN 0-201-69438-7. (pages 56, 146).

[215] Detlof Von Winterfeldt and Ward Edwards. *Decision Analysis And Behavioral Research*. University Press, 1986. ISBN 9780521253086. (page 108).

[216] Tewfik Ziadi and Jean-Marc Jézéquel. Software product line engineering with the uml: Deriving products. In Timo Käkölä and Juan C. Dueñas, editors, *Software Product Lines*, pages 557–588. Springer, 2006. ISBN 978-3-540-33252-7. (pages 31, 35).

[217] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Towards a uml profile for software product lines. In Frank van der Linden, editor, *PFE*, volume 3014 of *Lecture Notes in Computer Science*, pages 129–139. Springer, 2003. ISBN 3-540-21941-2. (page 31).

[218] Rainer Züst. *Systems Engineering – very briefly*. 1999 Orell Füssli Verlag, Zürich Verlag Industrielle Organisation, Zürich, 1999. (page 82).