TU WIEN Informatics

Model Difference Analysis for CPPS Engineering Models with Variability

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Christoph Burger, BSc
Matrikelnummer 01526642

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. **Stefan Biffl**
Mitwirkung: Projektass. Dipl.-Ing. **Kristof Meixner**, BSc

Wien, 25. November 2022

Christoph Burger	**Stefan Biffl**

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Model Difference Analysis for CPPS Engineering Models with Variability

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Christoph Burger, BSc
Registration Number 01526642

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. **Stefan Biffl**
Assistance: Projektass. Dipl.-Ing. **Kristof Meixner**, BSc

Vienna, 25th November, 2022

Christoph Burger | **Stefan Biffl**

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Erklärung zur Verfassung der Arbeit

Christoph Burger, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 25. November 2022

Christoph Burger

Danksagung

Ich möchte mich bei meinem Betreuer Stefan Biffl und Nebenbetreuer Kristof Meixner für ihre unendliche Geduld und die außerordentliche Unterstützung während der gesamten Arbeit bedanken. Außerdem möchte ich mich bei allen bedanken, die beim MDRE-Forschungsprojekt beteiligt waren und somit diese Arbeit erst möglich gemacht haben.

Was als zusammengewürfelte Gruppe von Studierenden begann, endete in einer Freundschaft, die über das Studium hinausgeht. Mein besonderer Dank gilt Christian Engelbrecht, Dominik Kretz, Mustafa Isikoglu und Alexander Prock für die Unterstützung und Motivation im Studium und während dieser Arbeit.

Zu guter Letzt möchte ich mich bei meiner Familie, meinem Partner und meinen Freunden bedanken, die viel Geduld bewiesen und mich bei all meinen Entscheidungen unterstützt haben. Ich bin mir dessen bewusst, dass die Zeit während dieser Arbeit nicht leicht mit mir war.

Acknowledgements

I would like to thank my supervisor Stefan Bifflˆ and co-supervisor Kristof Meixner for their infinite patience and extraordinary support throughout the thesis. Furthermore, I would like to thank everyone involved in the MDRE research project and thus made this work possible.

What started as a random group of students ended up in a friendship that goes beyond the studies. My special thanks go to Christian Engelbrecht, Dominik Kretz, Mustafa Isikoglu and Alexander Prock for their support and motivation during my studies and during this thesis.

Finally, I would like to thank my family, partner, and friends who have shown a lot of patience and supported me in all my decisions. Especially, in the time-intensive months of writing this thesis.

Kurzfassung

Cyber-Physical Production Systems (CPPSs), z. B. automatisierte Autofabriken, sind flexible Systeme, die aus intelligenten Maschinen und Software bestehen und sich selbstständig an wechselnde Anforderungen anpassen können. Ein CPPS produziert verwandte Produkte mit einer gemeinsamen Plattform, aber unterschiedlichen Eigenschaften, z. B. Autos in verschiedenen Konfigurationen, und bildet so eine Produktfamilie. Bei der Planung von CPPSs entwerfen die Ingenieure das CPPS unter Berücksichtigung von Produktfamilien. Daher modellieren sie die Produktionsprozesse für jedes Produkt und ermitteln deren Gemeinsamkeiten und produktspezifische Variabilität für eine integrierte Sicht auf die CPPS.

Die unzureichende Wissensdarstellung der Produktionsprozesse mit ihrer Variabilität macht diese Identifizierung zeitaufwändig, fehleranfällig und schwer reproduzierbar. Wenn sich die Produktfamilie weiterentwickelt, d. h. wenn sich die Anforderungen an das Produkt oder das CPPS ändern, führen die Ingenieure die Identifizierung der Variabilität in der Regel von Grund auf neu durch. Darüber hinaus erschwert eine begrenzte Werkzeugunterstützung mit unklarer Semantik und ohne maschinenlesbare Struktur die Analyse und das Testen und verhindert effiziente Prozessoptimierungen und Qualitätssicherung.

Diese Arbeit verwendet die Design-Science-Methodik, um den Model Variant Analysis (MVA)-Ansatz zu entwickeln, der aus (i) dem MVA-Metamodell zur Verbesserung der formalen Darstellung von Produktionsprozessen, (ii) einer Methode zur Identifizierung und Verbesserung von Merkmalen und (iii) einer Variabilitätsanalyse auf der Grundlage des Product-Process-Resource (PPR)-Ansatzes besteht.

In dieser Arbeit wird der MVA-Ansatz qualitativ evaluiert, wobei die identifizierten Anforderungen mit den bereitgestellten Fähigkeiten verglichen werden. Dazu werden typische Anwendungsfälle aus dem CPPS-Engineering verwendet und es wird untersucht, wie diese vom MVA-Ansatz profitieren können. Die Auswertung zeigt, dass der MVA-Ansatz Fähigkeiten bietet, die CPPS-Ingenieure bei ihren typischen Planungsaktivitäten unterstützen. Die Evaluierung zeigt auch, dass das MVA-Metamodell sowie die Identifizierungs- und Analyseansätze für andere Modelltypen geeignet sind.

Die Ergebnisse dieser Arbeit sollen ein wohldefiniertes Metamodell und strukturierte Modellanalyseansätze für Ingenieure bereitstellen. Allerdings ist eine empirische Evaluierung mit Branchenexperten und einer breiteren Palette von Anwendungsfällen erforderlich, um die Erkenntnisse zu erhärten.

Abstract

Cyber-Physical Production Systems (CPPSs), such as automated car manufacturing plants, are flexible systems, combined of smart machines and software, that can adapt to changing requirements autonomously. A CPPS produces related products with a shared platform but different characteristics, e.g., cars in different configurations, building a product family. In CPPSs planning, engineers design the CPPS considering product families. Therefore, they model production processes for each product and identify their commonalities and product-specific variability for an integrated view on the CPPS.

Insufficient knowledge representation of production processes with their variability makes this identification time-consuming, error-prone, and hard to reproduce. Furthermore, if the product family evolves, i.e., product or CPPS requirements change, engineers typically conduct the variability identification task from scratch. On top, limited tool support with unclear semantics and without machine-readable structure makes analysis and testing difficult, preventing efficient process optimizations and quality assurance.

This thesis uses the Design Science methodology to develop the Model Variant Analysis (MVA) approach consisting of (i) the MVA metamodel to improve the formal representation of production processes, (ii) a feature identification and improvement method, and (iii) a variability analysis based on the Product-Process-Resource (PPR) approach.

The thesis evaluates the MVA approach in a qualitative evaluation, comparing identified requirements with the provided capabilities. Therefore, this thesis utilizes typical use cases from CPPS engineering and investigates how they can benefit from the MVA approach. The evaluation indicates that the MVA approach provides capabilities that aid CPPS engineers in their typical planning activities. The evaluation also shows that the MVA metamodel as well as the identification and analysis approaches are open for other types of models.

The results of this thesis are envisioned to provide a well-defined metamodel and structured model analysis approaches for CPPS engineers. However, an empirical evaluation with industry experts and a broader range of use cases is required to harden the evidence.

Contents

Kurzfassung **xi**

Abstract **xiii**

1 Introduction **1**
1.1 Context . . . 1
1.2 Problem Description . . . 3
1.3 Aims and Results . . . 7
1.4 Thesis overview . . . 8

2 State of the Art **9**
2.1 Cyber-Physical Production Systems Engineering . . . 9
2.2 Model-Driven Engineering in the CPPS Domain . . . 11
2.3 Variability Modeling in the CPPS Domain . . . 16

3 Research Questions and Approach **19**
3.1 Research Questions . . . 19
3.2 Methodology . . . 21

4 Illustrative Use Cases **23**
4.1 Product Lines . . . 23
4.2 UC-1: Model Difference Analysis . . . 27
4.3 UC-2: Superimposed PPR Model Creation . . . 29
4.4 UC-3: Superimposed PPR Model Evolution . . . 32
4.5 UC-4: Superimposed PPR Model Variants . . . 32
4.6 UC-5: Superimposed PPR Model Analysis - Capacity Utilization . . . 35
4.7 Summary . . . 35

5 A Metamodel for Model Variant Analysis **37**
5.1 Metamodel Requirements . . . 37
5.2 Model Variant Analysis Metamodel . . . 40

6 PPR Variability Modeling Methods **47**
6.1 PPRVM-MDA - Model Difference Analysis . . . 47

6.2 PPRVM-Evolution - Superimposed Model Improvement 53
6.3 PPRVM-Evolution - Superimposed Model Analysis 61

7 Evaluation **65**
7.1 Evaluation Procedure 65
7.2 Evaluation Environment 65
7.3 Capability 1: Model Difference Analysis 66
7.4 Capability 2: Superimposed PPR Model Creation 73
7.5 Capability 3: Superimposed PPR Model Evolution - Add Variants . . 83
7.6 Capability 4: Superimposed PPR Model Evolution - Derive Variants . 90
7.7 Capability 5: Superimposed PPR Model Analysis 94

8 Discussion **99**
8.1 RQ1: Model Variability Analysis Metamodel 99
8.2 RQ2: Model Difference Analysis 100
8.3 RQ3: Model Evolution 101
8.4 Limitations 105

9 Conclusion and Future Work **107**
9.1 Conclusion 107
9.2 Future Work 110

Bibliography **111**

List of Figures **116**

List of Tables **121**

List of Algorithms **123**

Acronyms **125**

CHAPTER 1

Introduction

This chapter introduces the context of the work with a problem description and the expected aims and results of this thesis. Finally, the structure of this work is presented.

1.1 Context

Cyber-Physical Production Systems (CPPSs), like autonomous automotive plants, are highly integrated within their physical environment using sensors and software which is why they can adapt autonomously to changing conditions [Monostori, 2014, Biffl et al., 2017b]. Such CPPSs often produce a range of product variants, i.e., a product family [Van der Linden et al., 2007]. Figure 1.1 shows an example of a CPPS from the automotive manufacturing domain to assemble cars. The main aspects represented in a CPPS are the Product-Process-Resource (PPR) concepts [Schleipen et al., 2015]. For instance, in Figure 1.1, the product (red) is moved on a conveyor belt (process) to the working cells where robots (resource) assemble the car step by step. Products in a product family share a common platform but have specific additional features [Meyer and Utterback, 1992]. Kang et al. [1990] define a feature as *"a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems."* Feature examples are the color of a car or the type of rims.

The upper part of Figure 1.2 illustrates the basic principles of a production process to prepare a burger with two examples. Each rectangle describes one production step. The edges between the steps describe the flow of production. Regarding the first example **(1)** in Figure 1.2, the first step of preparing a burger is to place the bottom of the burger bun on the table, a plate, or a conveyor belt. Then the patty, cheese, and tomatoes are placed onto the bun. Finally, the top of the burger bun completes the burger. Regarding the second example **(2)** in Figure 1.2, the first step is again to place the bottom of the burger bun on the table, on a plate, or on a conveyor belt. Then the patty, tomatoes,

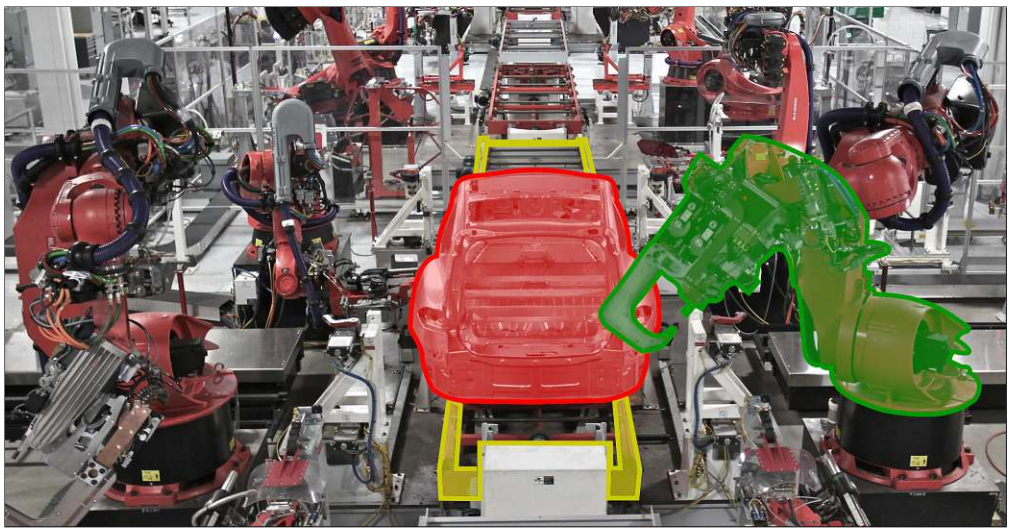


Figure 1.1: Automotive factory as an example of a CPPS with the product (red) that is assembled in a process (indicated by the conveyor belt in yellow) using a working unit (green) - *(derivative of Tesla Robot Dance by Steve Jurvetson, used under CC-BY 2.0, licensed under CC-BY 2.0 by Kristof Meixner).*

and pickles are placed on the bun. Finally, the top of the burger bun again completes the burger.

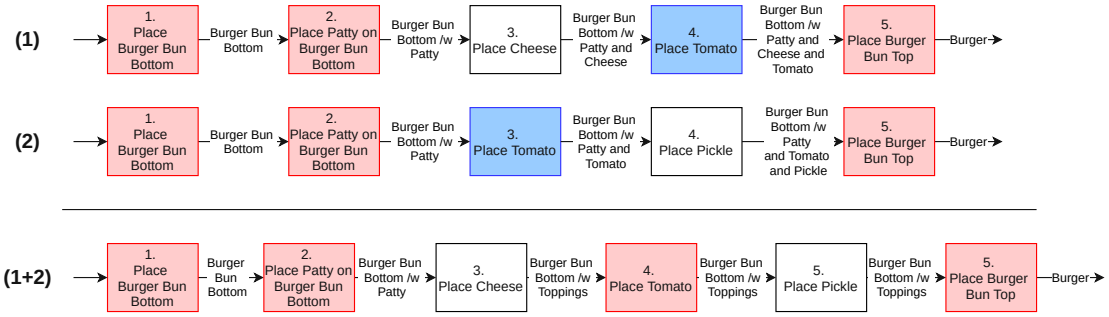


Figure 1.2: Example of preparation processes (1 and 2) for *Burgers* with different toppings and the combined production process (1+2) for both burger variants. Equal production steps are colored red. Equal production steps but in different positions are colored blue. White production steps are variant-specific and not part of the other one.

Obviously, some of the steps are equal (marked red in Figure 1.2), also regarding their position in the production process, some of them are present in both production processes but in a different position (marked blue), and some of the steps are only present in one of the production processes (marked black and white). The description of product variants regarding the production steps that the variants share (commonalities) and don't share (variability) can be described as *variability modeling*. *Variability modeling* is the

systematic modeling and managing of commonalities and variability to achieve reuse, reproducibility and to reduce maintenance effort [Bosch, 2001].

CPPSs engineering is a multidisciplinary process, where engineers of different domains work together to plan and build a CPPS [Biffl et al., 2017a]. In the basic planning phase of CPPSs, engineers have to create production processes for each product (of the product family) that the CPPS should automate. Further, engineers have to merge all relevant production processes together by identifying commonalities and variant-specific features to derive an overall design of a CPPS. The lower part of Figure 1.2 illustrates the two production processes merged together into a single model which is also called *integrated* or *superimposed* model in the frame of this work.

Merging the production processes is trivial in the case of the presented burger product line, because of only two production processes. However, merging and analyzing production processes becomes *time-consuming, error-prone, and hard to reproduce* with an increasing number of production processes Meixner et al. [2020c]. On top, the required knowledge to suitably find commonalities to merge the processes is, on the one hand, *scattered and implicit* and, on the other hand, *insufficient regarding the representation*. For instance, engineers use large spreadsheets with a few hundred rows and columns to design and merge individual production processes, due to the limitations of current tools Meixner et al. [2020c].

1.2 Problem Description

CPPS engineering comprises the design, construction, and commissioning of the production systems. Weber [2014] splits Production Systems Engineering (PSE) into nine phases, with *Phases 1-5* for planning and *Phases 6-9* for implementation (cf. Figure 1.3). Phases 1-2 concern the general project management activities to define the setting of the engineering project. Phases 3-5 concern the basic planning, which results in cost calculations, a rough concept of the production system and permit preparations that serve as the basis for binding contracts to engineers and to implement a production system. Phases 6-9 concern the detailed engineering of the CPPS, its implementation and the commissioning.

This work focuses on *Phase 3: Basic Engineering* of the presented model by Weber [2014] which involves *Basic Planners*, *Production Process Engineers* and *Quality Engineers*. The selected phase is essential for subsequent phases because the rough concept of the CPPS serves as basis for binding contracts. Planning errors that occur in this phase typically lead to high costs in subsequent phases and possibly to contract violation.

The lower part of Figure 1.3 shows the identified engineering activities in the Engineering Phase 3 with the identified challenges, respectively. *Basic Planners* design the production processes with the product requirements set by the customer. The designed production processes are then input for optimization and quality assurance, with the results being integrated back into the production processes iteratively.

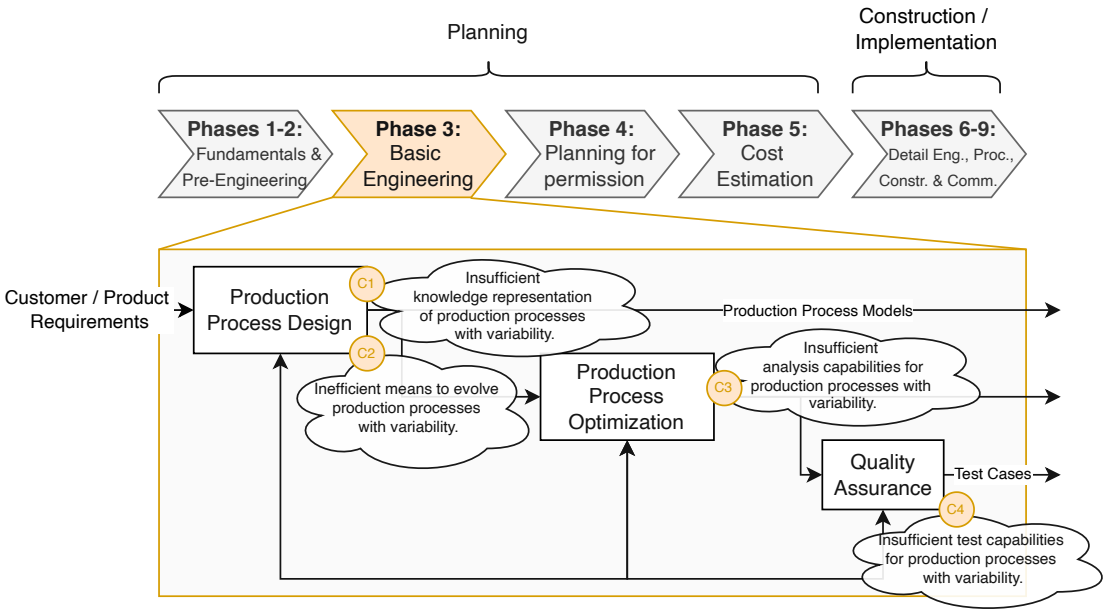


Figure 1.3: Challenges in the activities of basic engineering, based on [Weber, 2014].

This thesis focuses on the identified challenges in the activities of basic engineering and, therefore, presents the involved stakeholders, their goals, and challenges within the remainder of this subsection.

Basic Planners. *Basic Planners* are domain experts *responsible for creating an initial CPPS design* based on product and customer requirements. With the provided product requirements, basic planners create production processes with constraints between products, processes and resources, which are further used to create an initial CPPS design [Meixner et al., 2020c, Kathrein et al., 2018].

They strive for high-value solutions for customers and the engineering organization. Basic planners *want to reuse existing partial solutions*, e.g., production steps, from past projects to increase productivity [Biffl et al., 2021]. Therefore, they need to create and organize solution parts in a structured way. For instance, the *VDI 3695 - Part 3* [VDI, 2010] describes a reference model to create reusable artifacts inside and across projects.

Meixner et al. [2019] reported practitioners in industry to use spreadsheets to represent product variant matrices. Such a variant matrix comprises the product variants and their features. For example, the rows represent the available features in a product family and the columns describe the (final) product variants. Spreadsheets often have no predefined structure and semantics, such that the structure can be different for each engineering discipline or even in similar projects. The limitation of structure and semantics makes spreadsheets difficult to process by humans and machines [Chambers and Scaffidi, 2010].

Table 1.1 shows a small example of a variant matrix for the Burger product line.

Features \ Burgers	w/ Patty & Cheese	w/ Cheese	w/ Cheese & Tomato	w/ Patty & Tomato
Base (Burger Bun Top and Bottom)	×	×	×	×
Patty	×			×
Tomato			×	×
Cheese	×	×	×	

Table 1.1: Variant Matrix Example of the Burger Product Line. Rows represent the features while columns represent burger configurations.

The product line comprises four types of Burgers, *Burger with Patty and Cheese*, *Burger with Cheese*, *Burger with Cheese and Tomato* and *Burger with Patty and Tomato*. The rows contain the parts the burger is made of, i.e., the burger bun top and bottom (common for all burgers) and the other parts *Patty*, *Tomato*, and *Cheese*. For each type of burger, the respective features are marked with an *X* in the table. For example, *Burger with Patty and Cheese* comprises the base (burger bun top and bottom) and two features *Patty* and *Cheese*.

The burger product line has a rather small variant matrix. In [Meixner et al., 2019], the authors describe a real-world variant matrix with around 300 rows and 45 columns of a *rocker switch* product line. With the complexity of the variant matrix, the risk of errors increases, e.g., taking feature assignment of the wrong row, especially with the limitations of used tools, e.g., missing semantics and visualization capabilities in spreadsheet tools [Pett et al., 2019, Chambers and Scaffidi, 2010].

The variant matrices are used to derive the production processes that forms the rough concept of the CPPSs. Therefore, engineers manually identify commonalities of the production processes by comparing the production process variants and transferring the identified components into a component matrix within the same spreadsheets [Meixner et al., 2020c].

The VDI 3682 [VDI, 2005] is a notation to model the concepts of the PPR approach. To this end, the VDI 3682 allows describing and structure production processes with their products and resources, readable both by humans and machines. Production processes are represented as a sequence of *process steps* that transform one or more input-*products* to an intermediary or final *product* using specific *resources* [VDI, 2005, Schleipen et al., 2015]. Meixner et al. [2019] utitlized the approach to derive and visualize an integrated model that combines multiple production processes, named *superimposed PPR model*.

However, the superimposed PPR model has *limitations concerning the representation of*

knowledge (C1.), i.e., concepts are not well-defined. Furthermore, the *methods to evolve the superimposed PPR model are limited (C2.)*, i.e., requires recalculation from scratch every time an individual production process changes.

Production Process Optimizers. *Production Process Optimizers* are domain experts who aim to improve production process efficiency. Among others, optimal utilization of production processes is one way to increase efficiency [Weichert et al., 2019]. They analyze production processes to identify possible bottlenecks or waste of resources. Bottlenecks may be caused, e.g., by undersized machines with low throughput, which slows down the entire production process. Oversized machines can otherwise process more units than provided by the previous production step, which leads to high idle time-wasting energy and resources.

Thus, the goal of *Production Process Optimizers* is to find a balance between throughput and utilization to equally utilize components of the CPPS. They also try to find an optimal order of production steps to find an optimal production process for each product variant in the product line.

However, *insufficient analysis capabilities of current tools (C3.)*, e.g., spreadsheet tools, makes the utilization analysis error-prone and time-expensive. To reach their goal more efficiently, *Production Process Optimizers* require analysis capabilities for production processes with variability to find bottlenecks or oversized components in a CPPS. For example, they have given a superimposed model to identify optimization opportunities given the planned production volumes of the relevant products.

Quality Engineers *Quality Engineers* are domain experts whose goal is to ensure the correct behavior of production steps in production processes. In software engineering, the quality of software can be improved by using test automation, i.e., test cases are defined by engineers and automatically executed when the software changes [Garousi and Mäntylä, 2016]. Test cases are defined to show defects, especially if parts of the software change. Regarding CPPS engineering, it is also desired to ensure that certain parts of the CPPS, i.e., production processes, behave correctly. However, in contrast to software engineering, the definition of test cases requires not only the knowledge of the basic planner but also of the software engineer [Meixner et al., 2020b]. Especially for models with variability, *Quality Engineers* ideally want to test all feature combinations.

However, due to the combinatorial explosion with the increasing number of features, it is not feasible to test all feature combinations. Additionally, with spreadsheet tools, the derivation of a representative set of feature combinations is error-prone and requires significant human effort. This leads to *insufficient testing capabilities for production processes with variability (C4.)*.

Challenges To sum up, major challenges for engineers in the basic planning phase of CPPSs include (c.f. Figure 1.3)

(C1.) *Insufficient knowledge representation of production processes with variability.*

(C2.) *Inefficient means to evolve production processes with variability.*

(C3.) *Insufficient analysis capabilities for production processes with variability.*

(C4.) *Insufficient testing capabilities for production processes with variability.*

The following section presents details on the goal and the contributions of this work to address these challenges.

1.3 Aims and Results

The goal of this thesis is the design and evaluation of knowledge representations of production processes and methods to evolve, analyze, and test production processes to address the aforementioned challenges. To reach this goal and address the challenges, this thesis first derives research questions from the challenges, followed by the design for the problem solution.

In detail, the key contributions to address the challenges and research questions are:

(Co1) **The Model Variant Analysis (MVA) metamodel to represent superimposed PPR models** with concepts to visualize variability, differences, analysis results, and application-defined information. This contribution addresses challenge *(C1.)* such that engineers are provided with a suitable metamodel to represent knowledge of production processes with variability.

(Co2) **The PPRVM Model Difference Analysis (PPRVM-MDA) approach** to find differences in PPR models that conform to the MVA metamodel. This contribution addresses challenge *(C3.)* to provide a tool to analyze production processes with variability.

(Co3) **The PPRVM-FCI Add Variants (PPRVM-FCI-AV) approach** to extend the existing feature identification approaches by the ability to add production processes to existing superimposed models. The PPRVM-FCI Add Variants (PPRVM-FCI-AV) approach aims to provide an efficient tool for the evolution of production processes with variability and thus addresses the challenge *(C2.)*.

(Co4) **The PPRVM-FCI Derive Variants (PPRVM-FCI-DV) approach** to derive production process variants from superimposed models. The PPRVM-FCI Derive Variants (PPRVM-FCI-DV) approach derives representative sets of feature combinations from superimposed models that, among others, can be used for quality assurance, and thus addresses challenge *(C4.)*.

(Co5) **The PPRVM Analysis Framework (PPRVM-ANALYSIS)**, with a proof of concept analysis to calculate the utilization rate, that serves as the foundation for future types for the analysis of production processes with variability. This analysis

framework aims at providing a foundation to design analysis types for production processes with variability, which addresses challenge *(C4.)*.

The evaluation with typical engineering use cases and product lines from literature indicates that the developed approach and tools provide valuable insight to production process evolution and analysis. It showed that the MVA metamodel is suitable to represent superimposed PPR models with variability and is open to other types of models, like Unified Modeling Language (UML). The evaluation also showed a trend toward a more efficient approach to evolve superimposed models.

A more thorough and extensive empirical evaluation is required to harden the results, due to limited availability and number of industry experts and practitioners. Even though this evaluation was conducted with representative use cases from literature and industry, an evaluation with a broader range of use cases is recommended. Further, the evaluation indicated that the visual representation of the analysis of the engineering models and the superimposed model could be improved for usability to better guide the engineering experts.

1.4 Thesis overview

The remainder of this thesis is structured as follows.

Chapter 2 outlines the state of the art in the field of variability modeling in CPPS engineering and introduces the background of the concepts on which this thesis builds on.

Chapter 3 outlines the research questions of this thesis and the methodologies used to address them.

Chapter 4 outlines case studies from literature and typical use cases of engineers in CPPS engineering that are later used for evaluation.

Additionally, requirements were derived from the use cases, which significantly shape the design of the MVA metamodel presented in Chapter 5.

Chapter 6 presents methods to evolve, analyze and test superimposed models.

Chapter 7 describes the evaluation method and results.

Chapter 8 discusses the evaluation results regarding the research questions and capabilities raised in Chapters 3 and 4.

Finally, Chapter 9 concludes this thesis with the discussion of limitations and future research work.

CHAPTER 2

State of the Art

This chapter discusses the state of the art of approaches and methods relevant for this thesis in related work.

2.1 Cyber-Physical Production Systems Engineering

The term Cyber-Physical System (CPS) was coined by Lee [2006], who defines it as *"integration of computation with physical processes"*. Both, computation and physical processes affect each other, leading to feedback loops. Lee [2006] describes numerous application domains, like medical devices and systems, automotive systems, environmental control and critical infrastructure control. Research interest gained traction between 2007 and 2009, as the analysis of publications for CPS by Cardin [2019] reveals. This gain also led to several definitions in the product domain. Monostori [2014] defines CPPSs as production systems interconnected with their environment using sensors and software. CPPSs are considered to build the foundation of the fourth industrial revolution [Monostori, 2014]. The *Federal Ministry of Education and Research* in Germany, coined the term *Industry 4.0 (I4.0)* [Federal Ministry for Economic Affairs and Energy] to name the fourth industrial revolution. The main goals are increased efficiency of factories, a shorter production systems engineering cycle, and highly customizable customer-oriented products. Monostori [2014] further defined the main characteristics of CPPSs to be smart, connected, and responsive. CPPSs are, therefore, able to connect to external systems and act autonomously on internal or external changes.

Weber [2014] splits the CPPS engineering process in two main segments *Planning and Decision Phase* and *Construction / Implementation Phase* with several phases (c.f. Figure 1.3). Each phase refines the results of the previous phase. The first phase is concerned with the scope of the project and the feasibility of the CPPS. Subsequently, engineers detail on possible solutions and select the most promising. The third phase, comprises the basic design of the CPPS based on the selected, most promising, solution of

the previous phase. This is the main phase of engineering, also called *Basic Engineering* phase. With the basic design, permits are created and submitted within the next phase. The fifth phase comprises the cost calculations for implementation and operation. These calculations serve to decide on the investment. If the investment is approved, the next engineering phase - *Detail Engineering* - starts. In this phase, the basic design of phase 3 is further refined to have commission ready planning assets of the CPPS. Phases 7 and 8 comprises the procurement of the components and the construction of the CPPS. The engineering project ends with the ninth phase that comprises the commissioning of the CPPS with a final, customer approved, report.

Within the *Basic Engineering* and the *Detail Engineering* phases, engineers of different disciplines are involved. Mechanical, electrical and software engineers work in parallel to plan certain elements of the CPPS leading to a multidisciplinary environment Biffl et al. [2017a]. Each engineering discipline requires specific planning assets/perspectives of the CPPS, e.g., software engineers focus on the software while electricians create wiring plans of the CPPS components [Biffl et al., 2017a].

Due to the involvement of different engineering disciplines, also discipline-specific tools are used that require domain knowledge to understand and interpret the planning assets correctly. Especially, the correct interpretation of discipline-specific planning assets is essential to verify project-level requirements, i.e., cost or progress. Unfortunately, most domain-specific tools are not designed to seamlessly integrate with tools of other disciplines [Moser and Biffl, 2011]. However, engineers of different disciplines have to communicate their interfaces of shared concepts to ensure compatibility, e.g., the software needs to understand and process the values of a sensor. Misconceptions within this multidisciplinary environment bear the risk of costly errors if not identified early in the process. For example, misunderstandings or implicit knowledge of engineers may lead to incompatible interfaces of shared components, e.g., the software for the sensor is incompatible with the hardware interface of the sensor s.t. the software cannot interpret and process the values.

As previously described, CPPSs are considered as an integral part of *Industry 4.0*. Thus, the *Reference Architecture Model Industrie 4.0 (RAMI4.0)* [ZVEI, 2016] was developed for the interoperability of I4.0 components. The authors' goal was to provide a standardized model to describe I4.0 components. With standardization, it is possible to also have interoperability outside the business border.

The model consists of three axes.

1. Axis one (*Hierarchy Levels*) describes the hierarchy levels of standards for integrating enterprise IT and control systems.
2. Axis two (*Layers*) describes the architectural layers of an asset and its functions, communications abilities, and business behaviors.
3. Axis three (*The Life Cycle & Value Stream*) describes the lifecycle of the described asset, from planning, and production until disposal.

The contributions of this thesis are situated in the *Type – Development* phase of the *Life Cycle & Value Stream* axis, since this thesis introduces methods to create planning assets for production processes.

Further, this thesis aims at optimizing the reuse of planning assets. Thus, this thesis is situated in the project-dependent and project-independent activities described by the VDI 3695 [VDI, 2010]. The VDI 3695 is a guideline with the goal to support engineering organizations in optimizing themselves regarding internal processes, quality assurance, and risk management. The VDI 3695 provides a procedure model for project-dependent and -independent activities, to formalize engineering processes. Project-dependent activities describes task that are conducted within the scope of a specific project, e.g., tasks that apply to a specific project. Project-independent activities describe tasks that are outside the scope of a specific project, e.g., tasks that apply to multiple projects or the whole organization.

2.2 Model-Driven Engineering in the CPPS Domain

Models are abstractions of reality, often only partially or tailored to fulfill a specific task or to reach an agreement [Brambilla et al., 2017]. Traditionally in software engineering, models are used to define tasks, visualize source code for understanding, and communicate with stakeholders or customers [Brambilla et al., 2017].

In Model Driven Engineering (MDE), models are first-class citizens that drive the engineering process - *"everything is a model"* [Bézivin, 2005]. Models are the central artifact for fast prototyping, simulations, testing, and code generation. [Brambilla et al., 2017] Models are used, e.g., in the planning phase (e.g., Class-Diagrams in UML notation) or to auto-generate parts of software [Fowler, 2018].

Production Processes as Models. In discrete manufacturing, products get assembled from several parts and resources in several steps—called *production steps*. The authors of [Gupta and Krishnan, 1998] define an Assembly Sequence (AS) as *"the sequence in which the components of a product are assembled together to make up the finished product"* They emphasized, that ASs should be considered early in the design process, especially when the production system should assemble more than one product. Similarly, in process manufacturing, several production processes are required to craft a product. For example, several processes, in a specific order, are required to brew beer from the ingredients. The main distinction between discrete and process manufacturing is that the final product of discrete manufacturing can be (theoretically) disassembled into its parts while the final product of process manufacturing cannot be disassembled, e.g., due to chemical processes [Cheng et al., 2017, Brush et al., 2022].

In this thesis, the term *production process* is used as a synonym for a sequence of *assembly steps* and *production steps*.

The PPR approach by Schleipen et al. [2015] in combination with the Formalized Process Description (FPD) (VDI 3682) VDI [2005] is suitable to represent production processes, defining the concepts of products, processes, resources, and their relationships. The FPD (VDI 3682) VDI [2005] defines the semantics of *Products*, *Processes*, *Resources*, *Energy* and their dependencies using directed edges. Furthermore, *system limits* represent a system of interest. The graphical representation allows non-technical stakeholders to discuss the planning assets of the production system. Especially, when it comes to superimposed models, further described in Section 2.3.

Kathrein et al. [2019] extended the FPD (VDI 3682) VDI [2005] with the concepts of abstract types and consistency constraints. They introduced abstract types, e.g., abstract resources, to support basic engineers in early phases when there is not yet knowledge on which resource type is used. Further, consistency constraints, e.g., which dimensions of the grip are allowed to place the burger patty successfully on the burger, support engineers in their decisions on the concrete machines.

With the increasing pace of market changes also production systems need to adapt quickly to changing market needs [Biffl et al., 2017a]. Consistency constraints support engineers to make decisions on parts of the CPPS to be adapted more quickly, e.g., if the dimension of the grip is not in the allowed range for the adapted patty, the consistency constraint is violated. Further, consistency constraints enable CPPS to self-adapt to changing market needs within the limits of the used machines, e.g., robots or conveyor belts.

Meixner et al. [2020c] further enriched the FPD (VDI 3682) VDI [2005] to represent *superimposed PPR models*. They introduced visual concepts to mark elements with feature annotations and merge multiple variant models into a single model to support engineers to get an overview of the whole CPPS. The feature annotations help engineers to see commonalities and variable parts of the CPPS. However, the feature annotations do not allow tracing back the model elements to their origin variant model.

Figure 2.1 shows three production processes of different rocker switch variants in extended VDI 3682 [VDI, 2005, Meixner et al., 2019] notation. A *Rocker Switch* consists of a socket with multiple contacts and one or more rockers. *Variant 1* (leftmost in Figure 4.3) shows a basic variant with two rockers glued to the socket. *Variant 2* (second from left in Figure 4.3) shows an advanced variant with only one rocker but two changeover contacts. *Variant 3* (third from left in Figure 4.3) extends the second variant with a neutral contact. All three variants share some elements, like the socket, but also have distinct parts.

The rightmost model in Figure 4.3 shows the superimposed model resulting from combining the variants into a single production sequence. Black elements represent commonalities. Optional features, annotated using a yellow-colored trapezoid with the corresponding feature number in it. The annotated element is also highlighted with a yellow border.

This work uses the extended FPD (VDI 3682) to represent superimposed PPR models with feature annotations.

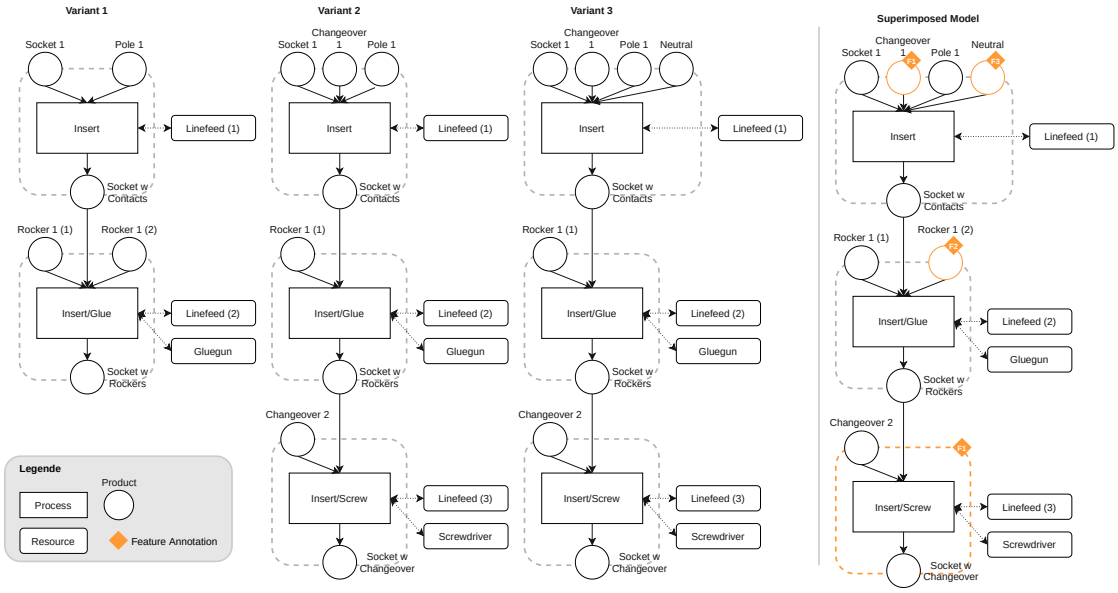


Figure 2.1: Three variants of the *Rocker Switch* product line (on the left) and the resulting superimposed model (on the right) [Meixner et al., 2019] in extended VDI 3682 [VDI, 2005] notation. The superimposed model shows optional features in hatched orange and annotates the corresponding features.

PPR Model Analysis. Difference analysis on code artifacts, i.e., formalized text, is an integral part of software engineering [Zoubek et al., 2018, Schipper et al., 2009, Ohst et al., 2003a]. Usually, a side-by-side view is used to compare two versions of the same code artifact. Textual artifacts can be compared line by line to determine which parts of the text changes, i.e., line addition, line removal [Hunt and MacIlroy, 1976].

Difference analysis on visual artifacts is different due to possibly hidden attributes of elements that are not visualized, e.g., properties. Therefore, comparison manually may only yield layout changes but may miss changes of properties that are not visualized [Schipper et al., 2009]. Therefore, algorithmic support to calculate differences is crucial for modern MDE.

Regarding differences on visual artifacts, multiple approaches were proposed lately. Schipper et al. [2009] proposed a generic approach based on the widely used side-by-side approach for textual artifacts using the Eclipse Modeling Framework (EMF). Brun and Pierantonio [2008] introduced EMF Compare, a generic approach to calculate differences on different types of models presented using UML models. Zoubek et al. [2018] emphasized the importance of visualizing model differences and introduces a *Unified Difference View* which shows both models that are compared, merged with highlights on the changes. Brunelière et al. [2015] proposed an approach to have multiple views on a model using the EMF. Ohst et al. [2003a], Schipper et al. [2009] and Zoubek et al. [2018] concluded

that a smart and optimized placing algorithm is needed to maintain the mental map. Therefore, they used either only manual or semi-automatic positioning, which uses the information on positions that are known, i.e., appear in one or both models.

This work builds on research on model difference analysis [Ohst et al., 2003a, Schipper et al., 2009, Zoubek et al., 2018, Brunelière et al., 2015] to design a model difference algorithm for (superimposed) PPR models.

Tool Support Nowadays, modeling is supported by different tools. Drawing tools, like *Microsoft Visio*[1] or *draw.io*[2], can be used to model production processes. However, these tools only provide a visual representation of the models and do not provide a suitable machine-readable format. On top, these tool often do not provide functionality to create relationships between model elements, e.g., two models may represent the same artifact but are not linked and therefore changes have to be made in both models.

The EMF [Steinberg et al., 2008] is a modeling framework built into the *Eclipse Integrated Development Environment (IDE)*[3]. It provides software developers a toolkit to define a relationship between the models and the implementation [Steinberg et al., 2008]. Models are defined in *Ecore*, which is itself an EMF model and its own metamodel. Developers can build their own applications upon *Eclipse* by defining the tool set and functionality with the tools provided by the EMF. Compared to drawing tools, developers can provide a visual and machine-readable representation of the models with EMF. However, the definition of model types and the delivery of the tailored application requires significant human effort of developers and engineers.

Another tool is the *Model Design and Review Editor (MDRE)* that was first presented in [Prock et al., 2021]. It is a web-based application to create visual models with built-in review capabilities. This editor comprises two core components, **(i)** the modeling component and **(ii)** the review component. The big advantage of this tool is that the model types can be defined using a configuration file in JavaScript Object Notation (JSON). With this, basically every model type with nodes and edges can be represented. Additionally, users can organize their models within projects to group related models. Each project has a name, project owners and project members.

The *modeling component* of the MDRE allows end-users to create models of various types. Each model has a user-defined name and a configuration assigned. Figure 2.2 shows the user interface of the modeling component of the MDRE. The left sidebar is used to manage general settings of the drawing area and contains a toolbox with available nodes and edges (defined in the configuration assigned to this model). In the center of the screen is the drawing area. New nodes can be added using drag-and-drop from the sidebar to the drawing area. Nodes can be moved / resized by the user using the mouse (click and drag) as well as multiple selected nodes can be grouped together. Nodes can

[1] https://www.microsoft.com/en-us/microsoft-365/visio/flowchart-software
[2] https://drawio-app.com/
[3] https://www.eclipse.org/ide/

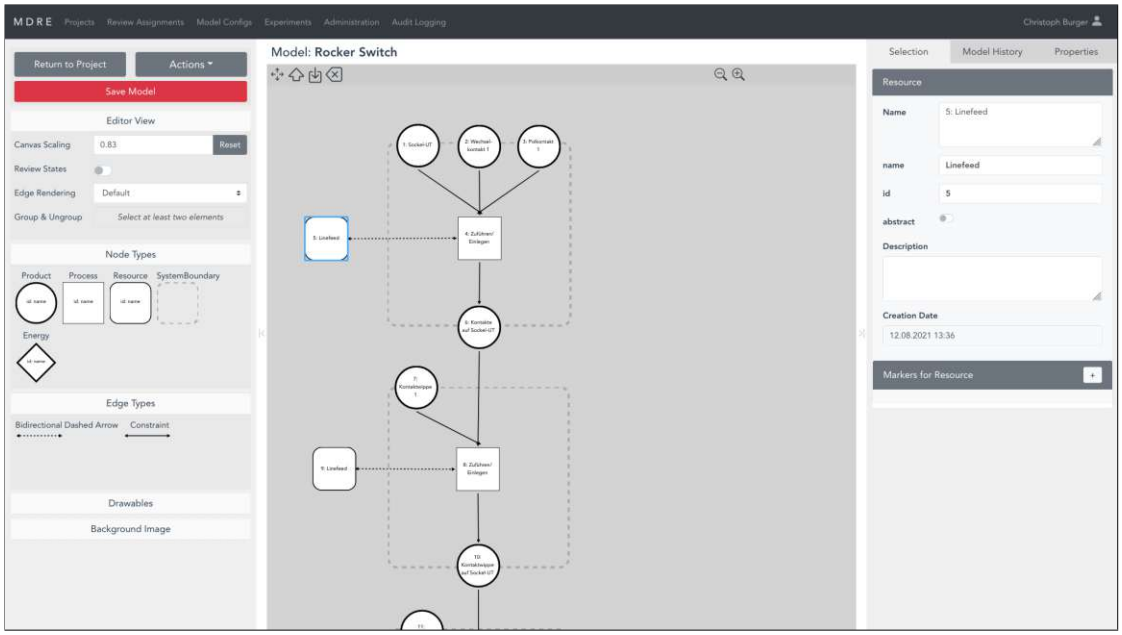


Figure 2.2: Screenshot of the MDRE Model Component

also have a parent-child relationship, if defined in the configuration. The right sidebar is used (i) to manage attributes of nodes and edges, (ii) to list the edit history of the model, and (iii) to manage properties of the model. Besides the name, description, and the creation date, other attributes defined in the configuration can be modified. Within the properties of the model, relationships to other models can be established, i.e., link from one model to another. These relationships enable the system to automatically propagate changes of model elements to linked models.

The *review component* provides engineers with a set of tools to review models as a whole or partially. Engineers can leave comments on models under review to provide feedback to modeling engineers. Since this work focuses on modeling, this component will not be discussed further.

[Engelbrecht, 2021] extended the MDRE with coordination capabilities such that engineers can link models together and make use of data propagation to automatically update dependent models. Coordinated modeling can, for example, be used to model the same artifact, e.g., a CPPS working cell, from different engineering views. For example, electricians model the electric parts of the working cell, while mechanics model the mechanical aspects of the working cell. With the coordination extension, both engineering disciplines share the same data since the update on one view updates the respective other view.

Within this thesis, the MDRE is extended with the proposed metamodel and engineering approaches for evaluation purposes. This is due to the flexibility of this editor, i.e., the definition of modeling languages in runtime configuration files. The PPR concepts of the

FPD will be defined in a configuration file to be able to create PPR models within the MDRE.

2.3 Variability Modeling in the CPPS Domain

Every product in a product family shares a common platform but has certain distinct characteristics on top Meyer and Utterback [1992], also called *features*. Kang et al. [1990] defines a feature as: *A prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems.*

Variability Modeling is the systematic modeling and managing of commonalities and variability [Bosch, 2001]. It is considered one of the most important activities for creating and managing product families [Chen and Babar, 2011]. According to Czarnecki et al. [2012], *Feature Modeling* and *Decision Modeling* are the most prominent approaches for Variability Modeling.

Feature Modeling. Feature Modeling focuses on common features (used by all variants) and variant-specific features. It was introduced by Kang et al. [1990] in 1990 as part of their work *Feature-Oriented Domain Analysis (FODA)*. They describe a feature model as a tree with the product as root and features as nodes and leaf nodes. Features can be mandatory, optional, or alternative (c.f. Legend in Figure 2.3). Figure 2.3 shows a feature model to configure a sandwich[4] [Sprey and Sundermann, 2018]. Abstract feature *Bread* is mandatory while *Cheese*, *Meat* and *Vegetables* are optional. One and only one sub-feature of *Bread* has to be selected (*xor* or *alternative*), while multiple sub-features of *Meat* can be selected (*or*). The selection of a set of features is also called *configuration*. Configurations can be valid or invalid. Invalid configurations contain features that violate feature constraints, e.g., *Full Grain* and *Flatbread* are selected but violate the *xor*-constraint, i.e., one and only one has to be selected. In valid configurations, the set of features satisfies all constraints.

This definition builds the basis of today's *Feature Modeling* approaches [Czarnecki et al., 2012]. Tools like FeatureIDE[5] can be used to create feature models in a graphical model editor.

Decision Modeling. Decision Modeling, introduced by McCabe et al. [1993] as part of the Synthesis methodology, focuses on decisions that have to be made to configure a valid product variant. Starting at a common product platform, engineers make decisions, usually by answering questions [Schaefer et al., 2012], to select features to derive a product variant. A decision model comprises product requirements, engineering decisions and the relationship between them. Engineers usually visualize decision models in a tabular or textual form [Czarnecki et al., 2012]. Figure 2.4 shows a decision model in tabular form.

[4] https://github.com/FeatureIDE/FeatureIDE/tree/develop/plugins/de.ovgu.featureide.examples/featureide_examples/ExtendedFeatureModeling/Sandwich

[5] http://www.featureide.com/

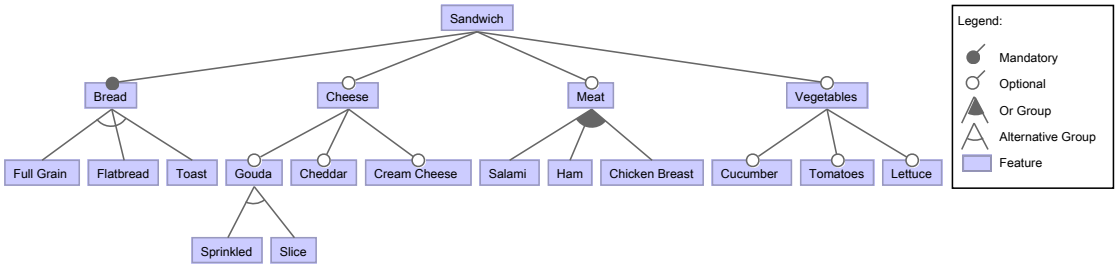


Figure 2.3: Example Sandwich Feature Model [Sprey and Sundermann, 2018] — in a tree notation from FODA [Kang et al., 1990]

decision name	description	type	range	cardinality	visible/relevant if
Bread	Which bread to use for sandwich?	Enum	Full Grain | Flatbread | Toast	1:1	
Vegetables	Put vegetables on sandwich?	Boolean	true | false		
Vegetables_Cucumber	Put cucumbers on the sandwich?	Boolean	true | false		Vegetables == true
Meat	Which meat to put on sandwich?	Enum	Salami | Ham | Chicken Breast	1:3	

Figure 2.4: Decision Model of Sandwich Example [Sprey and Sundermann, 2018] — in a tabular notation [Czarnecki et al., 2012]

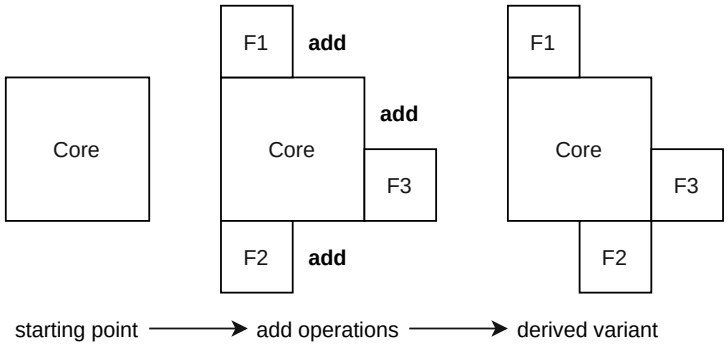


Figure 2.5: Positive variability (compositional approach) refers to adding features to a starting point to derive variants [Wille, 2019]

Positive and Negative Variability. Voelter and Groher [2007] introduced two concepts, *positive variability* and *negative variability*. Positive variability (cf. Figure 2.5) refers to a minimal starting point which gets iteratively extended by adding features. Schaefer et al. [2012] calls this approach *compositional approach*. Negative variability (cf. Figure 2.6) refers to removing parts of the model to derive a valid product configuration. This means that all product variants are included within one model. Schaefer et al. [2012] calls this approach *annotative approach* or *superimposed variants*. The initial model used, is also called *150%-model*.

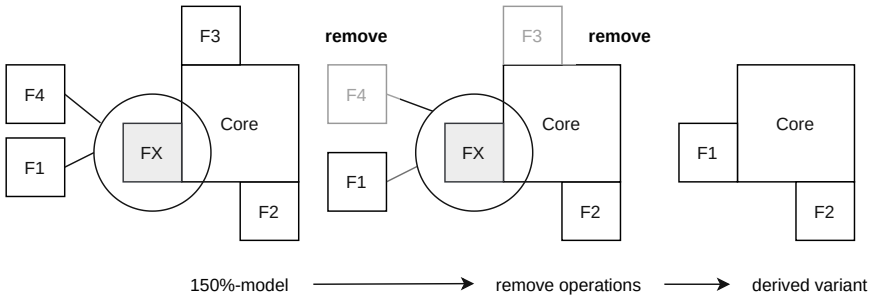


Figure 2.6: Negative variability (annotative approach) refers to removing features of a 150%-model to derive variants [Wille, 2019]

Lity et al. [2018] introduced *175% modeling* as an extension to *150% modeling* to also capture the evolution over time. Therefore, they annotate model elements not only with feature identifiers but also with element versions.

Creating such 150% or 175% models requires the identification of features in existing artifacts, such as code and models.

Feature Identification and Superimposed Models The authors of [Ziadi et al., 2012], introduced the algorithm *FCIdentification* to identify features from source code using reverse engineering and UML class diagrams. Meixner et al. [2020c] used this algorithm as a basis to create the *PPR-FCI* algorithm for Product-Process-Resource Assembly Sequence (PPR AS) and to create annotative PPR variability models – called *Superimposed PPR Models*.

The authors of [Martinez et al., 2015], also uses the *FCIdentification* algorithm by Ziadi et al. [2012] in their tool *But4Reuse*[6] to calculate features from engineering artifacts, like images, source code, EMF models, or graphs.

In this thesis, the annotative approach is used to create superimposed PPR models using the *PPR-FCI* as basis. This should help CPPS engineers to gain an overview, analyze and further improve their model of the planned CPPS.

[6] https://but4reuse.github.io/

CHAPTER 3

Research Questions and Approach

This chapter motivates the research questions and describes the research methodology.

3.1 Research Questions

Chapter 1 introduced challenges that engineers face in the basic planning phase of CPPSs. Based on these challenges and gaps in research, we derived the following research questions.

RQ1. ***MVA Metamodel.*** **What metamodel facilitates the knowledge representation required for manipulating the variability of production processes?**
The main motivation for this research questions, is the *insufficient knowledge representation of production processes with variability (C1)*. Especially for CPPS engineers, who use non-specialized tools for tasks like comparing or combining production processes, due to the limited availability of specialized tools. Insufficient semantics and visualization possibilities of spreadsheet tools [Chambers and Scaffidi, 2010], further motivate this RQ.

To address RQ1, this thesis introduces the Model Variant Analysis (MVA) metamodel to represent the required knowledge on **(i)** production processes as PPR models, **(ii)** differences between two PPR models, **(iii)** superimposed PPR models, and **(iv)** results of superimposed PPR model analysis.

In the context of RAMI4.0, it is beneficial to represent knowledge on products, processes, and resources and their relationships explicitly to standardize planning assets across teams within an organization. With the MVA metamodel, this thesis aims at less communication overhead, increased efficiency and fewer errors.

RQ2. ***Model Difference Analysis.*** **What semi-automated approach can effectively identify differences between PPR models with variability?** In Chapter 1,

we identified that engineers face the challenge of having *insufficient analysis capabilities for production processes with variability (C3)*. Especially, but not exclusively, when comparing production processes to find differences. However, the difference analysis of PPR variant models is an integral step toward the superimposed model used to build the CPPS. Thus, the main motivation of this RQ is to provide a difference analysis approach to facilitate the comparison of production processes with variability.

To address RQ2, this thesis introduces the PPRVM Model Difference Analysis (PPRVM-MDA) to support engineers in identifying differences between PPR models with variability, i.e., superimposed PPR models. The PPRVM-MDA approach builds on the MVA metamodel from *RQ1* s.t. the approach is capable to process models that comply to the MVA metamodel. Due to the automated identification of differences, the PPRVM-MDA approach should reduce errors and the required engineering effort.

RQ3. *Model Evolution.* What semi-automated approach facilitates the evolution and analysis of superimposed PPR models? The first part of this RQ concerns the evolution of superimposed PPR models. As outlined in Section 2.1, the design of CPPSs is an iterative process, starting with a rough concept that gets improved and detailed iteratively. A superimposed PPR model is usually not created only once, but extended and improved during engineering. However, in Chapter 1 we identified that engineers face *inefficient means to evolve production processes with variability (C2)*. Especially, when engineers have to take an existing superimposed PPR model and integrate further variants. On top, Quality Engineers face the challenge of having *insufficient test capabilities for production processes with variability (C4)*. Especially, due to non-specialized tools, like spreadsheets, that makes the derivation of a representative set of features to test, error-prone and inefficient.

The second part of this RQ concerns the analysis capabilities for superimposed PPR models. As outlined in Chapter 1, *Production Process Optimizer* aim to improve production process efficiency. However, we identified that Production Process Optimizer face *insufficient analysis capabilities for production processes with variability (C3)* what negatively affects their efficiency.

To address RQ3, this thesis proposes the *PPR Variability Modeling (PPRVM)-Evolution* approach to facilitate extending superimposed PPR models, i.e., add further variant models to an existing superimposed PPR model. On top, this approach facilitates the derivation of representative feature sets from superimposed PPR models. Finally, the *PPRVM-Evolution* approach provides a framework to define application-specific analysis types. This thesis provides a *proof of concept* analysis for superimposed PPR models, i.e., to analyze the CPPS component utilization in a superimposed PPR model based on the planned production volume of the variant models.

In the context of the VDI 3695 [VDI, 2010], we aim at optimizing project-dependent and project-independent analysis tasks using the *PPRVM-Evolution* approach. For example, to analyze and improve superimposed PPR and PPR variant models to standardize existing production processes for reuse in later projects.

3.2 Methodology

We use the Design Science methodology by Hevner [2007] to answer the research questions of this thesis. The Design Science framework consists of three cycles, as shown in Figure 3.1.

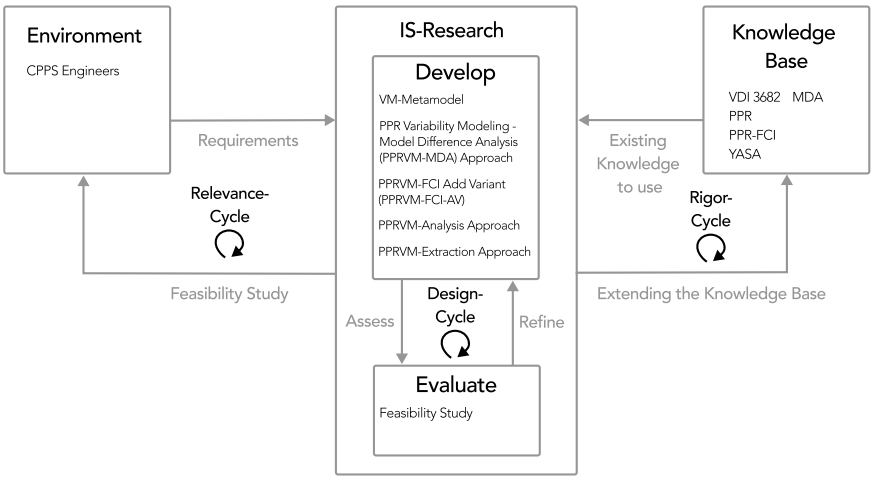


Figure 3.1: Design Science Approach based on [Hevner, 2007].

Relevance Cycle. The *Relevance Cycle* defines the context of the application and also the acceptance criteria for the evaluation [Hevner, 2007]. The cycle may be conducted multiple times, e.g., if field-testing does not provide the desired results, i.e., the designed artifact does not fulfill the defined acceptance criteria.

We described challenges that engineers face in the basic planning phase of CPPSs in Section 1.2. This thesis aims at addressing these challenges to support engineers in the basic planning phase. Therefore, we derived research questions in the previous Section 3.1 and defined an evaluation process and the acceptance criteria in the following Chapter 7.

Rigor Cycle. The *Rigor Cycle* provides the required knowledge (past knowledge seen as state of the art) for the design science project, and requires the design science project to contribute to the knowledge base [Hevner, 2007].

This thesis started with a set of base papers and builds on well-grounded concepts as described in Section 2. Among others, this thesis utilizes the FPD (VDI 3682) [VDI, 2005], its extension for superimposed PPR models by Meixner et al. [2019], the feature identification algorithm for PPR models *PPR-FCI* by Meixner et al. [2020c], and the algorithm provided by [Boubakir and Chaoui, 2018] to update a variability model given a set of variant models. These concepts are used to address *RQ2.* and *RQ3.*

This thesis goes beyond the state of the art and extends the scientific knowledge base by providing the means to (i) effectively compare, (ii) efficiently extend, and to (iii) effectively analyze superimposed PPR models.

Design Cycle. The *Design Cycle* is the heart of every design science project, with the development and evaluation of an artifact. It includes the design of alternatives and their evaluation against the requirements. This cycle iterates more often than the others to refine the design of the artifact [Hevner, 2007].

This thesis introduces the MVA metamodel that provides the basis for the approaches to compare (superimposed) PPR models (*PPRVM-MDA*), extend existing superimposed PPR models (*PPRVM-FCI-AV*), derive variant PPR models using feature combinations (*PPRVM-Extraction*), and to analyze superimposed PPR models (*PPRVM-Analysis*).

To automate selected steps of the proposed approaches, we design the *Variability Modeling Editor (VME)* based on the MDRE. We evaluate the Variability Modeling Editor (VME) in a feasibility study with the use cases presented in Chapter 4.

CHAPTER 4

Illustrative Use Cases

This chapter presents three product lines used to evaluate the MVA metamodel and the approaches presented in this thesis. Additionally, five use cases that engineers conduct in the planning phase of CPPSs are described.

All presented use cases are used to evaluate RQ1 since each of them requires the MVA metamodel for representation. UC-1 is further used to evaluate the PPRVM-MDA approach, planned to address RQ2. RQ3 is evaluated using UC-3, UC-4 and UC-5. The PPRVM-FCI-AV approach is built on UC-2 and UC-3 and evaluated using UC-3. The PPRVM-FCI-DV approach is built on and evaluated using UC-4. The PPRVM Analysis Framework (PPRVM-ANALYSIS) and the proof of concept *Capacity Utilization Analysis* is evaluated using UC-5.

4.1 Product Lines

This section introduces three every day product lines of CPPS engineers that are used in the use cases to create, evolve and analyse PPR models.

4.1.1 The Rocker Switch Product Line

Rocker Switches are everyday appliances widely used to control electronic devices, such as the lighting in buildings or small devices like coffee makers. Due to their universal application, these switches are manufactured in numerous variants Meixner et al. [2020c]. Figure 4.1 shows an example of a rocker switch and Figure 4.2 shows a schematic drawing of a rocker switch.

Rocker switches differ in size and color, and in the number of circuits they control. However, they can be assembled in similar production steps.

Figure 4.1: Rocker switch (Electronicgrup, CC BY-SA 3.0 — https://w.wiki/4Drq).

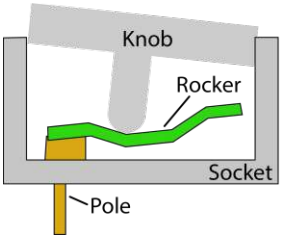


Figure 4.2: Rocker Switch Schema

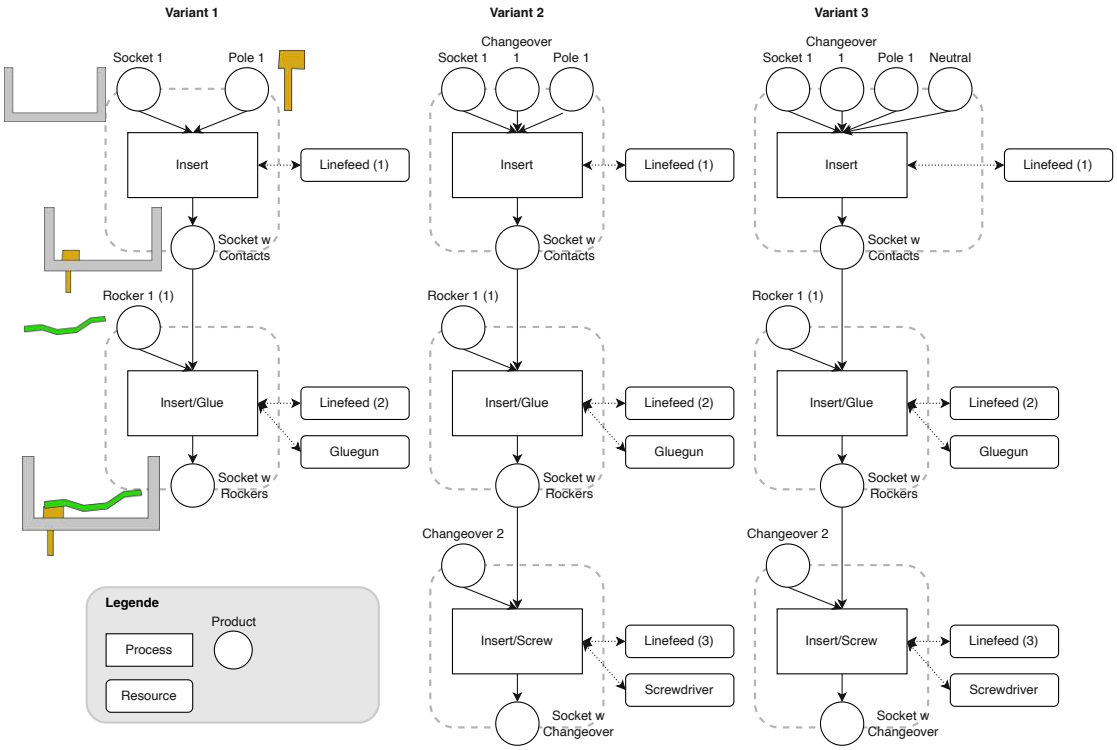


Figure 4.3: Three Assembly Sequences of different rocker switch variants, based on [Meixner et al., 2019]

Figure 4.3 shows three production processes of different rocker switch variants in VDI 3682 notation. A *Rocker Switch* consists of a socket with multiple contacts and one or more rockers. *Variant 1* (leftmost in Figure 4.3) shows a basic variant with two rockers glued to the socket. *Variant 2* (middle in Figure 4.3) shows an advanced variant with only one rocker but two changeover contacts. *Variant 3* (rightmost in Figure 4.3) extends the second variant with a neutral contact. All three variants share some elements, like the socket, but also have distinct parts.

The *Rocker Switch* product line is best described in the literature on (superimposed) PPR models, which allows us to compare our results. Therefore, this product line serves as a running example in this thesis to describe the proposed engineering approaches. Additionally, this product line was used to harden the results of the evaluation using the *Water Filter Product Line*, introduced in the next section. The *Rocker Switch* product line consists of ten variants, each with up to ten production steps. In total, this product line consists of ten unique processes, 30 unique input/output products, and 12 unique resources. This product line is considered medium-complex.

4.1.2 The Water Filter Product Line

Meixner et al. [2021] first described the *Water Filter - NanoFilter ®* product line in the context of PPR models. The *NanoFilter ®* is an invention to purify unsafe water sources like in Africa. To increase the range of applications, the filter is available in different sizes (small and large) and with different filter types (charcoal bone and charcoal active). In total, this product line consists of eight variants with up to 14 production steps. This product line is also considered medium-complex and was selected due to

The authors published a feature model and a description in the PPR Domain-Specific Language (DSL) [Meixner et al., 2020a] notation in an online repository[1]. The PPR DSL was presented by [Meixner et al., 2020a] to represent PPR models with constraints in a way that machines can process efficiently.

For the evaluation, the model description in the PPR DSL was used to recreate the variants as MDRE variant models (available in an online repository[2]). This product line is mainly used to evaluate the proposed MVA metamodel and the engineering approaches.

4.1.3 The Washing Machine Controller Product Line

This case study represents a product line for washing machine controllers in UML state chart notation, published by Rubin and Chechik [2012]. The authors describe three different controllers, comprising seven distinct states, and their respective superimposed models combining *Controller A+B* and *Controller A+B+C*.

[1] https://github.com/tuw-qse/cpps-var-case-studies/tree/main/waterfilter

[2] https://bitbucket.org/tuw-qse/msc-thesis-cburger/src/master/evaluation-models/02-water-filter/

Figure 4.4: The NanoFilter®[Model, 2022]

For the evaluation, the figures of the state charts [Rubin and Chechik, 2012] were used to recreate them as MDRE models. The created MDRE models are available in an online-resource[3].

The purpose of this case study is to show that the proposed approaches are capable of processing other models and are not custom-tailored to PPR models.

4.1.4 Summary

The presented production lines are input to the following engineering use cases. Each of the use cases is first motivated by literature and described in Restricted Use Case Modeling (RUCM) notation that specifies the context, scope, and the steps to conduct. On top, each use case results in requirements that are input to the solution approach design and the evaluation afterwards.

[3] https://bitbucket.org/tuw-qse/msc-thesis-cburger/src/master/evaluation-models/04-uml-statechart/

4.2 UC-1: Model Difference Analysis

The use case *model difference analysis* originated from an industry partner and was already abstracted as use case by Meixner et al. [2019]. This use case describes a typical task of basic planners during the initial design of CPPSs, based on customer and product requirements. This use case concerns *RQ2. Model Difference Analysis*, detailed in Section 3.1.

4.2.1 Motivation

Basic planners create a high number of production processes in the planning phase of CPPSs [Meixner et al., 2019]. During the design of new production processes, a key capability is comparing production process variants to identify similarities and differences of production process variants either designed in another project or in the current project. In the context of the VDI 3695 [VDI, 2010] it is possible, that a very similar production process was already designed in another project and generalized for reuse. CPPS engineers in later phases can, for example, reuse machine parts, like robots and conveyor belts of existing production processes.

The goal of this use case is to describe the task of identifying differences between PPR models with a semi-automated process to reduce errors that result from overlooking important aspects. Manual difference analysis is tedious and prone to error.

4.2.2 Use Case Definition in RUCM Notation

Table 4.1 represents this use case in RUCM notation [Jacobson, 1993], describing, among others, the preconditions, actors, dependencies, and the basic and alternative steps for executing this use case. The actors conduct the steps using the proposed solution for model difference analysis of PPR models, described in Chapter 6. In Chapter 7, we evaluate the fulfillment of the elicited requirements of this use case.

4.2.3 Elicited Requirements

The requirements derived from this use case focus on capabilities to analyze differences of PPR models and to visualize the results.

UC-1.R1: PPR metamodel. Requirement UC-1.R1 defines that a suitable metamodel should provide the foundation for basic planners to design ASs as PPR models with their particular elements and relations.

UC-1.R2: Selection of two models to compare. This capability should allow engineers to select the models they want to compare. These models have to follow the same metamodel.

Use Case Name	UC-1 Model Difference Analysis for a water filter product line.
Brief Description	This use case represents the workflow of an engineer to compare PPR models of the water filter product line.
Precondition	A set of PPR models for a water filter product line exists.
Primary Actor	Basic Planner.
Secondary Actors	Further engineers, like Quality Engineers or Production Process Optimizer, who view the variant models or extract information from the variant models. Further engineers who want to extract information from the variant models, e.g., Production Process Optimizer look for similarities in production process variants for sizing the required components
Dependency	No related use cases.
Generalization	No generalization.
Basic Flow	1. Basic Planner chooses two models to compare. 2. Basic Planner chooses to view the differences in an integrated manner. 3. Basic Planner chooses to highlight all types of differences. 4. The system shows an integrated model and highlights the differences, i.e., addition, deletion, change. 5. Basic Planner browses the model to see all differences.
Alternative Flows	Alternative 1 2. Basic Planner chooses to view the differences in a side-by-side manner. 4. The system shows both models side-by-side and highlights the differences on both sides. 5. Basic Planner scrolls left model and see the corresponding elements in the right model. Alternative 2 3. Basic Planner chooses not to see *addition* differences. 4. The system highlights *deletion* and *change* differences. Alternative 3 6. Basic Planner chooses another target model.

Table 4.1: UC-1 - Model Difference Analysis for a water filter product line, in RUCM notation. [Jacobson, 1993]

UC-1.R3: Highlight changes to provide visual feedback to engineers. Certain types of changes, i.e., *additions*, *deletions* and *modifications*, have to be highlighted. This capability should provide visual feedback to engineers that is easy to perceive. Engineers may want to display only certain types of changes, so the change types must be switchable.

UC-1.R4: Efficient overview on model variants. Basic Planners usually design a high number of variant models. Therefore, it is important, that engineers can compare model variants efficiently. Thus, this requirement facilitates to change the models under comparison quickly.

UC-1.R5: Side-by-side model comparison, integrated model comparison. This requirement aims to address the efficiency of the model comparison result visualization. The *side-by-side view* places both models beside and highlights differences in both models. The *integrated view* combines both models into one and highlights differences in the combined model. This results in the ability to switch between the views to have the most appropriate view for different sizes of models.

UC-1.R6: Navigate comparison view of large models. PPR models can quickly become considerable. For example, the previously described *Rocker Switch* product line has up to 9 production steps within a single PPR model. Since the screen height is limited with the vertical layout of PPR models, it is required to be able to navigate inside the comparison view (side-by-side and integrated view).

This requirement facilitates navigating large models in the comparison view. This includes the ability to zoom and move the viewport and move nodes. The latter is useful to resolve overlaps.

4.3 UC-2: Superimposed PPR Model Creation

This use case emerged during the analysis of the work of Meixner et al. [2020c] and provides the basis for the proposed approach of superimposed model improvement. Engineers usually get product requirements from customers and have to design production processes for each product that should be produced by the CPPS. Since they design a single CPPS that can produce each product of the customer, they have to merge the designed production processes into a single plan of the new CPPS. This use case describes a typical task of basic planners to derive Feature Candidates (FCs) from production processes, which are further used to detail the concrete CPPS modules. While this use case does not contribute directly to a research question of this thesis, it is added for completeness for further use cases that builds up on this use case.

4.3.1 Motivation

Basic planners need to find commonalities in the production processes relevant to the planned CPPSs. Meixner et al. [2020c] identified, that this is mainly a manual approach

Use Case Name	UC-2 Superimposed PPR Model Creation for a water filter product line.
Brief Description	This use case represents the workflow of an engineer to create a superimposed PPR model from a set of PPR variant models.
Precondition	A set of PPR variant models for a water filter product line.
Primary Actor	Basic Planner.
Secondary Actors	None.
Dependency	No related use cases.
Generalization	None.
Basic Flow	1. Basic Planner selects a set of PPR variant models. 2. The System calculates the superimposed PPR model and shows a preview of the superimposed PPR model with feature annotations. 3. Basic Planner confirms the superimposed PPR model with the identified features. 4. The System stores the superimposed PPR model and presents it to the engineer.
Alternative Flows	Alternative 1 3-4. IF the Basic Planner manually adjusts the identified features. 3. The Basic Planner adjusts the identified features, by adding/deleting a node to/from an identified feature. 4. The system stores the superimposed PPR model, preserving the customized features.

Table 4.2: UC-2 Superimposed PPR Model Creation for a water filter product line, represented in RUCM notation.

that is prone to error and requires high engineering effort. Therefore, they proposed a semi-automated approach to identify commonalities and variability—namely Feature Candidate (FC)s. However, only a prototypical implementation is available Meixner et al. [2020c] which is, at the time of writing, not integrated into an engineering tool.

While this thesis focuses on the improvement and analysis of superimposed models, this use case is integral for completeness and builds the basis for the next use cases.

4.3.2 Use Case Definition in RUCM Notation

Table 4.2 represents this use case in RUCM notation. The steps are executed using the described approach for superimposed models, depicted in Section 6.2.1. In Chapter 7, we evaluate the fulfillment of the elicited requirements of this use case.

4.3.3 Elicited Requirements

The requirements derived from this use case focus on the creation and visualization of superimposed models. Requirement *UC-1.R1* also applies to this use case.

UC-2.R1: Select a set of PPR variant models. This capability should enable engineers to select which variant models they want to use to calculate a superimposed model.

UC-2.R2: Preview of identified feature candidates. Identified feature candidates have to be presented to engineers before persisting the superimposed model.

UC-2.R3: Highlight nodes in optional features with markers. Nodes that are comprised of optional features should be highlighted with a marker, s.t. they are recognizable at first glance.

UC-2.R4: PPR metamodel representing a feature marker. The PPR metamodel, required by use case UC-1, needs to be extended to highlight model elements of identified feature candidates.

UC-2.R5: Persistent superimposed models. Superimposed models have to be persistent and modifiable like any other model.

UC-2.R6: Linking of variant models and superimposed model to propagate changes. Superimposed models have to be linked (bidirectional) to the variant models used to create. This automates the propagation of changes from superimposed model to variant models and vice-versa.

UC-2.R7: Manual adjustment of feature candidates. Engineers need the possibility to adjust the identified feature candidates. This may be required due to specific product requirements or the engineer's experience.

UC-2.R8: Deterministic feature candidate calculation. Identified feature candidates need to be deterministic, i.e., the same set of variant models yield the same set of feature candidates.

UC-2.R9: Select layers to show or hide. Superimposed models can become large. For example, the previously described *Rocker Switch* product line, has up to ten production process steps. Therefore, the capability to toggle layers facilitates showing/hiding relevant/irrelevant variants of the model to simplify the editor's view Model elements of variant models are assigned to layers. Layers logically group together model elements (nodes and edges). In this thesis, for example, model elements of the same originating variant models are grouped together using layers.

4.4 UC-3: Superimposed PPR Model Evolution

This use case describes a usual task of basic planners to improve an existing superimposed model by adding new variant models [Meixner et al., 2020c]. This use case supports *RQ3. — Model Evolution.*

4.4.1 Motivation

Meixner et al. [2020c] proposed a semi-automated approach for feature candidate identification and a visualization concept for superimposed models. However, the proposed approach is only capable to calculate feature candidates from scratch but is incapable of adding new variant ASs to an existing set of feature candidates.

4.4.2 Use Case Definition in RUCM Notation

Table 4.3 represents this use case in RUCM notation, describing, among others, the preconditions, actors, dependencies, and the basic and alternative steps for executing this use case. The steps are executed using the described approach to add variants to an existing superimposed model, described in Section 6.2.2. In Chapter 7, we evaluate the fulfillment of the elicited requirements of this use case.

4.4.3 Elicited Requirements

The requirement derived from this use case focuses on the improvement of superimposed models. Since this use case aims to be conducted using PPR models, requirement UC-1.R1 also applies to this use case. UC-2.R1 – UC-2.R9 also applies to this use case, since the resulting model, after conducting this use case, is also a superimposed model.

UC-3.R1: Integration of variant models into an existing superimposed model. It is required, that there is an algorithm that calculates feature candidates but preserves existing feature candidates of the existing superimposed model.

4.5 UC-4: Superimposed PPR Model Variants

This use case emerged during analysis of [Meixner et al., 2020b]. It describes a typical engineering task to derive a complete set of configurations from the superimposed model that serves as input for further testing tasks. This use case supports *RQ3 - Model Evolution.*

4.5.1 Motivation

Software Quality Assurance (SQA) is an integral part of software engineering to find and reduce defects that can have severe effects. Therefore, it is important that all components can be tested sufficiently. Meixner et al. [2020b] proposed an approach to generate test

Use Case Name	UC-3 Superimposed PPR Model Evolution - Add a PPR Variant Model to an existing Superimposed PPR Model for a water filter product line.
Brief Description	This use case represents the workflow of an engineer to create a superimposed PPR model from a set of PPR variant models.
Precondition	A superimposed PPR model for a water filter product line exists (UC-2). A PPR variant model for a water filter product line exists which was not used to create the superimposed PPR model.
Primary Actor	Basic Planner.
Secondary Actors	None.
Dependency	UC-2
Generalization	None.
Basic Flow	1. The Basic Planner opens the superimposed PPR model. 2. The Basic Planner chooses a PPR variant model to add to the superimposed PPR model. 3. The System calculates the updated superimposed PPR model and shows a preview of the updated superimposed PPR model with feature annotations. 4. The Basic Planner confirms the superimposed PPR model with the identified features. 5. The System updates the superimposed PPR model and presents it to the engineer.
Alternative Flows	-

Table 4.3: UC-3 Superimposed PPR Model Evolution - Add a PPR Variant Model to an existing Superimposed PPR Model for a water filter product line, represented in RUCM notation

cases from PPR models. The output of this use case may be used as input to production testing.

Engineers may define constraints (*requires*, *excludes*) between feature candidates that have a direct effect on the derived artifacts.

The goal of this use case is to support quality engineers to derive a (complete) set of configurations (variant models) from an existing superimposed model.

4.5.2 Use Case Definition in RUCM Notation

Table 4.4 represents this use case in RUCM notation, describing, among others, the preconditions, actors, dependencies, and the basic and alternative steps for executing this use case. The steps are executed using the described approach to derive variants from an existing superimposed model, described in Section 6.2.3. In Chapter 7, we evaluate the fulfillment of the elicited requirements of this use case.

Use Case Name	UC-4 Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model for a water filter product line.
Brief Description	This use case represents the workflow of an engineer to derive model variants from a superimposed PPR model for a water filter product line. The goal of this use case is, that Reuse-Engineers can derive model variants and make them reusable, e.g., for other projects/products.
Precondition	A superimposed PPR model for a water filter product line exists (UC-2).
Primary Actor	Quality Engineer.
Secondary Actors	None.
Dependency	UC-2
Generalization	None.
Basic Flow	1. The Engineer opens the superimposed PPR model. 2. The Engineer chooses to derive model variants. 3. The System calculates relevant model variants that can be extracted from the superimposed PPR model and presents it to the engineer. 4. The Engineer selects a set of model variants for extraction. 5. The System extracts the selected set of model variants and stores them in independent models.
Alternative Flows	Alternative 1 2. The engineer defines constraints on Features. The use case continues with *The Engineer chooses to derive model variants.*

Table 4.4: UC-4 Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model for a water filter product line, represented in RUCM notation,

4.5.3 Elicited Requirements

UC-4.R1: Define feature constraints. Features may require other features or must not be combined with other features. Therefore, this capability provides constraints that can be defined between features.

UC-4.R2: Calculate possible configurations to derive. This capability provides the calculation of valid configurations from feature candidates that can be used to derive variant models. The calculation shall consider feature constraints.

UC-4.R3: Preview configurations. This capability provides engineers a preview of the calculated possible configurations.

UC-4.R4: Persist derived variant models. Engineers can then select relevant configurations, which get persisted by the system as stand-alone models.

4.6 UC-5: Superimposed PPR Model Analysis - Capacity Utilization

This use case emerged during discussions with practitioners. It describes an engineering task to analyze the utilization of processes in superimposed models. This use case supports *RQ3 - Model Evolution*.

4.6.1 Motivation

Production Process Optimizers focus on improving the system under plan to optimize throughput, energy consumption and costs, among others. Therefore, they analyze ASs to find bottlenecks or waste of resources.

To prevent waste of resources, Production Process Optimizers need to analyze superimposed models to balance the utilization of the processes. Therefore, they have production volumes for each variant as input to identify bottlenecks or under-utilized processes. The results of the analysis may, for example, influence the order of ASs, i.e., a certain order of ASs better utilizes the processes.

4.6.2 Use Case Definition in RUCM Notation

Table 4.5 represents this use case in RUCM notation, describing, among others, the preconditions, actors, dependencies, and the basic and alternative steps for executing this use case. The steps are executed using the described approach to derive variants from an existing superimposed model, described in Section 6.3. In Chapter 7, we evaluate the fulfillment of the elicited requirements of this use case.

4.6.3 Elicited Requirements

UC-5.R1: Visualization of the utilization of each model element. Engineers need to see the utilization of a component at the first glance, like a heatmap provides.

UC-5.R2: Definition of planned production volume for each variant. The utilization of all components can only be calculated if it is known how often a certain variant is planned to be produced. Therefore, engineers need to define the planned production volume for each variant.

4.7 Summary

This chapter introduced three product lines from the literature with low to medium complexity that are later used to evaluate the approaches presented in the following

Use Case Name	UC-5 Superimposed PPR Model Analysis - Capacity Utilization.
Brief Description	This use case represents the workflow of an engineer to analyze the component utilization in a superimposed PPR model. The goal of this use case is, that Production Process Optimizer can identify low-/high-utilized components to further optimize the production processes.
Precondition	A superimposed PPR model exists (UC-2).
Primary Actor	Production Process Optimizer
Secondary Actors	-
Dependency	No related use cases.
Generalization	-
Basic Flow	1. The Production Process Optimizer opens the superimposed PPR model. 2. The Production Process Optimizer chooses to analyze the component utilization within this model. 3. The Production Process Optimizer enters the planned production volume of each variant. 4. The System calculates the utilization of each component and presents the result to the engineer by adding a heat-map overlay to the model. High utilization is *red*-colored, low utilization is *blue*-colored.
Alternative Flows	-

Table 4.5: UC-5 Superimposed PPR Model Analysis - Capacity Utilization, represented in RUCM notation

chapters. On top, this chapter presented five typical use cases of engineers in the basic planning phase of CPPSs. In the next chapter, this thesis uses the identified requirements of the use cases to design a metamodel to represent production processes with variability.

CHAPTER 5

A Metamodel for Model Variant Analysis

This section introduces the Model Variant Analysis (MVA) metamodel to represent and manipulate models with variability. The MVA metamodel builds the foundation for the approaches presented in the subsequent sections of this chapter.

The MVA metamodel is introduced step-wise, i.e., starting with the core (required by all presented use cases) and extending the metamodel step-wise to support Model Difference Analysis (MDA) and Model Evolution. Therefore, requirements of the metamodel were derived from the use case requirements and grouped. Finally, the proposed metamodel is introduced in the same step-wise manner.

5.1 Metamodel Requirements

In Chapter 4, we introduced five typical use cases for engineers in the planning phase of CPPSs. For each use case, we derived requirements that a suitable metamodel for variant analysis has to address.

To answer *RQ1*, we derived a set of requirements for the metamodel from the use case requirements. Table 5.1 shows which metamodel requirements maps to which use case requirement.

5.1.1 Core Requirements

This section defines the requirements for the core of the proposed metamodel. These requirements need to be addressed to conduct all the presented use cases, in Chapter 4.

Metamodel Requirements \ Relevant Use Case Requirements	UC-1.R1	UC-1.R3	UC-2.R3	UC-2.R4	UC-2.R6	UC-2.R9	UC-5.R1	UC-5.R2
MM.R1: Represent Nodes	×							
MM.R2: Represent Edges	×							
MM.R3: Group Nodes and Edges	×							
MM.R4: Node Linking	×							
MM.R5: Parent-Child Relation	×							
MM.R6: Properties	×							
MM.R7: Generic Markers								
MM.R8: Change Markers		×						
MM.R9: Cross-Model Linking					×			
MM.R10: Feature Marker			×	×				
MM.R11: Model Properties								×
MM.R12: Analysis Result Marker							×	
MM.R13: Layers						×		

Table 5.1: Mapping of Metamodel Requirements to relevant Use Case Requirements

MM.R1: Represent Nodes The metamodel shall support the definition of different types of nodes. For example, regarding use case *UC-1* such types are *Products*, *Processes*, *Resources* and *System Boundaries* for PPR models.

MM.R2: Represent Edges The metamodel shall support the definition of different kinds of links between nodes. For example, regarding use case *UC-1*, links are needed to connect products and processes, and to connect processes and resources.

MM.R3: Group nodes and edges together as model. According to *UC-1.R1*, the metamodel shall support the collection of nodes and edges into a *Model* to improve the organization.

MM.R4: Linking of nodes using edges. The metamodel shall support linking two nodes visually and to establish a relationship between them. This requirement addresses use case requirement *UC-1.R1*, such that there is not only the possibility to represent edges but also to semantically link nodes with edges.

MM.R5: Group nodes. According to *UC-1.R1*, the metamodel shall support the definition of parent-child relationships between nodes, e.g., a relationship between a system boundary and the process, resources, products comprising a production step.

MM.R6: Flexible definition of node and edge properties. The metamodel shall support a flexible definition of certain characteristics or properties of PPR aspects. For instance, a process *Bake* may have a property *temperature* which defines at what

temperature it should be baked in the oven. This metamodel requirement addresses the use case requirement *UC-1.R1*.

5.1.2 Model Difference Analysis Requirements

This section defines the requirements of the proposed metamodel to conduct use case *UC-2 - Superimposed PPR Model Creation*, presented in Chapter 4.

MM.R7: Flexible definition of generic markers for application-specific use-cases. The metamodel shall support the definition of generic markers to highlight nodes or edges. Such markers can be individually defined for each application / DSL. For example, task progress markers to mark model elements in a certain progress state: *To do*, *In Progress*, *In Review*, *Done*.

MM.R8: Mark type of change on nodes and edges. *UC-1 Model Difference Analysis*, presented in Section 4, assumes that differences are visually represented to engineers. Therefore, the metamodel shall support the representation of changes, i.e., *addition*, *deletion* and *modifications*.

5.1.3 Model Evolution Requirements

This section defines the requirements of the metamodel to conduct the use cases *UC-3*, *UC-4* and *UC-5*, presented in Chapter 4.

MM.R9: Link model elements cross-model. The metamodel shall support the definition of cross-model relationships to automatically propagate changes.

MM.R10: Mark membership of nodes and edges to features. The metamodel shall enable the visual representation of model elements that belong to the base feature or to an optional feature directly in the model.

MM.R11: Flexible definition of properties of models. Especially for analysis purposes, engineers have to define certain properties on models. For instance, for PPR variant models, engineers may want to define the planned production volume to analyze profitability. Therefore, the metamodel shall support the definition of custom properties on models.

MM.R12: Mark analysis results on nodes and edges. Model analysis requires presenting results to engineers visually. Therefore, the metamodel shall support visual markers for nodes or edges, e.g., in the form of a heatmap.

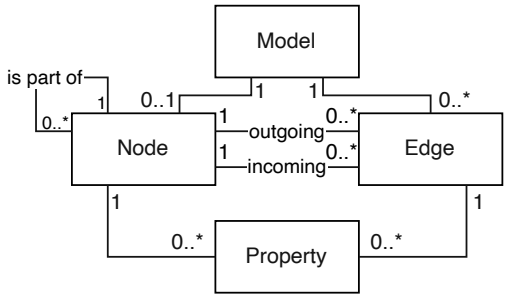


Figure 5.1: MVA Core Metamodel

Concept	Description
N; N_t	A *Node* of type t
E; $E_t(N_1, N_2)$	An *Edge* of type t between two nodes N_1 and N_2
M; $M(N, E)$	A *Model*, consists of *Nodes* and *Edges*
$partOf(N_1, N_2)$	A *part-of*-relationship between two nodes N_1 and N_2, i.e., N_1 is part of N_2.
$p(N), p(E)$	A *Property* of a Node or Edge

Table 5.2: Concepts of the MVA Metamodel Core

MM.R13: Layers containing nodes and edges. Models can quickly become large, especially superimposed models. Therefore, the metamodel shall support layers to which nodes and edges are assigned. Layers may be used to declutter the model by toggling the visibility of the layers (and the model elements, respectively).

5.2 Model Variant Analysis Metamodel

Based on the challenges, described in Section 1.2, and the elicited metamodel requirements, we derived a set of key concepts to design, update and analyze PPR models with variability, the *Model Variant Analysis (MVA) metamodel*. The MVA metamodel is introduced step-wise, starting with the core and extensions for MDA, superimposed model creation, and finally superimposed model evolution and analysis.

5.2.1 MVA Metamodel Core

The core of the MVA metamodel builds the graph-based basis for PPR modeling. The single elements are described in the following paragraphs.

Table 5.2 gives an overview of the introduced concepts, which are described below.

Model. A *Model* consists of a set of *Nodes* and *Edges*. For example, in PPR models, a model represents an assembly sequence of one or more products. This concept addresses the metamodel requirement *MM.R3*.

Node. A *Node* represents an entity that can have *Properties* and *Edges* to other/from other nodes. This concept addresses the metamodel requirement *MM.R1*. Nodes can have another node as parent or multiple nodes as children specified by the *part-of* relationship, which addresses the metamodel requirement *MM.R5*. For example, in PPR models, a *Node* represents either a *Product*, a *Process* or a *Resource*, e.g., a screw (product), an insert-and-position process, or a linefeed (resource) in an assembly sequence.

Edge. An *Edge* represents a relationship between two nodes. Different types of edges represent different types of relationships. For example, in PPR models, a directed edge is used to indicate the order of assembly and a bidirectional edge between a resource and a process indicates that the process needs this resource to work. This concept addresses the metamodel requirement *MM.R2*.

Property. Nodes and edges can have *properties* to further describe it, e.g., a node *screw* (of type product) may have a property *length* that describes the length of the screw that has to be used. This concept addresses the metamodel requirement *MM.R6*.

This metamodel core already allows representing models such as PPR ASs or UML state charts (see Chapter 7 for example models). The next section extends the MVA metamodel core to support MDA.

5.2.2 MVA Metamodel Extension - Model Difference Analysis

The previous section introduced the core of the MVA metamodel to create arbitrary graph-based models. In the planning phase of CPPSs, engineers need to compare models to further improve/evolve the models. As described in Chapter 2 and Section 4.2, differences have to be calculated and appropriately visualized. Therefore, the resulting model should have annotations using markers to highlight differences (additions, deletions, or modifications) on nodes and edges.

Figure 5.2 shows the elements that extend the MVA metamodel core to support MDA. Table 5.3 show that this extension introduces two new concepts, which are described below.

Generic Marker. *Generic Markers* are custom defined and application-specific marker types that can be assigned to nodes and edges. This concept addresses the metamodel requirement *MM.R7*.

Change Marker. A *Change Marker* is assigned to nodes and edges to indicate which type of change *addition*, *deletion*, *modification* was applied. These markers are used to visualize the differences between two models. This concept addresses the metamodel requirement *MM.R8*.

5.2.3 MVA Metamodel Extension - Superimposed Model

To support the creation and improvement of superimposed models, we further extend the metamodel. CPPS engineers require a proper representation to create and improve superimposed models. Therefore, this extension introduces *Feature Markers*, *Model Links*, and *Layers*.

Figure 5.3 shows the elements that extend the MVA metamodel core to support superimposed models. Table 5.4 show that this extension introduces three new concepts, which are described below.

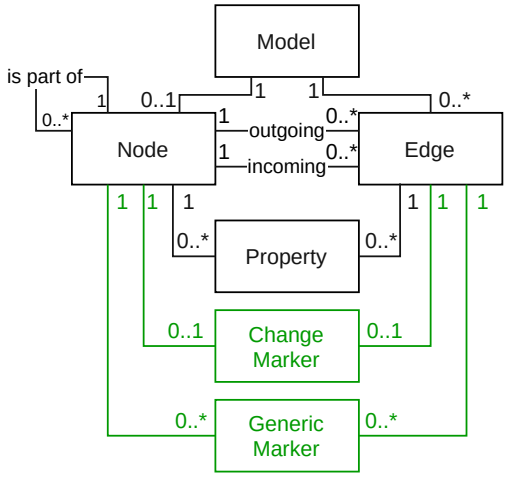


Figure 5.2: MVA Metamodel with MDA Extension

Concept	Description
$GM_t(N)$; $GM_t(E)$	A *Generic Marker* of type t of a *Node* or *Edge*
$CM(N)$; $CM(E)$	A *Change Marker* of a *Node* or *Edge*

Table 5.3: Concepts of the MDA extension for the MVA metamodel

Feature Marker. A *Feature Marker* is assigned to nodes to indicate to highlight which node belongs to which feature. This is required to distinguish between nodes of the base feature (required by all variant models) and the optional features (required only by a subset of variant models). This concept addresses the metamodel requirement *MM.R10*.

Model Link. A *Model Link* represents a link between two models. A model can have multiple incoming and outgoing links defined. These links are used to link variant models and superimposed models together, s.t. changes in the superimposed model are propagated to variant models and to correctly calculate and visualize the utilization of the components, e.g., using a heatmap. This concept addresses the metamodel requirement *MM.R9*.

Layer. *Layers* are introduced for readability and are assigned a set of nodes and edges. A *Layer* logically group nodes and edges within a model. Further, these layers are used to declutter the modeling view by toggling the visibility of the corresponding nodes and edges while working with the model. Each model can have an arbitrary number of *Layers*. This concept addresses the metamodel requirement *MM.R13*. This thesis uses this concept to toggle the visibility of nodes and edges of variant models in a superimposed model to declutter the modeling view.

5.2.4 MVA Metamodel Extension - Superimposed Model Analysis

Production Process Optimizers aim to improve the production process efficiency, as outlined in Section 1.2. For example, they may want to analyze the costs to reduce them, to arrange the production steps to minimize idle time of components, or to minimize risks

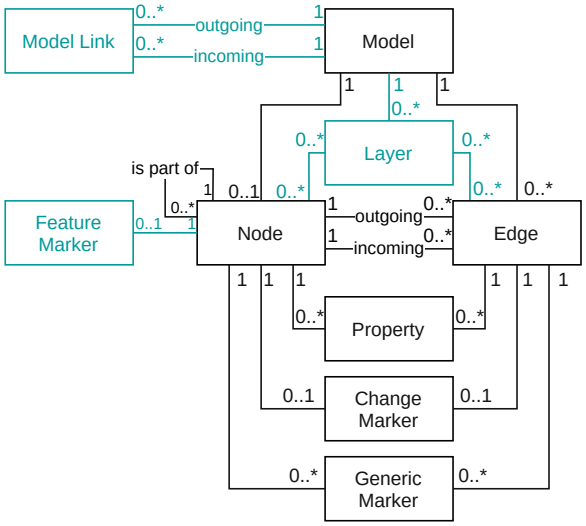


Figure 5.3: MVA metamodel with Superimposed Model extension

Concept	Description
$FM(N)$	A *Feature Marker* of a *Node*
ML; $ML(M_1, M_2)$	A *Model Link* between two models M_1 and M_2
L; $L(N)$; $L(E)$	A *Layer* containing nodes and edges

Table 5.4: Concepts of the Superimposed Model extension for the MVA metamodel

by moving high-risk production steps to the end of the production process. [Trojanowska et al., 2018]

As shown in the typical use case *UC-5*, engineers require defining properties on models, nodes, and edges which are further use to calculate some metrics. The analysis result needs to be represented properly, either in a textual or visual form.

Properties of nodes and edges were already introduced in the core metamodel, while properties on models are not included yet. Further, the metamodel lacks of a marker type to represent the analysis result of a node or edge.

Figure 5.4 shows the elements that extend the MVA metamodel to support model analysis. Table 5.5 show that this extension introduces two new concepts, which are described below.

Analysis Marker. *Analysis Markers* are used to highlight nodes and edges that are contained within the result set of a model analysis. This concept addresses metamodel requirement *MM.R12*.

Model Property. A *Model Property* further describes a model, e.g., *planned production quantity*, which defines how often it is planned to conduct an assembly sequence. These properties can, among others, be used to analyze the component utilization of a superimposed model. This concept addresses metamodel requirement *MM.R11*.

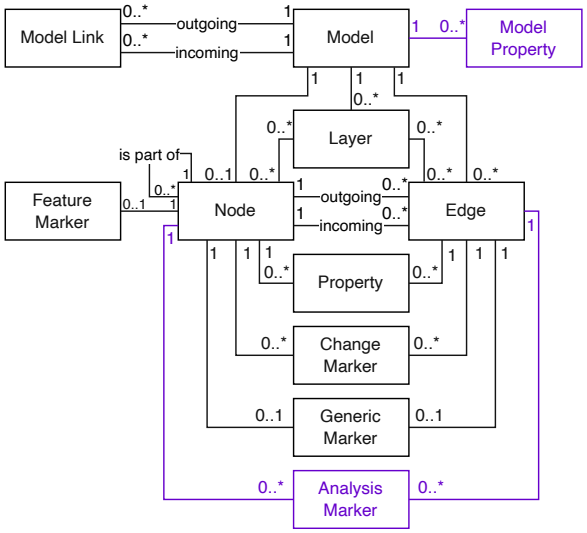


Figure 5.4: MVA Metamodel with Model Analysis extension

Concept	Description
$AM_t(N)$; $AM_t(E)$	An *Analysis Marker* of type t of a *Node* or *Edge*
$mp(M)$	A *Model Property* of a Model
ML; $ML(M_1, M_2)$	A *Model Link* between two models M_1 and M_2

Table 5.5: Concepts of the Model Evolution extension for the MVA metamodel

5.2.5 Full MVA Metamodel

To sum up, this thesis introduces the MVA metamodel using the concepts of graphs as basis with extensions to (i) support MDA, (ii) superimposed models, and (iii) their improvement and analysis.

Figure 5.5 shows the full MVA metamodel. Table 5.6 summarizes all introduced concepts with the metamodel requirements that they address.

Concept	**Concept Description**	**Metamodel Requirement**
N; N_t	A *Node* of type t	MM.R1
E; $E_t(N_1, N_2)$	An *Edge* of type t between two nodes N_1 and N_2	MM.R2, MM.R4
M; $M(N, E, FM, CM, GM, AM)$	A *Model*, consists of *Nodes*, *Edges*, *Feature Markers*, *Change Markers*, *Generic Markers* and *Analysis Markers*	MM.R3
$partOf(N_1, N_2)$	A *part-of*-relationship between twe nodes N_1 and N_2, i.e., N_1 is part of N_2.	MM.R5
$p(N), p(E)$	A *Property* of a Node or Edge	MM.R6
$GM_t(N)$; $GM_t(E)$	A *Generic Marker* of type t of a *Node* or *Edge*	MM.R7
$CM(N)$; $CM(E)$	A *Change Marker* of a *Node* or *Edge*	MM.R8
ML; $ML(M_1, M_2)$	A *Model Link* between two models M_1 and M_2	MM.R9
$FM(N)$	A *Feature Marker* of a *Node*	MM.R10
$mp(M)$	A *Model Property* of a Model	MM.R11
$AM_t(N)$; $AM_t(E)$	An *Analysis Marker* of type t of a *Node* or *Edge*	MM.R12
L; $L(N)$; $L(E)$	A *Layer* containing nodes and edges	MM.R13

Table 5.6: Complete list of concepts of the MVA metamodel

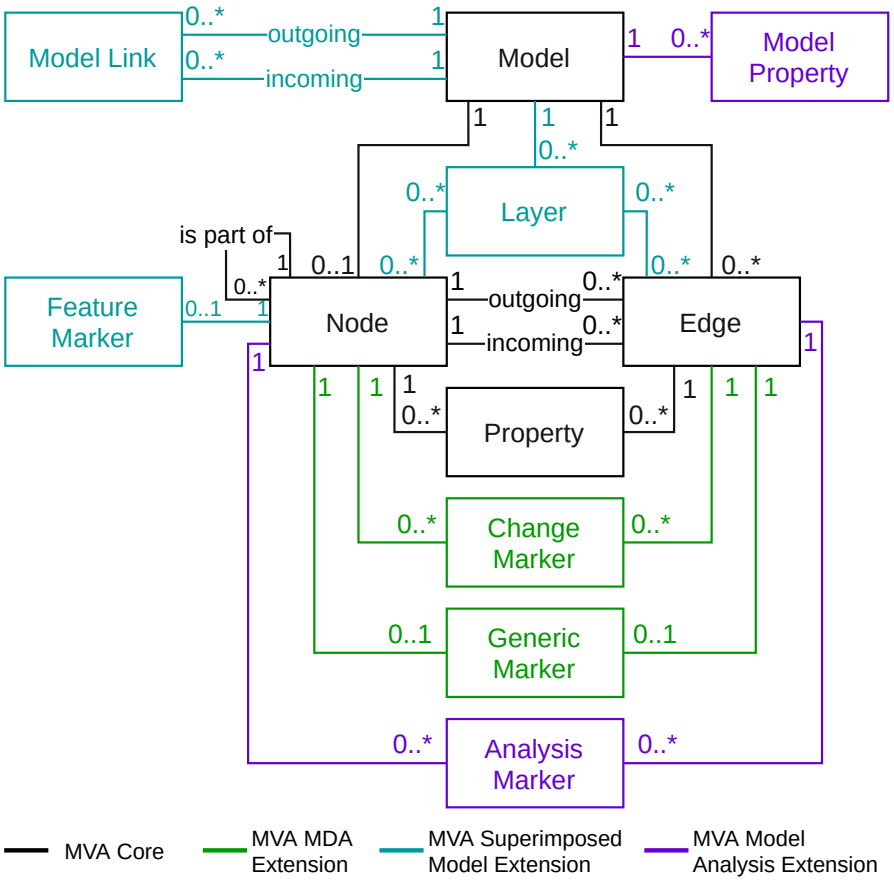


Figure 5.5: *MVA Metamodel*. Core elements are shown in *black*. Elements to support model difference analysis are shown in *green*. Elements to support superimposed models are shown in *cyan*. Elements to support analysis of (superimposed) models are shown in *violet*

CHAPTER 6

PPR Variability Modeling Methods

This chapter introduces variability modeling methods (i) to compare PPR models, (ii) to create and extend superimposed PPR models, (iii) to derive variant models from superimposed PPR models, and (iv) to analyze superimposed PPR models.

At first, Section 6.1 introduces the Model Difference Analysis (MDA) method for PPR models, which is based on the MVA metamodel, to address *RQ2*. Then, Section 6.2 describes the PPRVM-FCI Add Variants (PPRVM-FCI-AV) approach to iteratively improve existing superimposed PPR models to address the first aspect of *RQ3*. Afterwards, Section 6.2.3 describes the PPRVM-FCI Derive Variants (PPRVM-FCI-DV) approach to derivce variant models from superimposed PPR models. Lastly, Section 6.3 introduces a generic analysis framework to define different model analysis types to address the second aspect of *RQ3*.

6.1 PPRVM-MDA - Model Difference Analysis

In Section 2.2 we discussed existing approaches to calculate differences of models. This section introduces the PPRVM Model Difference Analysis (PPRVM-MDA) approach, which identifies differences between two PPR models and builds on the previously defined MVA metamodel. This approach is inspired by the work of [Zoubek et al., 2018] for visualization and by the work of [Brun and Pierantonio, 2008] for calculation.

The PPRVM-MDA approach is optimized for PPR models, but also supports other model types that conform to the MVA metamodel. This approach follows the framework proposed by Brun and Pierantonio [2008] who split calculation and representation into two separate steps. Thus, the Section 6.1.1, a model difference algorithm is presented, and Section 6.1.2 presents the visualization of the results to the engineer.

Attribute	Description
name	Arbitrary textual identifier of the model element. Does not need to be unique in a model.
type	The type of the model element, e.g., *Product*. The FPD defines the type implicitly by the representation of the element, e.g., the type *Product* is represented as circle. To be machine-readable, this type has to be defined explicitly.
parent	A link to the parent of a parent-child relationship between two nodes. Used, for example, to associate model elements of types *Product, Process, Resource, or Energy* to the system limit *SystemBoundary*.
attributes	A list of user-defined attributes that represents properties of the node.
children	A list of children belonging to the node.

Table 6.1: Attributes that extend the FPD (VDI 3682) [VDI, 2005]

To recap, the FPD (VDI 3682) [VDI, 2005] provides the means to create PPR models. A PPR model is a *Graph* with *Nodes*, of different type, and *Edges*, of different type, connecting the nodes. The FPD (VDI 3682) [VDI, 2005] defines five node types *Product*, *Energy*, *Process*, *Resource* and *System Limit* and two edge types *Flow* and *Usage*. Each node type has a unique identifier by definition.

In this thesis, we extend the concepts of the FPD VDI [2005] to have attributes besides the unique identifier. This allows to provide further details on a model element, e.g., a node of type *Process* named *Bake Dough* may have an attribute *Temperature* to give engineers a hint on what temperature the dough has to be baked. This extension has a direct effect on how to calculate differences. Table 6.1 presents the attributes introduced by this thesis that are used in the following section to describe the calculation of differences.

6.1.1 Step 1: Difference Calculation

The calculation of differences of two PPR models is inspired by the work of [Brun and Pierantonio, 2008] and custom tailored to the defined model elements and their properties. Table 6.2 details the mapping from the concepts of the Formalized Process Description (VDI 3682) to concepts of the MVA metamodel.

Equality Characteristics We previously defined the extension of the Formalized Process Description VDI [2005], which has a direct effect on how to calculate differences. Next, we define when elements are considered the equal.

In general, two model elements are considered equal if both elements have the same **ID** regardless of other attributes.

In modeling, two models may have elements that represent the same object but have different IDs, e.g., two engineers independently model the same objects. Therefore, it is required that the algorithm finds equal objects even if the IDs are different. Weighting

VDI 3682 Concept	MVA Concept
Product	Node
Energy	Node
Process	Node
Resource	Node
System Limit	Node
Flow	Edge
Usage	Edge

Table 6.2: Mapping from VDI 3682 [VDI, 2005] Concepts to MVA Concepts

Attribute	Weight
name	10%
attributes	50%
parent	20%
children	20%

Table 6.3: Attribute Weight for Equality Score Calculation

of attributes is commonly used in other difference calculation approaches [Rubin and Chechik, 2012, Kelter et al., 2005]. Thus, in this thesis, the other attributes **name**, **type**, **ID**, **parent**, **attributes** and **children** have a weighted impact on the equality score $[0, 1]$. Elements are considered equal if the equality score is above 0.8, i.e., 80 percent of the data is equal. Obviously, model elements of different type cannot be equal at all. Table 6.3 gives an overview on the attribute weights used to calculate equality. The list of *attributes* is considered most important. The more attributes are equal, the higher is the calculated equality score. The *parent* and the *children* are considered equally important, while the *name* has the less impact on the equality score. This increases the chance to find equal elements even if the model element has a different parent or different children. Especially important for assembly sequences to find equal assembly steps even if the order of steps is not the same. Due to the previous definition for type equality, the *type* is not considered in the weighted score calculation.

Differences in positions are not considered at all to preserve the mental map of engineers [Ohst et al., 2003b, Schipper et al., 2009, Diehl and Görg, 2002]. It would also prevent to find similar assembly steps when in different order. Therefore, different positions of the same model element is not considered by the algorithm. In practice, this may lead to overlapping in the visualization, but can be compensated by giving engineers the ability to modify the positioning within the result model. This issue and our solution are further described in Section 6.1.2.

Model Difference Algorithm The designed algorithm to calculate differences in PPR models is presented as pseudocode in Algorithm 6.1. It takes two PPR models M_1

and M_2 as input and returns a set of operations to be applied to M_1 to derive M_2. This algorithm handles three cases, which were identified the most common for PPR models:

1. *Addition*: Set of model elements (nodes or edges) that are present in M_2 but not in M_1 (see Algorithm 6.1 line 2).
2. *Deletion*: Set of model elements that are present in M_1 but not in M_2 (see Algorithm 6.1 line 3).
3. *Modify*: Set of model elements that are present in both models M_1 and M_2 but have different parents in the respective model. (see Algorithm 6.1 line 4)

Each of these sets are then mapped to sets of operations with details including the operation type (*ADD*, *REMOVE* or *MODIFY*), the value of the given operation and a path that operation has to be applied on (see Algorithm 6.1 lines 5-7). The structure of an operation is based on the JavaScript Object Notation (JSON) Patch (RFC-6902) [Bryan and Nottingham, 2013].

Algorithm 6.1: Model Difference Algorithm for MVA Models, inspired by [Rubin and Chechik, 2012, Brun and Pierantonio, 2008]

Input: Two PPR AS models M_1 and M_2, where each M_x is a set of model elements

Output: DO, the set of operations to be applied to M_1 to derive M_2

1 $DO = \emptyset$;
2 $ME_{added} = \{me_i | me_i \notin M_1 \wedge me_i \in M_2\}$;
3 $ME_{removed} = \{me_i | me_i \in M_1 \wedge me_i \notin M_2\}$;
4 $ME_{modified} = \{me_i | me_i \in M_2 \setminus \{ME_{added} \cup ME_{removed}\} \wedge \exists me_j \in similarModelElementsInModel(M_2, me_i) \wedge me_j$ has different parent than $me_i\}$\$;
5 $DO := DO \cup \{mapToOperation(me_i, \text{"added"}) | me_i \in ME_{added}\}$;
6 $DO := DO \cup \{mapToOperation(me_i, \text{"removed"}) | me_i \in ME_{removed}\}$;
7 $DO := DO \cup \{mapToOperation(me_i, \text{"modified"}) | me_i \in ME_{modified}\}$;
8 **return** DO

Algorithm 6.2 formalizes the described similarity heuristics.

6.1.2 Step 2: Difference Visualization

The visualization aims at easing the identification of differences for engineers. Given two PPR models, selected by the engineer, the algorithm previously described, calculates the differences and returns a set of operations.

The PPRVM-MDA approach should display the differences (*addition*, *deletion*, *modify*) (inspired by [Zoubek et al., 2018]) within the models, providing two different view options

Algorithm 6.2: Similarity Heuristic of Model Difference Algorithm for MVA Models

similarModelElementsInModel (M, m)
  Input: A PPR AS model M and a model element m
  Output: SME, the set of similar model elements to m in model M
  $SME := \{me_i | me_i \in M \wedge calculateSimilarityOfModelElements(m, m_i) \geq 0.8\}$;
  return SME

calculateSimilarityOfModelElements (m_1, m_2)
  Input: Two model elements m_1 and m_2
  Output: s, the similarity value
  if $m_1.id = m_2.id$ **then**
    return 1.0
  end
  if $m_1.type \neq m_2.type$ **then**
    return 0.0
  end
  $s_{name} := 1.0$;
  if $m_1.name \neq m_2.name$ **then**
    $s_{name} := 0.0$;
  end
  $s_{parent} = calculateSimilarityOfModelElements(m_1.parent, m_2.parent)$;
  $s_{children} := 1.0$;
  foreach $m_{c1} \in m_2.children$ **do**
    $m_{c2} := findFirstSimilarModelElement(m_{c1}, m_1.children)$;
    $s_{children} := s_{children} * calculateSimilarityOfModelElements(m_{c1}, m_{c2})$;
  end
  $s_{attributes} := \frac{numberOfEqualAttributes(m_1.attributes, m_2.attributes)}{numberOfUniqueAttributes(m_1.attributes, m_2.attributes)}$;
  $s := s_{attributes} * 0.5 + s_{parent} * 0.2 + s_{name} * 0.1 + s_{children} * 0.2$;
  return s

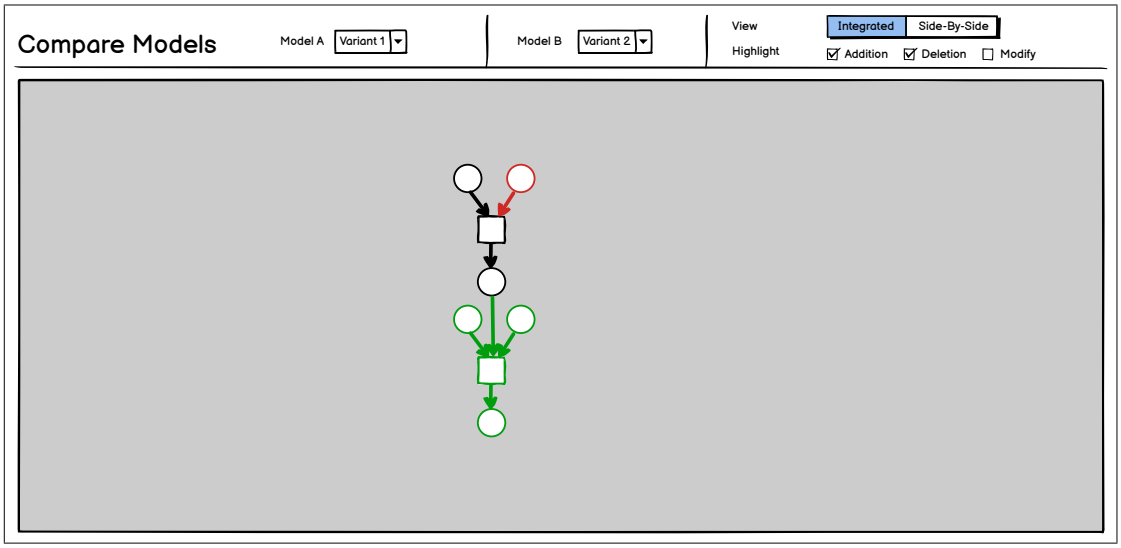


Figure 6.1: Concept of the integrated view of two models in the PPRVM-MDA

Integrated and *Side-By-Side*. The *integrated* view, as presented in Figure 6.1, combines both models into a single model and highlights the differences. The layout of the first model serves as basis for the merged layout to maintain the mental-map of the developers. Schipper et al. [2009] defined such merge approach as *incremental merge*. With the *Side-By-Side* view, as presented in Figure 6.2, the first model is presented on the left and the second model on the right. To increase usability, both model views can be linked (option *Mirror Interactions*) s.t. changing the viewport or the zoom-level is mirrored to the respective other canvas.

In both views, engineers can easily change the models under comparison. This enables engineers to compare product variants efficiently and without the need to go through the initial model selection. Further, engineers can customize which operation types (*addition*, *deletion*, *modify*) are highlighted.

Both views should allow moving model elements, changing the zoom-level and the position of the viewport of the canvas.

As described in the previous section, the algorithm, and the visualization approach should not lay out model elements automatically. Thus, the original positions of the model elements, in their origin models, should be used. Since this can lead to overlapping model elements, we require that it shall be possible to move model elements in both views.

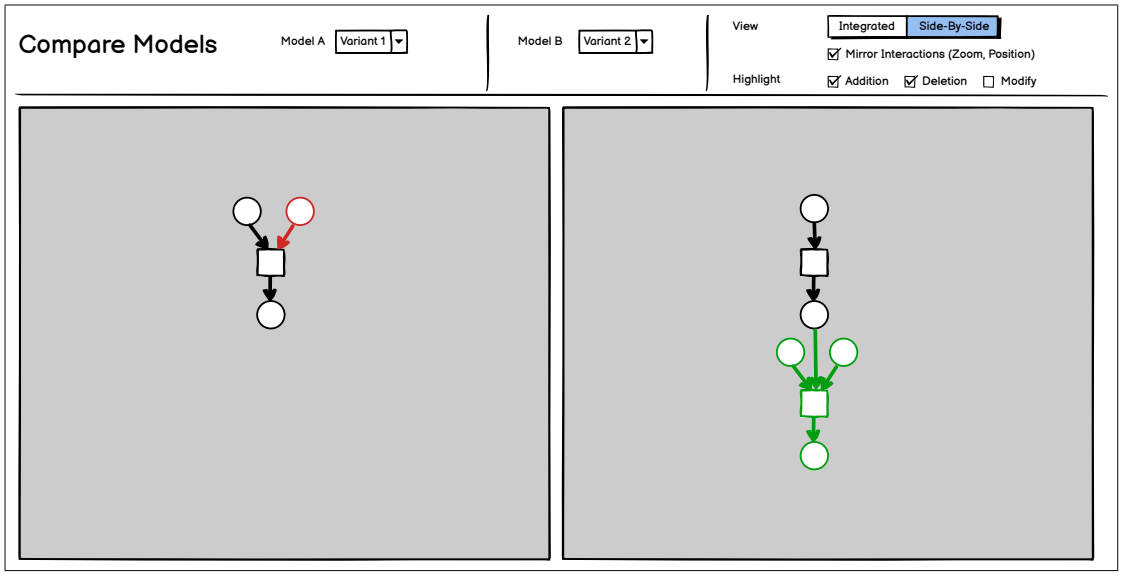


Figure 6.2: Concept of the side-by-side view of two models in the PPRVM-MDA

6.2 PPRVM-Evolution - Superimposed Model Improvement

The *PPR Model Feature Candidate Identification (PPRVM-FCI)* approach has limitations regarding model improvement, i.e., cannot use a superimposed model as input to integrate further variant models. Motivated by the typical engineering use case *UC-3 - Superimposed PPR Model Evolution* (see Section 4.4), this thesis introduces the PPRVM-FCI-AV approach to add variant models (PPR ASs) to an existing superimposed PPR model.

Further, this section deals with the derivation of variant models with feature combinations (configuration) from a superimposed model. Motivated by use case *UC-4* (see Section 4.5), this gives engineers the ability to derive configurations based on feature constraints, which is especially useful for quality engineers. Section 6.2.3 describes the integration of an existing sampling algorithm.

6.2.1 Feature Identification and Superimposed Models

As previously described, the existing *PPR-FCI* approach by Meixner et al. [2020c] is only suitable to calculate feature candidates from scratch. This thesis puts focus on the integration of further model variants to an existing superimposed model, i.e., an existing set of feature candidates. However, the *PPR-FCI* is used in this thesis to initially calculate feature candidates to create a superimposed model. Therefore, some adaptions are required to suite the MVA metamodel and the internal data structure of the MDRE.

This thesis follows the framework proposed by Brun and Pierantonio [2008] who splits calculation and representation into two separate steps.

Step 1: Calculation

In Section 2.3 we gave an overview about existing feature identification algorithms. We also concluded that the *PPR-FCI* algorithm by Meixner et al. [2020c] suits our needs best. The authors published a prototypical implementation of their algorithm in a GitHub repository[1]. Adaptions to the algorithm were necessary to handle the data structures of the MDRE.

Construction Primitives The *PPR-FCI* [Meixner et al., 2020c] algorithm is designed to find commonalities and variability in a set of *Construction Primitives (CPs)*. A CP is the smallest unit which describes an element uniquely but allows finding similar (or equal) elements. In general, as this approach is metamodel-independent, the data of the construction primitives depends on the respective metamodel. However, due to the architecture of the MDRE some properties are predefined such as the *name* for nodes and *source* and *target* node of edges. Further, some internal properties like the model element *ID* (nodes and edges) and *ID of the parent* (nodes only) are also contained within the CP. This is because these properties are required for each model element to be handled by the MDRE, regardless of the metamodel they belong to.

In case of PPR models, a CP of a *Node* consists of:

- *id*: The internal ID used by the MDRE and not defined by the engineer.
- *parentId*: The internal ID of the parent node used by the MDRE.
- *name*: The name of the node.
- *attributes*: A set of attributes with types defined in the metamodel and values defined by the engineer.
- *annotations*: A set of annotations that belongs to the node.

However, only the properties *name* and *attributes* are considered for comparison.

A CP of an edge consists of:

- *id*: The internal ID used by the MDRE and not defined by the engineer.
- *name*: The name of the node.
- *attributes*: A set of attributes with types defined in the metamodel and values defined by the engineer.
- *annotations*: A set of annotations that belongs to the node.
- *sourceId*: The internal ID of the source node.

[1] https://github.com/tuw-qse/ppr-fci

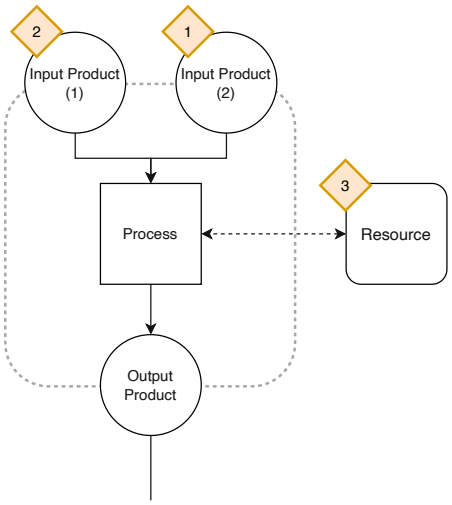


Figure 6.3: Concept of the representation of a production step in superimposed PPR models with feature annotation. Based on Meixner et al. [2019]

- *targetId*: The internal ID of the target node.

However, only the properties *name* and *attributes* are considered for comparison.

Feature Candidate Identification (FCI) algorithm for PPR models As already mentioned, we use the *PPR Feature Candidate Identification (PPR-FCI)* proposed by Meixner et al. [2020c] to calculate feature candidates of a set of models. The *PPR-FCI* itself is based on the *FCIdentificaton* algorithm proposed by Ziadi et al. [2012]. The *PPR-FCI* takes a set of models where each model is a set of CPs as input and returns a set of feature candidates of the input models.

Due to how the MDRE works internally, an additional step is necessary to update the internal IDs, especially of the source and target nodes of edges, to preserve the validity of the resulting feature candidates.

Step 2: Superimposed Model Visualization

The visualization of the resulting feature candidates is essential to engineers. As described in Section 2.3, there exists two different approaches of variability, namely *annotative* and *compositional* variability. Due to the elicited use case requirements, the annotative approach seemed promising.

Figure 6.3 presents the representation of an production step in a superimposed PPR model based on Meixner et al. [2019]. In contrast to other approaches, like the one used by But4Reuse [Martinez et al., 2014, Ziadi et al., 2012], this approach uses the visualization concept *Feature Annotation* proposed by Meixner et al. [2019]. This concept is used to mark which model element belongs to which feature. This can be seen in

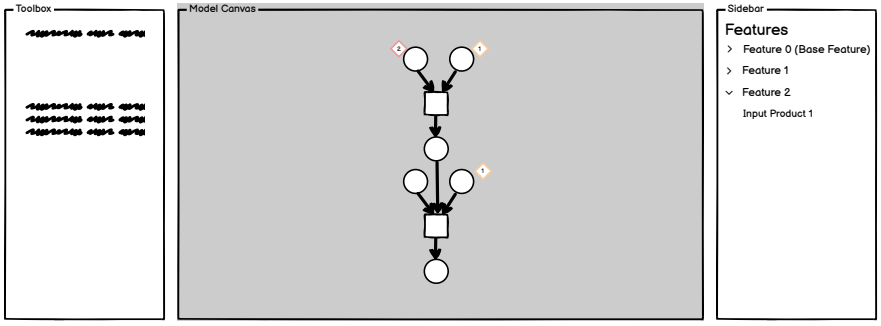


Figure 6.4: Concept of the Representation of Superimposed PPR Models with Feature Annotations

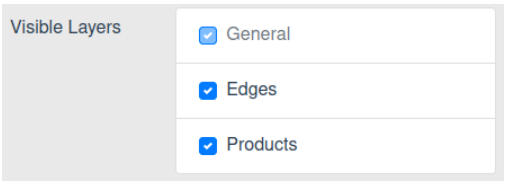
Figure 6.3 where the input product *Input Product (2)* has a yellow-bordered trapezoid with a light-yellow background and a number, representing the feature number, on the top left of the element.

Figure 6.4 shows a concept for a modeling view that supports superimposed PPR models. The view is split up in three areas **(i)** the toolbox on the left containing interaction options, e.g., to add new model elements or save the model, **(ii)** the model canvas in the center that renders the superimposed model with possibilities to manipulate (e.g., to move model elements) the model and **(iii)** a sidebar on the right that contains the features of the model as list to provide an overview for large models where only parts of the model fit into the model canvas.

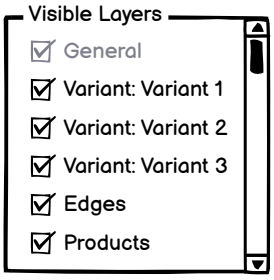
The MDRE already provided the concept of layers, which allows metamodel designers to define different layers and which node or edge types belong to exactly one layer. Layers allow the engineer to toggle the visibility of model elements inside a layer. In this thesis, the layer concept should be used to toggle the visibility of model elements belonging to a specific variant which were previously selected to create the superimposed model. Figure 6.5a shows layers defined by the metamodel, representing the current state in the MDRE. Figure 6.5b shows a concept of layers defined by the metamodel and dynamically defined by the selection of model variants which were used to create the superimposed model. The *General* layer is introduced by the system and contains all model element (types) not assigned to a user- or another application-defined layer.

6.2.2 Add Variants to Superimposed PPR Models

As described in Section 1.2, it is essential to engineers to add further variant models to an existing superimposed model. Stakeholders may come up with new product variants, which leads to new variant models created by engineers who later have to adapt the existing superimposed model to include the new variant model. Unfortunately, existing algorithms like the *PPR-FCI* by Meixner et al. [2020c] for PPR models and *FCIdentification* by Ziadi et al. [2012] are designed to calculate the feature candidates



(a) Screenshot of the list of layers in the MDRE defined by the Metamodel Designer.



(b) Concept of layers of Superimposed PPR Model, including dynamically layers defined by the selected variant models.

Figure 6.5: List of Layers in the MDRE and extension introduced by PPRVM-MDA to toggle visibility of model elements.

from scratch. Thus, adding a new variant requires repeating the feature candidate calculation.

This thesis proposes the *PPR-FCI-AV* algorithm, inspired by [Boubakir and Chaoui, 2018], that takes an existing set of feature candidates and a set of CPs of the variant PPR models to add. In an iterative process, each CP of the new variant models is processed without the need to calculate all feature candidates from scratch. The algorithm handles three possible cases:

Case 1: CP is only present in new variant model(s): This is the trivial case, which introduces a new feature candidate since the CP is not present in existing feature candidates. Figure 6.6 illustrates this case, showing three models, the starting point on the left, the new variant model in the middle and the resulting superimposed PPR model on the right. Starting point is a superimposed PPR model with three features in total, the base feature *Feature 0* and two optional features *Feature 1* and *Feature 2*. A new variant PPR model introduces a new resource *5: Resource* element which introduces a new feature *Feature 3* comprising this resource element. This results in a superimposed PPR model with four features in total, the base feature *Feature 0* and three optional features *Feature 1-3*.

Case 2: CP is present in existing optional feature candidate(s) but not in new variant model(s): In this case, a new feature candidate is created if the other CPs in the same feature candidate the CP belongs to, are not present in the new variant models. Otherwise, feature candidates remain the same. Figure 6.7 illustrates this case, showing the base superimposed PPR model on the right, the new variant PPR model in the middle and the resulting superimposed PPR model on the right of the figure. The base superimposed PPR model has two features, the base feature *Feature 0* and one optional feature *Feature 1*. *Feature 1* comprises two model elements, *4: Product* and *5: Resource*. The variant PPR model to be added

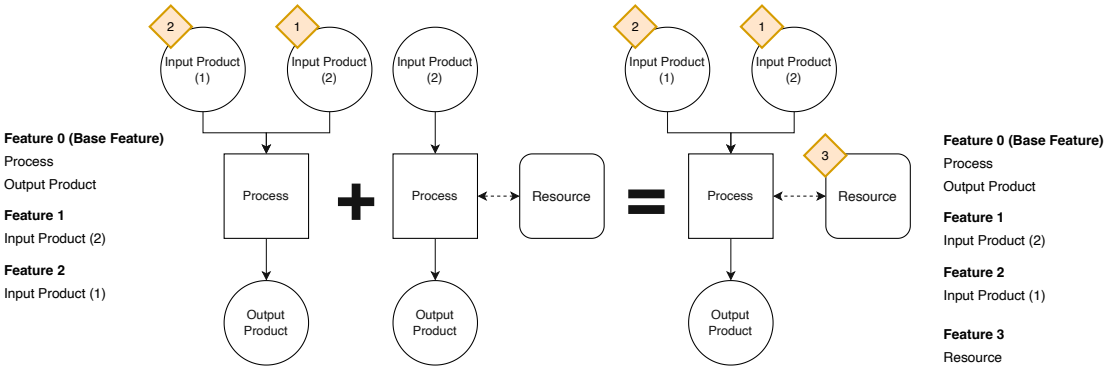


Figure 6.6: PPR-FCI-AV Case 1: *Resource* creates new *Feature 3*

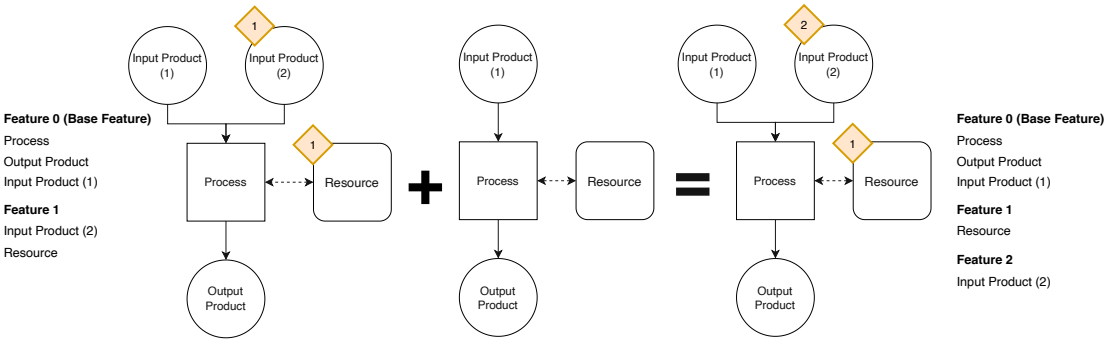


Figure 6.7: PPR-FCI-AV Case 2: *Input Product (2)* moves from *Feature 1* to new Feature *Feature 2*

does not have the model element *4: Product*. Thus, the resulting superimposed PPR model gets an additional feature *Feature 2* comprising the model element *4: Product* and *Feature 1* only comprises the model element *5: Resource*.

Case 3: CP is present in base feature candidate but not in new variant model(s): In this case, a new feature candidate is introduced and the CP removed from the base feature candidate. Figure 6.8 illustrates this case, showing three models, the initial superimposed PPR model on the left, the variant PPR model to be added in the middle and the resulting superimposed PPR model on the right. The initial superimposed PPR model has three features, the base feature *Feature 0* and two optional features *Feature 1-2*. The model element *5: Resource* is comprised in the base feature but not present in the variant PPR model that gets added. Thus, the resulting superimposed PPR model gets a new feature *Feature 3* comprising the model element *5: Resource*, which is not a part of the base feature anymore.

This algorithm reduces the calculation resources needed due to the iterative process to add a new variant CPs to an existing set of feature candidates. In addition, changes to

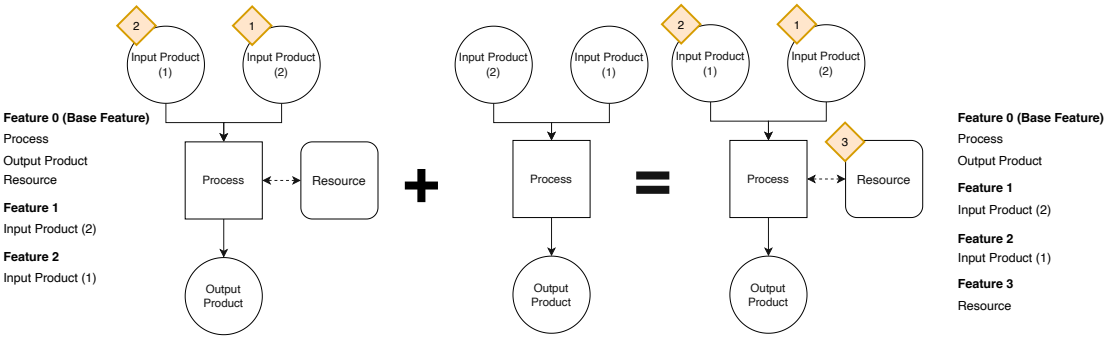


Figure 6.8: PPR-FCI-AV Case 3: *Resource* moves from *Feature 0 (Base Feature)* to new *Feature 3*

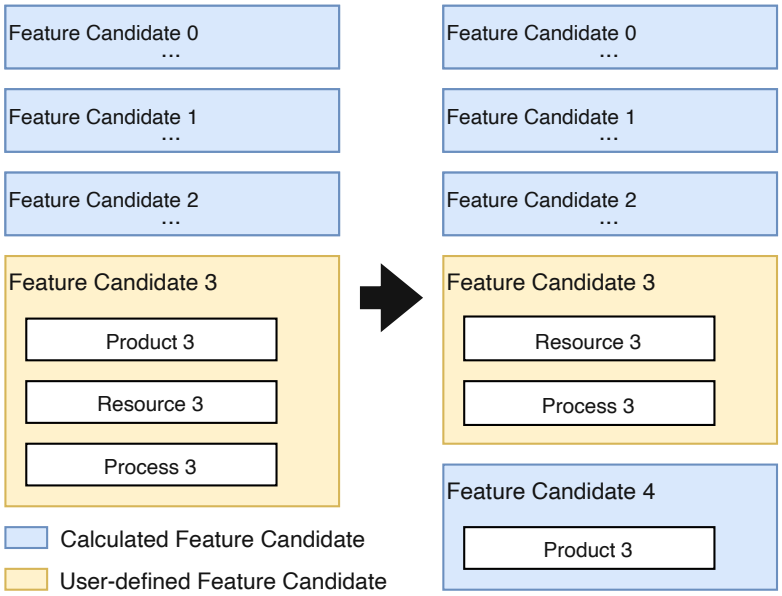


Figure 6.9: PPR-FCI-AV Edge Case: *Product 3* introduces a new feature candidate s.t. the user-defined feature candidate cannot be fully preserved.

the feature candidate set taken by the engineer, that would be lost at recalculation from scratch, are considered. However, preserving user-defined feature candidates is not always possible. For example, if some elements of a feature candidate introduce a new feature. Figure 6.9 illustrates this edge case. The left-hand side shows the feature candidates after the creation of the superimposed model. *Feature Candidate 3* is user-defined, while the system calculated the others. The right-hand side shows the result after adding another variant which introduced a new feature candidate containing *Product 3*. Obviously, user-defined feature candidates cannot be fully preserved if *Case 2* occurs.

The described algorithm is formalized in Algorithm 6.3. Lines *5-6* describe *Case 1*, lines

7-11 describe *Case 2* and lines *12-15* describe *Case 3*. Lines *17-18* ensure that existing feature numbers are preserved.

Algorithm 6.3: PPRVM-FCI Add Variants (PPRVM-FCI-AV) inspired by [Boubakir and Chaoui, 2018]

Input: CPs of base feature CP_b, a set $FC = \{FC_1, ..., FC_n\}$ of optional feature candidates, where each FC_x is a tuple (fc_n, CP_n) of feature number fc_n and a set of construction primitives CP, and a set of PPR models $AllM = \{M_1, .., M_m\}$, where each M_y is a set of PPR CPs

Output: $FC_{updated}$, the set of feature candidates of the models including the base feature

1 $FC_{updated} = FC$;
2 $CP_{all} =$ *distinct list of construction primitives* $CP_b \cup \bigcup_{(fc_i, CP_i) \in FC} CP_i$;
3 $CP_{optional} = CP_{all} \setminus CP_b$;
4 **foreach** $m_i \in AllM$ **do**
5 $\quad cp = \{cp_j | cp_j \in m_i \wedge cp_j \notin CP_{all}\}$;
6 $\quad FC_{updated} = FC_{updated} \cup \{(|FC_{updated}| + 1, cp)\}$;
7 $\quad$ **foreach** $(fc_k, cp_k) \in FC$ **do**
8 $\quad\quad cp = cp_k \setminus m_i$;
9 $\quad\quad FC_{updated} = FC_{updated} \cup \{(|FC_{updated}| + 1, cp_k \setminus cp)\}$;
10 $\quad\quad cp_k = cp_k \setminus cp$;
11 $\quad$ **end**
12 $\quad cp = \{cp_j | cp_j \in CP_{all} \wedge cp_j \notin m_i\} \cap CP_b$;
13 $\quad FC_{updated} = FC_{updated} \cup \{(|FC_{updated}|, cp)\}$;
14 $\quad CP_b = CP_b \setminus cp$;
15 $\quad CP_{all} = CP_{all} \setminus cp$;
16 **end**
17 *sort* $FC_{updated}$ *by feature number ascending*
18 *reassign feature numbers in* $FC_{updates}$ *to index + 1*
19 **return** $(0, CP_b) \cup FC_{updated}$

6.2.3 Extract Variants from Superimposed Models

The last part for superimposed model improvement concerns extracting/derivation variant models from existing superimposed models. As already mentioned in Chapters 1 and 4, the derivation of variants from a superimposed model is an integral task while planning CPPSs.

In the context of testing, the derivation of a representative set of configurations (variant models) is important. Due to the fast-growing configuration space with each added variant, it is important to retrieve a small but representative set of configurations that covers each feature candidate at least once [Krieter et al., 2020].

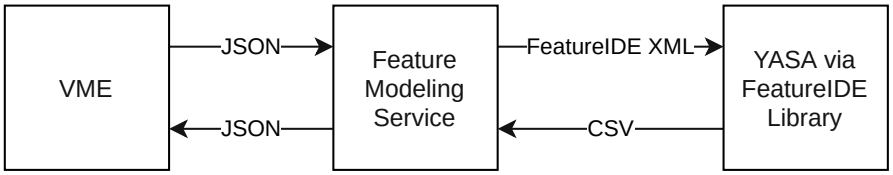


Figure 6.10: Architecture of the integration of the YASA

This thesis introduces the semi-automated approach PPRVM-FCI-DV to derive variant models (configurations) from superimposed models that **(i)** automatically calculates possible configurations, **(ii)** let the engineer choose which configurations to derive, and **(iii)** automatically persists all selected configurations as models.

There exists a multitude of sampling techniques in the literature, as summarized in Chapter 2. In this thesis, we utilize the *yet another sampling algorithm (YASA)* proposed by Krieter et al. [2020] to derive a set of variant models. We chose YASA due to its scalability capability for large configuration spaces and its availability as standalone Java library[2].

The YASA java library only supports a few input formats, with *FeatureIDE XML* among others. Thus, we decided to map our superimposed PPR models to a feature model in *FeatureIDE XML* format first. Afterwards, the XML data is provided to the Java library, which outputs the sampled configuration in a Comma-separated values (CSV) format.

The integration of the YASA is loosely coupled to the VME, i.e., we introduced the component *Feature Modeling Service* to wrap the FeatureIDE library with a standardized Application Programming Interface (API). This approach allows changing the sampling implementation easily, e.g., change from YASA provided by the FeatureIDE library to a custom-tailored sampling algorithm. Figure 6.10 presents a schematic view on how the YASA is integrated into the VME. The *VME* communicates with the *Feature Modeling Service* through a JSON API which then transforms the input to a FeatureIDE XML and calls the YASA sampler on the XML file. After the execution of the YASA finishes, a CSV list with a set of configurations is returned, transformed by the *Feature Modeling Service* back to a JSON format and sent back to the VME.

6.3 PPRVM-Evolution - Superimposed Model Analysis

To reasonably improve the superimposed models, engineers must be able to analyze the superimposed models. For example, to find bottlenecks or to improve throughput.

This thesis introduces a generic analysis framework PPRVM-ANALYSIS for engineers to define analysis-types. Further, we provide an exemplary analysis type to analyze *component utilization* [Ragan, 1976] in superimposed models.

[2] `https://github.com/FeatureIDE/FeatureIDE/tree/develop/plugins/de.ovgu.featureide.fm.core/library`

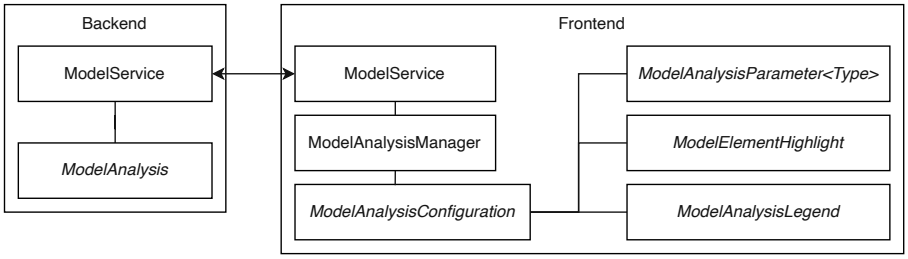


Figure 6.11: Concept of the PPRVM-ANALYSIS framework. The framework separates the concerns into the *Frontend* that handles the representations, and the *Backend* that does the calculation.

The MDRE already provides the possibility to define arbitrary analysis types. However, without the possibility to provide (user-defined) parameters. Therefore, the MDRE gets extended with the following framework.

Figure 6.11 illustrates the concepts of the framework. The *Backend* module consists of a *ModelService* that has multiple model analysis types (interface *ModelAnalysis*) registered. This interface is used for the concrete analysis implementation.

The *Frontend* module consists of a *ModelService* that communicates with the backend module and has a *ModelAnalysisManager* which is used to register analysis implementations. These analysis implementations are named *ModelAnalysisConfiguration* that comprises a list of parameters of any type (*ModelAnalysisParameter*), a definition to highlight model elements (*ModelElementHighlight*) that are part of the analysis result, and a legend definition of the analysis result (*ModelAnalysisLegend*).

A concrete example for this analysis framework is provided in the following subsection.

6.3.1 Component Utilization in Superimposed (PPR) Models

This subsection describes a concrete example of a model analysis type with the previously described analysis framework.

This example is motivated by typical engineering use case *UC-5 - Superimposed PPR Model Analysis - Capacity Utilization*. Production process optimizer tries to identify bottlenecks, waste of resources or energy. Therefore, they need to analyze the superimposed model created and improved previously.

This concrete example analyzes the component utilization of all processes within the superimposed model by providing the planned production volume of all model variants (used to create the superimposed model). The goal is to visualize processes that are well utilized, i.e., used for every variant and those which are under-utilized, e.g., only used for a small set of variants with low production volume.

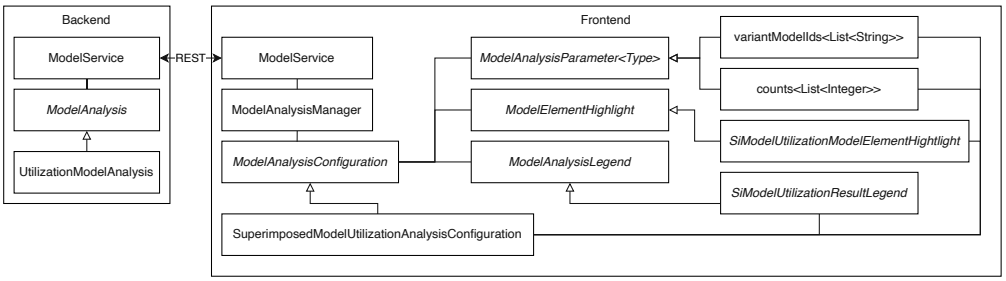


Figure 6.12: Component Utilization Analysis as Example of the Usage of the PPRVM-ANALYSIS Framework.

Figure 6.12 shows the introduced design of the interfaces provided by the framework.

The *Backend* module gets a concrete implementation of the *ModelAnalysis* interface—called *UtilizationModelAnalysis*. This analysis implementation expects a list of the variant models and their respective production volume, besides the superimposed model. With the production volumes provided, the Utilization Ratio (UR) for each process is calculated:

$$UR(p) = \frac{\{counts_m | m \in variantModels \wedge p \in m\}}{\sum counts} \tag{6.1}$$

The *Frontend* module gets a concrete model analysis configuration—called *SuperimposedModelUtilizationAnalysisConfiguration*. This analysis configuration comprises two parameters, **(1)** a list of the variant models, and **(2)** a list with the production volume of each variant model. Further, a custom analysis legend (*SiModelUtilizationResultLegend*) should be shown, and each process should get highlighted (*SiModelUtilizationModelElementHighlight*), with a color representing the utilization, in the editor.

Figure 6.13 shows the visualization concept of the analysis result for the MDRE. The toolbox and the sidebar remain the same, while the model canvas gets adapted to the model analysis result. The processes gets highlighted with a colored border with the color spectrum from blue to red, while blue means low-utilized and red means well-utilized.

$$color = hsl(20 + \lfloor (1.0 - UR) * 180 \rceil, 90\%, 70\%) \tag{6.2}$$

A popup with the calculated UR is shown on mouse-over. At the bottom, there is the legend for the analysis result describing the meaning of the color highlights.

Table 6.4 lists planned production volumes for three model variants of the water filter product line. The first variant represents a small water tank with bone filter and has a planned production volume of 1000 pieces. The second represents a small water tank with bone filter and a nanofilter and has a planned production volume of 300 pieces. The

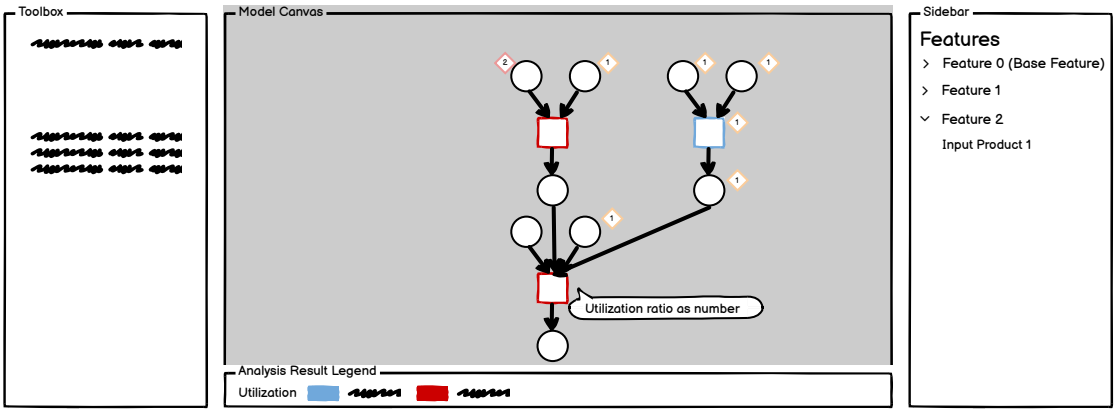


Figure 6.13: Utilization Analysis - Visualization Concept

Variant Model	**Planned Production Volume**
V1: Tank S with Bone Filter	1000
V2: Tank S with Bone and Nanofilter	300
V3: Tank XL with Bone	750

Table 6.4: Utilization Analysis - Model variants with planned production volumes

Process \ **Variant Model**	V1	V2	V3	**UR**	**hue**
1: Mount Freshwater Tank	×	×	×	1.00	20.0
6: Mount Water Tank	×	×	×	1.00	20.0
14: Fill Filter Tank	×	×	×	1.00	20.0
17: Mount Filter Tank	×	×	×	1.00	20.0
22: Mount Tubes	×	×	×	1.00	20.0
25: Mount Hull			×	0.37	133.4
31: Mount Wastewater Tank		×		0.15	173.0

Table 6.5: Utilization Analysis - *Process* nodes of Variant Models and their calculated UR

last variant represents a XL water tank with bone filter and has a planned production volume of 750 pieces.

The UR and hue of each process is calculated using Equations 6.1 and 6.2. Table 6.5 summarizes the calculated UR and hue of each process.

CHAPTER 7

Evaluation

This chapter describes the evaluation of the proposed solution approach, described in Chapter 6, using the illustrative use cases described in Chapter 4.

The remainder of this chapter is structured as follows. At first, the evaluation procedure and the evaluation environment are described in Section 7.1 and Section 7.2. Then, each subsequent section describes the evaluation of a use case.

7.1 Evaluation Procedure

The solution approach, described in Chapter 6, is evaluated in a qualitative feasibility study using the three product lines and illustrative use cases presented in Chapter 4.

For each use case, one or more test cases are derived to compare the provided capabilities to the requirements elicited for each use case. Each test case is executed and documented using screenshots and explanations. Finally, where applicable the described capability has to conform to the MVA metamodel, therefore a discussion about the conformity is provided for each capability.

The evaluation results are accepted if the use cases can be conducted using the proposed approaches.

7.2 Evaluation Environment

To conduct the evaluation, the MVA approach is integrated into the *MDRE* (c.f. Chapter 2), building the *Variability Modeling Editor (VME)*. The VME serves as prototype for the evaluation of the MVA metamodel and the approaches, described in Chapter 7. The following paragraphs describe further extensions made to the MDRE that were not already covered in Chapter 6.

MVA Model Configuration The modeling of PPR models in the VME requires a configuration file that contains the definition of the PPR notation. Table 7.1 shows the mapping of concepts provided by the FPD (VDI 3682) VDI [2005] to elements of the MVA metamodel and to the VME.

VDI 3682 Concept	MVA Concept	VME Element Type
Product	Node	Product
Energy	Node	Energy
Process	Node	Process
Resource	Node	Resource
System Limit	Node	SystemBoundary
Flow	Edge	Unidirectional Edge
Usage	Edge	Bidirectional Edge

Table 7.1: Mapping from VDI 3682 [VDI, 2005] Concepts to MVA Concepts and to VME Element Types

Model Links and Data Propagation Engelbrecht [2021] introduced links between models to enable data propagation when one of the connected models changes. Engineers can establish *Model Links* to control data propagation among multiple models. The *Model Links* are *unidirectional* in general, but can also be defined *bidirectional*. However, if defined *bidirectional*, cycles must be avoided, i.e., update each model only once.

In the MDRE, a model has a list of other model IDs representing the model links. For the *Data Propagation* to work properly, model elements need to have the same *ID* in all linked models. Unfortunately, the *ID* of a model element is randomly generated and cannot be changed by Engineers. However, this limitation can be mitigated by using the import function, that allows engineers to import model elements from other models within the same project. Using this workaround preserves the *ID*.

The VME uses the model links to trace the origin of model elements in the superimposed model. With the model links, the VME shows the dynamic layers for each variant model used to create the superimposed model, as required in Section 6.2.1. On top, the model links are used to provide the input fields to define the planned production volume for variant models in the *component utilization analysis*, described in Section 6.3.1

The following sections describe the test cases and their conduction to evaluate the proposed metamodel and approaches.

7.3 Capability 1: Model Difference Analysis

This section comprises the evaluation of the use case *UC-1: Model Difference Analysis* (see Section 4.2). The use case describes a typical workflow of engineers to find differences in PPR models.

Requirements / Test Case	UC-1.R1	UC-1.R2	UC-1.R3	UC-1.R4	UC-1.R5	UC-1.R6
TC-1.1	×	×	×		×	
TC-1.2				×		
TC-1.3			×		×	×
TC-1.4	×	×	×		×	

Table 7.2: Test cases used to evaluate UC-1: Model Difference Analysis (MDA)

The evaluation is conducted, as described in the previous Section 7.1, by conducting the use case flows step-by-step in the technical prototype developed during this thesis.

7.3.1 Test Cases

Table 7.2 lists the three test cases to evaluate the use case *UC-1: Model Difference Analysis (MDA)* with the respective use case requirements.

TC-1.1: Analyze differences of two related PPR models, created using a *clone-and-own* approach, using the integrated view with all highlights enabled. This test case focuses on the usual flow of analyzing differences of two related PPR models. In this test case, all change types are highlighted.

TC-1.2: Fast-switch source/target model. This test case focus on the ability to quickly switch models under comparison.

TC-1.3: Switch from integrated to side-by-side view, navigate through the model and toggle the visibility of change types. This test case focus on the interaction with models under comparison. At first, the view is switched from *integrated* to *side-by-side*. Then, the viewport is moved and zoomed out to see a different part of the models. Lastly, the highlighting of *deletion* is turned off.

TC-1.4: Analyze differences of two related UML state chart models. This test case represents the basic flow of the use case *UC-1* but uses UML state chart (see Section 4.1.3 for model description) models instead of PPR models. The goal of this test case is to show that the PPRVM-MDA approach is also applicable to other modeling metamodels than PPR.

This test case uses the variant models *Controller A* and *Controller B* of the *Washing Machine Controller* product line.

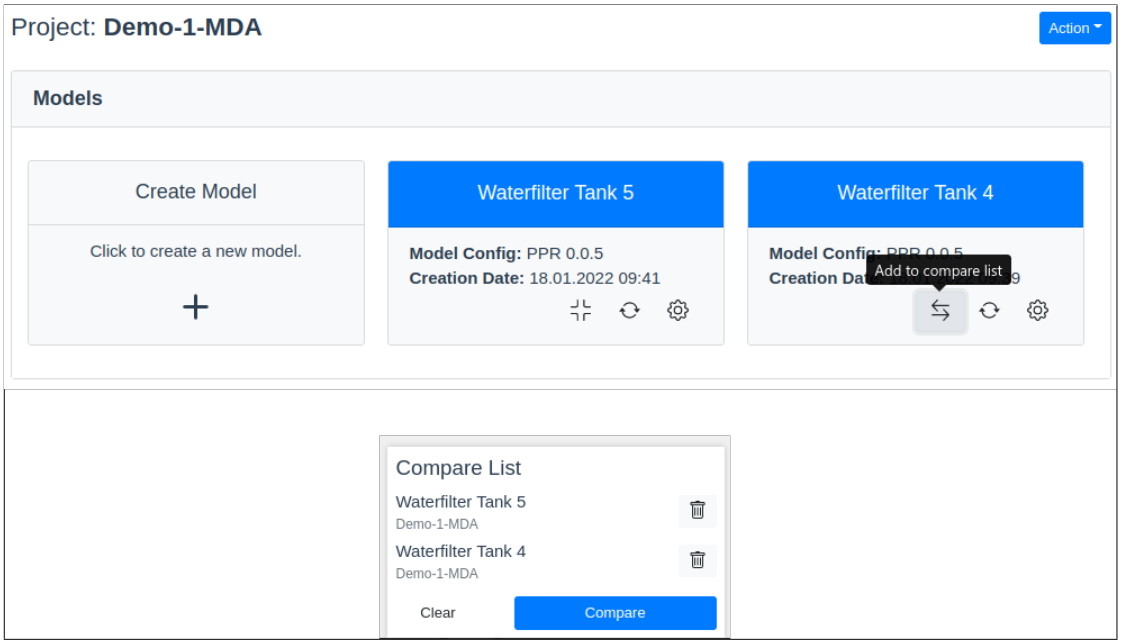


Figure 7.1: TC-1.1 - Screenshot of the variant model selection and the compare list in the VME

7.3.2 Test Case Execution

TC-1.1: Analyze differences of two related PPR models, created using a *clone-and-own* approach, using the integrated view with all highlights enabled. This test case is conducted using variant models of the *Water Filter* product line (introduced in Chapter 4) and relates to the basic flow of the RUCM steps of UC-1 outlined in Table 4.1.

Figure 7.1 shows the selection of models to be compared. Models of the same metamodel can be added to/removed from a *compare list* and finally be compared.

Figure 7.2 shows the resulting *integrated* comparison view. The view is divided into two areas.

The area at the top (marked (1) in the figure) contains options for the comparison view, i.e., which models to compare, which type of view to use, and which change types should be highlighted.

The bottom area (marked (2) in the figure) contains the integrated model with the visual feedback of differences.

In this specific case, the model *Waterfilter Tank 5* contains a *SystemBoundary* with process *23: Mount* with two input products *24: Wastewater Tank XL* and *25: Valve 2*, and an output product *26: Completed Wastewater Tank*. Due to the fact, that the model

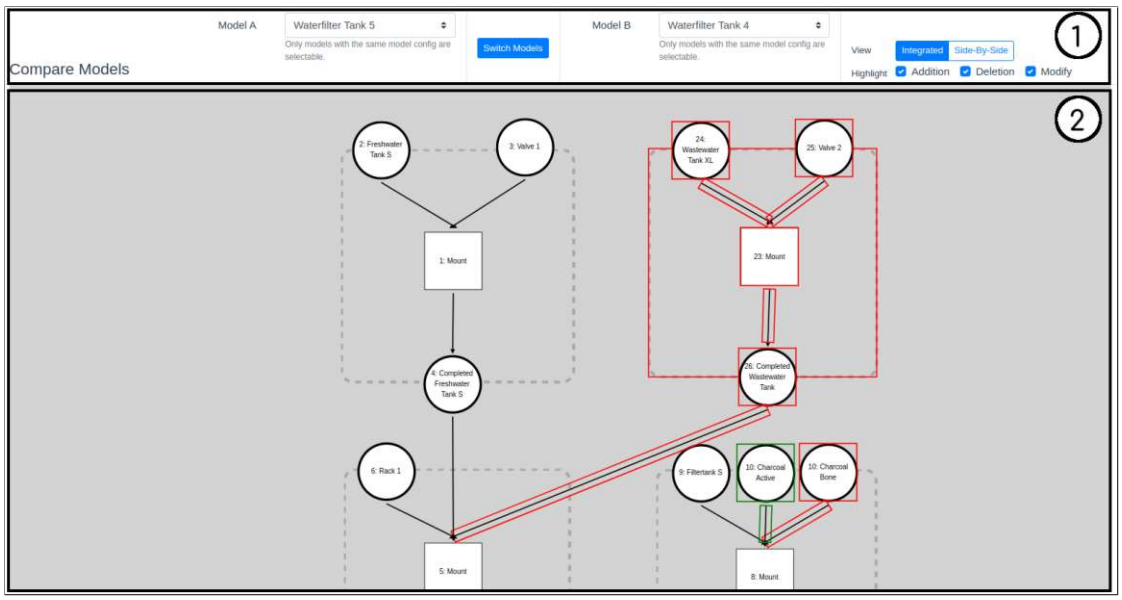


Figure 7.2: TC-1.1 - Screenshot of the integrated comparison view in the VME

Waterfilter Tank 4 does not contain that elements, all mentioned elements and their connecting edges are marked as *deleted* (red border).

Added elements are highlighted with a green border. In this specific case, the input product *10: Charcoal Active* is contained in model *Waterfilter Tank 4* but not in model *Waterfilter Tank 5* thus highlighted as *added* (green border).

Section 7.3.3 discusses the compliance of the resulting model of this approach to the MVA metamodel.

TC-1.2: Fast-switch source/target model. This test case is conducted using variant models of the *Water Filter* product line (introduced in Chapter 4) and relates to the *Alternative 3* flow of the RUCM steps of UC-1 outlined in Table 4.1.

Figure 7.3 shows the source model and target selection box with the possibility to choose another model in the same project as target model. Only models within the same project, and with the same metamodel, can be selected.

In this case, the target model is switched to *Waterfilter Tank 6*. The system recalculates the differences between the models *Waterfilter Tank 4* and *Waterfilter Tank 6* and shows the differences (as shown in Figure 7.4).

It is shown that the target or source model can be changed with one click, providing engineers a fast method to compare multiple models easily.

TC-1.3: Switch from integrated to side-by-side view, navigate through the model and toggle the visibility of change types. This test case is conducted using

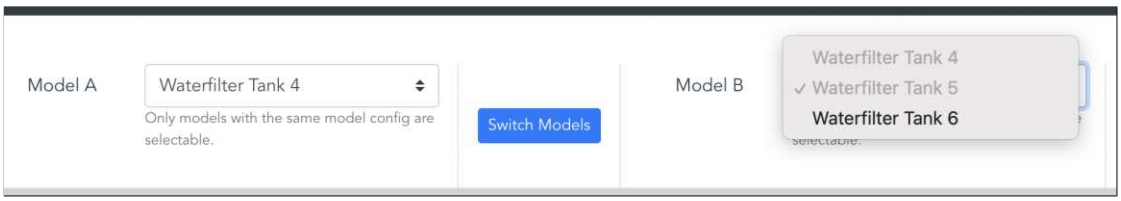


Figure 7.3: TC-1.2 - Screenshot of the target model selection box in the VME

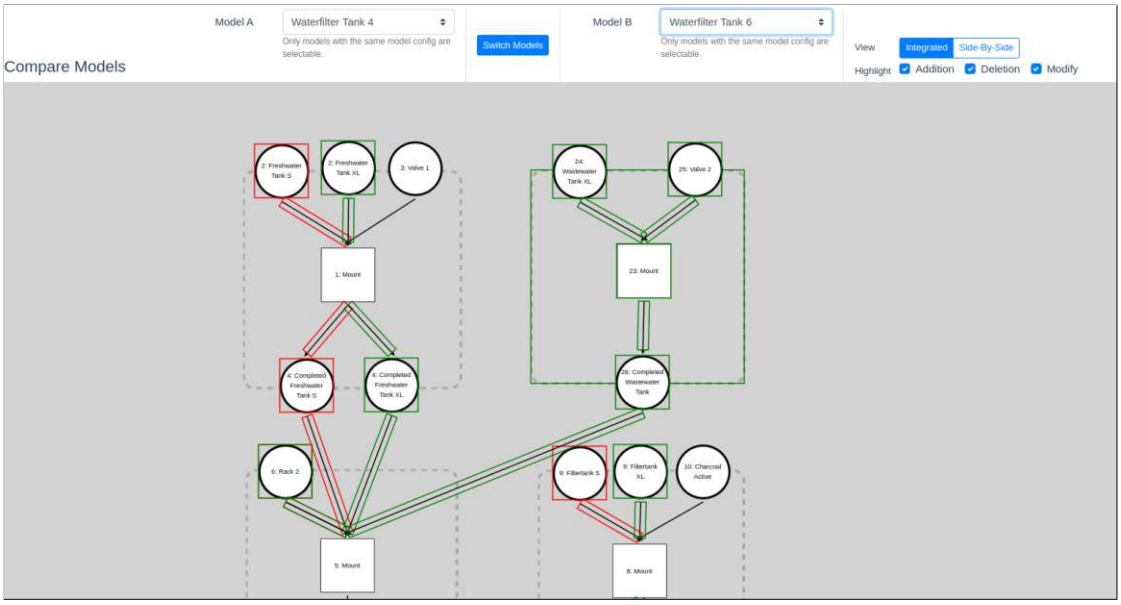
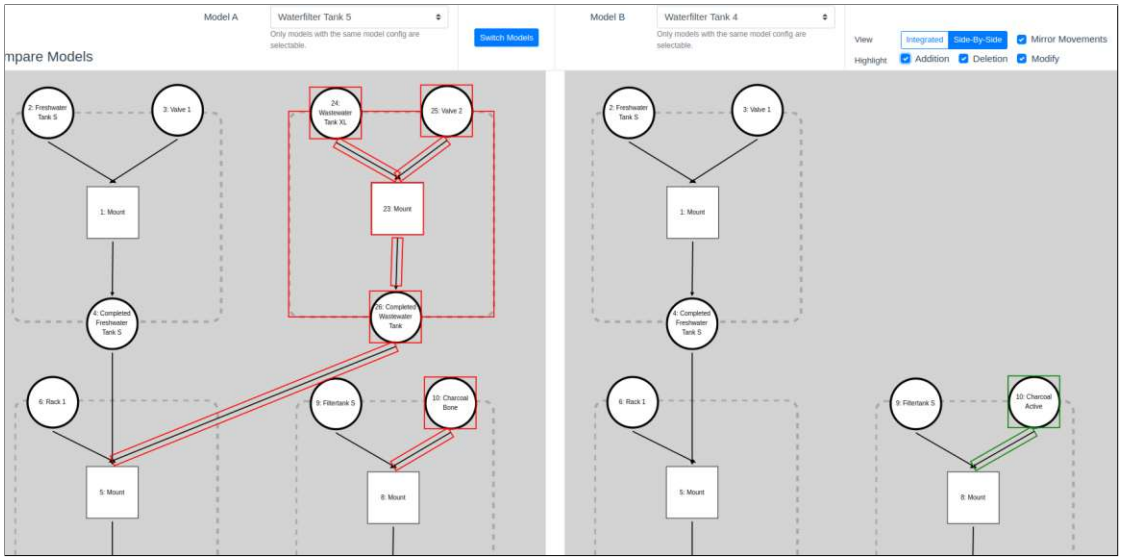


Figure 7.4: TC-1.2 - Screenshot of the integrated comparison view with the changed target model in the VME

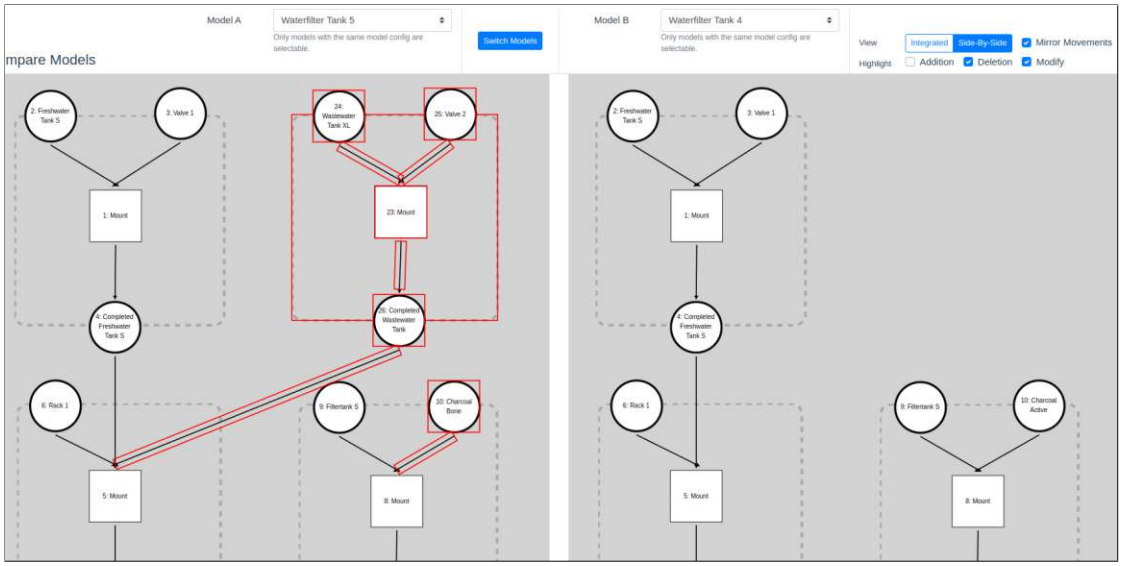
Figure 7.5: TC-1.3 - Screenshot of the view options in the VME

variant models of the *Water Filter* product line (introduced in Chapter 4) and relates to the alternative *Alternative 1* and *Alternative 2* flows of the RUCM steps of UC-1 outlined in Table 4.1.

In this test case, the engineer chooses to switch the differences view to have a side-by-side view of both models instead of the integrated view. With the view options, engineers can select which view to use (see Figure 7.5). Engineers can choose between an *integrated* and a *side-by-side* view. The side-by-side view reveals another option to mirror the movements of the viewport, i.e., if the viewport in the left model is changed, the viewport of the right model automatically mirrors the change. This allows viewing the same



(a) Side-by-side view with additions highlighted



(b) Side-by-side view without additions highlighted

Figure 7.6: TC-1.3 - Screenshot of the side-by-side comparison view with and without highlighting additions in the VME

regions of both models.

Additionally, engineers can choose which change types they want to highlight: *Additions*, *Deletions* and *Modifications*. In this case, the highlight of additions is disabled.

Figure 7.6 shows the side-by-side view of the two models compared without the highlighting of additions.

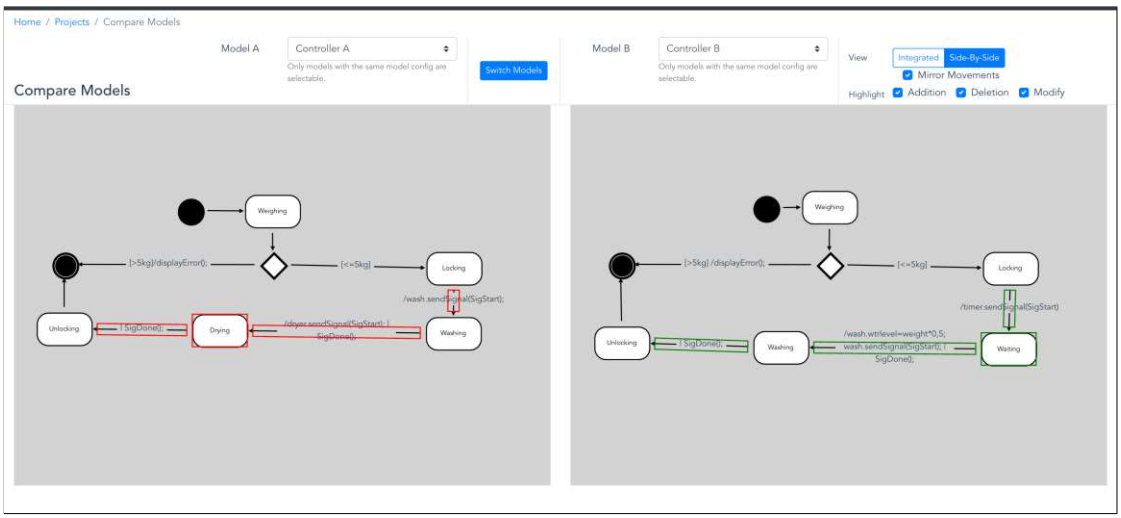


Figure 7.7: TC-1.4 - Screenshot of the differences of the UML state charts *Controller A* and *Controller B* in a side-by-side view

TC-1.4: Analyze differences of two related UML state chart models. The steps of this test case are similar to those of *TC-1.1* but with another type of VME models.

Figure 7.7 shows the identified differences in a side-by-side view. It can be seen that *Controller B* introduces a new state *Waiting* and does not have state *Drying*. Moreover, the guards of states *Washing* and *Unlocking* are different in *Controller B* than in *Controller A*.

7.3.3 Conformity to the MVA Metamodel

This section discusses the conformity of the result model of the PPRVM-MDA approach to the MVA metamodel.

Figure 5.2 shows to which parts of the MVA metamodel the PPRVM-MDA result model corresponds to. The result of the PPRVM-MDA approach is a *Model* comprising *Nodes* and *Edges* with *Change Markers*.

Figure 7.2 presents four production steps represented by *SystemBoundaries* containing one or more input *Products*, a *Process*, and one output *Products*. The input products are connected to the processes, and the processes are connected to the output product with an *Unidirectional Edge*. *SystemBoundary*, *Product* and *Process* corresponds to the *Node* element in the MVA metamodel. *Unidirectional Edge* corresponds to the *Edge* element in the MVA metamodel.

The MVA-metamodel defines, that a model can have zero or more nodes and edges. The resulting model contains *SystemBoundary*, *Product* and *Process* that corresponds to the *Node* element in the MVA metamodel.

Each node can be *part-of* another node, according to the MVA metamodel. Figure 7.2 also shows that each *Product* and *Process* is contained within a *SystemBoundary* which describes a parent-child relationship, e.g., the process *23: Mount is-part-of* a *SystemBoundary*.

Each node or edge can have zero or one *Change Marker*, according to the MVA metamodel. Figure 7.2 also shows that some products, processes, and system boundaries are highlighted with either a green or a red border. The green or red borders represent *Change Markers* of the types *addition* (green border) and *deletion* (red border). Each green or red border belongs to exactly one node or edge.

This shows that the result model of the PPRVM-MDA approach conforms to the MVA metamodel.

7.3.4 Evaluation Criteria

This section summarizes the characteristics of the PPRVM-MDA approach.

Reproducibility Compared to the manual comparison, the PPRVM-MDA approach yields the same results with the same input models every time it is executed. Therefore, the PPRVM-MDA approach yields reproducible results.

Soundness of the Comparison Test case *TC-1.1* shows that the PPRVM-MDA approach correctly identifies differences (*addition*, *deletion*) of two PPR models.

Flexibility Test case *TC-1.4* shows that the PPRVM-MDA approach is open to other types of models that comply to the VME metamodel.

7.4 Capability 2: Superimposed PPR Model Creation

This section comprises the evaluation of the use case *UC-2: Superimposed PPR Model Creation* (see Section 4.3). This use case describes the typical workflow of engineers to create a superimposed PPR model from a set of PPR variant models.

The evaluation is conducted, as described in the previous Section 7.1, by conducting the use case flows step-by-step in the technical prototype developed during this thesis.

7.4.1 Test Cases

Table 7.2 lists the seven test cases, with the respective use case requirements, to evaluate the use case *UC-1: Model Difference Analysis (MDA)*.

TC-2.1 - Create a Superimposed PPR Model with suggested Feature Assignments: This test case represents the basic flow of the use case UC-02 (see Table 4.2 for

Requirements / Test Case	UC-2.R1	UC-2.R2	UC-2.R3	UC-2.R4	UC-2.R5	UC-2.R6	UC-2.R7	UC-2.R8	UC-2.R9
TC-2.1	×	×	×	×	×				
TC-2.2		×	×	×			×		
TC-2.3		×	×						
TC-2.4		×	×						
TC-2.5								×	
TC-2.6						×			×

Table 7.3: Test cases used to evaluate UC-2: Superimposed PPR Model Creation

use case description in RUCM notation). All variants of the *Water Filter* product line are used.

TC-2.2 - Create a Superimposed PPR Model with customized Feature Assignments: This test case represents the alternative flow 1 of the use case *UC-2 - Superimposed PPR Model Creation* with a focus on the preservation of customized features during the creation process. Variants 4 and 5 of the *Water Filter* product line are used.

TC-2.3 - Compare Superimposed PPR Models created with VME and But4Reuse: This test case is executed twice, first in the VME, and second in another tool called *But4Reuse* [1] and visualized with *graphviz* [2]. Afterwards, the resulting superimposed PPR models from both tools are compared. The goal of this test case is to show the *soundness of the comparison*.

TC-2.4 - Repeatedly create a Superimposed PPR Model: This test case represents the basic flow of the use case UC-02 and is executed multiple times with the same set of variant models. The goal of this test case is to show *Reproducibility*, i.e., the same set of PPR variant models as input yields the same superimposed PPR model.

TC-2.5 - Create a Superimposed UML State-Chart Model: This test case represents the basic flow of the use case UC-02, but uses UML State-Chart models instead of PPR models. We recreated the UML state-chart models, originally described by Rubin and Chechik [2012], with the VME. The goal of this test case is to show *soundness of the comparison* and *Flexbility*. It shows that the proposed approaches are also applicable to other engineering DSLs.

[1] `https://but4reuse.github.io/`, last accessed 2022-10-18
[2] `https://graphviz.org/`, last accessed 2022-10-18

TC-2.6 - Toggle layers to adapt visibility of relevant parts of the superimposed PPR model This test case extends the basic flow of the use case *UC-02* by another step. After the superimposed PPR model is presented, the engineer can toggle layers to declutter the modeling view.

7.4.2 Test Case Execution

The execution of the test cases TC-2.1 - TC-2.6 is described in the following paragraphs.

TC-2.1 - Create a Superimposed PPR Model with suggested Feature Assignments

Engineers create superimposed PPR models in a three-step process.

1. Choose variant models At first, engineers select which variant models they want to use. Our prototype allows selecting models in the same project which has the same model configuration assigned. We assume that models with different model configurations do not have enough elements in common to be able to calculate a superimposed model.

Figure 7.8 shows a screenshot of the variant selection of the VME. The application shows available models as a list with details like name, and assigned model configuration (name and version) and allows showing a preview of the model. This allows engineers to have a look at the particular model if the name is not representative or the engineer is not familiar with the model.

2. Feature Identification and Preview of the resulting Superimposed PPR Model The application then calculates commonalities and variability and creates a preview of the resulting superimposed model. Basically, this algorithm compares all models to identify model elements (nodes and edges) that are present in the whole set or only a subset of the selected variant models. The processing time is optimized by processing the most frequent model elements first. Section 6.2.1 gives a detailed description on the used algorithm.

The VME visualizes the resulting superimposed model to give engineers the ability to modify the calculated features (cf. Figure 7.9). This may be useful, for example, if engineers have experience with some kind of assembly sequence and know that a certain way of feature assignment is more appropriate. Engineers can change the feature number with a click on an element of the model, which opens a dialog to change the feature number. The application adapts the feature list and the feature annotation automatically. Model elements that are assigned to features get annotated in the model for engineers to see the feature assignment at first sight. Currently, only nodes get annotated to increase readability. Additionally, the application provides a list of calculated features, including the base feature.

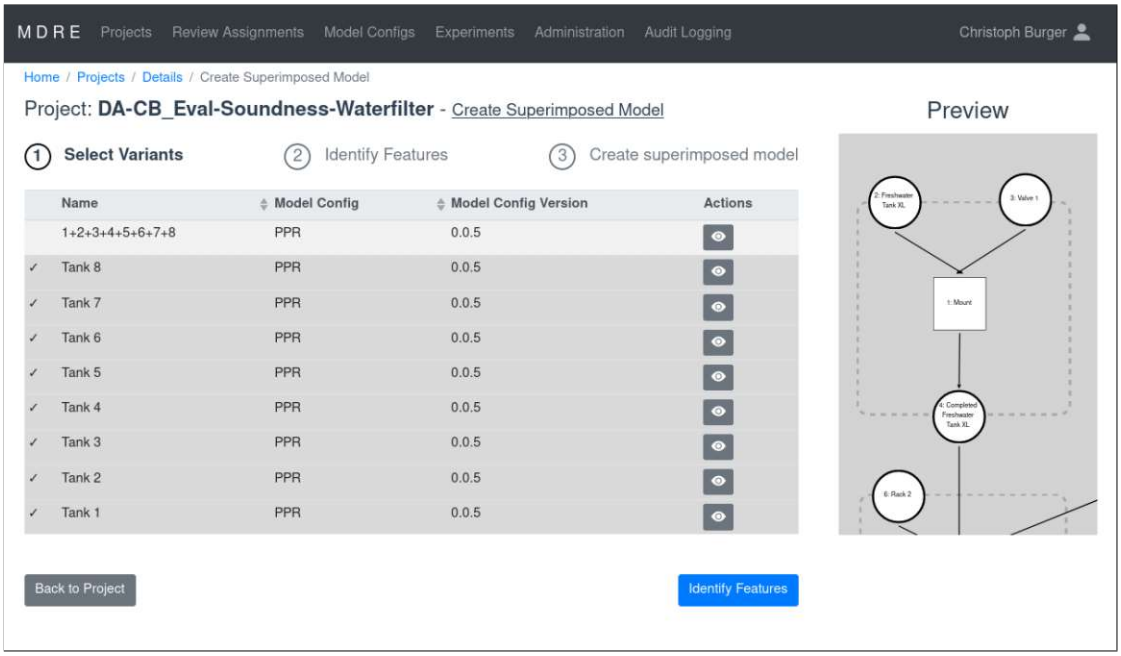


Figure 7.8: TC-2.1 - Screenshot of the Variant Model Selection in the VME

3. Save Superimposed PPR Model Finally, engineers can name and save the superimposed model. The application automatically links all variant models to the superimposed model and vice versa. This gives engineers the ability to propagate changes on elements in the variant or superimposed model to every model used to create the superimposed model that contains these elements.

Figure 7.10 shows a screenshot of the resulting superimposed model inside the modeling component of our prototype. The modeling component of our application handles superimposed models like any other model, with a few exceptions:

1. Model elements with a feature number assigned are annotated with the feature number.
2. The right sidebar contains a new tab *Features* which contains the list of features and the ability to add variant models to the superimposed model (see Section 6.2.2) and to extract variants models from the superimposed model (see Section 6.2.3). This gives engineers an overview of all features in the model.
3. Feature assignments can be modified with a click on a button inside the right pane that shows the properties of the selected model element. The dialog is the same as in the creation process, and so is the automatic update of the features if the feature number changes.

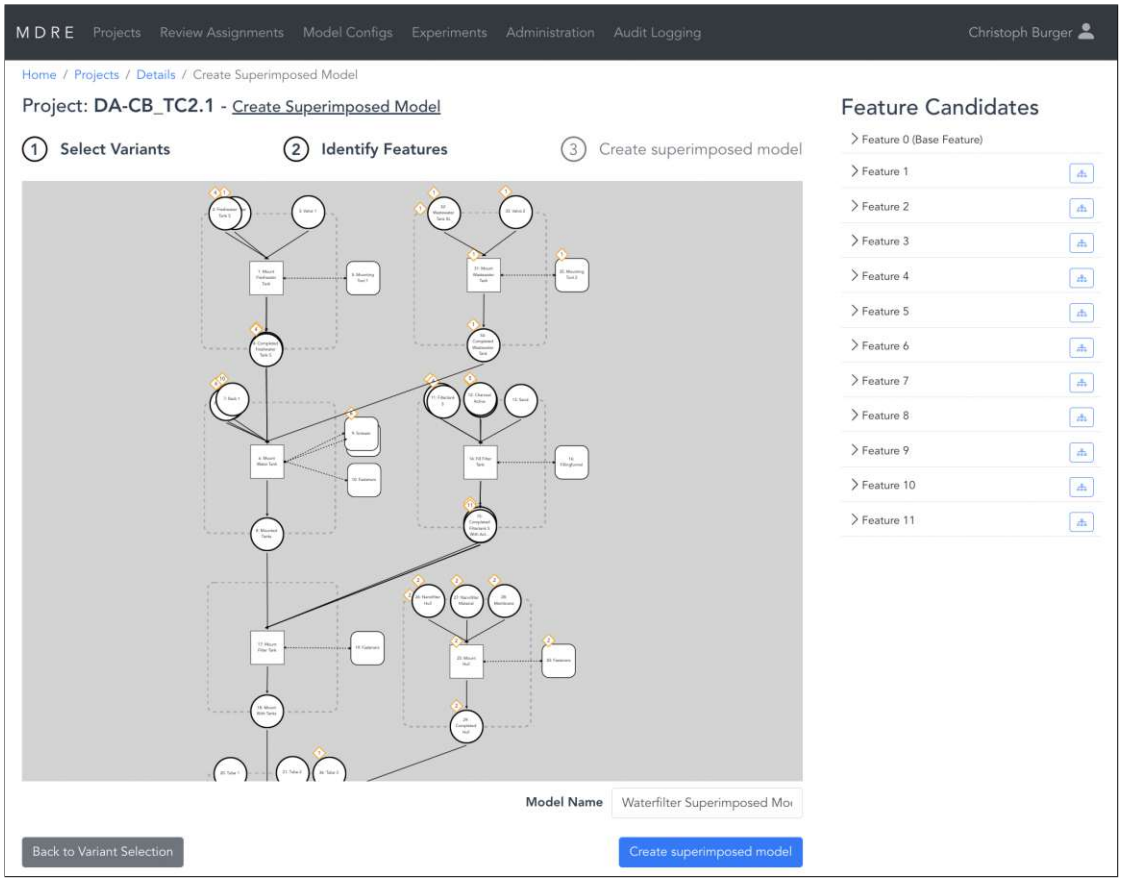


Figure 7.9: TC-2.1 - Screenshot of the calculated Features and parts of the resulting Superimposed PPR Model in the VME

TC-2.2 - Create a Superimposed PPR Model with customized Feature Assignments.

The process to create a Superimposed PPR Model with Customized Feature Assignments is mostly identical to TC-2.1 except for one step. At the second step *Feature Identification and Preview of the resulting Superimposed PPR Model*, the product *2: Freshwater Tank S* is moved from *Feature 2* into a new *Feature 20*. Therefore, engineers can click on model elements to open a modal window to change the feature the model element is assigned to. When saved, the new feature is created and the product is now assigned to the new feature. Figure 7.11 shows screenshots of the feature customization, during the creation of a superimposed PPR model, in the VME.

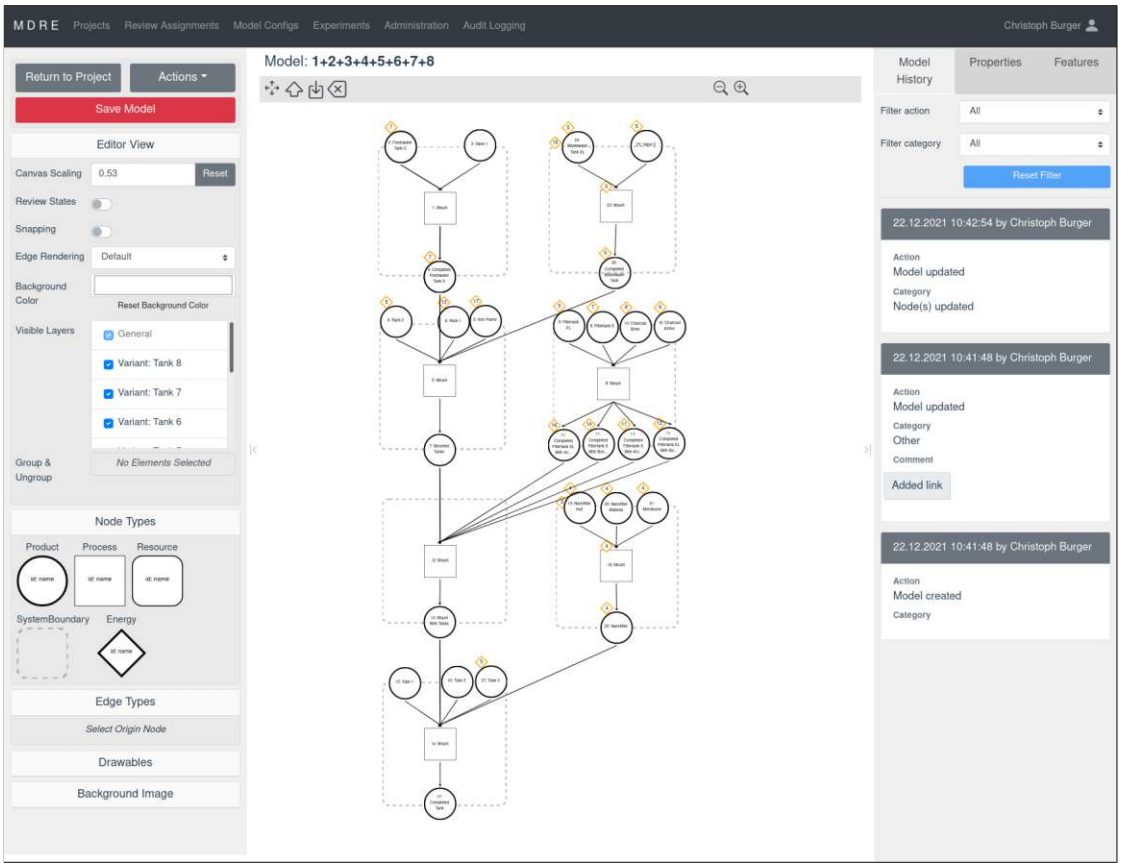


Figure 7.10: TC-2.1 - Resulting Water Filter Superimposed PPR Model in the VME Modeling Component

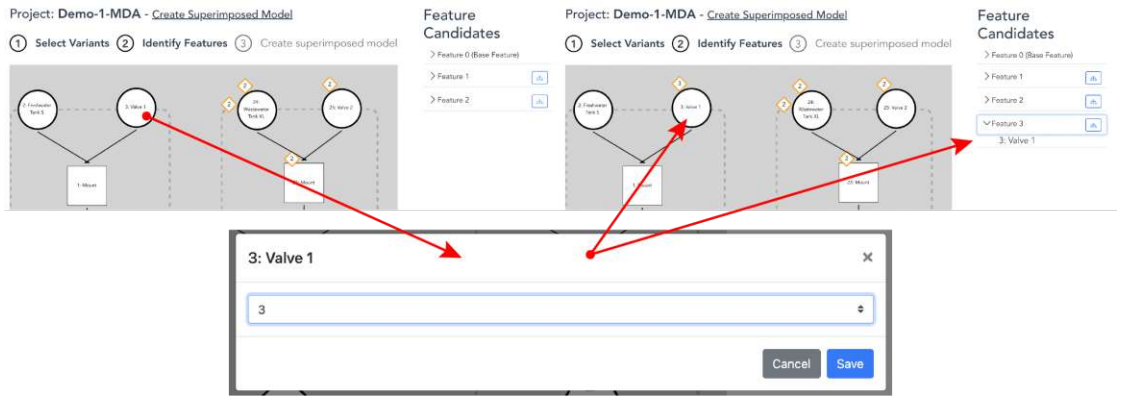


Figure 7.11: TC-2.2 - Feature Customization in the process of creating a superimposed model

TC-2.3 - Compare Superimposed PPR Models created with VME and But4Reuse

To show that our approach yields the correct results, the superimposed PPR model created with the VME is compared to the superimposed model that is generated by *But4Reuse*. Therefore, it is necessary to export the variant models of the *Water Filter* case study to *GraphML*[3] to use the graphs adapter provided by *But4Reuse*.

To export the models to *GraphML*, we extended the in-built export function of the VME to support *GraphML*. Figure 7.12 shows part of the resulting *GraphML* file.

Each VME model get mapped to a *GraphML graph* node, processes, resources, and products get mapped to *node* elements and edges get mapped to *edge* elements.

Custom attribute names, prefixed with *mdre:* are introduced to preserve element properties throughout the *But4Reuse* feature identification process.

Due to the limitation of *But4Reuse* to not fully support subgraphs, we flattened the model s.t. *SystemBoundaries* are not exported to the *GraphML* format.

After the export of all variant models, the superimposed model was created in the VME and in *But4Reuse* simultaneously. To visualize the superimposed model created by *But4Reuse*, we first converted the resulting *graphml* file to *graphviz* format and then used *graphviz* to generate a visualization of the superimposed model as a PDF file.

Both approaches yielded the same semantically superimposed model, except that the *But4Reuse* model does not contain *System Boundaries*. Further, the identified features are in a different order. Table 7.4 describes the mapping between the feature numbers of the VME and the block numbers of the *But4Reuse* tool.

The variant models, in both VME and *graphml* formats, and the resulting superimposed models are available in an online-resource[4].

TC-2.4 - Repeatedly create a Superimposed PPR Model.

This test case is conducted multiple times with all variants of the *Water Filter* case study. The approach yielded the same feature candidates in all runs.

TC-2.5 - Create a Superimposed UML State-Chart Model

To show the flexibility of the proposed approach, a superimposed model is created from three UML state chart variant models. The variant models represent washing machine controllers, originally published by the authors of [Rubin and Chechik, 2012], and modeled using the VME.

[3] `http://graphml.graphdrawing.org/`, last accessed 2022-06-23

[4] `https://bitbucket.org/tuw-qse/msc-thesis-cburger/src/master/test-cases/tc-2.3/`

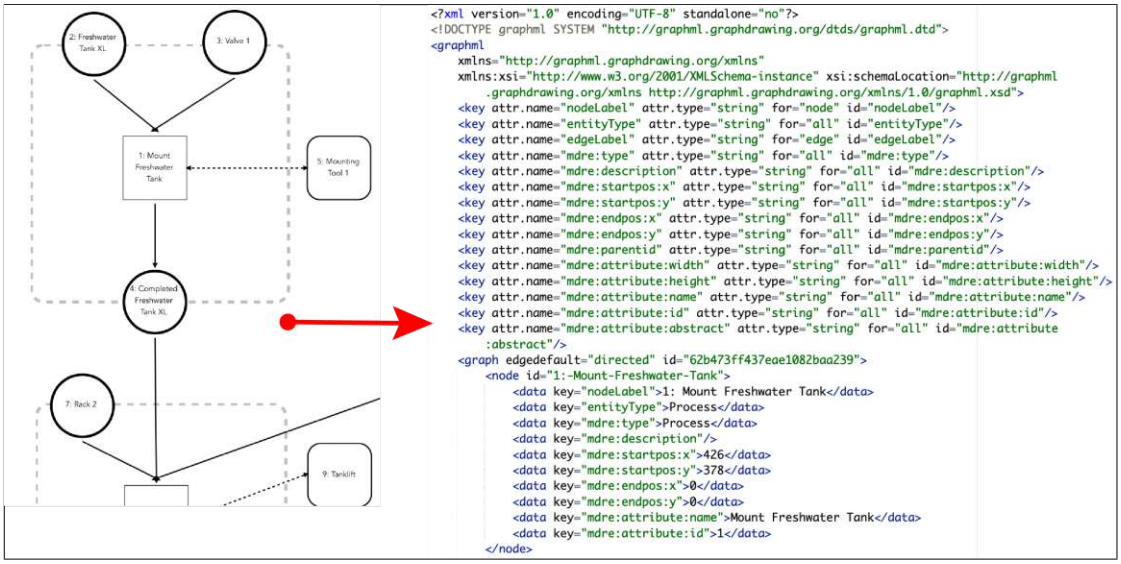


Figure 7.12: TC-2.3 - GraphML Export of VME models

VME Feature Number	But4Reuse Block Number
Feature 0	Block 00
Feature 1	Block 04
Feature 2	Block 05
Feature 3	Block 03
Feature 4	Block 01
Feature 5	Block 02
Feature 6	Block 08
Feature 7	Block 10
Feature 8	Block 09
Feature 9	Block 06
Feature 10	Block 11
Feature 11	Block 07

Table 7.4: Feature Mapping of the VME and But4Reuse

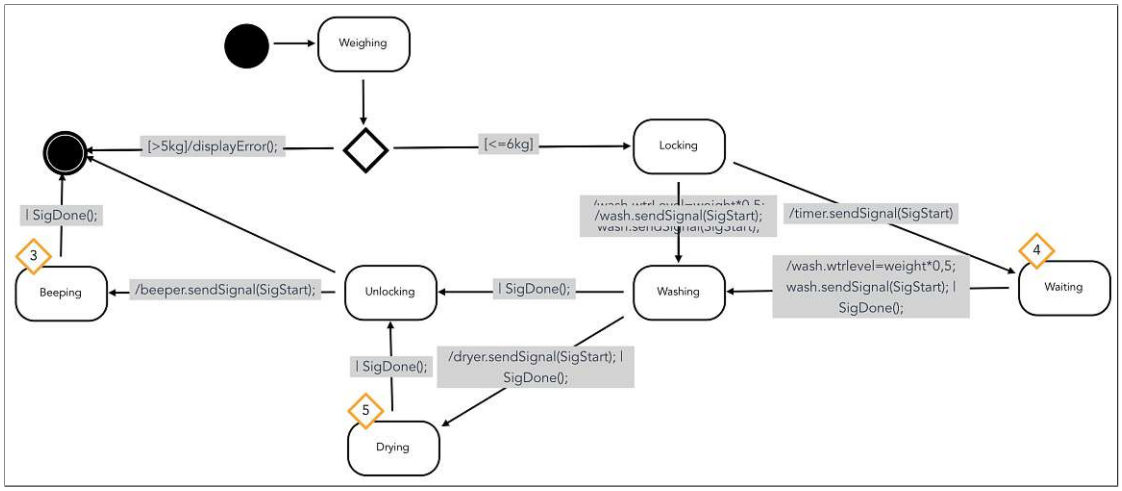


Figure 7.13: TC-2.5 - Superimposed model of Controller A, B, C, based on [Rubin and Chechik, 2012]

Figure 7.13 shows the resulting superimposed model. The proposed approach yields the same result compared to the resulting model in [Rubin and Chechik, 2012] with a slightly different visualization. The VME does not do well with multiple guards defined on the same edge. This visualization issue is not part of this thesis and may be resolved in future work.

This test case shows that the proposed approach not only works for PPR but also for UML state chart models.

TC-2.6 - Toggle layers to adapt visibility of relevant parts of the superimposed PPR model

This test case uses the superimposed PPR model created in test case *TC-1*.

Figure 7.14 shows some available layers, in the VME modeling component, in the middle. On the left side, there is an assembly step with input (*2: Freshwater Tank XL*) and output products (*4: Completed Freshwater Tank XL*) that belongs to *Feature 1*, as annotated. The right side, is the same assembly steps without the input and output products that were marked as part of *Feature 1* after hiding layer *Variant: Tank 5*.

7.4.3 Conformity to the MVA Metamodel

This section discusses the conformity of the result model of the PPRVM-FCI approach to the MVA metamodel.

Figure 7.15 shows to which parts of the MVA metamodel the PPRVM-FCI result model corresponds to. The result of the PPRVM-FCI approach is a *Model* comprising *Layers* with *Nodes* and *Edges* having *Properties*, *Feature Markers* and *Model Links*.

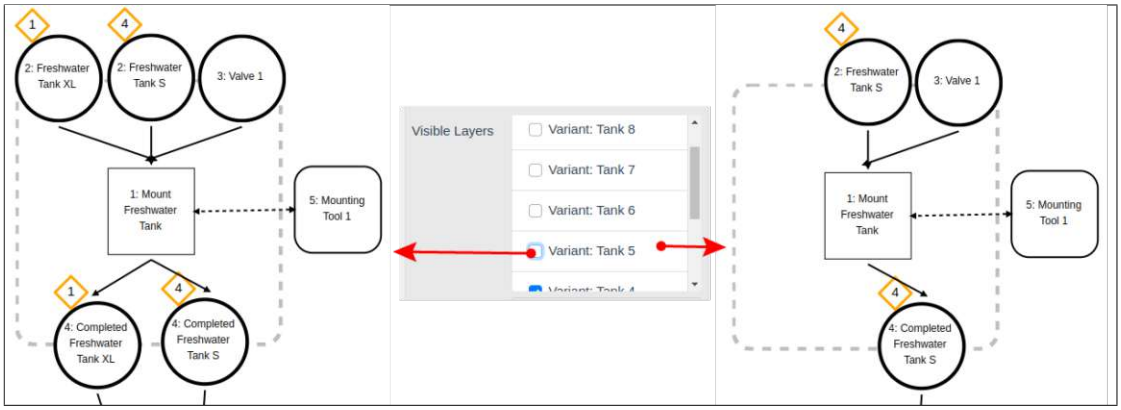


Figure 7.14: TC-2.6 - Hide Features of Superimposed PPR Model using the Layer Toggle of the VME Modeling Component

Figure 7.10 presents the resulting superimposed PPR model in the VME modeling component. The shown superimposed PPR model consists of *Nodes* and *Edges* that are annotated with *Feature Markers*.

The MVA metamodel defines that each node or edge can have at most one feature marker. Figure 7.10 also shows that each node and edge that corresponds to a feature has only one feature marker.

Further, the MVA metamodel defines that nodes and edges can be part of zero or more layers. Test case *TC-2.6* shows that the layers enables engineers to hide/show nodes and edges of certain variants. A node or edge is only hidden when all layers containing the respective node or edge are deselected.

To sum up, this shows that the resulting superimposed PPR model conforms to the proposed MVA metamodel.

This shows that the result model of the PPRVM-MDA approach conforms to the MVA metamodel.

7.4.4 Evaluation Criteria

This section summarizes the characteristics of the PPRVM-FCI approach.

Reproducibility In comparison to the manual comparison, the PPRVM-FCI approach yields the same feature candidates with the same input models every time it is executed. Therefore, the PPRVM-FCI approach yields reproducible results.

Soundness of the Comparison Test case *TC-2.3* shows that the PPRVM-FCI approach correctly identifies commonalities and variability.

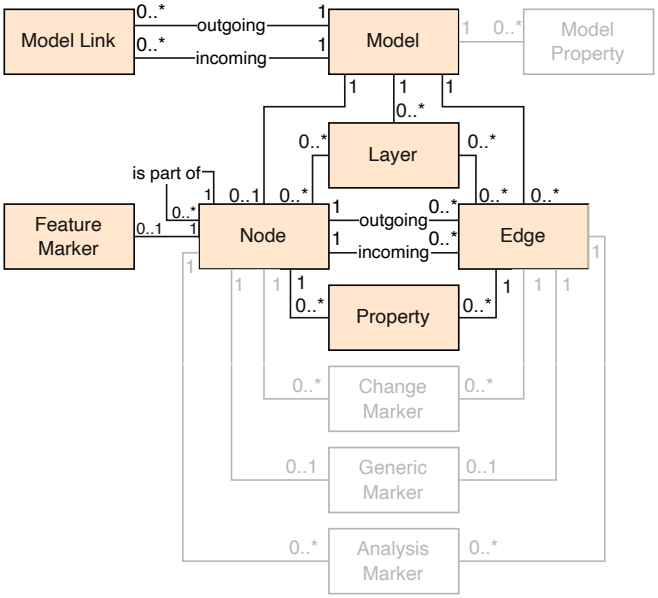


Figure 7.15: Relevant parts of the MVA Metamodel for the PPRVM-FCI Approach

Flexibility Test case *TC-2.5* shows that the PPRVM-FCI approach is applicable not only to PPR but also to UML state chart models.

7.5 Capability 3: Superimposed PPR Model Evolution - Add Variants

This section comprises the evaluation of the use case *UC-3: Superimposed PPR Model Evolution - Add a PPR Variant Model to an existing Superimposed PPR Model*(see Section 4.4). The use case describes a typical workflow of engineers to add variant models to existing superimposed PPR models.

The evaluation is conducted, as described in the previous Section 7.1, by conducting the use case flows step-by-step in the VME.

7.5.1 Test Cases

In the following, multiple test cases are described to evaluate the PPRVM-FCI-AV approach given the described use case *UC-3 - Superimposed PPR Model Evolution*. The test cases comprise the basic flow of the use cases and the three distinct cases and the edge case described in Section 6.2.2.

TC-3.1 - Add a variant PPR model to an existing superimposed PPR model. This test case represents the basic flow of the use case *UC-3 - Superimposed PPR Model*

Evolution (see Section 4.4). The goal of this test case is to show that the proposed PPRVM-FCI-AV algorithm is capable to integrate variant models into an existing superimposed model.

This test case is executed from the viewpoint of a basic planner whose task is to integrate a new model variant (*Tank 8*) into an existing superimposed PPR model already created using *Tank 1-7* of the *Water Filter* case study.

TC-3.2 - PPRVM-FCI-AV Case 1: CP is only present in new variant model(s). This test case represents the basic flow of the use case *UC-3 - Superimposed PPR Model Evolution* (see Section 4.4). The goal of this test case is to show *Case 1* of the proposed PPRVM-FCI-AV algorithm.

This test case is executed using the example model for *Case 1*, described in Section 6.2.2.

TC-3.3 - PPRVM-FCI-AV Case 2: CP is present in existing optional feature candidate(s) but not in new variant model(s). This test case represents the basic flow of the use case *UC-3 - Superimposed PPR Model Evolution* (see Section 4.4). The goal of this test case is to show *Case 2* of the proposed PPRVM-FCI-AV algorithm.

This test case is executed using the example model for *Case 2*, described in Section 6.2.2.

TC-3.4 - PPRVM-FCI-AV Case 3: CP is present in base feature candidate but not in new variant model(s). This test case represents the basic flow of the use case *UC-3 - Superimposed PPR Model Evolution* (see Section 4.4). The goal of this test case is to show *Case 3* of the proposed PPRVM-FCI-AV algorithm.

This test case is executed using the example model for *Case 3*, described in Section 6.2.2.

TC-3.5 - PPRVM-FCI-AV Edge Case: user-defined features cannot always be preserved. This test case represents the basic flow of the use case *UC-3 - Superimposed PPR Model Evolution* (see Section 4.4). The goal of this test case is to show the edge case of the PPRVM-FCI-AV algorithm, described in Section 6.2.2.

This test case is executed using the example model for the edge case, described in Section 6.2.2.

TC-3.6 - Adding variant models to an existing superimposed PPR model yields the same feature candidates as creating a superimposed PPR model from scratch The goal of this test case is to show that using a set of variant models to create a superimposed PPR model yields the same feature candidates, i.e., model elements are grouped together the same but maybe with different feature number, as if the superimposed PPR model is created with a subset of variant models and later extended with the variant models not used to create the superimposed PPR model. This test case represents the basic flow of use case *UC-3 - Superimposed PPR Model Evolution*

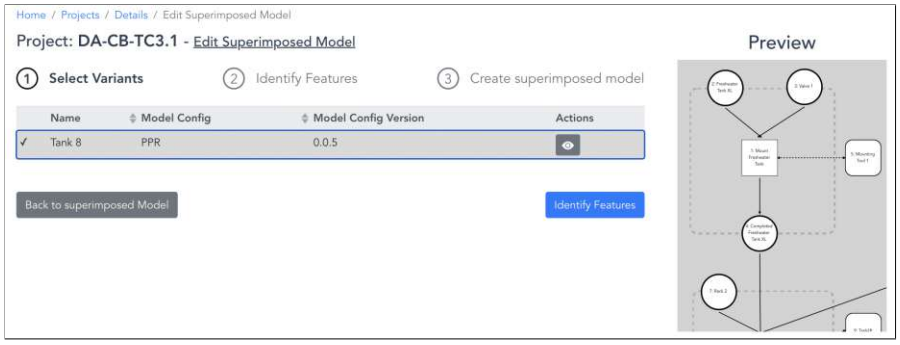


Figure 7.16: TC-3.1 - Screenshot of the Variant Model Selection in the VME. Only models not already part of the Superimposed PPR model are shown.

(c.f. Section 4.4), uses all variant models of the *Water Filter* case study and is executed multiple times. In total, it is executed 10 times with different order of variant models. The order is determined using a random number sequence generator[5] and the size of one batch is determined by an 8-sided dice[6].

7.5.2 Test Case Execution

The execution of the test cases TC-3.1 - TC-3.6 is described in the following paragraphs.

TC-3.1 - Add a variant PPR model to an existing superimposed PPR model. Table 4.3 describes the steps the basic planner has to conduct to integrate the new tank variant into the superimposed PPR model. We assume that the variant model of the new tank variant is already created and ready to be integrated.

The engineer opens the existing superimposed PPR model in the VME and starts the process to add a new variant. This process is similar to the creation of a superimposed PPR model (c.f. Section 7.4) but the wizard only shows available models that are not already part of the superimposed PPR model. Figure 7.16 shows the available models to be integrated into the superimposed PPR model.

After the selection, the PPRVM-FCI-AV algorithm (c.f Algorithm 6.3) calculates the new feature candidates and the VME presents the resulting superimposed PPR model to the engineer. Similarly to the process to create a superimposed PPR model, the feature candidates can be changed by the engineer before the model is saved. Figure 7.17 shows the preview of the resulting superimposed PPR model with the new variant model *Tank 8* integrated.

The next steps are equal to those of creating a superimposed PPR model (see Section 7.4).

[5] https://www.random.org/integer-sets/?sets=10&num=8&min=1&max=8&seqnos=on&commas=on&order=index&format=html&rnd=new, last accessed 2022-03-27

[6] https://rollthedice.online/en/dice/d8, last accessed 2022-03-27

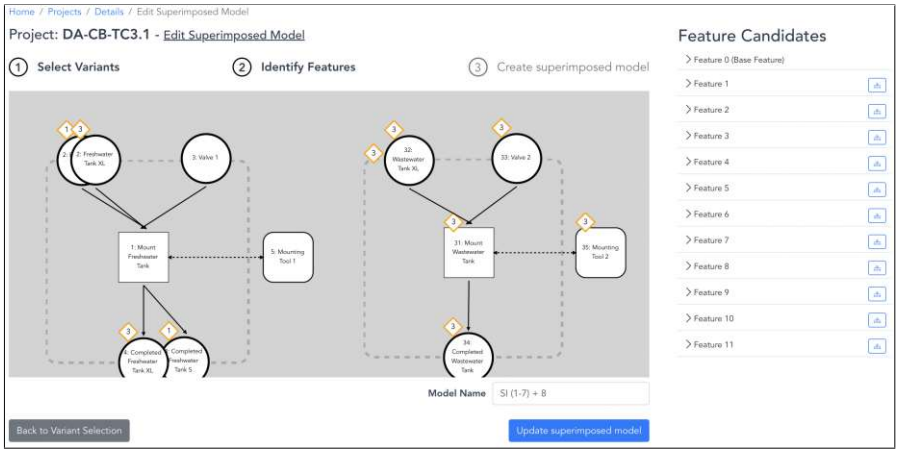


Figure 7.17: TC-3.1 - Screenshot of the calculated Features and the resulting Superimposed PPR model in the VME.

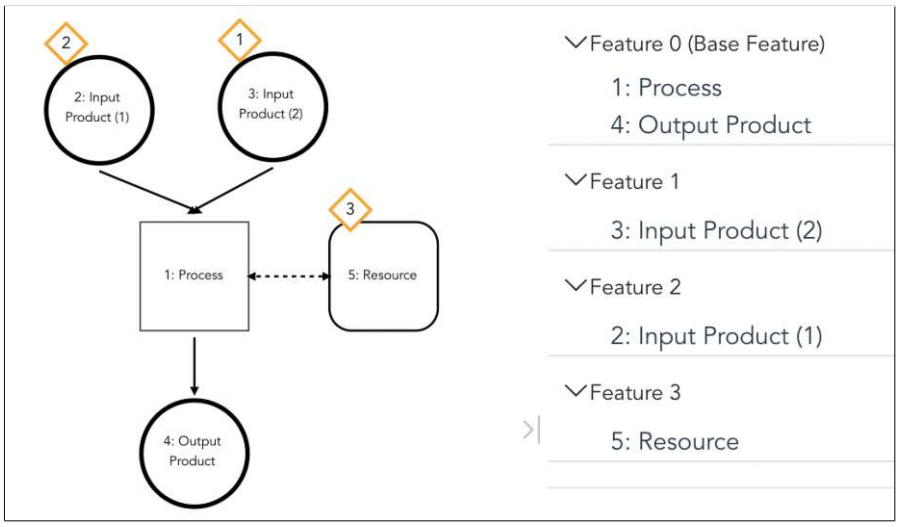


Figure 7.18: TC-3.2 - PPRVM-FCI-AV Case 1 - Screenshot of the resulting superimposed PPR model in the VME

TC-3.2 - PPRVM-FCI-AV Case 1: CP is only present in new variant model(s). The steps of this test case are equal to the test case *TC-3.1* but with tailored models to show *Case 1* of the PPRVM-FCI-AV approach.

Figure 7.18 shows the resulting superimposed PPR model with the calculated feature candidates. It can be seen that the calculated features are equal to those described in Section 6.2.2.

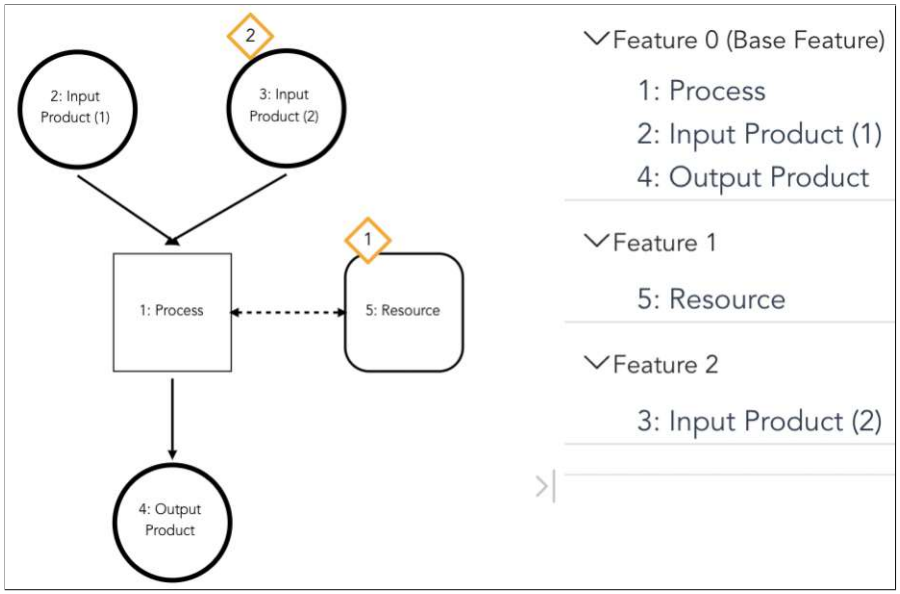


Figure 7.19: TC-3.3 - PPRVM-FCI-AV Case 2 - Screenshot of the resulting superimposed PPR model in the VME

TC-3.3 - PPRVM-FCI-AV Case 2: CP is present in existing optional feature candidate(s) but not in new variant model(s). The steps of this test case are equal to the test case *TC-3.1* but with tailored models to show *Case 2* of the PPRVM-FCI-AV approach.

Figure 7.19 shows the resulting superimposed PPR model with the calculated feature candidates. It can be seen that the calculated features are equal to those described in Section 6.2.2.

TC-3.4 - PPRVM-FCI-AV Case 3: CP is present in base feature candidate but not in new variant model(s). The steps of this test case are equal to the test case *TC-3.1* but with tailored models to show *Case 3* of the PPRVM-FCI-AV approach.

Figure 7.19 shows the resulting superimposed PPR model with the calculated feature candidates. It can be seen that the calculated features are equal to those described in Section 6.2.2.

TC-3.5 - Preserve Customized Features The steps of this test case are equal to the test case *TC-3.1* but with tailored models to show the *Edge Case* (c.f. Section 6.2.2) of the PPRVM-FCI-AV approach.

Figure 7.21 shows the resulting feature candidates. It can be seen that the feature candidates could not be fully preserved, i.e., *Product 3* introduced new feature *Feature 4*. Section 6.2.2 describes the basic principle of this edge case.

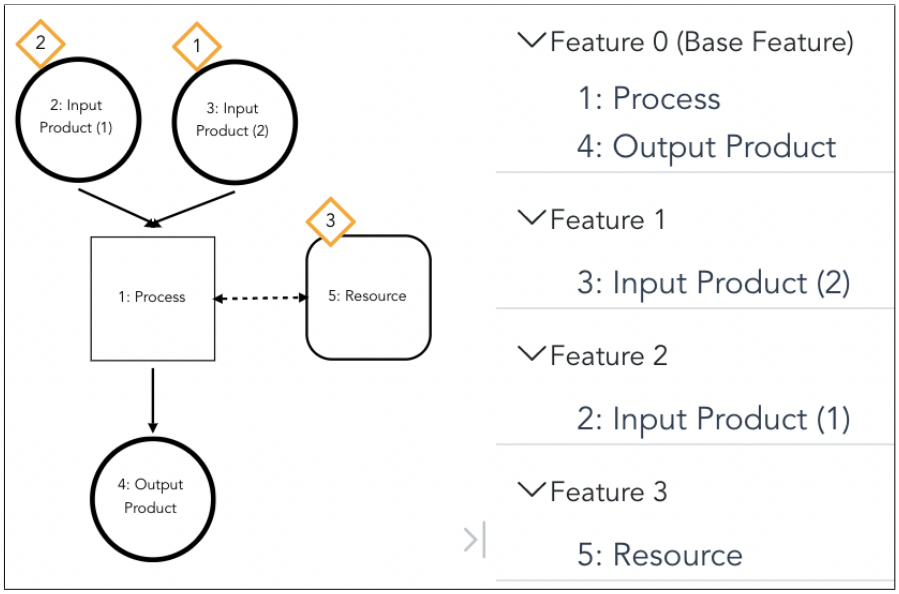


Figure 7.20: TC-3.4 - PPRVM-FCI-AV Case 3 - Screenshot of the resulting superimposed PPR model in the VME

TC-3.6 - Adding a variant model to an existing superimposed PPR model yields the same feature candidates as creating a superimposed PPR model from scratch A superimposed model created with all eight variants of the *Water Filter* case study is used as reference. The superimposed model has a base feature and eleven optional features *Feature 1-11*.

As indicated previously, this case is executed ten times with the models of the *Water Filter* case study in different order and in batches. The first batch describes the initial creation of the superimposed model. Each subsequent batch describes the task of adding a set of variant models to the existing superimposed model.

Table 7.5 shows ten sets, described by the batch size and the batches with the corresponding model variant number(s). For example, *Set 1* has a batch size of 6 s.t. the total number of variant models (8) is split into batches of 6 variant models. Therefore, set 1 comprises two batches, the first is the initial creation of the superimposed model with variant models *4,8,3,6* and *7* and afterwards the superimposed model gets extended using variant models *5,1* and *2*.

The test case revealed, that each execution yields the same number of feature candidates with the corresponding model elements but in a different order, e.g., *Feature 1* in the reference model corresponds to *Feature 8* in the resulting superimposed model of *Set 1*.

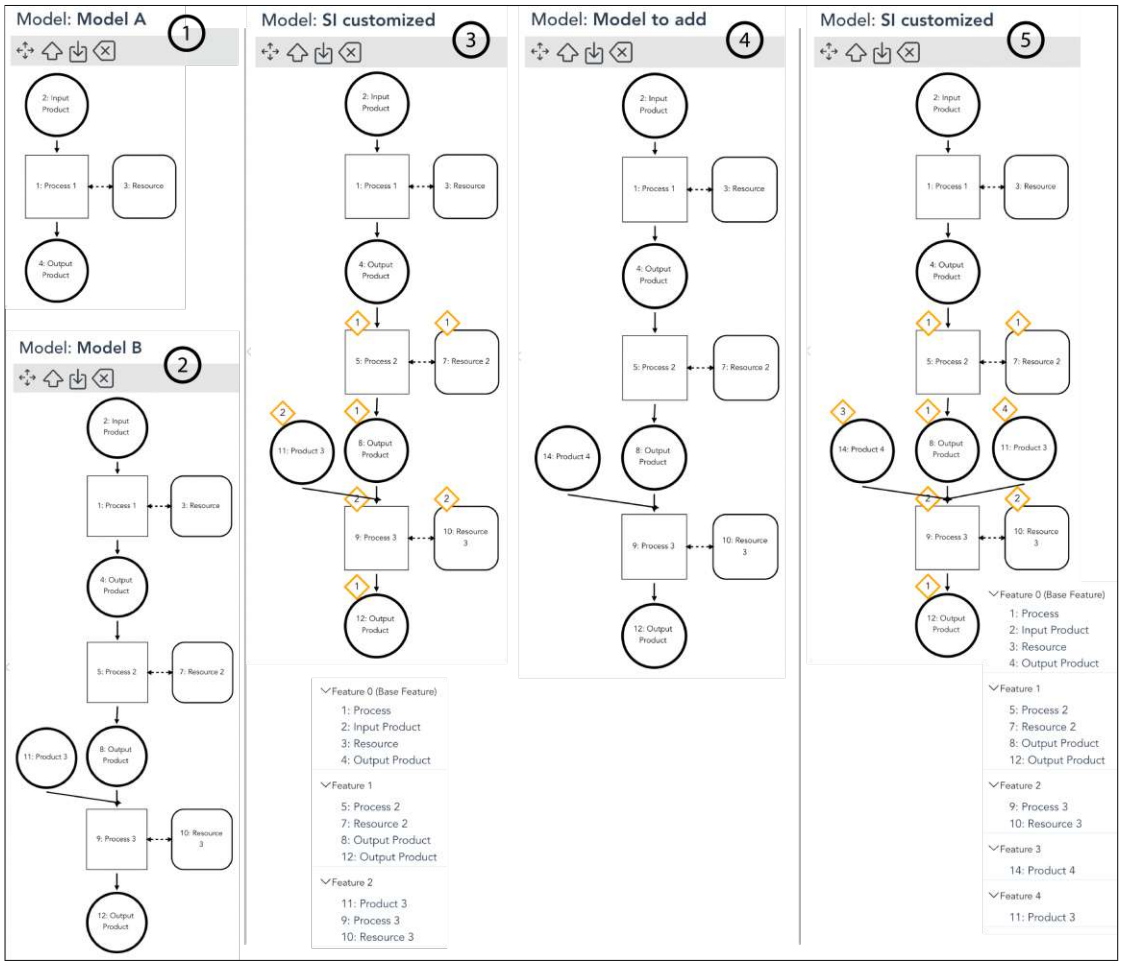


Figure 7.21: TC-3.4 - PPRVM-FCI-AV Edge Case. *(1-2)* are the base models used to create the initial superimposed PPR model *(3)* with customized feature candidates, s.t. *Product 3*, *Process 3* and *Resource 4* are contained in *Feature 2*. *(4)* is the model that is added to the superimposed PPR model (*3*). *(5)* is the resulting superimposed PPR model with *Product 3* introducing a new feature candidate.

	Batch Size	Batch 1	Batch 2	Batch 3	Batch 4
Set 1	6	4,8,3,6,7	5,1,2		
Set 2	4	5,4,1,7	2,8,6,3		
Set 3	8	5,1,7,6,4,3,2,8			
Set 4	5	8,2,5,7,3	6,4,1		
Set 5	7	1,3,7,2,4,6,8	5		
Set 6	3	7,8,5	6,2,3	4,1	
Set 7	6	3,7,1,2,6,8	5,4		
Set 8	2	5,6	1,3	7,8	4,2
Set 9	3	8,5,1	7,2,3	6,4	
Set 10	6	7,2,8,1,5,4	6,3		

Table 7.5: TC-3.6 - Ten different combinations of variant models of the *Water Filter* case study. Each set consists of the combination of variant models to initially create the superimposed PPR model and one or more batches to add variant models to the superimposed PPR model.

7.5.3 Conformity to the MVA Metamodel

Section 7.4.3 already describes the conformity to the MVA metamodel, since the PPRVM-FCI-AV approach only updates an existing superimposed model but does not introduce new concepts.

7.5.4 Evaluation Criteria

This section summarizes the characteristics of the PPRVM-FCI-AV approach.

Reproducibility, Soundness of the Comparison Test case *TC-3.6* shows that the PPRVM-FCI-AV approach repeatedly yields the same result, i.e., identifies same feature candidates but sometimes in different order, compared to the PPRVM-FCI.

7.6 Capability 4: Superimposed PPR Model Evolution - Derive Variants

This section comprises the evaluation of the use case *UC-4: Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model*(see Section 4.5). The use case describes a typical workflow of engineers to derive configurations (variant models) from superimposed PPR models.

The evaluation is conducted, as described in the previous Section 7.1, by conducting the use case flows step-by-step in the VME.

7.6.1 Test Cases

Table 7.6 lists the two test cases to evaluate the use case *UC-4* with the respective use case requirements.

Test Case \ Requirement	UC-4.R1	UC-4.R2	UC-4.R3	UC-4.R4
TC-4.1		×	×	×
TC-4.2	×			

Table 7.6: Test cases used to evaluate UC-4: Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model.

TC-4.1 - Derive suggested variant models from Superimposed PPR Model This test case evaluates the basic flow of the use case *UC-4* in the VME. It describes a typical task of *Production Process Optimizer* or *Quality Engineers* to derive a set of model variants from a superimposed PPR model. For *Quality Engineers*, it is required that every feature is contained at least once.

TC-4.2 - Derive suggested variant models from Superimposed PPR Model with defined Feature Constraints This test case focuses on the alternative flow of the use case *UC-4*. Engineers define feature constraints before deriving variants from the superimposed PPR model.

TC-4.3 - Derive suggested variant models from Superimposed PPR Model with conflicting Feature Constraints defined This test case focuses on the alternative flow of the use case *UC-4*. The goal of this test case is to show that not all features are contained at least once when conflicting feature constraints are defined.

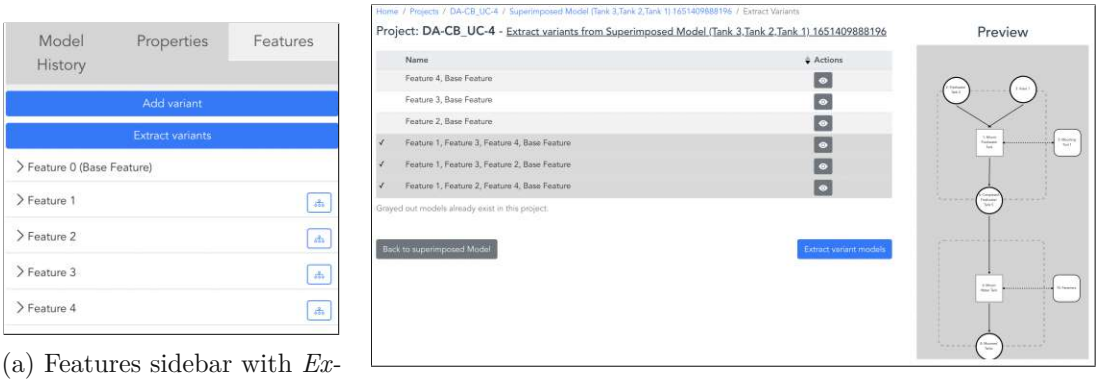
7.6.2 Test Case Execution

TC-4.1 - Derive suggested variant models from Superimposed PPR Model Engineers can start the derivation process by using the button provided within the *Features*-sidebar in the superimposed model details view (see Figure 7.22a).

Engineers then get a list of available configurations to choose from (see Figure 7.22b). They have the possibility to open a preview of the suggested models. Each line represents one configuration that can be selected.

The selected configurations get stored within the same project, after the selection. The models are named after the configuration.

TC-4.2 - Derive suggested variant models from Superimposed PPR Model with defined Feature Constraints For this test case, the steps of *Alternative 1* of use case *UC-4* (see RUCM notation of UC-4 in Table 4.4) are executed from the viewpoint of a *Quality Engineer*.



(a) Features sidebar with *Extract variants* button

(b) Configuration Selection with Preview

Figure 7.22: TC-4.1 - Screenshots of the derivation Process

Feature constraints can be edited within the superimposed model. In the *Features* sidebar on the right, there is a button for each feature (see Figure 7.23a) that opens a modal window after a click. Within this modal window, there are two drop-downs to define *requires* and *exclude* constraints for the selected feature. Figure 7.23b shows that *Feature 1* requires *Feature 2* but excludes *Feature 3*. These constraints have a direct impact on the calculation of valid configurations by the YASA (described in Section 6.2.3)

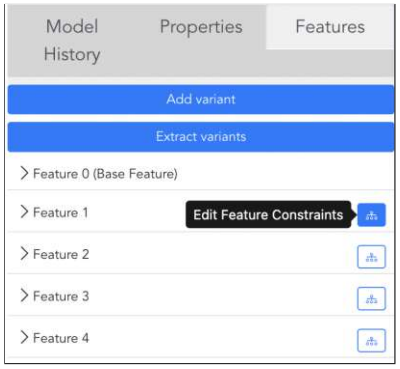
Figure 7.23c shows the available configurations calculated by the YASA algorithm. Compared to the available configurations of test case *TC-4.1* there is no configuration with both *Feature 1* and *Feature 3* (due to the defined *excludes*-constraint) It is also notable, that *Feature 1* is only present with *Feature 2* (also due to the defined *requires*-constraint).

Steps 4-5 of the use case *UC-4* remains the same as for the previous test case.

TC-4.3 - Derive suggested variant models from Superimposed PPR Model with conflicting Feature Constraints defined For this test case, the steps of *Alternative 1* of use case *UC-4* (see RUCM notation of UC-4 in Table 4.4) are conducted from the viewpoint of a *Quality Engineer*.

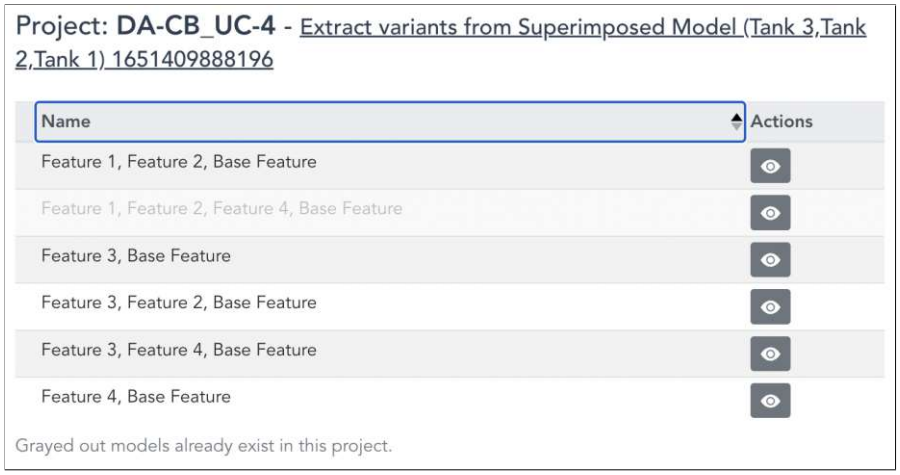
At first, feature constraints are defined. *Feature 1* should require *Feature 3* (see Figure 7.24a) and *Feature 3* should exclude *Feature 1* (see Figure 7.24b). Obviously, these constraints are conflicting and lead to *Feature 1* not present in any of the suggested configurations (see Figure 7.24c).

In the current state of the VME there is no warning about conflicting feature constraints. Neither in the editor at the time of definition, nor in the derivation process. Such a feature was out of scope of this thesis and may be the subject of future work.



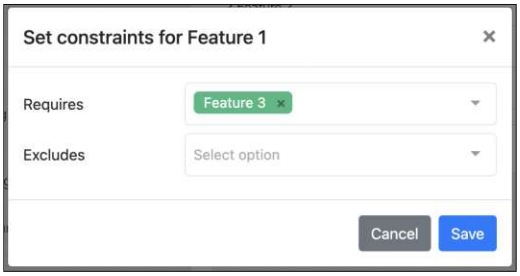
(a) Features sidebar with buttons to edit the constraints for each Feature

(b) Modal to define Feature Constraints. *Feature 1* requires *Feature 2* but excludes *Feature 3*.

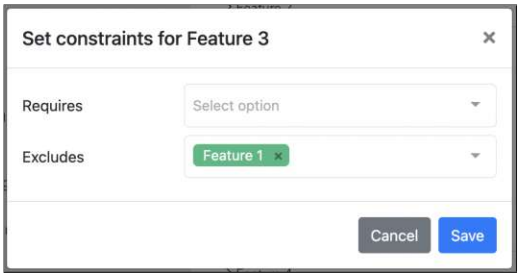


(c) Configuration Selection with Preview.

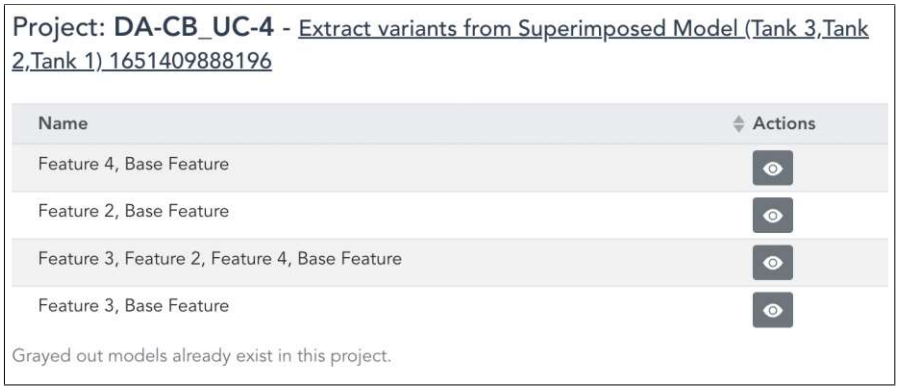
Figure 7.23: TC-4.2 - Screenshots of the derivation Process with defined Feature Constraints. Compared to TC-4.1 there is no configuration with *Feature 1* and *Feature 3* both present due to the defined *excludes* constraint.



(a) Feature Constraint Modal with *Feature 1* requires *Feature 3*.



(b) Feature Constraint Modal with *Feature 3* excludes *Feature 2*.



(c) Configuration Selection with Preview. *Feature 1* is not present in any configuration due to the conflicting constraints.

Figure 7.24: TC-4.3 - Screenshots of the derivation Process with conflicting Feature Constraints. *Feature 1* requires *Feature 3* but *Feature 3* excludes *Feature 1* results in *Feature 1* not present in any suggested configuration

7.6.3 Conformity to the MVA Metamodel

Section 7.3.3 already describes the conformity to the MVA metamodel, since the derived models are simple PPR models.

7.7 Capability 5: Superimposed PPR Model Analysis

This section comprises the evaluation of the analysis framework proposed in Section 6.3 with use case *UC-5: Superimposed PPR Model Analysis - Capacity Utilization*(see Section 4.6). The use case describes a typical workflow of engineers to analyze the process utilization in superimposed PPR models.

The evaluation is conducted, as described in the previous Section 7.1, by conducting the use case flows step-by-step in the VME.

Test Case \ Requirement	UC-5.R1	UC-5.R2
TC-5.1	×	×

Table 7.7: Test cases used to evaluate UC-5: Superimposed PPR Model Analysis - Capacity Utilization.

Variant Model	Planned Production Volume
Tank 1	8,304
Tank 2	12,385
Tank 3	6,253
Tank 4	2,148
Tank 5	2,482
Tank 6	2,947
Tank 7	1,957
Tank 8	568

Table 7.8: TC-5.1 - Planned production volume of Water Filter Variants

7.7.1 Test Cases

Table 7.7 lists the two test cases to evaluate the proposed framework (see Section 6.3) using the use case *UC-5* with the respective use case requirements.

TC-5.1 - Analyze Capacity Utilization of Water Filter Case Study This test case evaluates the basic flow of the use case *UC-5* in the VME. It describes a typical task of a *Production Process Optimizer* to analyze the utilization of processes in a superimposed PPR model given planned production volumes for each variant (used to create the superimposed model).

This test case is conducted using the *Water Filter* case study. The superimposed PPR model was created using all eight variant models. Table 7.8 shows the planned production volume for each variant. The planned production volumes were calculated using a random number generator[7].

7.7.2 Test Case Execution

TC-5.1 - Analyze Capacity Utilization of Water Filter Case Study For this test case, the steps of the basic flow of use case *UC-5* (see RUCM notation of UC-5 in

[7] https://www.random.org/integer-sets/?sets=1&num=8&min=100&max=20000&seqnos=on&commas=on&order=index&format=html&rnd=id.2022-05-01-19-58, last accessed 2022-05-01

Table 4.5) are executed from the viewpoint of a *Production Process Optimizer*.

At first, the engineer opens the superimposed PPR model and starts the analysis workflow. Therefore, the engineer selects the *Analyse Model* action shown in Figure 7.25a. Figure 7.25b shows the modal that opens up. Engineers select the analysis type *Superimposed (PPR) Model Utilization*, which is described in Section 6.1, and enters the planned production volume for each variant model. Finally, the button *Analyse* starts the analysis.

The analysis results are integrated into the model, s.t. model elements with an analysis results are highlighted using a heat-map coloring schema. Figure 7.26b shows the highlighted processes with the respective color of the UR. The VME also shows a legend to map the colors to the respective UR. 0.000 or (0%) means no utilization at all, while 1.000 or (100%) means that the process is utilized in all variants. Engineers can get the absolute utilization by moving the mouse over a process (c.f. Figure 7.26c)

7.7.3 Conformity to the MVA Metamodel

The conformity of the superimposed model itself, were already discussed in Section 7.4.3.

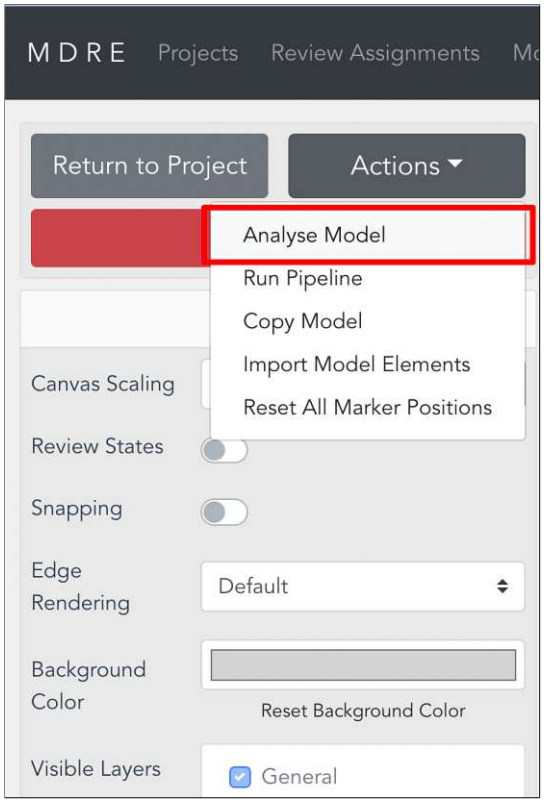
What leaves us with the discussion of the conformity of the analysis result to the MVA metamodel.

Figure 5.4 shows the relevant parts of the MVA metamodel for model analysis.

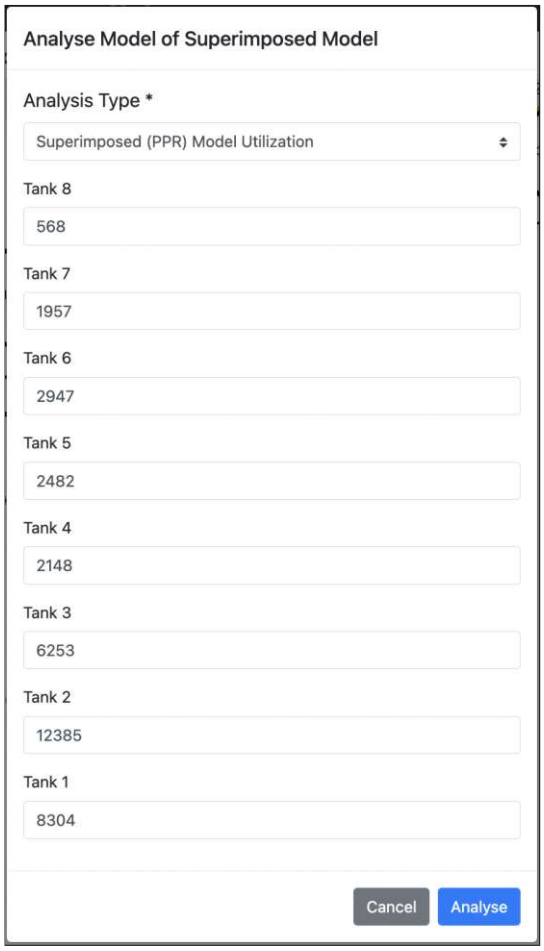
Figure 7.25b shows the input fields to enter the planned production volume of each variant model connected to the superimposed model. These fields correspond to the concept of *Model Properties* of the MVA metamodel, since the entered value characterizes the variant model.

Figure 7.26c shows the process *31: Mount Wastewater Tank* with a colored border. The colored border serves as a marker to highlight nodes and edges that are contained in the analysis result. These markers with colored border conform to the concept of *Analysis Marker* defined by the MVA metamodel.

To summarize, the definition of the planned production volumes and the superimposed model with marker to show the analysis results, conforms to the MVA metamodel.

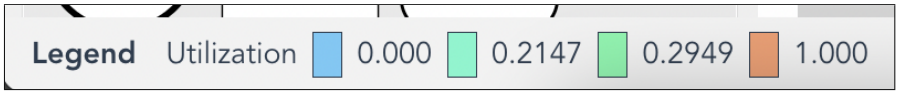


(a) Location of Model Analysis Option

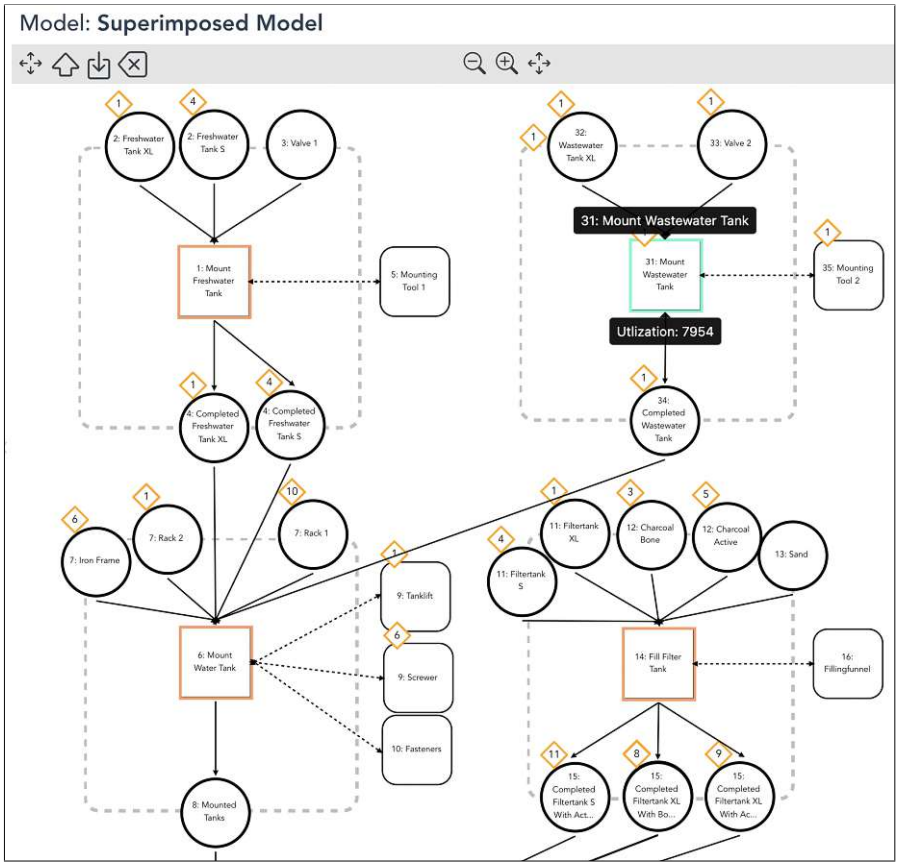


(b) Modal to select Model Analysis Type with Input fields for Planned Production Volume

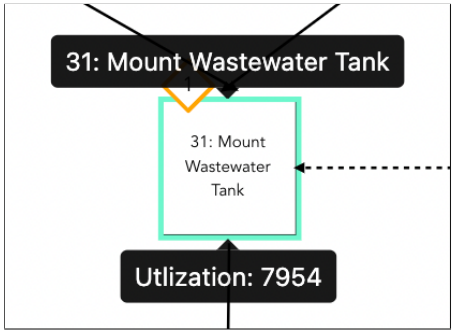
Figure 7.25: TC-5.1 - Screenshots of the Analysis Process in the VME



(a) Model Analysis Result Legend at the bottom of the Modeling Component in the VME



(b) The Processes are highlighted with a colored border representing the Utilization Ratio (UR)



(c) A tooltip shows the absolute UR

Figure 7.26: TC-5.1 - Screenshots of the Analysis Result

CHAPTER 8

Discussion

This chapter connects the results of this thesis with the evaluation and discusses the fulfillment of the research questions. Additionally, this chapter outlines the limitations of the proposed approaches.

8.1 RQ1: Model Variability Analysis Metamodel

The aim of research question *RQ1. MVA Metamodel. What metamodel facilitates the knowledge representation required for manipulating the variability of production processes?* was to find a suitable metamodel to not only visually represent production processes with variability for engineers, but also to have the production processes with variability in a machine-readable format.

This research question was motivated by the lack of an existing metamodel to represent production processes with concepts for feature annotations and model analysis (e.g., difference and utilization analysis). We assumed that such a metamodel provides a structured notation of production processes (and superimposed models) to be both human- and machine-readable.

During a basic literature search, we found the notation of the FPD [VDI, 2005] most promising. Especially, because of the extensions made regarding constraints and superimposed models [Kathrein et al., 2019, Meixner et al., 2019]. However, in contrast to AutomationML (what we found as the second notation), the FPD provides only visual representation but no hint on data structures.

Thus, this thesis introduced the *Model Variant Analysis (MVA)* metamodel that builds on the concept of *graphs* to model PPR models in VDI 3682 [VDI, 2005] notation. Further, we introduced concepts to support **(i)** the visualization of differences of two models (Model Difference Analysis (MDA)), **(ii)** the visualization of *superimposed (PPR) models*,

and **(iii)** the visualization of analysis results, i.e., mark model elements depending on the analysis result.

The MVA metamodel was evaluated using the use cases described in Chapter 4. In the evaluation of the metamodel we conducted the three approaches, i.e., PPRVM-MDA (cf. Section 6.1), PPRVM-FCI-AV (cf. Section 6.2.2), PPRVM-FCI-DV(c.f. Section 6.2.3), PPRVM-ANALYSIS (cf. Section 6.3). The evaluation indicated that the MVA metamodel is also suitable for other types of graph-based models, at least for UML state charts.

This thesis goes beyond the state of the art Meixner et al. [2019], Schleipen et al. [2015] by introducing the MVA metamodel to address challenge *C1 - Insufficient knowledge representation of production processes with variability.*

8.2 RQ2: Model Difference Analysis

The MVA metamodel provides the means to visualize PPR variant and superimposed models. To take full advantage of the metamodel, engineers need approaches to work with instance models that correspond to the metamodel.

Thus, the research question *RQ2. Model Difference Analysis. What semi-automated approach can effectively identify differences between PPR models with variability?* aimed at the design of an approach to identify differences of PPR variant models and superimposed PPR models. This research question was motivated due to the characteristics of PPR models, i.e., model elements can have properties that are not visually represented, introduced by the VDI 3682 [VDI, 2005] notation making simple visual comparison unsuitable.

We found many comparison and visualization techniques in the literature, which are described in detail in Chapter 2. However, we found [Brun and Pierantonio, 2008] and [Zoubek et al., 2018] most promising regarding algorithm design and visualization.

With the characteristics of PPR models, the requirements derived from use case UC-1, and the literature of existing difference comparison and visualization techniques, we designed the semi-automated PPRVM Model Difference Analysis (PPRVM-MDA) approach. The PPRVM-MDA approach takes two models, that conform to the MVA metamodel, as input and returns a list of operations to apply to the first model to get the second model. Further, the PPRVM-MDA approach automates the calculation and visualization step but leaves the model selection upfront and the result analysis afterward to the engineers.

Within this thesis, we integrated the PPRVM-MDA approach into the VME which is a web-based engineering tool based on the MDRE[Prock et al., 2021].

The PPRVM-MDA approach was evaluated with the use case *UC-1*, described in Chapter 4, in a qualitative feasibility study. Based on the requirements resulting from the use case, test cases were defined. The execution of the test cases has shown that the PPRVM-MDA approach meets all the requirements of the use case UC-1.

With the proposed approach, this thesis extends the research knowledge base Zoubek et al. [2018] by providing a tool for engineers in the CPPS domain to improve the overall quality of created models, especially in organizations following the guidelines of the VDI 3695.

Limitations The PPRVM-MDA approach is a solid model difference analysis approach that works with a variety of graph-based model types. However, the visualization of changes may be improved to improve usability.

For example, with large models, the navigation of the models can become tedious since there is no indicator of changes outside the visible part of the models. This affects both, the *integrated view* and the *side-by-side view*. In [Zoubek et al., 2018], the authors, propose the *Merged off-screen indicators*, which would improve usability considerably when integrated. However, the integration of these indicators was beyond the scope of this work.

Another limitation is, that the placement of model elements in the *integrated view* is not automatic to preserve the mental map of the engineers, which is considered crucial [Zoubek et al., 2018, Schipper et al., 2009, Ohst et al., 2003a]. In this work, the first model is used as the basis for the positions of the model elements, and the model elements of the second model are added. This may lead to overlapping of model elements. Although model elements can be moved during MDA, a smart and optimized placement algorithm would improve usability.

8.3 RQ3: Model Evolution

The third research question of this thesis, *RQ3. Model Evolution. What semi-automated approach facilitates the evolution and analysis of superimposed PPR models?*, aimed at different semi-automated approaches to support engineers to extend and analyze existing superimposed PPR models.

Incremental Feature Identification. We found that existing approaches for feature identification and superimposed models [Meixner et al., 2020c, Ziadi et al., 2012] only allow engineers to identify features from scratch and can not incrementally extend the identified features. In CPPS engineering, production processes can become considerable and with a high number of variants [Meixner et al., 2020c], the feature identification requires significant computational effort. It is desired to optimize computational effort for feature identification, s.t. newly identified features of new variant models are merged into the existing set of features of the superimposed model.

To optimize the computational effort, this thesis introduced the PPRVM-FCI Add Variants (PPRVM-FCI-AV) approach inspired by the work of [Boubakir and Chaoui, 2018]. The PPRVM-FCI-AV approach considers three cases. Case 1 considers model elements that are not yet part of the superimposed PPR model. Case 2 considers model

elements that are present in the base feature of the superimposed PPR model. Case 3 considers model elements that are present in optional features of the superimposed PPR model. Depending on the case, either a new feature candidate is created or an existing feature candidate is split into two feature candidates.

The evaluation procedure of the PPRVM-FCI-AV is the same as of the PPRVM-MDA approach. Test cases were derived from the requirements of the use cases *UC-2* and *UC-3*. The execution of the test cases has shown that the PPRVM-FCI-AV approach meets all requirements of the use cases *UC-2* and *UC-3*.

Additionally, we compared the computational steps and time for creating the superimposed PPR model from scratch with the incremental feature identification. Figure 8.1 shows the comparison of the computational steps on the left and the required computation time on the right of the *Rocker Switch* product line. It shows, that, in total, the combination of PPRVM-FCI and the PPRVM-FCI-AV requires more computational steps. The PPRVM-FCI takes 1.427 steps constantly, to create the superimposed PPR model from scratch, while the incremental feature identification using the PPRVM-FCI-AV takes a median of 1.626 steps. The range of steps can be explained by the order in which the model variants were added. The number of steps depends on which case is hit. The comparison of the required computational time shows that the incremental feature identification (*PPRVM-FCI + PPRVM-FCI-AV*) takes slightly less time than the calculation from scratch (*PPRVM-FCI* only).

Figure 8.2 shows the comparison of the computational steps on the left and the required computation time on the right of the *Water Filter* product line. In this case, the computational steps are similar at the median, and it also shows that incremental feature identification takes less time at the median, but varies more depending on the case.

To sum up, two medium complex case studies suggest that both approaches require similar steps and time to compute the feature candidates of the superimposed PPR model.

Derivation of Variant Models. In this thesis, model evolution also includes the derivation of variant models from superimposed models. Thus, this thesis introduced the PPRVM-FCI Derive Variants (PPRVM-FCI-DV) approach that supports engineers in the extraction of variant models from a superimposed model. The approach takes a superimposed model and uses the YASA sampling algorithm to calculate a representative set of feature candidates, and the resulting variant models are presented to engineers. Engineers can choose variant models that the approach automatically persists as VME models. The calculation of the representative set of feature candidates and the persisting as VME models is fully automated, while the selection of variant models is left to the engineers, which makes the PPRVM-FCI-DV approach semi-automated.

This approach is important, especially for quality engineers who can extract a representative set, i.e., each feature is covered at least once, of variant models and persist them in the VME. Engineers can use the derived variant models, for example, for testing

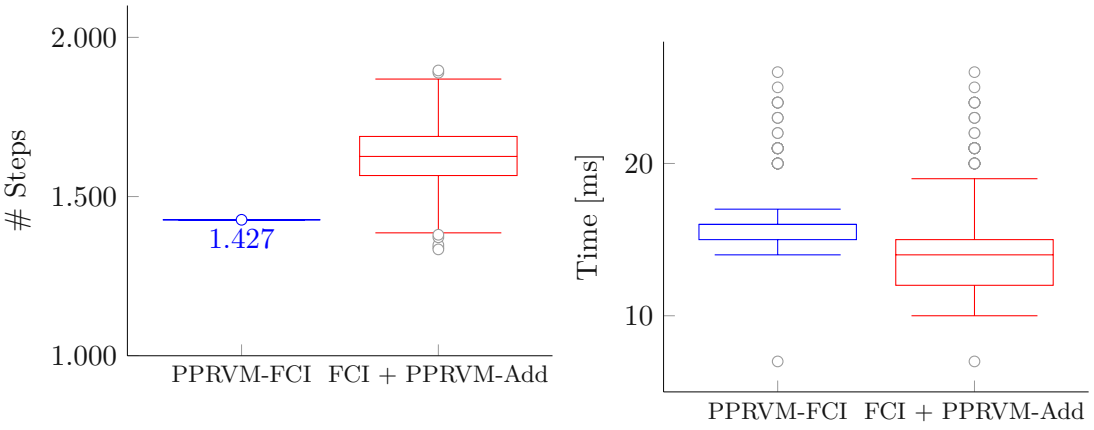


Figure 8.1: Rocker Switch Product Line - Steps and Timing - Comparison of the calculation of the feature candidates of all variant models from scratch (blue) and the incremental feature identification (red)

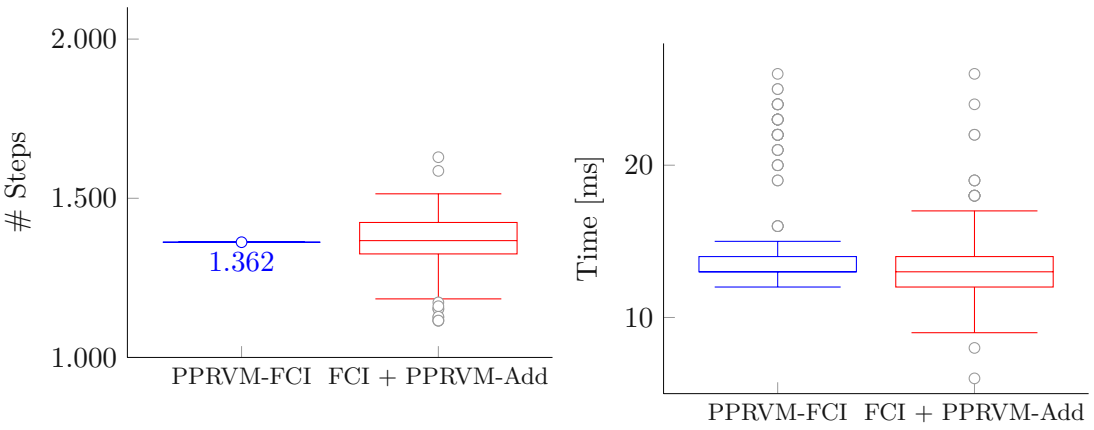


Figure 8.2: Water Filter Product Line - Steps and Timing - Comparison of the calculation of the feature candidates of all variant models from scratch (blue) and the incremental feature identification (red)

[Meixner et al., 2020b] or export them in data formats of other engineering tools, e.g., *GraphML* or *FeatureIDE*.

Due to the loosely coupled integration of the YASA sampling algorithm, it is easily possible to change the used sampling algorithm for other purposes, e.g., to create variant models of all combination of features.

Like the others, this approach was evaluated using test cases derived from the requirements of use case UC-4. The execution of the test cases has shown that the PPRVM-FCI-DV approach meets all requirements of the use case *UC-3*.

Model Analysis beyond Difference Analysis. The sizing of CPPS components is an integral task of *Production Process Optimizer* to find the optimal utilization of components required in the production processes. Thus, this thesis introduced the PPRVM Analysis Framework (PPRVM-ANALYSIS) which is a generic framework for model analysis in the VME.

To support *Production Process Optimizer* in their work, this thesis, further, introduced an example analysis using the PPRVM-ANALYSIS framework. The analysis comprises the utilization of superimposed PPR models, especially the utilization of *process* model elements. Within the VME, it enables engineers to take a superimposed PPR model, input the planned production volume of each variant (that were used to create the superimposed model), e.g., variant one of the *Rocker Switch* will be produced 1000 times, variant two 500 times, and so on. Subsequently, the VME calculates the utilization ratio and visualizes it directly in the superimposed PPR model by highlighting *processes* with a color depending on the utilization value.

The automatic calculation of the utilization ratio, makes this approach semi-automated, since the input values have to be given by the engineers.

This approach has also been evaluated using test cases derived from the requirements of use case UC-5. The execution of the test cases has shown that the PPRVM-ANALYSIS framework and the utilization analysis approach meets all requirements of the use case *UC-5*.

Limitations The proposed approaches for PPR variability modeling provide a solid tool set for a wide variety of models. However, the approaches have their limitations.

The PPRVM-FCI-AV approach can preserve custom feature assignments up to a certain point. However, engineers currently cannot trace visually how the approach splits the features when adding variants. Currently, the list of features presented to the engineer is the same as for creating a superimposed model from scratch and does not highlight split features.

A limitation of the PPRVM-FCI-DV approach is, that currently, and due to the use of the YASA sampling algorithm, it is not possible to specify a custom feature combination

to be derived as a feature model. However, using another sampling algorithm would release this limitation.

Summary Overall, the proposed approaches improve the state of the art. The PPRVM-FCI-AV approach improves the state of the art [Meixner et al., 2020c, Ziadi et al., 2012] by providing a semi-automated tool for engineers to extend an existing superimposed model with variant models without the need to calculate the superimposed model from scratch. Since engineers can customize features, the risk of loosing the customization is reduced to a single case.

The PPRVM-FCI-DV approach extends the research knowledge [Garousi and Mäntylä, 2016, Meixner et al., 2020b] by providing a semi-automated approach to derive variant models from superimposed models to address challenge *C4 - Insufficient testing capabilities for production processes with variability*. Finally, the PPRVM-ANALYSIS framework extends the research knowledge by providing a semi-automated approach to analyze superimposed PPR models.

8.4 Limitations

The following limitations apply to the thesis.

The proposed approaches were evaluated using three case studies from two different authors. Therefore, there is a risk that our results contain errors that are already present in the origin work. Nevertheless, one of the three case studies originated from another author. Thus, it is likely that the approaches bring similar results in other case studies. Further, the case studies are abstracted from the real world and, therefore, bear the risk of being too abstract which may influence the results of more advanced case studies.

The evaluation of the test cases using the proposed approaches were conducted by the author of the thesis. However, one supervisor conducted several of the test cases in a workshop to reproduce partial results of the evaluation. This provided a careful supervision of the test cases. However, an evaluation with a larger audience ...

In this work, the computation steps and timing for the combined use of the PPRVM-FCI and PPRVM-FCI-AV approach was compared to the use of PPRVM-FCI each time a superimposed model needs to be created. A trend toward more steps, but less time, can be observed. However, with only two medium-complex case studies, this trend is not representative because they may be outliers. A deeper evaluation using more representative case studies have to be conducted to make a valid statement on the efficiency of the approach.

CHAPTER 9

Conclusion and Future Work

This chapter summarizes the conducted work for this thesis and presents topics that remain as future work due to the limited scope of the thesis.

9.1 Conclusion

CPPSs are software-intensive production systems that are capable of autonomously adapting to changes in their environment, and are therefore seen as an integral part of *Industry 4.0*[Federal Ministry for Economic Affairs and Energy]. The planning phase of CPPSs is characterized by several disciplines working together to build and integrate the physical devices and the required software. Engineers in the basic planning phase design a rough concept of CPPSs that supports all relevant production processes based on the product line requirements. Therefore, engineers have to identify commonalities and variability of the product processes they created for each relevant product.

This identification is time-consuming, error-prone, and hard to reproduce due to insufficient knowledge representation of production processes with their variability. Furthermore, the limitations of current means to evolve the product family, i.e., product requirements change, requires engineers to conduct the identification of commonalities and variability from scratch. On top, non-specialized tools like spreadsheets without clear semantics impede optimization or analysis tasks for production process optimizer or quality engineers. Therefore, the goal of this thesis is to lay the foundation to support engineers in the basic planning phase of CPPSs to reduce errors, increase reproducibility, and reduce manual effort. To achieve this aim, this thesis outlines several typical use cases of CPPS engineers in the basic planning phase from literature and an industry partner in Chapter 4, which are used for evaluation.

In this thesis, the Design Science methodology was used to identify and investigate shortcomings in the design and analysis of production processes in the context of CPPS

engineering. The *FPD* [VDI, 2005] and its extensions [Kathrein et al., 2019, Meixner et al., 2020c] already provide a solid basis for visualization of PPR models with variability. However, these approaches have no underlying formal metamodel with support for model analysis and feature annotation. Additionally, we found opportunities for optimization in the calculation of superimposed models, especially regarding the evolution of superimposed models, i.e., the iterative extension of superimposed models. To evaluate the results of this thesis, a technical prototype is provided that builds on the already available Model Design and Review Editor (MDRE). The proposed metamodel and the approaches are evaluated in a qualitative approach, comparing identified requirements of the use cases with the provided capabilities of the prototype. The typical use cases were conducted step-by-step in the prototype and documented with screenshots and resulting models. The evaluation results in a set of capabilities the software prototype, the metamodel, and the approaches provide to CPPS engineers.

Based on the findings during literature research and the requirements derived from the use cases of CPPS engineers, this thesis proposes the Model Variant Analysis (MVA) metamodel and approaches to design, evolve and analyze (superimposed) PPR models. With the MVA metamodel, this thesis addresses the insufficient knowledge representation for production processes that is human and machine-readable. The MVA metamodel is graph-based, making nodes and edges first-class citizens with concepts for feature annotation and model analysis result visualization. It allows the designing of production processes, the annotation of features used for superimposed models, and the visualization of model analysis results, like model difference, i.e., comparison of two models, and utilization analysis, i.e., analyzing the utilization of production processes.

To support engineers in identifying commonalities and variability, this thesis proposes the PPRVM Model Difference Analysis (PPRVM-MDA) approach that gives engineers the ability to compare (superimposed) PPR models to find differences. The Model Difference Analysis (MDA) is an integral task of engineers during the design and optimization of production processes of CPPSs.

The PPRVM-FCI Add Variants (PPRVM-FCI-AV) approach is one of the major contributions of this work. During literature research, this thesis identified that feature identification and superimposed model calculation by existing approaches [Ziadi et al., 2012, Meixner et al., 2020c] are always conducted from scratch. This means that every time an engineer needs to adapt the superimposed model, the calculation is done from scratch. Thus, the PPRVM-FCI-AV approach extends the existing approaches [Meixner et al., 2020c, Ziadi et al., 2012] to be able to take an existing superimposed model and extend it with other production processes. The PPRVM-FCI-AV approach is inspired by the work of Boubakir and Chaoui [2018], who incrementally calculate a variability model from model variants.

Further, this thesis introduces the PPRVM-FCI Derive Variants (PPRVM-FCI-DV) approach and the PPRVM Analysis Framework (PPRVM-ANALYSIS) to assist engineers in optimization tasks and quality assurance. The PPRVM-FCI-DV approach enables engineers to derive variant models from superimposed models. Therefore, engineers can

define constraints between features that are considered by the PPRVM-FCI-DV. This approach is useful to derive a representative set of variant models with each feature covered, e.g., for testing.

Finally, this thesis introduces the PPRVM-ANALYSIS framework which provides extendible analysis capabilities to engineers. This thesis presents the calculation of the utilization rate of processes as a proof of concept of the framework. The utilization calculation enables, engineers to define the planned production volume of certain products that are planned to be produced by the CPPS and returns the results as colored overlays of the processes.

For evaluating the proposed metamodel and the approaches, a software prototype was created. The Variability Modeling Editor (VME) is a software prototype based on the existing engineering tool Model Design and Review Editor (MDRE) [Prock et al., 2021] with the proposed MVA metamodel and the engineering approaches integrated. The evaluation was done using the software prototype by conducting the use cases step-by-step, resulting in capabilities the VME provides to aid CPPS engineers. The evaluation shows that typical use cases of CPPS engineers, elicited from literature and an industry partner, profit off the proposed metamodel and approaches. It further shows that the MVA metamodel and the approaches are open to other types of models.

Overall, the evaluation results seem promising to be applicable in the field of CPPS engineering to support engineers in the basic planning phase of CPPSs. However, due to the limited scope of this thesis, further evaluation, i.e., with experts from industry and more use cases is required to harden the evidence.

Referring back to the stakeholders introduced in Section 1.2, each group of stakeholders can take up the results of this work.

Practitioners in the CPPS industry can use the VME to conduct tasks like modeling, optimizing and analyzing of their planning assets, like production processes. *Basic Planners* can use the VME to model production processes as PPR models. Additionally, they can create and extend superimposed PPR models of the planned CPPSs to deliver a rough design to the following engineering phases. *Production Process Optimizers* can use the VME to analyze (superimposed) PPR models created by basic planners and provide optimization feedback to subsequent engineering phases. *Quality Engineers* can use the derivation capabilities of the VME to derive a representative set of variant models and provide them to other tools, e.g., for testing.

Researchers can take up the results of this thesis to build upon and improve these results or derive other approaches, maybe also based on the future work presented in the next section.

Software Engineers can improve or extend the VME prototype to improve existing or integrate advanced approaches that aid engineers in planning CPPSs. For example, to integrate new analysis types using the proposed PPRVM-ANALYSIS framework.

9.2 Future Work

This section outlines opportunities for future work that partly results from the limitations outlined in Chapter 8.

Empirical Evidence The qualitative evaluation of the MVA approach was conducted with representative product lines and use cases abstracted from industry. Nevertheless, in the future, the approach needs to be evaluated with a broader range of product lines of different size. Furthermore, the approach should be thoroughly evaluated by industry partners and further practitioners from the industry.

Find Variability in Production Step Order in PPR Models The *FPD* (VDI 3682) introduces the concept of *system limit* to logically represent a process step comprising a process, products and resources. In this thesis, the *SystemBoundary* node is the counterpart in the PPR DSL of the VME. The PPRVM-FCI approach considers this relationship only in the comparison of CPs but not to check if whole production steps are equal, including all products, resources, and the process. Thus, a possible extension is to modify the PPRVM-FCI approach to also find whole production steps that are equal but, e.g., in a different order in the production process.

Constraints and Verification This thesis utilizes *constraints* on features, based on [Kathrein et al., 2019, Meixner et al., 2021], that allows engineers to define which features are incompatible with each other or required by another feature. Currently, these constraints are only considered in the PPRVM-FCI-DV approach at the time of the feature set calculation. A possible extension is to use these constraints also on production process steps (*SystemBoundaries*) to validate if there is any constraint violation. For example, engineers may define that a certain production process step must not follow a specific other, e.g., placing a topping on a cake before the cake is baked.

Improve Navigation of Large Models. Since the Variability Modeling Editor (VME) is only a prototype, some usability aspects can be improved. For example, the navigation of large models during the MDA. Large models exceed the viewport rapidly, which requires engineers to scroll through the model to see all differences. A possible extension is to create either a mini map which indicates the location on the model with changes highlighted or to show off-screen indicators like in [Zoubek et al., 2018].

Bibliography

J. Bézivin. On the unification power of models. *Software & Systems Modeling*, 4(2): 171–188, 2005.

S. Biffl, D. Gerhard, and A. Lüder. Introduction to the Multi-Disciplinary Engineering for Cyber-Physical Production Systems. In S. Biffl, A. Lüder, and D. Gerhard, editors, *Multi-Disciplinary Engineering for Cyber-Physical Production Systems, Data Models and Software Solutions for Handling Complex Engineering Projects*, pages 1–24. Springer, 2017a. doi: 10.1007/978-3-319-56345-9_1.

S. Biffl, A. Lüder, and D. Gerhard. *Multi-Disciplinary Engineering for Cyber-Physical Production Systems: Data Models and Software Solutions for Handling Complex Engineering Projects*. Springer, 2017b.

S. Biffl, J. Musil, A. Musil, K. Meixner, A. Lüder, F. Rinker, and D. Winkler. Industry 4.0 artifact-based coordination in production systems engineering processes, Jan. 2021.

J. Bosch. Software product lines: Organizational alternatives. In *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, pages 91–100. IEEE, 2001.

M. Boubakir and A. Chaoui. An incremental approach for the extraction of software product lines from model variants. In *International Conference on Computer Science and Its Applications*, pages 124–134. Springer, 2018.

M. Brambilla, J. Cabot, and M. Wimmer. Model-driven software engineering in practice. *Synthesis lectures on software engineering*, 3(1):1–207, 2017.

C. Brun and A. Pierantonio. Model differences in the eclipse modeling framework. *UPGRADE, The European Journal for the Informatics Professional*, 9(2):29–34, 2008.

H. Brunelière, J. G. Perez, M. Wimmer, and J. Cabot. EMF Views: A View Mechanism for Integrating Heterogeneous Models. In P. Johannesson, M.-L. Lee, S. W. Liddle, A. L. Opdahl, and O. P. López, editors, *Conceptual Modeling - 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings*, volume 9381 of *Lecture Notes in Computer Science*, pages 317–325. Springer, 2015. doi: 10.1007/978-3-319-25264-3_23.

K. Brush, B. Cole, and J. O'Donnell. What is process manufacturing? Definition and examples, 2022.

P. C. Bryan and M. Nottingham. JavaScript object notation (JSON) patch. RFC 6902, Apr. 2013.

O. Cardin. Classification of cyber-physical production systems applications: Proposition of an analysis framework. *Computers in Industry*, 104:11–21, 2019.

C. Chambers and C. Scaffidi. Struggling to excel: A field study of challenges faced by spreadsheet users. In *2010 IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 187–194. IEEE, 2010.

L. Chen and M. A. Babar. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology*, 53(4): 344–362, 2011.

H. Cheng, L. Xue, P. Wang, P. Zeng, and H. Yu. Discrete manufacturing ontology development. In *2017 IEEE International Conference on Industrial Technology (ICIT)*, pages 1393–1396, 2017. doi: 10.1109/ICIT.2017.7915568.

K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wasowski. Cool features and tough decisions: A comparison of variability modeling approaches. *ACM International Conference Proceeding Series*, (May 2014):173–182, 2012. doi: 10.1145/2110147.2110167.

S. Diehl and C. Görg. Graphs, They Are Changing. In S. G. Kobourov and M. T. Goodrich, editors, *Graph Drawing, 10th International Symposium, GD 2002, Irvine, CA, USA, August 26-28, 2002, Revised Papers*, volume 2528 of *Lecture Notes in Computer Science*, pages 23–30. Springer, 2002. doi: 10.1007/3-540-36151-0_3.

C. Engelbrecht. *Coordinated Multi-View Graph Analysis and Improvement with Applications in Cyber-Physical Production Systems Engineering*. PhD thesis, Wien, 2021.

Federal Ministry for Economic Affairs and Energy. What is Industrie 4.0?

M. Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.

V. Garousi and M. V. Mäntylä. When and what to automate in software testing? A multi-vocal literature review. *Information and Software Technology*, 76:92–117, 2016.

S. Gupta and V. Krishnan. Product family-based assembly sequence design methodology. *IIE transactions*, 30(10):933–945, 1998.

A. R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

J. W. Hunt and M. D. MacIlroy. *An Algorithm for Differential File Comparison*. Bell Laboratories Murray Hill, 1976.

I. Jacobson. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Pearson Education India, 1993.

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 November, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

L. Kathrein, A. Lueder, K. Meixner, D. Winkler, and S. Biffl. Process analysis for communicating systems engineering workgroups. *Technical report, Technical Report CDL-SQI-2018-11*, 2018.

L. Kathrein, K. Meixner, D. Winkler, A. Lüder, and S. Biffl. Extending the Formal Process Description towards Consistency in Product/ion-Aware Modeling. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 679–686. IEEE, 2019.

U. Kelter, J. Wehren, and J. Niere. A generic difference algorithm for uml models. *Software Engineering 2005*, 2005.

S. Krieter, T. Thüm, S. Schulze, G. Saake, and T. Leich. YASA: Yet another sampling algorithm. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, pages 1–10, 2020.

E. A. Lee. Cyber-physical systems-are computing foundations adequate. Citeseer, 2006.

S. Lity, S. Nahrendorf, T. Thüm, C. Seidl, and I. Schaefer. 175% Modeling for Product-Line Evolution of Domain Artifacts. In *Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, VAMOS 2018, Madrid, Spain, February 7-9, 2018*, pages 27–34, 2018. doi: 10.1145/3168365.3168369.

J. Martinez, T. Ziadi, J. Klein, and Y. Le Traon. Identifying and visualising commonality and variability in model variants. In *European Conference on Modelling Foundations and Applications*, pages 117–131. Springer, 2014.

J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon. Bottom-up adoption of software product lines: A generic and extensible approach. In D. C. Schmidt, editor, *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 101–110. ACM, 2015. doi: 10.1145/2791060.2791086.

R. McCabe, G. Campbell, N. Burkhard, S. Wartik, J. O'Connor, J. Valent, and J. Facemire. Reuse-Driven Software Processes Guidebook. Techreport SPC-92019-CMC, Version 02.00.03, Software Productivity Consortium, Nov. 1993.

K. Meixner, R. Rabiser, and S. Biffl. Towards modeling variability of products, processes and resources in cyber-physical production systems engineering. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume B*, pages 49–56, 2019.

K. Meixner, J. Decker, H. Marcher, A. Lüder, and S. Biffl. Towards a domain-specific language for product-process-resource constraints. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1405–1408. IEEE, 2020a.

K. Meixner, L. Kathrein, D. Winkler, A. Lüder, and S. Biffl. Efficient test case generation from product and process model properties and preconditions. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 859–866. IEEE, 2020b.

K. Meixner, R. Rabiser, and S. Biffl. Feature identification for engineering model variants in cyber-physical production systems engineering. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, pages 1–5, 2020c.

K. Meixner, K. Feichtinger, R. Rabiser, and S. Biffl. A reusable set of real-world product line case studies for comparing variability models in research and practice. In *Proceedings of the 25th ACM International Systems and Software Product Line Conference - Volume b*, SPLC '21, pages 105–112, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 978-1-4503-8470-4. doi: 10.1145/3461002.3473946.

M. H. Meyer and J. M. Utterback. The product family and the dynamics of core capability. 1992.

G. Model. The NanoFilter®. https://twitter.com/GongaliModel/status/1217727935744004096, 2022.

L. Monostori. Cyber-physical production systems: Roots, expectations and R&D challenges. *Procedia Cirp*, 17:9–13, 2014. doi: 10.1016/j.procir.2014.03.115.

T. Moser and S. Biffl. Semantic integration of software and systems engineering environments. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(1):38–50, 2011.

D. Ohst, M. Welle, and U. Kelter. Difference Tools for Analysis and Design Documents. In *19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22-26 September 2003, Amsterdam, The Netherlands*, pages 13–22. IEEE Computer Society, 2003a. doi: 10.1109/ICSM.2003.1235402.

D. Ohst, M. Welle, and U. Kelter. Differences between versions of UML diagrams. In J. Paakki and P. Inverardi, editors, *Proceedings of the 11th ACM SIGSOFT Symposium on Foundations of Software Engineering 2003 Held Jointly with 9th European Software*

Engineering Conference, ESEC/FSE 2003, Helsinki, Finland, September 1-5, 2003, pages 227–236. ACM, 2003b. doi: 10.1145/940071.940102.

T. Pett, T. Thüm, T. Runge, S. Krieter, M. Lochau, and I. Schaefer. Product sampling for product lines: The scalability challenge. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, SPLC '19, pages 78–83, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-7138-4. doi: 10.1145/3336294.3336322.

A. Prock, C. Engelbrecht, M. Isikoglu, C. Burger, and D. Kretz. Model design and review editor: Featureliste. Technical report cdl-sqi 2021-12, Sept. 2021.

J. F. Ragan. Measuring capacity utilization in manufacturing. *Federal Reserve Board New York Quarterly Review*, 1:13–28, 1976.

J. Rubin and M. Chechik. Combining related products into product lines. In *International Conference on Fundamental Approaches to Software Engineering*, pages 285–300. Springer, 2012.

I. Schaefer, R. Rabiser, D. Clarke, L. Bettini, D. Benavides, G. Botterweck, A. Pathak, S. Trujillo, and K. Villela. Software diversity: State of the art and perspectives. *Int. J. Softw. Tools Technol. Transf.*, 14(5):477–495, 2012. doi: 10.1007/s10009-012-0253-y.

A. Schipper, H. Fuhrmann, and R. von Hanxleden. Visual Comparison of Graphical Models. In *14th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2009, Potsdam, Germany, 2-4 June 2009*, pages 335–340. IEEE Computer Society, 2009. doi: 10.1109/ICECCS.2009.15.

M. Schleipen, A. Lüder, O. Sauer, H. Flatt, and J. Jasperneite. Requirements and concept for plug-and-work. *at-Automatisierungstechnik*, 63(10):801–820, 2015.

J. Sprey and C. Sundermann. Computing attribute ranges for partial configurations with JavaSMT. 2018.

D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro. *EMF: eclipse modeling framework*. Pearson Education, 2008.

J. Trojanowska, A. Kolinski, D. Galusik, M. L. Varela, and J. Machado. A methodology of improvement of manufacturing productivity through increasing operational efficiency of the production process. In *Advances in Manufacturing*, pages 23–32. Springer, 2018.

F. J. Van der Linden, K. Schmid, and E. Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer Science & Business Media, 2007.

VDI. VDI/VDE 3682: Formalised Process Descriptions, 2005.

VDI. VDI/VDE 3695 - Engineering of Industrial Plants - Evaluation and Optimization, Part 2: Subject Processes, 2010.

M. Voelter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference (SPLC 2007)*, pages 233–242, 2007. doi: 10.1109/SPLINE.2007.23.

K. H. Weber. *Engineering Verfahrenstechnischer Anlagen*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-43528-1. doi: 10.1007/978-3-662-43529-8.

D. Weichert, P. Link, A. Stoll, S. Rüping, S. Ihlenfeldt, and S. Wrobel. A review of machine learning for the optimization of production processes. *The International Journal of Advanced Manufacturing Technology*, 104(5):1889–1902, 2019.

D. Wille. *Custom-Tailored Product Line Extraction*. PhD thesis, Technische Universität Carolo-Wilhelmina zu Braunschweig, 2019.

T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane. Feature identification from the source code of product variants. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 417–422. IEEE, 2012.

F. Zoubek, P. Langer, and T. Mayerhofer. Visualizations of Evolving Graphical Models in the Context of Model Review. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 381–391, 2018.

ZVEI. DIN SPEC 91345:2016-04, Referenzarchitekturmodell Industrie 4.0 (RAMI4.0), Apr. 2016.

List of Figures

1.1 Automotive factory as an example of a CPPS with the product (red) that is assembled in a process (indicated by the conveyor belt in yellow) using a working unit (green) - *(derivative of Tesla Robot Dance by Steve Jurvetson, used under CC-BY 2.0, licensed under CC-BY 2.0 by Kristof Meixner).* . 2

1.2 Example of preparation processes (1 and 2) for *Burgers* with different toppings and the combined production process (1+2) for both burger variants. Equal production steps are colored red. Equal production steps but in different positions are colored blue. White production steps are variant-specific and not part of the other one. 2

1.3 Challenges in the activities of basic engineering, based on [Weber, 2014]. 4

2.1 Three variants of the *Rocker Switch* product line (on the left) and the resulting superimposed model (on the right) [Meixner et al., 2019] in extended VDI 3682 [VDI, 2005] notation. The superimposed model shows optional features in hatched orange and annotates the corresponding features. . . . 13
2.2 Screenshot of the MDRE Model Component 15
2.3 Example Sandwich Feature Model [Sprey and Sundermann, 2018] — in a tree notation from FODA [Kang et al., 1990] 17
2.4 Decision Model of Sandwich Example [Sprey and Sundermann, 2018] — in a tabular notation [Czarnecki et al., 2012] 17
2.5 Positive variability (compositional approach) refers to adding features to a starting point to derive variants [Wille, 2019] 17
2.6 Negative variability (annotative approach) refers to removing features of a 150%-model to derive variants [Wille, 2019] 18

3.1 Design Science Approach based on [Hevner, 2007]. 21

4.1 Rocker switch (Electronicgrup, CC BY-SA 3.0 — https://w.wiki/4Drq). . 24
4.2 Rocker Switch Schema 24
4.3 Three Assembly Sequences of different rocker switch variants, based on [Meixner et al., 2019] 24
4.4 The NanoFilter®[Model, 2022] 26

5.1 MVA Core Metamodel 40
5.2 MVA Metamodel with MDA Extension 42
5.3 MVA metamodel with Superimposed Model extension 43
5.4 MVA Metamodel with Model Analysis extension 44
5.5 *MVA Metamodel*. Core elements are shown in *black*. Elements to support model difference analysis are shown in *green*. Elements to support superimposed models are shown in *cyan*. Elements to support analysis of (superimposed) models are shown in *violet* 46

6.1 Concept of the integrated view of two models in the PPRVM-MDA 52
6.2 Concept of the side-by-side view of two models in the PPRVM-MDA . . . 53
6.3 Concept of the representation of a production step in superimposed PPR models with feature annotation. Based on Meixner et al. [2019] 55
6.4 Concept of the Representation of Superimposed PPR Models with Feature Annotations 56
6.5 List of Layers in the MDRE and extension introduced by PPRVM-MDA to toggle visibility of model elements. 57
6.6 PPR-FCI-AV Case 1: *Resource* creates new *Feature 3* 58
6.7 PPR-FCI-AV Case 2: *Input Product (2)* moves from *Feature 1* to new Feature *Feature 2* 58

6.8 PPR-FCI-AV Case 3: *Resource* moves from *Feature 0 (Base Feature)* to new *Feature 3* . . . 59
6.9 PPR-FCI-AV Edge Case: *Product 3* introduces a new feature candidate s.t. the user-defined feature candidate cannot be fully preserved. . . . 59
6.10 Architecture of the integration of the YASA . . . 61
6.11 Concept of the PPRVM-ANALYSIS framework. The framework separates the concerns into the *Frontend* that handles the representations, and the *Backend* that does the calculation. . . . 62
6.12 Component Utilization Analysis as Example of the Usage of the PPRVM-ANALYSIS Framework. . . . 63
6.13 Utilization Analysis - Visualization Concept . . . 64

7.1 TC-1.1 - Screenshot of the variant model selection and the compare list in the VME . . . 68
7.2 TC-1.1 - Screenshot of the integrated comparison view in the VME . . . 69
7.3 TC-1.2 - Screenshot of the target model selection box in the VME . . . 70
7.4 TC-1.2 - Screenshot of the integrated comparison view with the changed target model in the VME . . . 70
7.5 TC-1.3 - Screenshot of the view options in the VME . . . 70
7.6 TC-1.3 - Screenshot of the side-by-side comparison view with and without highlighting additions in the VME . . . 71
7.7 TC-1.4 - Screenshot of the differences of the UML state charts *Controller A* and *Controller B* in a side-by-side view . . . 72
7.8 TC-2.1 - Screenshot of the Variant Model Selection in the VME . . . 76
7.9 TC-2.1 - Screenshot of the calculated Features and parts of the resulting Superimposed PPR Model in the VME . . . 77
7.10 TC-2.1 - Resulting Water Filter Superimposed PPR Model in the VME Modeling Component . . . 78
7.11 TC-2.2 - Feature Customization in the process of creating a superimposed model . . . 78
7.12 TC-2.3 - GraphML Export of VME models . . . 80
7.13 TC-2.5 - Superimposed model of Controller A, B, C, based on [Rubin and Chechik, 2012] . . . 81
7.14 TC-2.6 - Hide Features of Superimposed PPR Model using the Layer Toggle of the VME Modeling Component . . . 82
7.15 Relevant parts of the MVA Metamodel for the PPRVM-FCI Approach . . . 83
7.16 TC-3.1 - Screenshot of the Variant Model Selection in the VME. Only models not already part of the Superimposed PPR model are shown. . . . 85
7.17 TC-3.1 - Screenshot of the calculated Features and the resulting Superimposed PPR model in the VME. . . . 86
7.18 TC-3.2 - PPRVM-FCI-AV Case 1 - Screenshot of the resulting superimposed PPR model in the VME . . . 86

7.19 TC-3.3 - PPRVM-FCI-AV Case 2 - Screenshot of the resulting superimposed PPR model in the VME 87
7.20 TC-3.4 - PPRVM-FCI-AV Case 3 - Screenshot of the resulting superimposed PPR model in the VME 88
7.21 TC-3.4 - PPRVM-FCI-AV Edge Case. *(1-2)* are the base models used to create the initial superimposed PPR model *(3)* with customized feature candidates, s.t. *Product 3*, *Process 3* and *Resource 4* are contained in *Feature 2*. *(4)* is the model that is added to the superimposed PPR model (*3*). *(5)* is the resulting superimposed PPR model with *Product 3* introducing a new feature candidate. 89
7.22 TC-4.1 - Screenshots of the derivation Process 92
7.23 TC-4.2 - Screenshots of the derivation Process with defined Feature Constraints. Compared to TC-4.1 there is no configuration with *Feature 1* and *Feature 3* both present due to the defined *excludes* constraint. 93
7.24 TC-4.3 - Screenshots of the derivation Process with conflicting Feature Constraints. *Feature 1* requires *Feature 3* but *Feature 3* excludes *Feature 1* results in *Feature 1* not present in any suggested configuration 94
7.25 TC-5.1 - Screenshots of the Analysis Process in the VME 97
7.26 TC-5.1 - Screenshots of the Analysis Result 98

8.1 Rocker Switch Product Line - Steps and Timing - Comparison of the calculation of the feature candidates of all variant models from scratch (blue) and the incremental feature identification (red) 103
8.2 Water Filter Product Line - Steps and Timing - Comparison of the calculation of the feature candidates of all variant models from scratch (blue) and the incremental feature identification (red) 103

List of Tables

1.1 Variant Matrix Example of the Burger Product Line. Rows represent the features while columns represent burger configurations. 5

4.1 UC-1 - Model Difference Analysis for a water filter product line, in RUCM notation. [Jacobson, 1993] 28
4.2 UC-2 Superimposed PPR Model Creation for a water filter product line, represented in RUCM notation. 30
4.3 UC-3 Superimposed PPR Model Evolution - Add a PPR Variant Model to an existing Superimposed PPR Model for a water filter product line, represented in RUCM notation 33
4.4 UC-4 Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model for a water filter product line, represented in RUCM notation, ... 34
4.5 UC-5 Superimposed PPR Model Analysis - Capacity Utilization, represented in RUCM notation 36

5.1 Mapping of Metamodel Requirements to relevant Use Case Requirements 38
5.2 Concepts of the MVA Metamodel Core 40
5.3 Concepts of the MDA extension for the MVA metamodel 42
5.4 Concepts of the Superimposed Model extension for the MVA metamodel . 43
5.5 Concepts of the Model Evolution extension for the MVA metamodel . . . 44
5.6 Complete list of concepts of the MVA metamodel 45

6.1 Attributes that extend the FPD (VDI 3682) [VDI, 2005] 48
6.2 Mapping from VDI 3682 [VDI, 2005] Concepts to MVA Concepts 49
6.3 Attribute Weight for Equality Score Calculation 49
6.4 Utilization Analysis - Model variants with planned production volumes . 64
6.5 Utilization Analysis - *Process* nodes of Variant Models and their calculated UR 64

7.1 Mapping from VDI 3682 [VDI, 2005] Concepts to MVA Concepts and to VME Element Types 66
7.2 Test cases used to evaluate UC-1: Model Difference Analysis (MDA) . . . 67
7.3 Test cases used to evaluate UC-2: Superimposed PPR Model Creation . . 74
7.4 Feature Mapping of the VME and But4Reuse 80

7.5 TC-3.6 - Ten different combinations of variant models of the *Water Filter* case study. Each set consists of the combination of variant models to initially create the superimposed PPR model and one or more batches to add variant models to the superimposed PPR model. 90
7.6 Test cases used to evaluate UC-4: Superimposed PPR Model Evolution - Derive Model Variants from a Superimposed PPR Model. 91
7.7 Test cases used to evaluate UC-5: Superimposed PPR Model Analysis - Capacity Utilization. 95
7.8 TC-5.1 - Planned production volume of Water Filter Variants 95

List of Algorithms

6.1 Model Difference Algorithm for MVA Models, inspired by [Rubin and Chechik, 2012, Brun and Pierantonio, 2008] 50

6.2 Similarity Heuristic of Model Difference Algorithm for MVA Models . . 51

6.3 PPRVM-FCI Add Variants (PPRVM-FCI-AV) inspired by [Boubakir and Chaoui, 2018] 60

Acronyms

API Application Programming Interface. 61

AS Assembly Sequence. 11, 27, 32, 35

CP Construction Primitive. 54, 55, 57, 58, 84, 86, 87, 110

CPPS Cyber-Physical Production System. xi, xiii, 1–6, 8–10, 12, 15, 18–21, 23, 27, 29, 36, 37, 41, 60, 101, 104, 107–109, 116

CPS Cyber-Physical System. 9

CSV Comma-separated values. 61

DSL Domain-Specific Language. 25, 39, 74, 110

EMF Eclipse Modeling Framework. 13, 14, 18

FC Feature Candidate. 29, 30

FCI Feature Candidate Identification. 55

FODA Feature-Oriented Domain Analysis. 16

FPD Formalized Process Description. 12, 16, 21, 48, 66, 99, 108, 110, 121

IDE Integrated Development Environment. 14

JSON JavaScript Object Notation. 14, 61

MDA Model Difference Analysis. 37, 40, 41, 44, 47, 67, 73, 99, 101, 108, 110

MDE Model Driven Engineering. 11, 13

MDRE Model Design and Review Editor. 14–16, 22, 25, 26, 53–57, 62, 63, 65, 66, 100, 108, 109, 117

MVA Model Variant Analysis. xi, xiii, 7, 8, 19, 20, 22, 23, 25, 37, 40, 41, 43, 44, 47, 48, 53, 65, 66, 69, 72, 73, 81–83, 90, 94, 96, 99, 100, 108–110, 118

PPR Product-Process-Resource. xi, xiii, 1, 5–8, 12, 14–16, 18–23, 25–34, 36, 38–41, 47–50, 53–58, 60, 61, 66–68, 73–77, 79, 81–91, 94–96, 99–102, 104, 105, 108–110, 118, 119, 121, 122

PPR AS Product-Process-Resource Assembly Sequence. 18, 41, 50, 51, 53

PPR-FCI PPR Feature Candidate Identification. 55

PPRVM PPR Variability Modeling. 20

PPRVM-ANALYSIS PPRVM Analysis Framework. 7, 23, 61–63, 100, 104, 105, 108, 109, 118

PPRVM-FCI PPR Model Feature Candidate Identification. 53, 81–83, 90, 102, 105, 110, 118

PPRVM-FCI-AV PPRVM-FCI Add Variants. 7, 22, 23, 47, 53, 60, 83–90, 100–102, 104, 105, 108, 118, 119, 123

PPRVM-FCI-DV PPRVM-FCI Derive Variants. 7, 23, 47, 61, 100, 102, 104, 105, 108–110

PPRVM-MDA PPRVM Model Difference Analysis. 7, 20, 22, 23, 47, 50, 52, 53, 57, 67, 72, 73, 82, 100–102, 108, 117

PSE Production Systems Engineering. 3

RAMI4.0 Reference Architecture Model Industrie 4.0. 10, 19

RUCM Restricted Use Case Modeling. 26, 27, 30, 32, 33, 35, 68–70, 91, 92, 95

SQA Software Quality Assurance. 32

UML Unified Modeling Language. 8, 11, 13, 18, 25, 41, 67, 72, 74, 79, 81, 83, 100, 118

UR Utilization Ratio. 63, 64, 96, 98, 121

VME Variability Modeling Editor. 22, 61, 65, 66, 68–70, 72–83, 85–88, 90–92, 94–98, 100, 102, 104, 109, 110, 118, 119, 121

YASA yet another sampling algorithm. 61, 92, 102, 104, 118