Model-Based Software Derivation for Industrial Automation Management Systems

Thesis approved by
the Department of Computer Science
Technische Universität Kaiserslautern
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)

to

Miao Fang

Date of Defense:	July 15, 2019
Dean:	Professor Dr. Stefan Deßloch
Reviewer:	Professor Dr. Dr. h.c. Dieter Rombach
Reviewer:	Professor Dr. Klaus Schmid

D 386

Abstract

The systems in industrial automation management (IAM) are information systems. The management parts of such systems are software components that support the manufacturing processes. The operational parts control highly plug-compatible devices, such as controllers, sensors and motors. Process variability and topology variability are the two main characteristics of software families in this domain. Furthermore, three roles of stakeholders – requirement engineers, hardware-oriented engineers, and software developers – participate in different derivation stages and have different variability concerns. In current practice, the development and reuse of such systems is costly and time-consuming, due to the complexity of topology and process variability.

To overcome these challenges, the goal of this thesis is to develop an approach to improve the software product derivation process for systems in industrial automation management, where different variability types are concerned in different derivation stages. Current state-of-the-art approaches commonly use general-purpose variability modeling languages to represent variability, which is not sufficient for IAM systems. The process and topology variability requires more user-centered modeling and representation. The insufficiency of variability modeling leads to low efficiency during the staged derivation process involving different stakeholders. Up to now, product line approaches for systematic variability modeling and realization have not been well established for such complex domains.

The model-based derivation approach presented in this thesis integrates feature modeling with domain-specific models for expressing processes and topology. The multi-variability modeling framework includes the meta-models of the three variability types and their associations. The realization and implementation of the multi-variability involves the mapping and the tracing of variants to their corresponding software product line assets. Based on the foundation of multi-variability modeling and realization, a derivation infrastructure is developed, which enables a semi-automated software derivation approach. It supports the configuration of different variability types to be integrated into the staged derivation process of the involved stakeholders.

The derivation approach is evaluated in an industry-grade case study of a complex software system. The feasibility is demonstrated by applying the approach in the case study. By using the approach, both the size of the reusable core assets and the automation level of derivation are significantly improved. Furthermore, semi-structured interviews with engineers in practice have evaluated the usefulness and ease-of-use of the proposed approach. The results show a positive attitude towards applying the approach in practice, and high potential to generalize it to other related domains.

Contents

List of Figures

List of Tables

1
Introduction

"A journey of thousand miles begins with a single step."

–Laozi

The original motivation of this thesis was based on the derivation challenges observed in the industrial automation domain from a practical viewpoint. In order to help practitioners in this domain to improve the reuse during development, this thesis presents a model-based derivation approach, which integrates multiple variability types, such as features, processes and topology, into a staged derivation process. Compared to the prior approaches, the main benefits of using such an approach presented in this thesis include both the improvement of stakeholders' satisfaction during their work and the automation of derivation steps to achieve better efficiency.

This chapter introduces the background and motivates the research topics of this thesis. It presents the overall contributions of the thesis, as well as an outline of the following chapters.

1.1. Context

Software reuse is of interest because software organizations tend to build systems that are more complex, more reliable, less expensive, and that are delivered on time [FK05]. To achieve systematic software reuse, software product line engineering (SPLE) has been proposed as a set of methods and techniques for managing and maintaining core assets and systematically reusing them during the development of new software products [HP03]. Since approximately two decades, a wide variety of companies and organizations have achieved substantial adoption of SPLE with reported benefits, such as reduced development costs, improved quality and shortened time-to-market [Bos02].

The overview of SPLE processes and activities is presented in Figure 1.1. The two main processes in SPLE are family engineering (or domain engineering) and application engineering [HP03]. Scoping is fundamentally important to decide which features and assets should be covered by family engineering or be developed in application engineering [DS99]. During family engineering, commonalities and variability of a product family are identified. Reusable artifacts are developed and managed as core assets, such as requirement scenarios, reusable architectures and reusable components. During application engineering, customer-specific software products are developed based on the assets established in family engineering [HP03, Mut02].

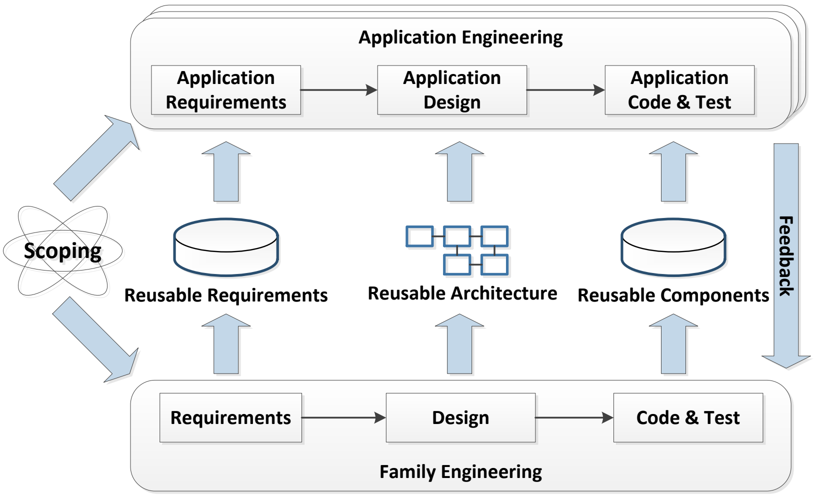


Figure 1.1.: Overview of Activities and Processes in SPLE [HP03]

1.1.1. Software Product Line Engineering Background

In SPLE family engineering, variability expresses the assumptions about how members in a software family may differ from each other [HP03, WL99]. Modeling variability is essential to define and manage commonalities and variability [CGR$^+$12]. Many existing variability modeling techniques have been proposed in academia, for example feature modeling, decision modeling and orthogonal variability modeling [CABA09]. Among them, feature modeling has gained importance especially in practice, since it is intuitive to understand and is suitable to express both commonalities and variability [CGR$^+$12]. Many family modeling approaches have originated based on Feature-Oriented Domain Analysis (FODA), proposed by Kang et al. [KCH$^+$90]. Furthermore, in application engineering, variability models empower the software product derivation [HP03]. During product derivation, the variability models are configured by the software engineers or involved stakeholders. The variants in the models are selected, configured and bound, to fulfill the customers' specific requirements.

Although many product derivation approaches have been proposed in the last two decades, the derivation in SPLE remains being an expensive and an error-prone activity, so that it is still hard to automate and support by tools [ROR11]. As O'Leary et al. pointed out: if a product derivation approach is supposed to be used in many different domains, it may become too generic that impedes the adoption in practice [ORRT09]. Instead, approaches and tool support should focus on the needs of the domain context and involved stakeholders to be adopted in practice [ORRT09].

1.1.2. System Background

In the industrial automation domain, systems are commonly modeled as an automation pyramid to structure the different types of applications, as presented in Figure 1.2 [ISA, IEC]. The top level appearing in the pyramid is the enterprise level. The systems on this layer manage the business-related or strategic activities of the manufacturing tasks [ISA]. Enterprise Resource Planning (ERP) systems, for example, are typical for systems on this level. They provide solutions for business process integration within or across organizations. The second level is the management level, which includes the systems of manufacturing operations management (MOM). They manage the creation, production and distribution of end products. For instance, manufacturing execution systems, laboratory information management systems, warehouse management systems, or computerized maintenance management systems are in this level [McC97, ISA]. Below MOM, the supervision level contains the systems for monitoring and supervision, such as Supervision Control and Data Acquisition (SCADA) systems. SCADA systems interoperate with the systems in control level to manipulate the physical manufacturing processes. The lowest level contains physical devices to perform concrete tasks. The sensors, actuators and embedded systems are on this level [ISA].

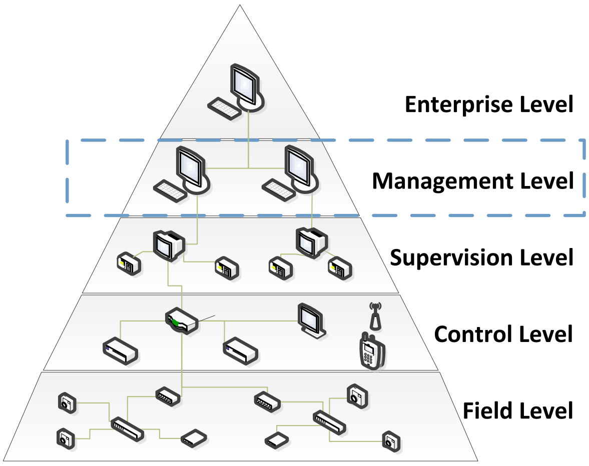


Figure 1.2.: The Industrial Pyramid

The type of software systems that this thesis focuses on is on the management level. In general, many systems on this level are information systems that manage manufacturing processes. They also coordinate numerous distributed hardware components, as well as human workers in case not all manufacturing tasks are automated. The management parts are software components and services that support the manufacturing processes. The operational parts control and communicate with highly plug-compatible devices, such as controllers, sensors and motors. Based on this understanding, the definition of the targeted software systems in the context of this thesis is given as following:

Industrial Automation Management (IAM)
The industrial automation management systems are the information systems of manufacturing operations managements that are used for the creation, development, production and distribution of products and services [ISA, IEC].

1.2. Motivation

This section shortly presents the problems observed in practice to motivate the whole thesis. A more complete problem formulation will be in Chapter 2. In Figure 1.3 [SIE], a product family of IAM among three customers is illustrated. As can be seen, three customers share some common hardware devices. The software systems managing the factories share also commonalities among customers. The difference is that *Customer A* requires integrating the IAM with ERP systems. *Customer B* and *Customer C* have no integration requirement, but they use different communication mechanism inside their factories.

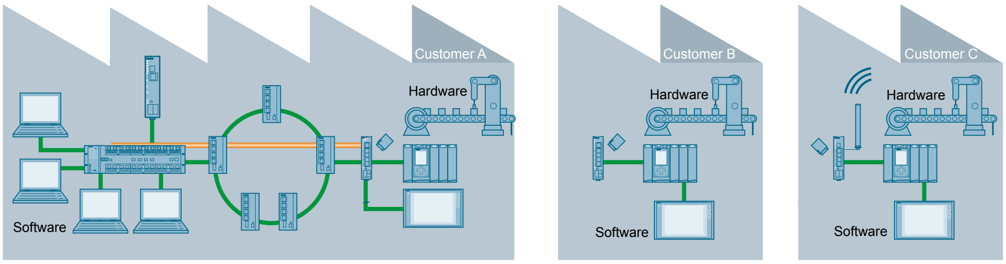


Figure 1.3.: Commonalities and Variability in IAM Product Families

The requirements among customers of IAM share considerable commonalities, as shown in Figure 1.3. With the success and benefits of SPLE adoption reported in the last two decades, the initial idea to promote reuse of IAM systems was to bring variability modeling of SPLE into IAM. However, the domain complexities, the process orientation and topological hardware configuration, cause the difficulty of variability representation, and further hinder the configuration during software derivation [FLED13b].

Furthermore, the development and derivation of IAM involves people from different backgrounds, such as requirement engineers, hardware-oriented engineers and software developers. These stakeholders have different focus and concerns, but they need to work together to deliver the final systems to customers. Up to now, the development of such systems cannot reach a satisfactory level of reuse. Especially, for some complex components handling business logic involving process and topology variability, the development still follows the clone-and-own approach to achieve ad-hoc reuse. Therefore, a systematic reuse approach is desired by practitioners to improve on this situation [FLED13b, FLE+15].

1.3. Thesis Contribution

This thesis aims at developing an approach for software product derivation of IAM systems. The solution idea to achieve this goal is to involve domain-specific modeling techniques into variability representation during family engineering. During application engineering, the domain-specific models can then be utilized as configuration mechanism to achieve better derivation and development efficiency. The solution outline is presented in Figure 1.4, in which the core elements of the approach as the core contributions of this thesis are highlighted:

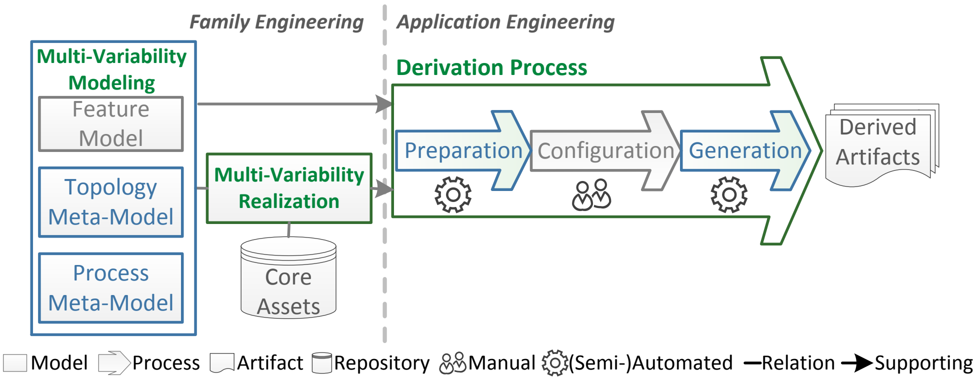


Figure 1.4.: Overview of Multi-Variability Derivation Approach

- *A multi-variability modeling framework* represents feature, topology and process variability types with configuration tooling. The framework includes also the associations among the variability (meta)-models.
- *The multi-variability realization techniques* link the variability modeling elements with the software core assets. The techniques enable the implementation of variability in both architecture and code derivation.
- *A semi-automated approach for product derivation* takes the advantages of the establishment mentioned above during application engineering. The derivation process, as presented in Figure 1.4, can be further split into three sub-processes: 1) preparation supported by variability model instantiation; 2) configuration supported by variability modeling editors; and 3) automated code generation.

The validation of the proposed approach in this thesis has two parts. The first part is a case study based on a "slice-of-life" example of an industrial IAM system, to test the feasibility, to characterize the variability models, and to evaluate the level of automation during derivation. The second part of the validation contains semi-structured interviews involving domain experts from practical viewpoints, to evaluate users' perceived usefulness and ease of use.

1.3.1. Research Process with Scientific and Practical Benefits

Figure 1.5 shows the research process followed in this thesis, and the expected benefits at the scientific and the practical level.

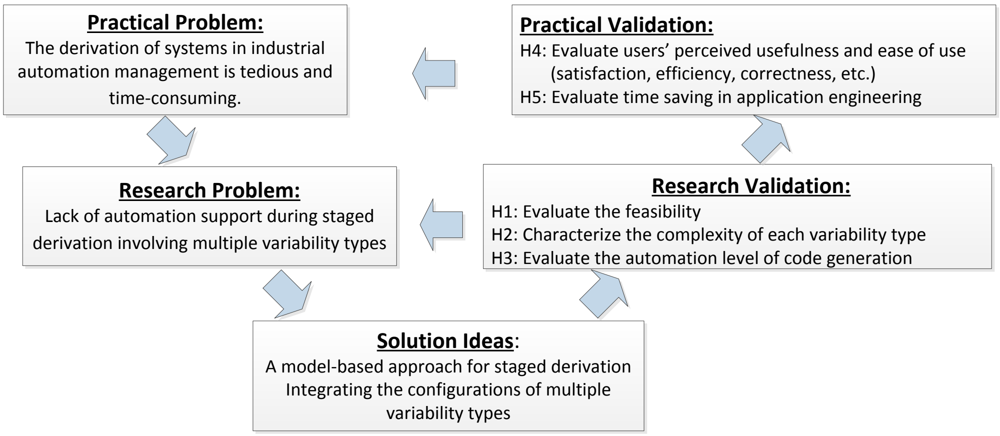

Practical Problem:
The derivation of systems in industrial automation management is tedious and time-consuming.
Practical Validation:
H4: Evaluate users' perceived usefulness and ease of use (satisfaction, efficiency, correctness, etc.)
H5: Evaluate time saving in application engineering
Research Problem:
Lack of automation support during staged derivation involving multiple variability types
Research Validation:
H1: Evaluate the feasibility
H2: Characterize the complexity of each variability type
H3: Evaluate the automation level of code generation
Solution Ideas:
A model-based approach for staged derivation Integrating the configurations of multiple variability types

Figure 1.5.: The Research Process

At the research level, benefits and applicability of the proposed approach, shown in Figure 1.4, have been evaluated according to the following hypotheses:

- *Hypothesis 1 (H1) – It is feasible to apply the approach in family engineering.* Implementing and using the approach in practical settings should be practically feasible, which means that the required effort on family engineering of the proposed approach is reasonable to afford by derivation stakeholders. It is expected that the effort of getting more reusable assets into core assets base should be reasonable.
- *Hypothesis 2 (H2) – The multi-variability modeling, by introducing process and topology models, can capture significantly more variability (>50%), compared to feature modeling.* The expectation is that by using additional topology and process variability models, instead of a general-purpose variability modeling technique (such as feature modeling), more variability in core assets can be modeled, so that reuse during application engineering can be improved.
- *Hypothesis 3 (H3) – The approach improves the code generation rate significantly (>35%), compared with the derivation without using the approach.* With

the developed variability modeling mechanism and the established assets, the proposed approach is expected to derive more source code automatically, to reduce the effort of development in application engineering.

At the practical level, the proposed approach is expected to help practitioners to improve their derivation and development tasks. With the proposed approach, it is expected to shorten the time-to-delivery for the development of customer-specific application. Therefore, the benefits to practical users should be assessed by:

- *Hypothesis 4 (H4) – The approach in application engineering can be easily accepted by users for their derivation tasks.* The developed approach in this thesis is expected to be easily accepted by users from a practical viewpoint. For this hypothesis, the metrics of perceived usefulness and ease-of-use aspects are assessed.
- *Hypothesis 5 (H5) – Applying the derivation approach in application engineering can shorten 1/3 of time spent on application engineering.* It is expected that when applying the proposed approach, the stakeholders can improve their efficiency, so that it can shorten the time spent in application engineering.

1.3.2. Assumptions and Limitations

The approach presented in this thesis with the contributions and benefits mentioned above is based on the following assumptions:

1. *The software organizations that intend to use the approach have existing knowledge, experiences and existing artifacts of the software family.* The rationale to have this assumption is that the approach depends on the development of domain-specific modeling and core assets of the software family. This assumption is reasonable to have, since introducing SPLE to a software family is usually not done in an unfamiliar domain of software organizations. Commonly, the decision of adopting SPLE is made, when software organizations have accumulated certain domain knowledge, and when they want to achieve better productivity or shorten the time-to-delivery.
2. *To adopt the approach proposed in this thesis, the extractive manner is encouraged.* Krueger classifies three SPLE adoption models [Kru02]: (1) The proactive strategy is to design and implement a complete software product line with the necessary infrastructure to support the full scope of products needed on the foreseeable horizon; (2) The reactive strategy is to incrementally grow the product line infrastructure when the demand arises; (3) The extractive strategy is to develop software artifacts by extracting the common and varying source code based on existing artifacts. This assumption is related to the first assumption, since the proposed approach acquires the reusable artifacts, such as requirements documentation, software architecture and source code from re-engineering of existing artifacts. However, it does not restrict the approach to be only adopted

in an extractive manner. Applying it with proactive and reactive strategies is also possible, in case the development team is able to design the domain variability models, the corresponding reference architecture, as well as the foreseeable code artifacts.

3. *The derivation approach only focuses on the complexities of the management level of industrial automation (see Figure 1.2); Complexities, such as interoperability or data variability during sub-system interactions, are excluded from consideration.* Firstly, the scope of this thesis has to be limited to keep the thesis manageable. It is not possible to take into account all types of variability existing in real industrial settings. Secondly, sub-system complexities usually can be abstracted from the lower layers of the industrial pyramid (e.g., using an interpreter pattern or adapters). For this reason, complexities on field, control, or supervision level are not considered in the proposed approach.

1.4. Thesis Outline

The thesis is organized and aligned with the research process shown in Figure 1.5. This section shows a short summary of each chapter.

Chapter 2 - Foundation and Problem Formulation: The chapter presents the characteristics of IAM systems with a running example. This example is used then in the rest of the thesis to illustrate the problems with corresponding solutions in depth. The state of practice of product derivation is analyzed. The three roles of stakeholders with their variability concerns are analyzed in this chapter. Based on the practical problem, Chapter 2 presents the analysis about how product derivation of IAM systems should be supported in terms of user-centered variability modeling and software derivation automation.

Chapter 3 - State of the Art and Limitation Analysis: The chapter collects and analyzes the existing product derivation approaches. It reports a systematic literature review (SLR) targeting the most recent publications between the years 2008 and 2015, extending the results of an existing SLR covering the years 1996 to 2007. It presents, how far existing approaches can fulfill the needs of product derivation in IAM systems. The shortcomings of the existing approaches build the foundation of the research questions of the thesis and the solution ideas in the next chapters.

Chapter 4, Chapter 5 and Chapter 6 propose the three main components of the derivation approach for IAM systems:

- *Chapter 4 - Hierarchical Multi-Variability Modeling*: Following the meta-modeling architecture of Meta-Object Facility (MOF), the meta-models for feature, topology and process are proposed and associated in this chapter.
- *Chapter 5 - Multi-Variability Realization*: The chapter presents the implementation of the multi-variability, by tracing the (meta-)models to the technical

solution space. A taxonomy of reusable asset types using feature-like, topology and process variability modeling elements is proposed in this chapter as well.

- *Chapter 6 - Automation of Model and Code Derivation*: This chapter presents the architecture of the derivation infrastructure. Two automated steps are introduced to the derivation process: model instantiation and code generation.

Chapter 7 - Validation: The chapter reports on the validation of the derivation approach. The goal-question-metric method is used to trace the overall goal of this thesis down to metrics for validation purposes. The chosen evaluation methodologies consist of two parts: a case study and semi-structured interviews. The chapter also includes the experiences and lessons learned during the validation.

Chapter 8 - Conclusions and Further Research Directions: The chapter concludes the thesis including the results, the contributions, as well as the limitations of adopting the approach in practice. Furthermore, it points out the potential future research directions in the area of domain-specific modeling and product derivation in software product line engineering.

1.5. Supporting Publications

The publications supporting this thesis are listed as following, as well as the related content in corresponding chapters.

1. Miao Fang, Georg Leyh, Joerg Doerr, and Christoph Elsner. Multi-variability modeling and realization for software derivation in industrial automation management. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 2–12, Saint Malo, France, October 2016. ACM.
    - Supporting the hierarchical multi-variability modeling in Chapter 4
    - Supporting the multi-variability realization in Chapter 5
    - Supporting the validation with a case study in Chapter 7
2. Miao Fang, Georg Leyh, Christoph Elsner, Joerg Doerr, and Jingjing Zhao. Towards model-based derivation of systems in the industrial automation domain. In *Software Product Line Conference (SPLC)*, pages 283–292, Nashville, USA, July 2015. ACM.
    - Supporting the derivation process in Chapter 6
    - Supporting the validation with semi-structured interviews in Chapter 7
3. Miao Fang, Georg Leyh, Christoph Elsner, and Joerg Doerr. Experiences during extraction of variability models for warehouse management systems. In *Software Engineering Conference (APSEC)*, pages 111–116, Bangkok, Thailand, December 2013. IEEE.

- Supporting the practical problem in Chapter 2

4. Miao Fang, Georg Leyh, Christoph Elsner, and Joerg Doerr. Challenges in managing behavior variability of production control software. In *International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, pages 21–24, San Francisco, USA, May 2013. IEEE.
    - Supporting the variability modeling challenges in Chapter 2

2
Foundations and Problem Formulation

"Successful problem solving requires finding
the right solution to the right problem."
–Russell Lincoln Ackoff

The chapter serves as the foundation of this thesis. At the first place, it introduces the terms and definitions focusing on application engineering activities of software produce line engineering. At the second place, this chapter presents the practical problems with a running example, which shows that the current development of systems of industrial automation management (IAM) is tedious and time consuming. The difficulty of using software product line approaches in IAM is that these systems do not only have feature-like variability, but also topology and process variability. Moreover, three roles of stakeholders – requirement engineers, hardware-oriented engineers, and software developers – participate in different development tasks and derivation stages. They have their particular variability concerns. To help these stakeholders, a systematic product line approach is necessary to support (1) representing multiple variability types, and (2) automating the derivation process by integrating different variability models.

2.1. Foundations and Definitions

According to Clements and Northrop [CN02], the term, "software product line", is "a set of software-intensive systems sharing a common, managed set of **features** that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way". As one of the essential concepts, this section starts with the notion of feature.

2.1.1. The Notion of Feature

From existing literature, a feature can be defined as "a logical unit of behavior specified by a set of functional and non-functional requirements" [Bos00] or as "a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems" [KCH+90]. In fact, the understanding of what a feature is varies significantly from people to people [BLR+15]. Berger et al. argue that features do not necessarily address functional or non-functional concerns that end up in a software product. Instead, features can be used atypically, for example to support internal development planning [BLR+15]. Furthermore, features have certain architectural responsibility, which means that some features can be located at a certain "place", for example in a component or in a unit of implementation-level software assets. Some other features can be crosscutting, which means that they scatter over several parts of software components or artifacts. Feature can represent requirements, architectural elements, parameters or non-functional requirements [BLR+15]. Considering these existing definitions, feature in this thesis is defined as:

Feature:
A feature is an abstraction of a characteristic of a software family, which represents a comprehensible aspect of a company's business [SVGB05, BLR+15].

The reason to define the feature in this way is that the IAM systems are highly influenced by real-world hardware configurations. In this case, features refer to a graspable aspect that can be easily understood by domain experts. Such features can be crosscutting. This means that the related artifacts may be scattered throughout the software implementation.

2.1.2. Classification of Software Product Lines

Bosch distinguishes four main maturity levels of software product lines [Bos02]. Table 2.1 summarizes these four levels with the efforts needed in family engineering (FE), application engineering (AE), and typical assets of reuse [Bos02].

Name	FE Effort	AE Effort	Typical Core Assets
Standardized infrastructure	Very low	Very high	Operating systems, database
Platform	Low/medium	High	A shared platform
Software product line	High	Low/medium	A product line architecture
Configurable product base	Very high	Very low	An automated SPL infrastructure

Table 2.1.: Software Product Line Maturity Levels

Firstly, the standardized infrastructure is the level with minimum possibility of reuse. Standardized tools or commercial software are the commonalities shared among different products, such as operating systems or databases. Secondly, at the platform level, the development teams develop a platform to include all common functionality of their software domains. The developed platform becomes the core assets of reuse. Thirdly, at the product line level, when functionalities are shared by a sufficient number of products, their artifacts can be included as software product line assets. Managing the variability of the shared artifacts becomes critical[1]. Reference architecture of the software family is a typical asset shared among different products. Finally, the configurable product base has the highest maturity. At this level, all variation points with all variants can be explicitly modeled and predicted, so that product derivation can be fully automated during application engineering[2] [Bos02].

Besides the overall SPL maturity, within one software family, individual artifacts may also have different maturity levels [Bos02]. For example, a component for data persistence may have higher maturity than a component handling user requests, because the latter has more unpredictable requirements.

However, the highest maturity level is not always desirable, because the costs invested in the product line infrastructure need to pay off. Depending on the number of new products, the possibility to anticipate the requirements of new customers, and the stability of the application domains, the development organizations should decide an optimal maturity level with business considerations. In contrast to the configurable product base, Adam defines a concept of a flexible software product line as [Ada12]:

Flexible Software Product Line:
A flexible software product line is a product line, for which it is either not possible or not economic during family engineering to explicitly anticipate most requirements that may possibly occur in application engineering projects [Ada12].

According to the reference [Ada12], software product lines of information systems particularly belong to the category of flexible software product lines, since the systems need to support customer-specific business processes, and need to integrate with other legacy software systems [Ada12]. In flexible product lines, upgrading to higher maturity levels may be not possible to achieve, due to the instability and variability of the domains. It is also possible that the cost of upgrading the maturity level of a software product line is too high, so that the return on investment is insufficient from an economic perspective. Besides these aspects, the circumstances of development teams and their readiness can also influence the product line maturity level [Bos02].

The IAM systems belong to the category of flexible software product lines. They are responsible for management in manufacturing factories (see the definition of

[1] Level 2 and Level 3 may result in product populations, in which the variability does not only settle in components but also in the composition of components. In this case, the derived products may employ totally different configurations [Bos02].

[2] Level 3 and Level 4 may result in a program of product lines, especially for large and complex software systems. The challenges in this situation are, for example, to manage the large amount of variability and the dependencies among them [Bos02].

IAM in Section 1.1.2). These systems are information systems, which manage manufacturing processes in factories. They also communicate with highly plug-compatible devices, such as controllers, sensors and motors. Based on the current observation in practice, many IAM software families are at low maturity levels, that is, they may share standardized infrastructure or common platforms. Industrial practitioners and researchers are looking for solutions of this type of software systems to improve the reuse of their existing legacy systems, and to increase the efficiency of product development [GSC15, WC15].

2.1.3. Variability and Variability Modeling

Software product line engineering distinguishes commonalities and variability. Commonalities are the software artifacts that are shared by all members of the product family. Variability is the "ability of a software system or artifact to be efficiently extended, changed, customized or configured to be used in a particular context" [SVGB05]. Variability is the core concept in software product line engineering, as application-engineering activities rely on the configuration of variability.

Variability Model:
Variability model represents and describes how the various assets in a product family may differ from one another [WL99].

Based on the configuration of variability models, the new software product can be derived to fulfill the customer requirements. Variability has its own life cycle, for example, to be identified, scoped, bound and resolved [Els11].

Many variability modeling techniques have been proposed and used in academia and practice [CABA09], such as feature modeling [KCH+90], decision modeling [CGR+12], orthogonal variability modeling [HP03] and the common variability language [cvl12]. Among all of the existing variability modeling techniques, feature and decision modeling have attracted most attention [CGR+12]. Feature modeling has relatively broad adoption in practice, because it models both commonalities and variability, and it seems to be more intuitive to understand, while decision modeling focuses more on variability. Commonly, feature modeling is used just like decision modeling in an orthogonal manner [CGR+12], which means that the variability model is separated to other artifacts in the product line. The variation points and the corresponding artifacts are related via trace links. There are existing commercial and academic tools to support variability modeling. For example, there are GEARS [Kru07b], pure::variants [pur], FeatureIDE [KTS+09] for feature modeling and DOPLER for decision modeling [RGD07].

Both feature models and decision models are able to describe similar variant types [CGR+12]. Feature models describe the *boolean* type as optional features; the selection of one out of several possibilities as alternative features. Decision models can model these as decisions. Both of them also support other primitive types, such as strings or integers. Figure 2.1 depicts these data types of feature models with minor adaptations, based on the feature meta-models proposed by [CGR+12, KCH+90]. In Figure 2.1, Timer is an optional feature. Database is for example a mandatory feature. SQL Server and Oracle are alternative features: one of them needs to be chosen during application engineering. For identification, both radio-frequency identification (RFID) and barcode scanners can be used, and at least one of them needs to be chosen.

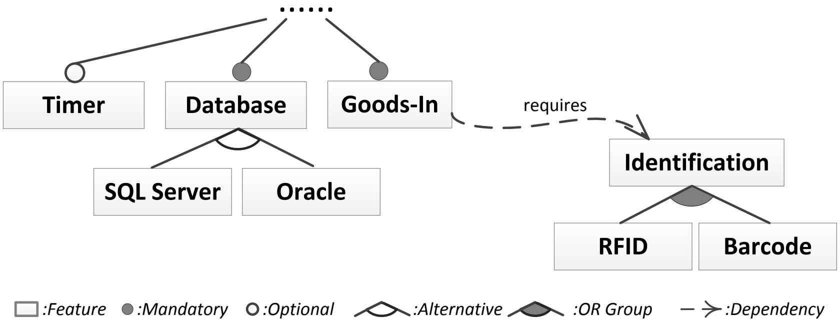


Figure 2.1.: Variants in Feature Modeling

Halmans and Pohl distinguish variability as essential and technical variability [HP03]. Essential variability is oriented to end-users and customers. Essential variability does not contain technical implementation details. When negotiating with customers, understanding customer requirements or documenting requirements, requirement engineers usually consider and resolve essential variability at the early phase of application engineering [HP03].

> **Essential Variability:**
> Essential variability describes the variability of a software family from a viewpoint of customers or users without implementation thinking [HP03, NE08].

Technical variability is oriented to implementation. Technical stakeholders are concerned with this variability, such as software architects and developers. Technical variability occurs mostly not at the very beginning of the application engineering processes. Technical variability often is resolved later than essential variability [HP03].

> **Technical Variability:**
> Technical variability describes the variability of a software product line from an implementation perspective [HP03, NE08].

To better illustrate the occurrence time of essential and technical variability according to [HP03], Figure 2.2 shows the application engineering process from left to right. At the early phase of application engineering, the development team receives customer requirements. At the beginning, more essential variability are concerned than technical variability. During negotiation and requirement engineering with customers, more and more essential variability should be decided. With the time, more technical decisions have to be made to finally decide and configure all variability.

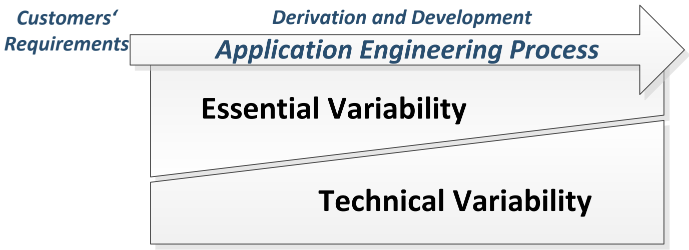


Figure 2.2.: Variability Concerns During Application Engineering Process

The IAM systems also have both essential and technical variability. In this domain, the variability communication is non-trivial, since the customers are from various manufacturing domains, such as food or car production. Essential variability as requirements is very challenging to collect and understand by software engineers. Inside the development organizations, the technical variability may also include both software and hardware aspects. This creates understanding difficulties that may cause significant communication overhead and may slow down the development. Section 2.2.3 will discuss this situation with more details.

2.1.4. Product Derivation

After the definition of the variability types, this section starts to introduce the configuration of variability during application engineering. Product derivation includes the core activities during application engineering. It addresses the construction of a customer-specific product based on the product line core assets [DSB05]. Thereby, we define product derivation as follows:

> **Product Derivation:**
> Product derivation includes the activities of selecting and configuring core assets to create the product to fulfill customers' specific requirements [HP03, ORRT09].

Rabiser et al. summarize key activities in product derivation, which are further categorized into three main steps [ROR11, ORRT09] : *Preparing for Derivation*, *Product Derivation* and *Application-Specific Development and Testing* [ROR11]. Based on this work, Elsner further separates *Product Derivation* into *Product Configuration* and *Product Generation* [Els11]. Figure 2.3 presents these four main steps of product derivation. In the context of this thesis, the last step, *Application-Specific Development and Testing* is not of focus, since it has less relation to the reuse of core assets.

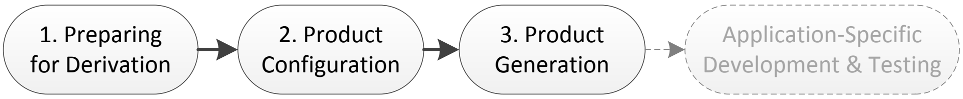

1. Preparing for Derivation
2. Product Configuration
3. Product Generation
Application-Specific Development & Testing

Figure 2.3.: General Steps in Product Derivation

Depending on the maturity of software product lines, at the end of the Step 3, the derived product can be a full product or a partial product that needs further application-specific development [ROR11, ORRT09]. Since the target IAM systems belong to the category of flexible product lines, the expectation of derivation results would be a partial product. The next three subsections will explain the details of Step 1, Step 2 and Step 3 respectively.

2.1.4.1. Preparing for Derivation

The most important task in preparing derivation is to understand the customers' needs and to translate their needs into internally understandable languages by development teams [ROR11]. In configurable software product lines, the "translation" is straightforward, because the domain is relatively mature and stable, so that variability can be well captured, modeled and predicted during family engineering. In flexible software product lines, in particular the targeted IAM systems, understanding customers is much more complicated, since IAM systems have the nature of information systems, as argued in Section 2.1.2. The requirements of IAM systems are process-oriented, and the desired products need to fit the customers' business environments. Depending on the manufacturing domains, understanding customers' requirements can be challenging, because it needs knowledge of customers' engineering disciplines.

In addition, derivation starts usually not from scratch. A base configuration is very helpful to have as a stepping-stone to start derivation [ROR11, ORRT09]. This base configuration is especially important for systems with a large number of variation points. To support it, a configuration with some pre-configured decisions or values should be automatically initialized to prepare for derivation. In software product line engineering, typical assets are requirement specifications, architecture, components and test cases (see Figure 1.1 in Chapter 1). To come up with the base configuration, the core asset base should be extended to include reusable configurations, as presented in Figure 2.4.

Figure 2.4 shows that, the variability model is a very important concept in software product derivation. Based on existing configurations, some reusable decisions and their combinations can be extracted and maintained as parts of the core asset base as well, because customers potentially share requirements. During product derivation for a new customer, a base configuration can be initialized to help on maximizing reuse. In other words, the configurations of variability models of existing customers

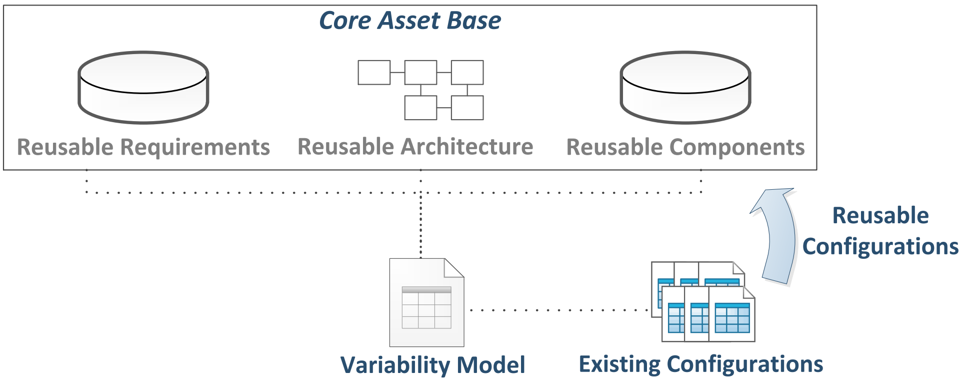


Figure 2.4.: Taking Reusable Configurations as Core Assets

can become kinds of templates to be reused as base configurations for new projects. This is especially beneficial for flexible product lines to improve derivation efficiency, as they may have an enormous amount of variants to configure.

2.1.4.2. Product Configuration

Product configuration is usually an iterative process [ORRT09]. The procedure of configuration can be considered as following the hierarchies of feature or decision models to resolve the variations.

> **Product Configuration:**
> Product configuration refers to the activity of a human stakeholder, who translates product requirements into a machine interpretable form by using tool support [Els11].

Many research papers propose support to help on product configuration activities. Figure 2.5 shows the different kinds of configuration support characterized by means of a feature model. Most of these “features” presented in this feature model are optional features, since the existing derivation approaches do not necessarily cover all kinds of configuration supports. Many existing research papers only cover a single or several of these features.

Stakeholder Support and *Configuration Verification* are the two major types of configuration support. The following gives a short description of the sub-features, together with related references in the state of the art:

- *Separation of views*: Hubaux et al. formalize and implement an approach to support separation of concerns for feature-based configuration [HHSD10, HHS$^+$13]. With this approach, users during configuration get customized views of variations that they should focus on. Mannion et al. also propose to use

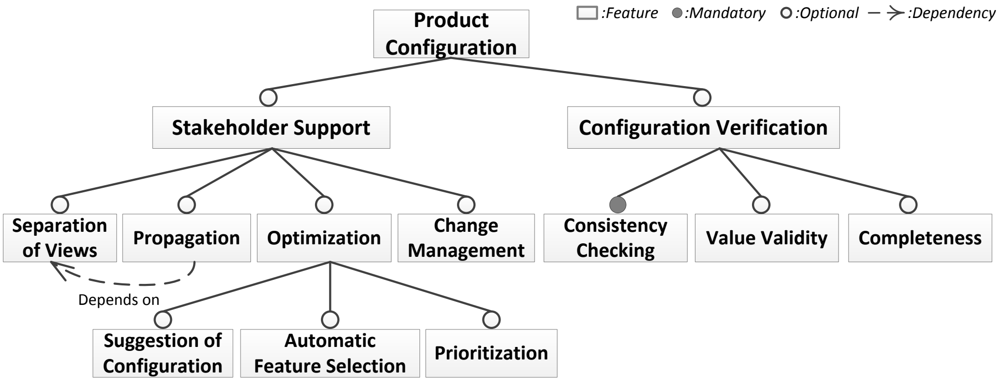


Figure 2.5.: Features of Product Configuration Support

multiple viewpoints to capture the needs of different stakeholders to avoid that they face the complexity of a large feature model [MSA09]. Other works closely related to separation of concerns include approaches on multiple-staged derivation [CHE04, Els12]. They provide not only separated views, but also sequentially ordered views according to variability binding time during derivation.

- *Propagation*: During configuration, when one decision is made, or one feature is (de-)selected, the impacted decisions or features can be automatically navigated to by the configuration tools [RGL12]. The goal of developing such a function is to improve on configuration consistency [CNCJ14, UBFC14]. For complicated configuration processes, derivation stakeholders need to participate in a work-flow-like process, as the propagated views may have different users [AHH11, ACG$^+$12]. To support this function, separation of concerns of variability is usually required.
- *Optimization*: Rabiser et al. propose that the derivation tooling should provide a function to suggest configuration values [RGL12]. Similarly, Guo et al. propose an approach, which supports the automatic selection of features during configuration time to save user effort [GWW$^+$11]. Features are prioritized based on their selectivity of existing configurations of the product line, proposed by Chen and Erwig [CE11]. During configuration time, features with higher selectivity are brought to users earlier to optimize users' selection and configuration tasks [CE11]. Integrating such functions into the derivation tooling can improve the productivity of the derivation stakeholders.
- *Change management* [RGL12]: In practice, the configuration of complex systems is commonly an iterative process. The decisions sometimes are made by multiple roles of derivation stakeholders. For this reason, it is helpful to trace the changes of configurations during the whole configuration process, in order to maintain the modification history of decisions and values. Some additional functions can be also helpful to the derivation stakeholders, for example, to enable annotations to document the rationales of a decision.

- *Configuration Verification*: Most of the approaches in this category focus on error detection, to diagnose constraint violations or to ensure consistency [WBS+10, TBG13]. Error fixing can be done either automatically or manually by generating a report [ELSP11]. A few of the works in this category support also completeness checking [AHH11] or correctness checking of configured values [RGL12].

This thesis does not intend to fully cover all of these aspects shown in Figure 2.5. The goal to analyze the product configuration is to understand how good the configuration of variability can be supported by existing approaches and tooling. It is however notable, that most of the existing approaches only cover a single or a subset of these features. Among them only DOPLER [RGL12] has explicitly developed the user guidance functions aiming at covering majorities of these functions. As a commercial feature modeling tool, pure::variants has also considered usability to facilitate configuration. This is an indication that covering all of these functions with tool support is probably very difficult to achieve. On the one hand, it can be very useful to integrate the existing approaches and their tooling, to ease the configuration tasks. On the other hand, it might not be necessary to cover all these functions in a single tooling. The configuration and derivation of different software domains may require the tooling to be adjusted and adapted to fit to the needs of the derivation stakeholders.

2.1.4.3. Product Generation

According to the general derivation steps shown in Figure 2.3, the product generation takes the configurations as input to derive the software products.

Product Generation:
Product generation refers to the activity supported by automated tooling, which interprets the variability configuration into software artifacts, such as components, source code and configuration files.

A product generator uses the configurations as input and generates the expected software artifacts as output. The technologies, such as feature-oriented programming [BSR04, SBB+10], aspect-oriented programming [VBMD11, DPFSG13], and pre-processing embed such possibility already at programming level. In model-based engineering, the generation is actually a model transformation step, more specifically model-to-text transformation, in which the configured models are the input for generators. The code and configuration files are the output of the generators.

Configurable software product lines (see configurable product base in Section 2.1.2) have higher potential to achieve a high degree of product generation. However, for flexible product lines, the outcome of product generation can be derived software architecture, particularly a generated code structure, code fragments, or other non-code artifacts, such as requirement documents, derivation guidelines, or some intermediate models.

2.2. Practical Problem

From a practical viewpoint, the original motivation of conducting the research in industrial automation management (IAM) systems comes from the lack of satisfaction during development reported by software architects and developers in Siemens. The IAM product to deliver to customers is usually a whole solution for a factory. Such a solution includes not only software solutions, but sometimes also hardware systems, depending on whether the customer purchases hardware separately from other vendors. The IAM systems play both a role in implementing the customers' business and in managing the underling software and hardware systems. In this case, customer needs are not the only sources where requirements come from. The hardware decisions and restrictions may also have a strong impact on software implementation. Due to the expectation to understand and to cope with all these requirements and restrictions, software engineers feel highly challenged in this domain [FLED13a, FLED13b]. This situation results in a lack of satisfaction during their development work as we showed in [FLED13b, FLE^{+}15].

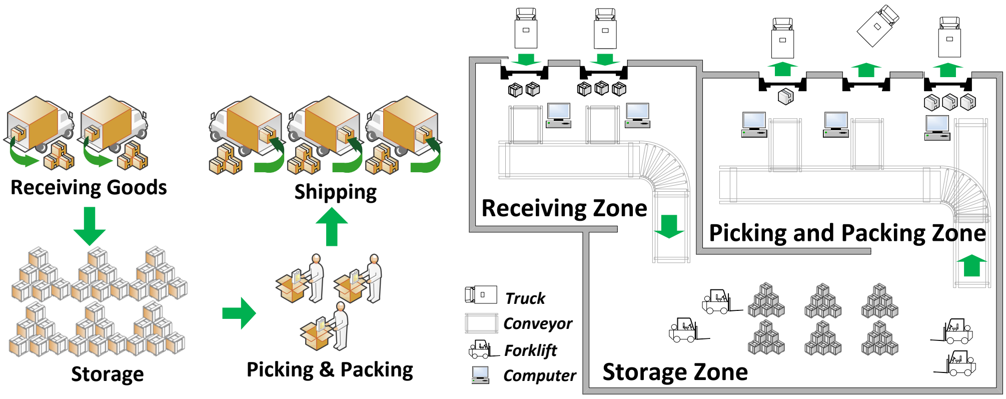
This section reports on the difficulties during development and derivation of IAM from two aspects: (1) the lack of derivation support when involving multiple variability types, and (2) the communication gap among stakeholders. The following sub-section provides the analysis of these two difficulties with a running example, which will be used throughout this thesis to illustrate the problems and the proposed solution.

2.2.1. Running Example

A software product family developed in Siemens for Warehouse Management Systems (WMS) was chosen to describe the running example, because it a typical sub-domain of industrial automation management. Manufacturers, importers, exporters and logistics businesses commonly use warehouses for the storage of goods or products. Warehouse management systems can be either stand-alone systems supporting logistics, or supporting other large-scale manufacturing factories, such as automobile and food production. When supporting other productions, WMS may receive orders from upper-level enterprise resource planning systems. They also control automatic and manual work flows inside warehouse plants.

Figure 2.6 shows the high-level features of WMS with an example of a factory layout. The arrows in Figure 2.6 show the material flows which are commonly used in the IAM domain to represent how manufacturing materials or goods are transported, stored, supplied and assembled in factories. Figure 2.6a presents the typical process-like functions in WMS, such as goods-in, storage, picking, packing and shipping. Figure 2.6b shows the zones of performing these tasks in a warehouse example.

Based on this running example, we see that IAM as management systems should be able to assign and allocate each of the tasks to its exact location for execution at run-time. The IAM systems are in charge of managing and coordinating these tasks, both from software and hardware perspectives. We further use the goods-in feature as an example to understand the systems' characteristics.



(a) High Level Features **(b)** An Example of a Factory Layout

Figure 2.6.: High-Level Features with Examples of Factory Layouts

Topology variability. For the goods-in feature, two customers require different configurations in the receiving zones in warehouse plants, as presented in Figure 2.7. On the left side, *Customer 1* has an automatic goods-in forklift and a manual goods-in workstation, whereas on the right side, *Customer 2* has at least three manual goods-in workstations. The topological configurations of the goods-in feature need to be addressed explicitly, during IAM development and derivation. As can be seen, various devices are installed at different locations of the plant floors. They are highly reusable and plug-compatible among customers. The variability is caused by the flexible combination of all these devices, since all the devices ideally can be wired directly or connected via communication networks.

Topology Variability:
Topology variability is the ability to combine, group and connect physical devices, which are related to a software family to fulfill a particular feature in a given zone [BSØ+14].

Process variability. Figure 2.8 illustrates the process variability related to the goods-in feature. It contains a material flow at the top part, and an informal process description shared among customers at the lower part. As can be seen, the processes for the same feature required by different customers share commonalities, and also comprise significant variability. For instance, Quality Control in Figure 2.8 contains variability that can be a optional process depending on the needs of customers. For automotive production and food industries, when and how to perform quality control can be completely different. Based on the analysis of existing requirement documents in practice, 50%-70% of requirement documents in IAM development are process descriptions, in both textual and graphical formats.

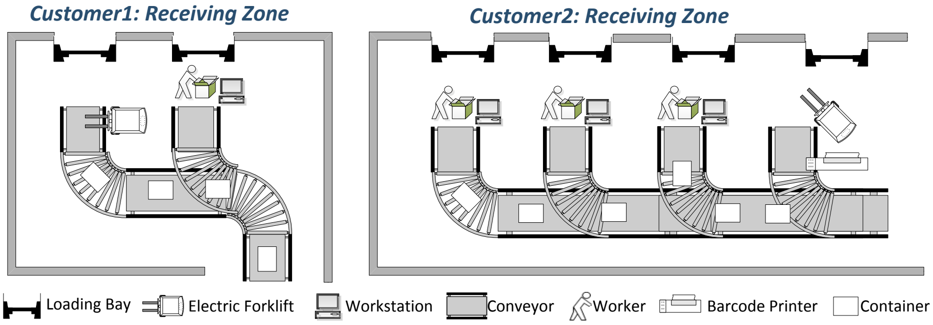


Figure 2.7.: Two Examples of Topology Configuration

Process Variability:
Process variability is the ability to configure and modify the execution sequences of process steps to fulfill a particular feature.

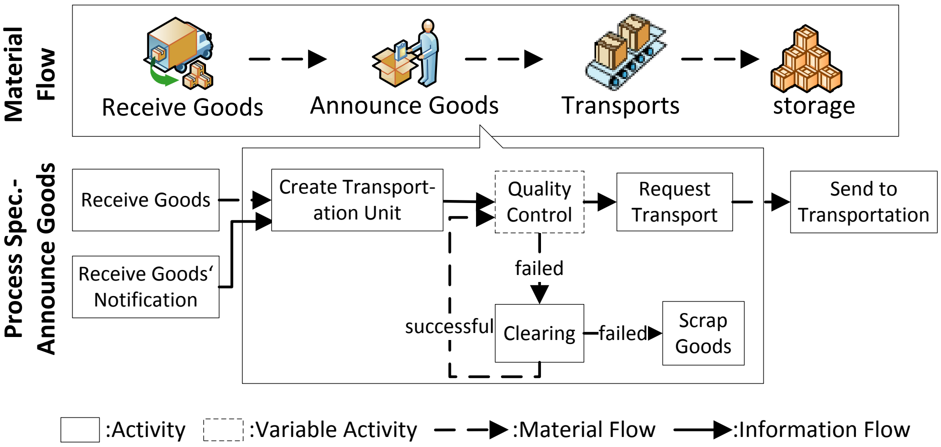


Figure 2.8.: Commonalities and Variability in Processes

The relationship between topology and process variability. Figure 2.9 shows an example of the relationships between some process steps and topological elements. The *Receive Goods' Notification* step has to be bound to the particular topological configuration of each customer project. This process can be triggered, only after the software system receives the notification of box arrivals at the corresponding locations.

To sum up, in this section, the goods-in feature has shown that the decision of a feature may impact the topology and processes configurations. The configuration of features could decide the inclusion or exclusion of topology and process variability models. Such relationships are explicit. Process variability should be bound to topology configuration during software derivation of the target software domain. The topology and process variability are domain-specific variability types. They lead to difficulties,

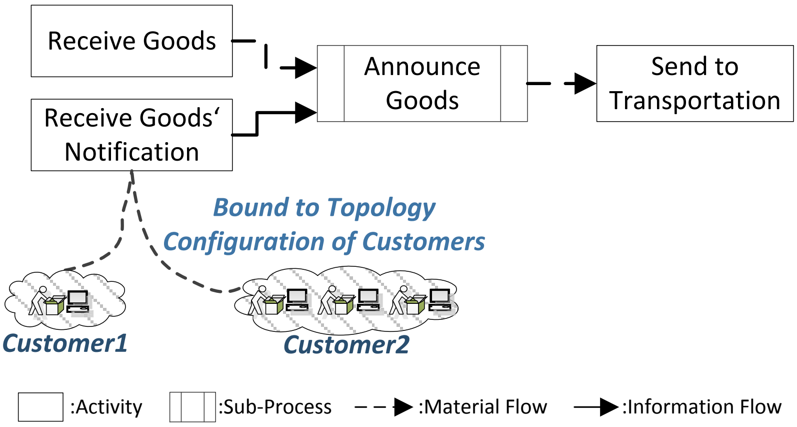


Figure 2.9.: Some Process Steps Are Bound to Topology Elements

when we try to use general-purpose variability modeling techniques to express them, as we pointed out [FLED13b]. There are possibly tacit relationships among variability models, such as *suggest* and *imply*. For example, a selected feature might imply the configuration of several other features, or even some process variability elements. These implicit relationships among variability models are difficult to give absolute semantics during family engineering [FFB02]. In the context of this thesis, the explicit relationships are of focus, especially the two domain-specific variability types. To come up with a systematic approach for the target domain, it is important to model the topology and process variability and their interrelationships.

2.2.2. Lack of Derivation Support Involving Multiple Stakeholders

Large-scale software product lines involve variability that cannot be managed centrally and evolved multiple teams that often work independently [Hol11]. The configuration of the variability in such large-scale systems often requires a staged derivation, where each stage eliminates some configuration choices [CHE04]. In IAM systems, three roles of stakeholders – requirement engineers, hardware-oriented engineers, and software developers – participate in a staged derivation process, as illustrated in Figure 2.10. These three roles may not only be distributed among people in one development team, but also among different departments or even different organizations. This section presents the analysis of the primary variability concerns of the three stakeholders.

According to the classification of Halmans et al. [HP03], in which essential and technical variability are defined (see Section 2.1.3), the main concerns of different stakeholders are summarized in Table 2.2.

Requirement engineers focus mostly on essential variability. At the beginning phase of this research, the concept of feature models with extracted domain features was presented to requirement engineers, which was reported in our work [FLED13b,

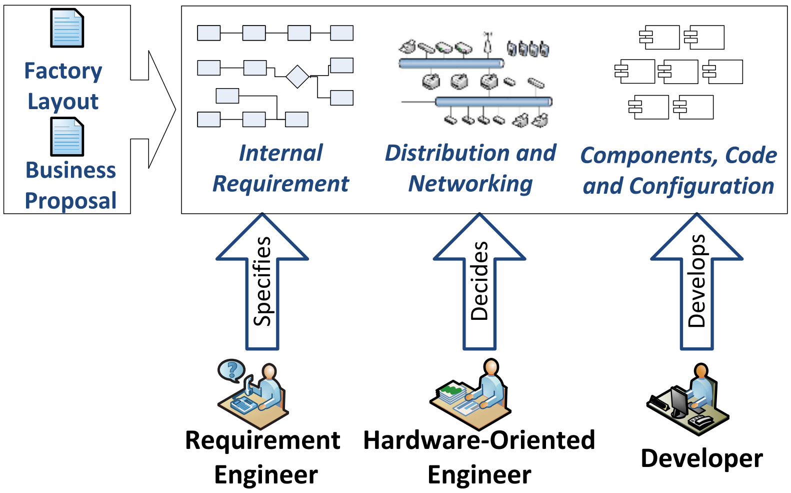


Figure 2.10.: The Staged Derivation in Practice

Role	**Variability in Target Domain**
Requirement Engineer	Features, Processes (Essential)
Hardware-Oriented Engineer	Topology (Technical)
Software Developer	Features, Topology, Processes (Essential, Technical)

Table 2.2.: Variability Concerns of Roles

FLE+15]. The feedback clearly indicates the importance of modeling the processes to support their work. Processes, as informal behavioral models, describe the behavior of systems that are closer to the material flows, which reflect directly what should happen in the real-world plants. Compared to configuring features, modeling the processes is more powerful to express the functional requirements in the viewpoint of requirement engineers [FLED13b]. To support their configuration tasks, a notation set for domain-specific processes is important to help them on modeling and expressing essential variability.

Hardware-oriented engineers are usually key engineers not only from hardware teams, but also from the requirement and development teams in practical settings. They join meetings together to communicate and decide the topology configuration. It is usually technical variability of a new factory, because the distributed hardware devices and the IT systems need to be integrated at the end to actually perform the manufacturing functions. For this reason, the role of hardware-oriented engineers serves as a “bridge” between the software and hardware teams. To support their work, domain topological models with additional grouping mechanisms are necessary to explicitly represent the layout of the systems [FLED13b]. In current practice, this information is not explicitly documented with proper details after the decisions has been made, which leads to significant communication overhead.

Software developers have both essential and technical variability concerns. Feature modeling can be used to trace variability to the components with high modularity level. However, in IAM systems, there are also many features that are driven by manufacturing processes. These components are process-oriented and topologically configurable. It is impossible to configure them with feature models [FLED13a, FLED13b]. Developers need a comprehensive derivation infrastructure, which integrates the different variability types during derivation time to facilitate automation of their tasks.

There are different strategies to separate configuration tasks by different stakeholders. Multi-staged derivation refers to the process of specifying and configuring variability of a software family member in stages, where each stage eliminates some configuration choices [CHE04, Els12]. Multi-staged derivation comprises, for example, development, compiling and setup time. To position the work of this thesis based on multi-staged derivation, the three roles of stakeholders – requirement engineers, hardware-oriented engineers, and software developers – participate mainly in the development stage, and their configuration choices and decisions of variability configuration can be considered as substages within the development stage. They face already variability configuration challenges in the target IAM domain.

2.2.3. Inefficiency during Communication and Development

Variability communication is non-trivial [HP03]. Figure 2.11 illustrates the communication links among stakeholders. The requirement engineers receive the requirements from two types of customers. Factory owners or managers are the first type of customers. They have high-level business concerns. Key users in factories are the second type of customers. They are very experienced technicians in factories, usually chosen by the factory managers to communicate with the IT development teams. Key users know clearly how material flows should be supported, and can tell very detailed requirements that the IT-systems should support, for example, the speed of a conveyor belt at a certain assembly station should be 0.2m/s. To understand both of these types of customers is challenging for requirement engineers, and they need to translate what the customers want to internal languages for further development and implementation.

However, even if requirement engineers could understand the customers well and convey the information as requirement specifications, there are still communication gaps among internal stakeholders, such as hardware-oriented engineers and software developers. The reason is that the three roles of stakeholders have also different concerns, as discussed in Table 2.2.

To understand the gap, we can still use the example shown in Figure 2.7. Customer 2 requires to have a feature *Attach New Barcode* to the automatic goods-in process. For each incoming box, a new matrix barcode shall be assigned for internal storage and optimization, whereas Customer 1 uses the common linear barcodes within the factory, which are already on the boxes. This feature for Customer 2 requires the installation of a barcode printer at the conveyor belt, as illustrated in Figure 2.7. The

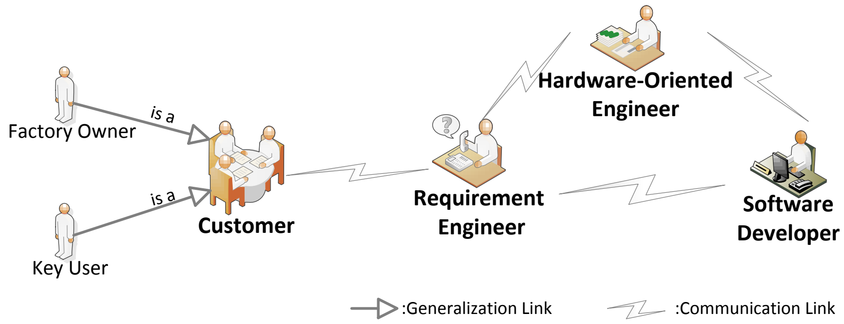


Figure 2.11.: Communication Among Stakeholders

three roles of stakeholders have their own perspectives to understand such a feature. The differences lead to further communication difficulties among them.

- **Requirement engineers** have essential variability concerns to specify the behavioral change to the systems. To attach the new matrix barcodes to boxes, their original linear barcodes are needed. For each box, the system shall retrieve the related order, assign a new barcode to the box, and communicate with the corresponding printer to print and attach it, as illustrated in Figure 2.12.
- **Hardware-oriented engineers** are concerned with the required topology configuration of hardware devices to support the feature. To implement the *Attach New Barcode* feature, for each of the automatic goods-in conveyors, a new barcode reader and a printer should be installed. They should be wired to the communication module so that they can interact with IAM systems to perform the tasks. The physical constraints of hardware devices have strong influences to the decision made by the hardware-oriented engineers, which could have relatively orthogonal dependencies software constraints. For example, choosing proper barcode readers for automatic goods-in conveyors might depend on the consideration response time, the material and costs.
- **Software developers** are responsible for developing the *Attach New Barcode* feature. They need to understand the requirements of the other two types of stakeholders and bring them together to develop the IAM systems. Within this barcode example, it requires several changes to components and their interactions. For instance, the Printer component is necessary to coordinate with the barcode printers. This component needs to know how many printers are installed in the factory and their communication ports in order to communicate with them. A change to the goods-out component is as well necessary, since the identification mechanism is changed from scanning linear barcodes to scanning matrix barcodes for the goods-out process as well, as shown in Figure 2.12.

The variability concerns of the three stakeholders have dependencies among one another, especially for software developers. They need to understand both of the sequence of

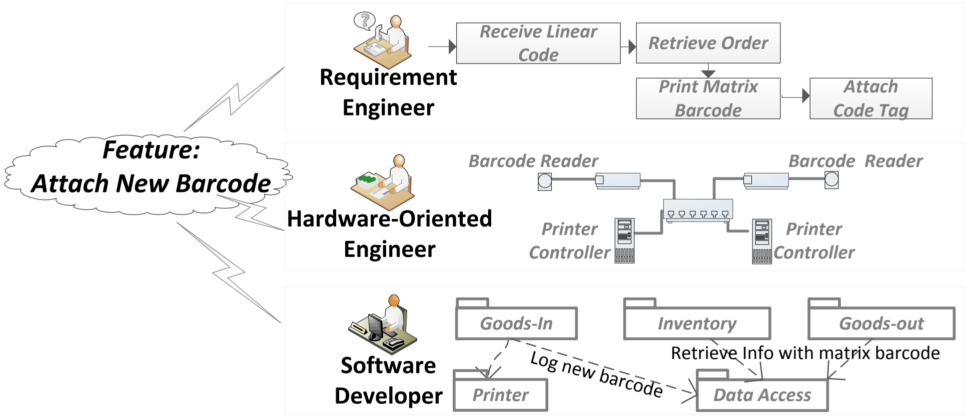


Figure 2.12.: Stakeholders and Their Different Concerns to Communicate

process steps defined by the requirement engineers, as well as the hardware configuration to enable the communication between the hardware and software components. Some process steps are bound to certain locations (see Section 2.2.1 as well). In this example shown in Figure 2.12, the *Print Matrix Barcode* has to be bound to the two *Printer Controllers* in the hardware configuration. Such information is very important for software developers to implement the *Printer* component. The complexity of the IAM domain determines that no single role of stakeholders may fully understand the whole systems. Currently in practice, the communication gaps among stakeholders cause low efficiency during development and derivation.

2.2.4. Summary of Practical Problem

We have seen that the IAM systems of different customers share a considerable amount of commonalities that can be beneficial to adopt software product line engineering. However, the characteristics of IAM, topology configurability and process orientation determine that two more variability types are necessary to be expressed and modeled. Three roles of stakeholders are identified, who participate in the derivation of IAM. They have different variability concerns during the software derivation process. This creates heavy communication overhead and low development efficiency as well. Consequently, the current development and derivation in practice is unsatisfactory for the stakeholders. Thus, we come up with the summary of practical problems in this thesis as:

Practical Problem:
In practice, the development and derivation of IAM systems commonly still follows a clone-and-own approach, which is tedious and time-consuming. An approach to facilitate the communication and improve derivation efficiency is lacking to help stakeholders to achieve systematic reuse.

2.3. Required Capabilities to Facilitate Product Derivation

With the identified practical problem, this section discusses how the software product derivation should be supported in IAM systems. In order to help the derivation stakeholders, it is essential to develop a derivation infrastructure to ease their work and to automate their tasks, where possible.

> **Derivation Infrastructure:**
> A derivation infrastructure consists of a number of technical tools facilitating derivation activities.

The derivation infrastructure refers to the technical tooling that allows stakeholders to configure and manipulate reusable software artifacts. To support product derivation in the context of this thesis, the derivation infrastructure needs to fulfill certain capabilities. The required capabilities will be analyzed according to two aspects (1) user-centered variability modeling and (2) derivation automation.

The need of supporting user-centered variability modeling stems from a systematic literature review regarding the requirements of derivation support reported by Rabiser et al. [RGD10], where the authors pointed out that users of variability models find, it is difficult to understand variability models due to the size of variability modeling elements and complex dependencies among them. O'Leary et al. have also reported that the representation of variability should be in the language of the product derivation stakeholders [ORRT09]. In the targeted IAM domain, three roles of stakeholders participate in the derivation process with different variability concerns, as analyzed in Table 2.2. In particular, requirement engineers are mainly concerned with process variability. Hardware-oriented engineers are concerned with topology variability. Software developers need to understand both of the other two types of stakeholders and bring their requirements together to develop the software systems. The derivation infrastructure should support the modeling of the three main variability types:

- **C1. Support for modeling feature variability**: Feature or decision models are able to represent the basic variability, such as mandatory, optional, alternative, multiplicity and cardinality. The IAM systems have feature-like variability that is common for many other product lines as well. The derivation infrastructure should be able to support the modeling of this type of variability.
- **C2. Support for modeling topology variability**: It is important to allow the representation of topological configuration of physical layouts in factories for hardware-oriented engineers to support their derivation tasks. This variability type cannot be expressed with feature-like variability modeling techniques [FLED13b], especially for the relationships among the physical-world hardware devices.
- **C3. Support for modeling process variability**: Process models are used in practice to describe functional requirements. The process variability is difficult to be expressed by feature-like variability modeling techniques, as we pointed

out in our work [FLED13b]. The reason is that complex dependencies are created among features, when using feature models to express process-oriented variability. In this case, the derivation infrastructure should be able to facilitate the representation of process variability during derivation.

To support the overall derivation process, including preparation, configuration and generation (see Figure 2.3), it is necessary to integrate the three variability models into the derivation approach, so that the outcome of derivation can take advantage of variability modeling to achieve a higher level of automation and efficiency. For this purpose, the derivation infrastructure requires the following three capabilities:

- **C4. Support for model-level reuse**: The reuse of non-code artifacts has not attracted enough attention in product derivation approaches, as reported by O' Leary et al. [ORRT09]. For complex software product lines, the configuration of variability models can be a tedious task for derivation stakeholders. To support derivation, the reuse of existing configured variability models can ease tasks of derivation stakeholders, since customers may share requirements.

- **C5. Support for staged derivation**: This capability comes from the derivation process presented in Figure 2.10. In practice, requirement engineers, hardware-oriented engineers and software developers participate in different stages during derivation. A derivation approach should not significantly change the derivation process. It should naturally align with the original stages to help the stakeholders.

- **C6. Support for integrating multi-variability configuration**: The product derivation tools should be adaptable and extensible to organizational and technological contexts when adopting the approach in practice [RGD10]. For the targeted IAM systems, a tailored approach should be developed, in which the three variability types and the different stakeholders should be integrated into a derivation process with their own variability concerns. As presented in Section 2.2.1, some process steps are bound to certain locations as topological elements. The tailored derivation approach should allow the configuration and integration of these variability and the interrelationship.

In the context of IAM systems, these six required capabilities (from C1 to C6) are originated from derivation process, which involves multiple stakeholders with individual variability concerns. Some other software families have more homogeneous variability types, in which a single variability model can support the automation of derivation well enough. These six capabilities motivated in this section are not intended to cover all variability modeling problems and derivation. As pointed out by O'Leary et al., if a derivation approach aims at a too generic adoption, it is very difficult to achieve a satisfied automation level [ORRT09].

2.4. Chapter Summary

Firstly, this chapter introduces important definitions to provide a consistent foundation of related concepts of this thesis. The goal is to avoid ambiguities of terms being used by researchers and practitioners.

Secondly, this chapter presents the practical problems with a running example. IAM systems have topology variability and process variability. These two domain variability types are very difficult to be expressed with typical variability modeling techniques in SPLE, such as feature modeling and decision modeling. They lead to complex dependencies among the variations. Worse still, three derivation stakeholders – requirement engineers, hardware-oriented engineers, and software developers – participate in a staged derivation process with different variability concerns. The derivation stakeholders desire a systematic approach to improve this situation and to achieve better efficiency during derivation.

Lastly, the required capabilities to facilitate product derivation in industrial automation management are analyzed. Feature-like, topology and process variability are discussed to enable user-centered variability representation. During derivation, the configuration of these three types of variability should be integrated systematically, so that a higher level of derivation automation can be achieved. The reuse of non-code assets, such as process models, should be promoted during derivation.

3
State of the Art and Limitation Analysis

"All doctrine, and all intellectual discipline,
arise from pre-existent knowledge."
–Aristotle

This chapter aims at reporting the related approaches of software product derivation. The analysis of the existing approaches shows how far the existing approaches can fulfill the needs of product derivation in IAM systems, based on the expected capabilities for its derivation support in the previous chapter (see Section 2.3 in Chapter 2). To achieve these goals, two questions will be answered in this chapter:

- *Q1. Which approaches exist in SPLE that support product derivation?*
- *Q2. What are the limitations of using existing approaches considering the required capabilities for derivation support in IAM systems?*

3.1. Research Methodology

A systematic literature review (SLR) is a research method to identify, evaluate, and interpret available research with regard to particular research questions [Kee07]. It is a method that can be used to systematically discover existing product derivation approaches. To answer Q1 of this chapter, the first step is to collect the existing relevant approaches in the state of the art. For this, it is important to systematically identify studies from scientific databases. Then, the analysis can be performed to know whether existing approaches can satisfy the expected capabilities during derivation in IAM, which is the answer to Q2.

Rabiser et al. has reported a systematic literature review [RGD10], which collected existing product derivation approaches, covering the research publications from year 1996 to 2007. A research question solved in [RGD10], *SLR-RQ1*, is actually the same as Q1 of this chapter. As the results of this SLR, Rabiser et al. reported 13 product derivation approaches. However, the searching was conducted mainly in winter 2007, and partially in spring 2008, so that the collected publications are mainly from **1996 to 2007**, although the paper was later on published in year **2010**.

Considering the rapid changes and advances in SPLE research, it is necessary to also collect the latest updates in this research area. Therefore, in this chapter, a systematic literature review is chosen as the research methodology to summarize the existing approaches of product derivation, focusing one the years **2008 to 2015**. Combining the results of this SLR and the previous SLR reported by Rabiser et al. [RGD10], a foundation can be built as a starting point for this thesis to overcome the problems identified in industry.

According to [Kit04, Kee07], a systematic literature review in software engineering has three major steps: planning, conducting, and reporting. These three steps are addressed in Section 3.2, Section 3.3, and Section 3.4 respectively.

3.2. Planning the Systematic Literature Review

The goal of conducting this systematic literature review is to collect and summarize the most recent approaches for product derivation between year 2008 and 2015, to answer "Q1. Which approaches exist in SPLE that support product derivation"? Learning from search terms used in the previous SLR reported by Rabiser et al. [RGD10], the search terms in this literature review are formulated as presented in Table 3.1. The chosen search terms are reused based on the previous SLR [RGD10].

Search Terms	
1	Derivation
2	Configuration
3	Application Engineering
4	Product Line
5	Product Family
6	{1 OR 2 } AND {3 OR 4 OR 5}

Table 3.1.: Search Terms (Focusing on Publications Between 2008-2015)

As shown in the following list, four electronic databases are used in this literature review, including: IEEE Xplore, ACM Digital Library, Science Direct and Springer Link. Besides, the 13th Software Product Line Conference (SPLC) in 2009 was published in Carnegie Mellon University (CMU). Since SPLC is one of the main venues for SPLE research, collecting the papers of this event should not be missed, for which extra search was conducted.

- IEEE Xplore (`http://ieeexplore.ieee.org/`)
- ACM Digital Library (`http://portal.acm.org/`)
- Science Direct (`http://www.sciencedirect.com/`)
- Springer Link (`http://www.springerlink.com/`)
- CMU (Accessed through `http://portal.acm.org/`)

3.2.1. Inclusion and Exclusion Criteria

As a part of the review protocol according to [SGF$^+$10], Table 3.2 shows the inclusion and exclusion criteria of this SLR. These criteria are applied for the selection of primary studies, recommended by the previous SLR [RGD10].

Inclusion Criteria (IC)	
1	The paper is electronically and full-text accessible.
2	The paper is written in English.
3	The paper is peer reviewed.
4	The paper presents a derivation or a variability configuration technique.
5	The paper contains an evaluation or a review of several derivation approaches.
Exclusion Criteria (EC)	
1	The paper is only a position paper to point out an idea.
2	The paper is not closely related to the software product derivation topic.

Table 3.2.: Inclusion and Exclusion Criteria

For the inclusion criteria, only peer-reviewed, English-written papers were considered. Only the papers with full-text availability in the chosen scientific databases were considered. The 4th inclusion criterion restricts that the SLR focuses on application engineering for the derivation approaches or variability configuration approaches. The 5th inclusion criterion ensures that the SLR also includes the evaluation papers which compare several derivation approaches, or report experiences of applying them in different circumstances.

For the exclusion criteria, position papers presented only a possible research direction are excluded. Since this thesis has a strong focus on derivation, which belongs to the activities of application engineering, this SLR on purpose excludes those papers that are not closely related to this area, for example if they are about testing, or development of the SPL architecture.

3.2.2. Quality Assessment

According to [Kit04, SGF$^+$10], quality assessment (QA) criteria are important to analyze the selected studies and to perform data extraction. Table 3.3 defines the quality assessment of this SLR. They are defined based on the quality criteria of the previous SLR reported by Rabiser et al. [RGD10].

Quality Assessment (QA)	
1	Is the paper cited by at least one other qualified papers?
2	Does the paper show good completeness and credibility?
3	Does the approach have considerable adoption barriers?

Table 3.3.: Quality Assessment

QA1 checks whether the paper is cited by other qualified papers, since approaches with high impact to the research field are likely to be well referenced by other papers. This criterion is less strict than in the previous SLR [RGD10], in which only papers with at least five other peer-reviewed publications were selected. The reason is that, this SLR is aimed at collecting the most recent derivation approaches. QA2 is defined to assess the credibility of the proposed approach, in accordance with the previous SLR [RGD10]. It ensures that the paper has no unsupported claims. QA3 is defined to examine whether the approach is suitable to be used in an extractive or re-active manner, which is preferred by the targeted IAM systems in the context of this thesis (see Section 1.3.2 in Chapter 1). These criteria work as further requirements to analyze the importance of the primary studies [Kee07].

3.3. Conducting the Systematic Literature Review

Figure 3.1 shows the execution procedure of this literature review. With the search terms displayed in Table 3.1, 2189 hits were retrieved from the chosen electronic databases. With the inclusion criteria in Table 3.2, the execution of the SLR results in 109 primary studies. With the exclusion criteria, 60 studies were excluded and 49 studies remain in the collection. Many of the excluded studies propose particular support during configuration, either for stakeholder support or configuration verification. These functions have been introduced and categorized in Section 2.1.4.2, such as prioritization of features during configuration time or consistency checking. These aspects are important to help users, but they are not closely related to the research goal of this thesis.

After quality assessment, there are 42 studies in the collection. Two literature reviews are reported by Rabiser et al. [RGD10], including the known SLR and capabilities of multi-product lines by Holl et al. [HGR12]. One study reports on experiences during product derivation in practice [dSOdAdLM15]. By snowball searching, one more study, named FAST [WL99], is included into the results of the SLR. In total, the SLR comprises 43 primary studies (see Appendix A).

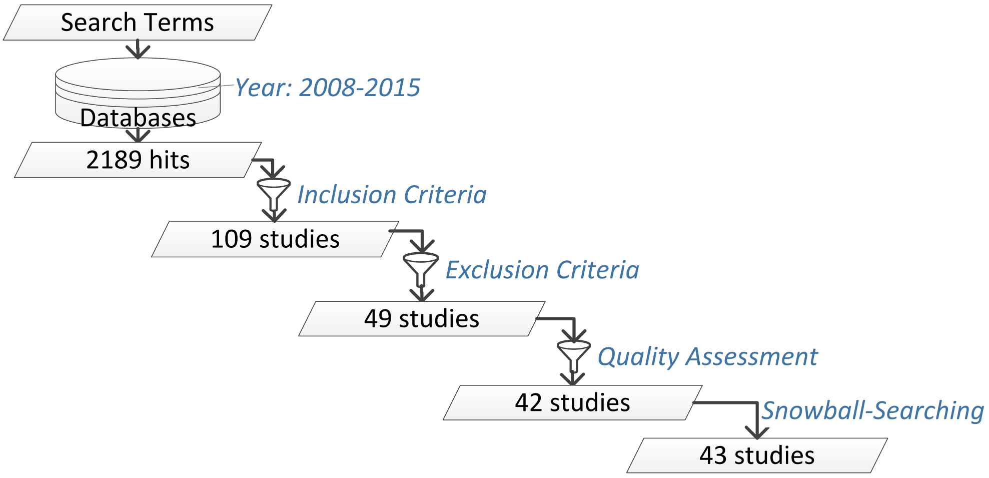


Figure 3.1.: Conducting the Systematic Literature Review

3.4. Reporting the State of the Art

The SLR results in 43 primary studies on derivation approaches[1]. They can be further grouped into 15 product derivation approaches. The studies are grouped based on either their variability mechanisms, or on (co-)authors who contributed to the same derivation approaches. Figure 3.2 shows that the two SLRs collect only four common approaches: COVAMOF, the 3-tiered methodology, feature-oriented programming (FOP) and DOPLER.

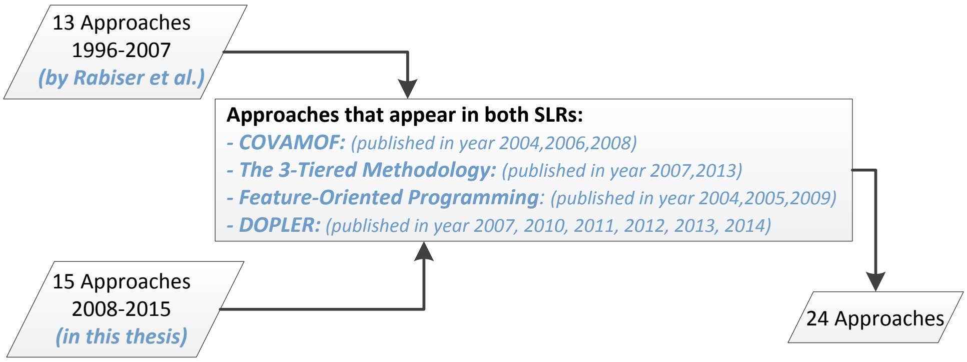

13 Approaches
1996-2007
(by Rabiser et al.)
Approaches that appear in both SLRs:
- COVAMOF: (published in year 2004,2006,2008)
- The 3-Tiered Methodology: (published in year 2007,2013)
- Feature-Oriented Programming: (published in year 2004,2005,2009)
- DOPLER: (published in year 2007, 2010, 2011, 2012, 2013, 2014)
15 Approaches
2008-2015
(in this thesis)
24 Approaches

Figure 3.2.: Common Approaches in the Two Literature Reviews

However, it should be noted that just because an approach has no recent publication updates in the chosen databases, it does not make it less relevant. Actually, several of the approaches (e.g., SEI Product Line Practices [CKSW13, Kru07a, Kru13]), are

[1] The author of this thesis does not include her own work [FLE^{+}15] in this reporting section, because the work will appear in Chapter 6 as a part of this thesis.

still of high relevance for practice, or have influenced many subsequent approaches in the field (e.g., staged derivation of Czarnecki et al. [CHE04]). The reason is probably that each derivation approach has its applicability to be applied to certain types of software product lines. As reported by O'Leary et al. [ORRT09], it is impossible to develop a generic enough derivation approach to fit to every type of systems.

It is important to emphasize that, the analysis of these 24 approaches are based on how they were explicitly presented and conveyed in the collected studies, as well as related supporting documents of the studies that are publicly available. Applying every of them directly in IAM systems is not feasible to achieve within the scope of this thesis. Since the goal of conducting the literature view is to get a good understanding of the research landscape, the author of this thesis decided to focus only on research tools. In case, some of the approaches are related to commercial tools, they are excluded.

To mitigate bias of assessment, the author of this thesis involved several experienced researchers in software product line engineering, in order to come up with a more objective assessment of each study. The analysis of the selected studies reveals that existing derivation approaches have mainly three different purposes. (1) The approaches with general-purpose variability models are those derivation approaches, in which a single variability modeling technique is used. The majority of them either use feature modeling or decision modeling as the general-purpose variability modeling technique. 12 approaches are in this group. (2) The approaches have domain-specific or model-specific concerns. In this group, variability models have a broader scope, for example domain-specific modeling languages or architectural description languages. Some of the 8 approaches in this group use several models or languages to express variability of software systems. (3) The approaches propose derivation guidelines, focusing on higher methodological level. These 4 approaches propose in particular reference activities during derivation.

The rest of this section organizes these 24 approaches into the three groups. Section 3.4.1, Section 3.4.2, and Section 3.4.3 address each of these groups respectively.

3.4.1. Derivation with General-Purpose Variability Models

This subsection starts with the introduction of the derivation approaches that are based on general-purpose variability modeling techniques. Afterwards, they are analyzed at the end of this subsection to what extent they can provide the expected capabilities for the software derivation of IAM domain.

1. **COVAMOF**: Sinnema et al. propose a variability modeling framework named COVAMOF [SDNB04]. COVAMOF consists of two variability views: a variation point view and a dependency view. The idea to develop COVAMOF is to support a consistent way of representation of variations across different development hierarchies, from requirements, to architecture and further to implementation details. The separate dependency view highlights the importance of complex

dependencies and interactions as first class citizens. As tool support, COVAMOF-VS tool suite was developed. The supported COVAMOF derivation approach contains four steps, including Product Definition, Product Configuration, Product Realization and Product Testing. The last three steps can occur in one or more iterations [SDH06]. An experiment of applying COVAMOF in intelligent traffic domain proves the reduction of derivation time and iterations [SD08].

2. **The 3-Tiered Methodology**: Krueger proposes the 3-tiered SPL methodology [Kru07a]. The base tier is concerned with variation management and automated production at product technical level. The middle tier focuses on core assets development and management across multitude of software products. The top tier manages the portfolios, business, and aims at ensuring the portfolio scalability and time-to-market [Kru07a]. The 3-tiered methodology was implemented in GEARS[1] by BigLever software, Inc. [Kru07b]. More recent work from BigLever shows the enhancements of GEARS by introducing staged derivation [Kru13] and auditing of product configuration [CKSW13].

3. **Feature-oriented programming (FOP)**: The idea of FOP is to base on a core product and to modify it via feature modules, which refine the core product. It leverages step-wise refinement, which is a paradigm to develop complex program by incrementally adding details to simple program [Dij76]. Batory et al. propose AHEAD (Algebraic Hierarchical Equations for Application Design) model, which shows how step-wise refinement can scale to synthesize multiple programs and multiple non-code representations [BSR04]. Díaz et al. based their approach on AHEAD to promote product derivation processes as "first-class artifacts" [DTA05]. Apel et al. [AKL09] propose FeatureHouse, which is a descendant of Batory's AHEAD program generator. The implementation of FeatureHouse is the tooling to unify the languages and tools that rely on superimposition by using the language-independent model of feature structure trees, which enable the composition of software artifacts [AKL09]. Besides these works, FeatureIDE [KTS$^+$09] is an Eclipse[2] based tool, originally designed in 2004/2005 as an IDE for the AHEAD tool suite. Kästner et al. report that FeatureIDE can support the entire life cycle of a product line coherently [KTS$^+$09], incorporating different tools including AHEAD, FeatureC++, FeatureHouse, etc.

4. **Delta-oriented programming (DOP)**: DOP is a compositional approach to flexibly implement software product lines [SBB$^+$10]. DOP uses a core module and a set of delta modules to represent a software product line. In contrast to FOP, the core module in DOP is always a valid product configuration. This valid core module serves as a basis for further configurations. The Delta modules define changes to be applied to the core module. By adding, modifying and re-moving code, new product can be implemented [SBB$^+$10]. Extensions of DOP include dynamic DOP [DS11], which supports changing the feature configurations of a

[1] GEARS: `http://www.biglever.com/solution/product.html`, accessed on 10.11.2015.

[2] Eclipse: Open Source Development Environment: `https://eclipse.org/`, accessed on 12.11.2015.

product at run-time, as well as verification of delta modules [SK13].

5. **DOPLER**: Decision-Oriented Product Line Engineering for Effective Reuse: User-centered Configuration (DOPLER$^{\mathrm{UCon}}$) [DGR11, RGD07] is a decision-model based derivation approach, developed at Johannes Kepler University Linz. DOPLER$^{\mathrm{UCon}}$ allows the customization of different views to different users during configuration time. From a high-level viewpoint, it supports the whole lifecycle of product derivation: configuration preparation, product configuration, application requirement engineering, additional development, integration, deployment, maintenance and evolution. Based on DOPLER$^{\mathrm{UCon}}$, additional components were developed and frequently published in scientific conferences, journals and workshops. For instance, DOPLER$^{\mathrm{VM}}$ [DGRN10] was developed by Dhungana et al. to support variability management in DOPLER [ROR11]. User guidance function was developed by Rabiser et al., named DOPLER ConfigurationWizard (DOPLER CW), to guide users to make their decisions during configuration [RGL12]. There are also several additional works, developed to enhance the DOPLER, such as multiple product line or hierarchical modeling [HEGV13, DSL^{+}14].
6. **EASy-Producer**: EASy-Producer is developed as a lightweight engineering tool for SPLs and variability-rich software ecosystems [EESKS14, SE15]. It is an open-source tool based on Eclipse. EASy-Producer supports different views and configuration modes for users during configuration time. It also includes a modeling language for expressing variability named IVML [ES15], which can be instantiated by Variability Instantiation Language (VIL) during project derivation. According to the available feature description [eas], EASy-Producer also allows staged configuration in multi-product lines.
7. **Staged Derivation**: Czarnecki et al. motivate the concept of staged configuration [CHE04], since in a realistic development environment, different stakeholders decide or configure product variability in different stages [CHE04]. The authors propose to use a cardinality-based feature modeling in a staged derivation process to formalize the definition of feature notations.
8. **Multi-product-line**: Complex software systems comprise several heterogeneous product lines. Different research groups or organizations are often in charge of the individual product lines. Commonly, staged derivation is required. Holl in his work [Hol11] presents an approach to facilitate variability configuration in multi-product lines. Typically, multi-product lines cannot be managed centrally, as the involved product lines are developed and evolved by multiple teams that often work independently. Holl et al. developed a constraint-checking tool, which has been also integrated into the tool suite of DOPLER [HGE^{+}13]. In the context of multi-product lines, Elsner proposed lightweight tool support [Els12], which explicitly distinguishes stages, stakeholders, and build tasks with constraints to handle heterogeneous assets.
9. **Orthogonal variability modeling (OVM)**: Halmans and Pohl proposed orthog-

onal variability modeling [HP03], which clearly differentiates the variability in the model and its realization in assets. An extension of the orthogonal variability modeling technique to use case diagrams based on Unified Modeling Language (UML) 1.4 notations is presented in [HP03]. Up to now, product derivation is not the main focus of the OVM technique.

10. **Product-Line Requirements Specification (PRS)**: The approach aims at systematically representing the requirements of software families [Fau01]. PRS is able to distinguish variations of different family members, and allows generating well-formed software requirements specifications [Fau01]. Other approaches focusing on requirement engineering of SPL are: Requirement Elicitation in flexible SPL by Adam [AS13, Ada12], Requirement Document Generation by Rabiser et al. [RHE+10], and Use Case Derivation by Yu et al. [YZZJ14]. These approaches mainly focus on the beginning phase of application engineering, which is valuable to learn from. However, the research goal of this thesis is not only to help requirement engineers who are mainly concerned with on essential variability, but also the hardware-oriented and software developers with technical variability. Consequently, approaches focusing only on requirements cannot fully solve the IAM derivation problems.

11. **VISIT-FC** Cawley et al. propose a feature configuration meta-model and introduce a prototype tool named Visual and Interactive Tool for Feature Configuration (VISIT-FC) [CNP+08]. The approach supports a variety of feature visualization and interactions during configuration time to help users to perform their configuration tasks.

12. **PuLSE^{TM}, PuLSE-I, and KobrA**: Bayer et al. present a method based on decision modeling, PuLSE^{TM} [BFK+99]. PuLSE^{TM} enables the conception and deployment of software product lines within different enterprise contexts. PuLSE^{TM} splits the life cycle of software product lines into four phases: initialization, product line infrastructure construction, usage and evolution. It also provides technical components for the deployment phase of software product line development [BFK+99]. As the application engineering process of this method, PuLSE-I was presented as a high-level guideline when performing activities during application engineering [BGMW00]. As one of the follow-up works of PuLSE^{TM}, Atkinson et al. [ABM00] describe a method, Component-Based Product Line Development (KobrA), which is considered to be a customization of PuLSE^{TM}. KobrA integrates product line and component-based approaches into a systematic, unified approach to software development and maintenance. Compared to PuLSE^{TM}, KobrA is more suitable to be introduced to development organizations when product families are not yet well defined [ABM00].

The advantages and limitations of using these existing approaches are analyzed in Table 3.4, based on how well they can fulfill the desired capabilities compared with the information conveyed in the selected studies. The columns of Table 3.4 are the desired capabilities of product derivation support in IAM systems as summarized in Section

2.3, and the rows include the 12 approaches mentioned above. If the approach fully supports the required capability, it is denoted by (+); if the approach partially fulfills the requirement with certain limitations or constraints, it is marked as (o); and if the approach does not provide technical supports to enable the corresponding requirement, the assessment is presented as (-). The assessment is based on the information shown in the related studies to the best of the author's knowledge and therefore could be subjective. It is possible that some of these approaches have not explicitly mentioned the desired capabilities, but they could support them to some extent.

Approach	**Desired Capabilities**					
	C1.	C2.	C3.	C4.	C5.	C6.
	Modeling Feature Variability	Modeling Topology Variability	Modeling Process Variability	Model-Level Reuse	Staged Derivation	Integrating Multi-Variability Config
COVAMOF	+	−	**o**	−	−	−
The 3-Tiered Methodology	+	−	−	**o**	+	−
FOP	+	−	−	−	**o**	−
DOP	+	−	−	−	**o**	−
DOPLER	+	**o**	−	**o**	+	−
EASy-Producer	+	+	**o**	**o**	+	−
Staged Derivation	+	−	−	**o**	+	−
Multi-Product Line	+	−	−	**o**	+	−
OVM	+	−	**o**	**o**	−	−
PRS	+	−	**o**	+	−	−
VISIT-FC	+	**o**	**o**	−	−	−
PuLSE^{TM}, PuLSE-I, and KobrA	+	−	**o**	+	**o**	−

+:good **o**:fair −:poor

Table 3.4.: Analysis of Approaches with General-Purpose Variability Models

As can be seen from Table 3.4, all the derivation approaches use a general-purpose variability modeling technique, either feature modeling, decision modeling, or orthogonal variability modeling. No matter which variability technique is chosen in these approaches, it is possible to express *feature-like* variability (C1).

Modeling topology variability (C2) was considered in several approaches as multiplicity. For example, DOPLER has taken multiplicity and hierarchy into consideration, which could partially fulfill the topology variability modeling need in the targeted IAM systems [DSL$^+$14]. EASy-Producer includes the capabilities of modeling, configuring and instantiating of topology variability as part of its toolset, which allows modeling complex constraints using an extended dialect of the Object Constraint Language. [ES15, EQSS16, Eic]

For modeling process variability (C3), several approaches resolve it as feature dependencies. For instance, in COVAMOF, the dependencies among features are also promoted as first class citizens in product derivation, but the variability dependencies are oriented more to the implementation, without the consideration of execution procedures to satisfy the need in IAM system derivation. To overcome the challenges of complexities brought by the large numbers of variability elements and the complicated dependencies, EASy-Producer supports a textual variability modeling using the INDENICA Variability Modeling Language (IVML) [ES15]. Textual variability modeling languages bring the flexibility to express variability, for example in a specific domain, or in a desired way. By using textual variability modeling languages with certain adjustment, it is possible to express the process-like relationships among variability in EASy-Producer. The evaluation of these approaches are positive, but only fair support (o). The reason is that using dependencies to express process variability leads to tacit relationships among features, instead of explicit relationships, because some of the process steps are optional. Fey et al. point out that such weak suggestion can be categorized as an "imply" relation [FFB02]. Modeling the process variability in this way could cause difficulties in feature configuration for derivation stakeholders, as discussed in our work [FLED13b].

As can be seen from Table 3.4, to support the *topology* (C2) and *process variability modeling* (C3), using general-purpose modeling techniques cannot satisfy the needs of the domain experts for representation and configuration in IAM systems very well. Without the configuration of these two variability types, it is unlikely to reach a satisfactory automation level during product derivation.

For the purpose of software derivation, the capability of *model-level reuse (C4)* has not attracted enough attention based on the analysis of Table 3.4. Several approaches collected in this subsection can support the reuse of non-code artifacts, such as requirements documents, so that the model-level reuse can be indirectly supported, such as in PRS or PuLSE^{TM}.

Regarding the staged derivation (C5), the assessment is based on whether the approaches have explicitly mentioned the support of staged derivation, either directly in the collected studies or in the accessible documentation. Among these studies in this subsection, the 3-tiered methodology, DOPLER, EASy-Producer and Multi-Product-Line are rated as positive.

The *support of integrating multi-variability configuration (C6)* requires the derivation approach to allow the configuration of multi-variability. Particularly in IAM domain, the process variability are necessary to bind to some of the topological elements, as motivated in Section 2.3 of Chapter 2. Furthermore, the multi-variability should be integrated in a derivation process. Such an integration of the variability models can significantly improve the efficiency of product derivation, as pointed out in Section 2.2. Due to the complexities of variability modeling in IAM systems, C6 as is presented in Table 3.4 cannot be well supported by the approaches in this group.

3.4.2. Derivation with (Domain-)Specific Variability Models

In this subsection, the derivation approaches with multiple modeling concerns or domain-specific concerns are summarized. Following the structure of the previous section, each of the approaches is introduced first. Afterwards, the limitations of using them are analyzed and investigated at the end of this subsection.

1. **PLUS**: Gomaa proposed Product Line UML-based Software engineering (PLUS) for designing software product lines [Gom04]. In PLUS, feature modeling is incorporated in UML for modeling common, optional and alternative product line features. Application engineering is carried out by tailoring the UML product line models, which addresses variability in use cases, state machines, classes and components, for example [Gom04, RGD10].

2. **Koala/Koalish/Kumbang**: Koala is an architecture description language developed by the architects and developers to reuse components within a family of television products in Philips [ASM04]. In Koala, the components are developed independently, and these components are wired and interacted via interfaces. The binding of variability in components is at configuration time, by using a compiler [VOVDLKM00]. Koalish is an extension of Koala, which combines the idea of an architecture description language with explicit variation modeling mechanisms to enable automatic configuration of products in the software family [ASM04]. Furthermore, the Kumbang configurator tool was developed to support the configuration and derivation of Koalish [RGD10]. Since this approach is clearly tailored towards the solution space of systems in the embedded domain, where the topology and processes are not concerned as variability types, it is not applicable to the IAM domain.

3. **DSL-based methodology**: Voelter and Visser propose to use domain-specific languages (DSL) in the context of SPLE [VV11]. The authors argue that using DSLs helps on filling the expressive gap between the variability in feature models and programming languages. The first benefit of using DSLs is that they provide a certain formalization of the core application logic in the solution space, which can be linked to the variability in the problem space. Secondly, well-designed DSLs give domain stakeholders the power to express the variability in a more precise way. However, the approach presented by Voelter and Visser did not include the situation, when multiple DSLs are required to be integrated. For this reason, the staged derivation (C5) is not supported to solve the problems in IAM systems.

4. **SimPL**: Behjati et al. propose SimPL as a methodology [BYBS13], which provides multiple views to the users to specify integrated control systems. In SimPL, the variability of software, hardware, and their dependencies are presented in different views. The derivation process in SimPL is focused on interactive user guidance and consistency checking of variability configuration. The application domain of SimPL seems to be similar to our target domain without the consideration of process variability.

5. **Topology-oriented methodologies**: Several approaches have been proposed to solve topology variability during modeling and derivation. Urli et al. propose

SpineFM [UBFC14], which is a tool-supported approach that uses domain models to represent topology variability. SpineFM interrelates domain models with feature models, and enables an order-free configuration process by propagation mechanisms to help users to perform their tasks. Berger et al. distinguish between feature-/decision-like variability and topological variability, and report the applicability and challenges of modeling topological variability with a case study in the domain of fire alarm systems [BSØ+14]. Fantechi raises the challenges of formal verification for system families with topological variability [Fan13]. These approaches concern the representation of topology variability and its interrelationship with feature-/decision-like variability. The authors did not mention process variability in the domains that their approaches address, neither the staged derivation process.

6. **Process-oriented methodologies**: There are also several existing works focusing on process variability [WMW11, GBPG13]. Gröner et al. proposed an approach for configuration and validation of business process families [GBPG13]. In this work, the authors developed generic business process models based on workflow control patterns [vDATHKB03]. A business process family consists of a feature model, a process model template and mappings among the elements in these two types of models [GBPG13]. The usage of business process modeling was not originated from the SPLE community, but research areas like business process management. The modeled business processes have usually execution semantics that can run directly on a workflow engine. For software systems, this requires a high maturity level of the software product line, so that process steps are able to be composed. However, this is not the case for IAM systems, because IAM systems belong to the kind of flexible SPLs and usually are at low maturity level (cf. Section 2.1.2 in Chapter 2). In IAM systems, process composition with full flexibility is not possible to reach according to our observation in practice, due to domain complexity.

7. **Aspect-oriented programming (AOP)**: Aspect-oriented programming (AOP) is a programming mechanism to clearly express additional behavior in source code. AOP involves aspects, such as isolation, composition and reuse of the aspect code [KLM+97]. The International Conference on Aspect-Oriented Software Development is one of the primary events for this area. Approaches closely related to software reuse have be proposed by, for example Lee. Lee et al. propose to integrate feature models with AOP mechanisms to achieve derivation automation [LBT09]. Since the AOP-based approaches have a very strong focus on the solution space, there are further works involve architecture models [DPFSG13, CNKL12]. The effort of re-work on a software family to apply an AOP-based approach at code level would be very high, which is very difficult to achieve in IAM systems.

8. **Domain-specific modeling (DSM) / modeling-driven development (MDD)**: Pillai et al. developed an approach to enable the specification of static and

behavior design information to be expressed with domain-specific models. The approach was applied to paper handling systems in printers to fully generate the control software [PFS09]. Kulkarni et al. propose a model-driven approach in business application product lines. This approach abstracts the composition, variability and solution in a uniform manner [KBR12]. Trask et al. combine model-driven engineering and SPLE to overcome the challenges of the high-level of variability in the radio domain [TPRB06], such as high performance, security, resource-constrained. In this approach, the authors use domain-specific modeling and model-driven engineering to tackle these variants and integrate them into SPLE. Brown et al. propose to attach behavioral models to feature nodes to compensate the representation limitation of feature modeling [BGB$^+$06]. Heinzemann and Becker propose to use adapted concepts of the UML, named MechatronicUML, to enable hierarchical re-configuration of complex software components for self-adaptive mechatronics systems [HB13]. These model-based approaches in this category show clearly the benefits on expressing variability for topology and processes. By developing proper domain-specific models, it is very promising to fulfill the variability representation requirements in this thesis. Until now, none of these approaches could also support staged derivation with multiple stakeholders.

Table 3.5 presents the limitation analysis for using these approaches in the IAM domain. Compared to Table 3.4 (see the approaches mentioned in Section 3.4.1), these approaches can provide obviously better support for topology and process variability representation. The reason is that, multiple modeling techniques are integrated into the derivation to enhance the variability representation.

Approach	**Desired Capabilities**					
	C1.	C2.	C3.	C4.	C5.	C6.
	Modeling Feature Variability	Modeling Topology Variability	Modeling Process Variability	Model-Level Reuse	Staged Derivation	Integrating Multi-Variability Config
PLUS	+	**o**	**o**	−	**o**	−
Koalish/Kumbang	+	−	−	−	**o**	−
DSL-based methodology	+	**o**	**o**	**o**	−	−
SimPL	+	+	**o**	−	**o**	−
Topology-oriented methodologies	+	+	−	−	**o**	−
Process-oriented methodologies	+	−	+	+	−	−
AOP	+	−	**o**	−	**o**	−
DSM-MDD	+	+	+	−	−	−

+:good **o**:fair −:poor

Table 3.5.: Analysis of Approaches with (Domain)-Specific Variability Models

The limitations of the approaches collected in this subsection have less focus on derivation but more on modeling. Enhanced variability modeling and representation may lead to better derivation automation through model transformation. However, in the IAM domain, derivation requires explicit support for stages. As can be seen in Table 3.5, only the process-oriented methodologies address the model-level reuse (C4), in which process templates are used to save the configuration effort. The support of different variability in staged derivation (C6) is still a research gap of the approaches in this category.

3.4.3. Derivation Activities and Processes

The derivation approaches in this subsection are not at a technical level. They are at a higher methodological level, aiming at providing guidelines, risk analysis, reference models for product derivation, or SPLE adoption practices.

1. **SEI Product Line Practice Initiative**: The previous SLR by Rabiser [RGD10] has already reported this approach. The SEI provides a high-level framework of practices for deciding the timing to automate product derivation, choosing the techniques, and carrying out the derivation process [RGD10]. McGregor in his work [McG05] points out that the decision to apply SPLE has both economic and technical dimensions. In the economic dimension, the decision should be made based on the cost of developing core assets, SPL infrastructure, and converting an organization to apply the product line strategy. Furthermore, during derivation practice, the effort of reusing the core assets also influences the benefits of applying automated derivation. In the technical dimension, achieving a higher automation level during derivation requires more investment in tooling, since more knowledge needs to be encoded in the infrastructure.
2. **FAST**: FAST was collected during snowball searching. It is not accessible from the chosen scientific databases. It was neither collected from the SLR reported by Rabiser et al. [RGD10]. However, some selected approaches in this chapter consider FAST as an important related work. Weiss et al. propose a family-oriented software process, named Family-Oriented Abstraction, Specification and Translation (FAST) [WL99]. The FAST method includes a full SPLE process with three phases: domain qualification, domain engineering and application engineering [WL99]. The derivation process in FAST includes requirements elicitation, requirements analysis, product configuration and additional development and testing. FAST has been applied successfully in the domains of telecommunication infrastructure and real-time systems. However, the derivation of FAST is strongly relying on precise system specifications [WL99], which is often not suitable for flexible SPLs, especially in the context of IAM systems.
3. **Pro-PD**: O'Leary et al. propose a reference process for software product derivation. Their Process Model for Product Derivation (Pro-PD) [OTBR08] defines six main process steps in Pro-PD: Initiate Project, Identify and Refine Requirements, Derive the Product, Develop the Product, Test the Product, and Management and Assessment. For each of the process steps, the tasks, roles of participants, and the related work artifacts are clearly defined as well. One of the main goals of

developing Pro-PD is to provide a reference model. When facing domain-specific needs, Pro-PD needs to be further customized [ODAR12].

4. **Reference process of product derivation**: Rabiser et al. propose key activities for product derivation, aiming at a reference product derivation process in SPLE [ROR11]. The reference activities are collected based on a comparison of the two derivation approaches Pro-PD and DOPLER^{UCon} [ROR11, ORRT09]. The three key activities are: Preparing for Derivation, Product Derivation/Configuration and Additional Development/Testing. The validation of these reference activities is conducted by analyzing other existing approaches to see whether they also support it, including COVAMOF, PuLSE-I, and FAST. The results of these comparisons emphasize that a preparing step is critical for derivation, especially to help on supporting the derivation stakeholders and creating guidance [ROR11].

These approaches have less technical aspects of software product derivation. They rather provide guidance for product derivation or even for the whole SPLE life cycle, with less emphasis on how to implement the derivation infrastructure technically. Nevertheless, Table 3.6 shows the analysis of the four approaches in terms of the expected software deviation capabilities of IAM systems.

Approach	**Desired Capabilities**					
	C1.	C2.	C3.	C4.	C5.	C6.
	Modeling Feature Variability	Modeling Topology Variability	Modeling Process Variability	Model-Level Reuse	Staged Derivation	Integrating Multi-Variability Config
SEI Product Line Practice Initiative	**o**	−	−	−	**o**	−
FAST	+	**o**	**o**	−	+	−
Pro-PD	**o**	**o**	**o**	−	**o**	−
Reference process of product derivation	**o**	−	−	**o**	**o**	−

+:good **o**:fair −:poor

Table 3.6.: Limitation of Using Methodological Level Derivation Approaches

3.5. Research Problems and Research Questions

Based on the results of the systematic literature reviews, a derivation approach that can satisfy the six desired capabilities of IAM systems is still missing. To look for a solution, the main research question in this thesis is defined as:

Main Research Question:
How can we improve the software product derivation process for systems in industrial automation management, where different variability types are concerned in different derivation stages?

Shaw classifies research questions into five types: (1) *method or means of development*, (2) *method for analysis or evaluation*, (3) *design, evaluation or analysis of a particular instance*, (4) *generalization or characterization*, and (5) *feasibility study or exploration* [Sha03]. Based on this classification, the main research question proposed above can be categorized as a *method or means of development*. The goal of the thesis aims at a new method of developing product derivation infrastructures for IAM systems.

To answer this main question, four sub-questions need to be answered:

- *RQ1: How can we structure the variability modeling space to support the representation of multiple variability types?* RQ1 is aiming at a method of development for a variability modeling framework to satisfy the three variability types: feature-like, topology and process variability. The answer to RQ1 establishes the modeling infrastructure to build the foundation of further automation to the overall approach.
- *RQ2. How can we establish the mapping between the different variability modeling elements with their associated core assets?* With multiple types of variability in the target domain, the traditional way to associate variability modeling elements with reusable artifacts is not sufficient anymore. The answer to RQ2 looks for a method to establish such a mapping between the modeling space and solution space.
- *RQ3. How can we use the multiple variability models during the staged derivation process?* This question aims at automating the derivation, in order to finally achieve the goal of efficiency improvement to help the domain engineers and stakeholders. The answer to this question actually assembles the outcomes of RQ1 and RQ2. With the answers to RQ1, RQ2 and RQ3 the overall solution is complete at this point.
- *RQ4. How good is the proposed approach on improving IAM product derivation?* RQ4 is an evaluation question to assess to which extent the approach can achieve the goal, and to help on solving the practical problem.

3.6. Chapter Summary

The chapter reports on the state of the art of software product derivation, based on a previous systematic literature review [RGD10] (covering the years 1996 to 2007) and a newly conducted systematic literature review (covering the years 2008 to 2015). As the answer to *Q1: Which approaches exist in SPLE that support product derivation*, Table 3.7 shows a summary of the 24 approaches. Each of the selected approaches has been introduced and analyzed with its advantages and limitations against the desired capabilities of the derivation infrastructure (cf. Section 2.3). Both of the execution of the literature review and the assessment have involved other experienced researchers in software product line engineering to ensure the credibility of the results.

Approach	Desired Capabilities					
	C1.	C2.	C3.	C4.	C5.	C6.
	Modeling Feature Variability	Modeling Topology Variability	Modeling Process Variability	Model-Level Reuse	Staged Derivation	Integrating Multi-Variability Config
COVAMOF	+	−	o	−	−	−
The 3-Tiered Methodology	+	−	−	o	+	−
FOP	+	−	−	−	o	−
DOP	+	−	−	−	o	−
DOPLER	+	o	−	o	+	−
EASy-Producer	+	+	o	o	+	−
Staged Derivation	+	−	−	o	+	−
Multi-Product-Line	+	−	−	o	+	−
OVM	+	−	o	o	−	−
PRS	+	−	o	+	−	−
VISIT-FC	+	o	o	−	−	−
PuLSE^{TM}, PuLSE-I, and KobrA	+	−	o	+	o	−
PLUS	+	o	o	−	o	−
Koalish/Kumbang	+	−	−	−	o	−
DSL-based methodology	+	o	o	o	−	−
SimPL	+	+	o	−	o	−
Topology-oriented methodologies	+	+	−	−	o	−
Process-oriented methodologies	+	−	+	+	−	−
AOP	+	−	o	−	o	−
DSM-MDD	+	+	+	−	−	−
SEI Product Line Practice Initiative	o	−	−	−	o	−
FAST	+	o	o	−	+	−
Pro-PD	o	o	o	−	o	−
Reference process of product derivation	o	o	o	o	−	−

+:good o:fair −:poor

Table 3.7.: Summary of State-of-the-Art Approaches

The analysis of the state of the art in Table 3.7 has shown that none of the existing product derivation approaches could fully satisfy all the desired capabilities. For C1, many of the existing approaches use either feature modeling or decision modeling for variability representation. SimPL, EASy-Producer, topology-oriented methodologies and DSM-MDD are able to express the topology variability (C2). There are process-oriented methodologies and DSM-MDD to model process variability (C3). For C4, it becomes also clear that model-level reuse has not attracted enough attention in SPLE derivation approaches. Many existing approaches are able to support staged derivation (C5), especially those designed for multiple roles of derivation stakeholders, such as DOPLER, or Multi-Product Line. Integrating different variability in stages (C6) until now has not been well recognized as a derivation challenge. Until now, there is no reasonable solution yet to technically tackle it. The analysis becomes the answer to Q2, about the limitations of using existing approaches considering the required capabilities for derivation support in IAM systems.

At first glance, the difficulty of reuse in the IAM domain seems to stem from the fact that it comprises topology and processes as domain-specific variability types. However, the actual reason is that many of the existing approaches try to use a general-purpose variability model to express all types of variability, without enough consideration of the needs of the model users, especially the approaches categorized in the first group. For example, it might be possible to use dependencies to describe weak sequential dependencies among features. It would lead to complicated feature dependencies among feature nodes, which makes the feature configuration tasks very difficult. A better solution would be to allow the users to express the variability in their own languages, which means to use domain-specific models to represent the variability. Domain-specific modeling helps engineers to express their knowledge in a clear and more concise way, which is closer to the problem space [Val10].

Additionally, the derivation process should integrate the different variability models and their configurations by different stakeholders. It can increase automation level by model transformation and code generation, so that the overall productivity and efficiency can be improved [TK05]. Based on the result of this chapter, the potential solution would be to use a model-based approach, which allows the configuration of three variability types and enhances the model-level reuse to automatically instantiate the variability models as base configuration. The configuration of the different variability types shall be integrated in derivation stages to improve the derivation efficiency.

Lastly, this chapter comes up the main research questions as: *to improve the product derivation of systems in industrial automation management, where different variability types are concerned in different derivation stages*. The problem is further broken down into four sub-research-questions. Chapter 4 proposes a multi-level variability modeling framework addressing RQ1. Chapter 5 presents the realization of variability, by tracing models to assets, which is the answer to RQ2. Chapter 6 integrates the outcome of the variability modeling framework and the core asset base to a semi-automated derivation approach as the answer to RQ3. Chapter 7 finally evaluates the whole solution to answer RQ4.

4

Hierarchical Multi-Variability Modeling

"Essentially, all models are wrong, but some are useful."

–George Box and Norman Draper

The overall solution outline of this thesis has been shortly introduced in Section 1.3 in Chapter 1. As shown in Figure 4.1, the solution includes three main components: (1) *a multi-variability modeling framework* for variability representation, (2) *the multi-variability realization techniques* to implement the multi-variability and linkages in the core asset base, and (3) *a semi-automated approach for product derivation* to integrate the multi-variability and using them to enable a derivation process. Starting from this chapter for the (1), and following by Chapter 5 for the (2) and Chapter 6 for the (3), we will see the details of these three solution components.

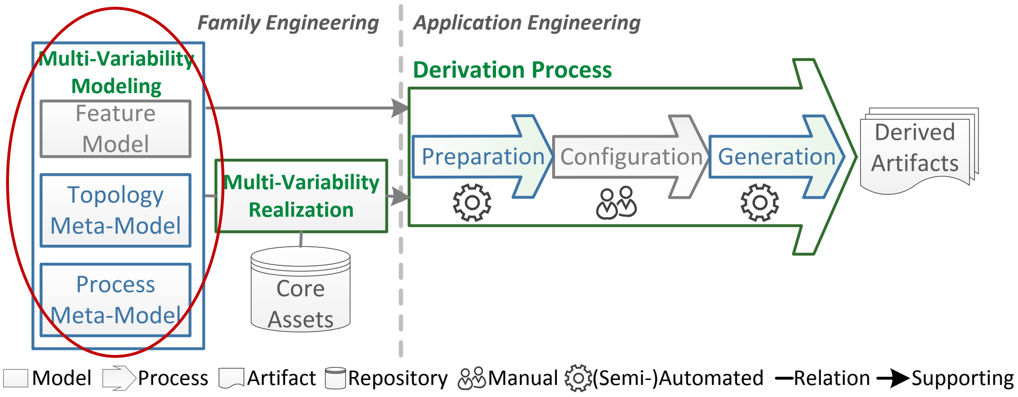


Figure 4.1.: Multi-Variability Modeling in Overall Solution

This chapter presents a multi-variability modeling framework for the systems of industrial automation management (IAM), as shown on the left hand side of Figure 4.1. The framework includes feature modeling for representing general software commonalities and variability, and two domain-specific models for expressing topology and process variability. Besides the meta-models of the two domain-specific models, this chapter also provides the principles and procedures to develop them. Additionally, the three types of variability models are further associated with components in the solution space hierarchically. This framework plays an essential role in the overall product derivation approach for variability modeling as marked in Figure 4.1, in order to represent and express more variability in the target software domain.

By proposing the modeling framework, this chapter answers the question, *"RQ1: How can we structure the variability modeling space to support the representation of multiple variability types?"* To answer RQ1, this chapter contributes to:

- Multi-variability modeling framework for feature, topology and process variability
- The principles, the procedures and the meta-models
- The hierarchical associations among the multi-variability types

The remainder of this chapter is organized as follows: Section 4.1 presents the architecture of multi-variability modeling framework. Section 4.2 introduces the meta-model. Section 4.3 defines the principles, the procedures and the meta-model of topology. Likewise, Section 4.4 defines the principles, the procedures and meta-model of processes. Section 4.5 associates the three variability types hierarchically to the components in the solution space. Section 4.6 discusses the limitations and lessons learned during the development of domain-specific models. Section 4.7 compares the proposed modeling approach with the prior approaches. Section 4.8 collects the evaluation goals. Finally, Section 4.9 provides the summary of this chapter.

4.1. Multi-Variability Modeling Overview

Figure 4.2 presents the overview of the (meta-)modeling framework, which is designed according to the four-layered architecture standardized by Meta-Object Facility (MOF) [OMG15]. The focus of this chapter is mainly at the M2 level in Figure 4.2 to propose the meta-models for the multi-variability modeling framework.

For variability modeling and representation, both feature modeling and decision modeling have gained the most attention in academia research [CGR+12]. Feature modeling can be used more intuitively to model both commonalities and variability, whereas decision modeling emphasizes more on variability of software systems. Feature modeling is often used just like decision modeling [CGR+12]. Other researchers [CBH11, KSRB13] report that feature modeling has been considered as the de facto method of variability modeling in software product line engineering. In the context of this thesis, feature modeling is chosen to represent the general variability. To achieve this, the modeling

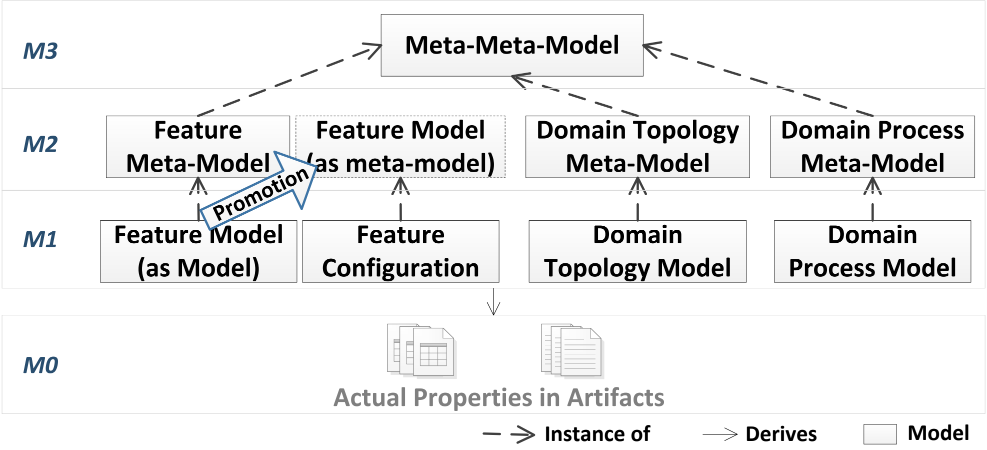


Figure 4.2.: Multi-Variability Modeling in MOF Architecture

framework includes a feature meta-model which is the first meta-model at the M2 level in Figure 4.2.

Feature models are one type of outcomes of the family engineering processes (or domain engineering / domain analysis) at the M1 level in Figure 4.2. They are the instances of the feature meta-model (at the M2 level). During application engineering, in particular product derivation, derivation stakeholders configure the feature model to decide features configurations. This requires a promotion of feature models to be used as meta-model during application engineering. With the promotion, it is possible to "instantiate" the Feature Model to Feature Configuration, when following the MOF architecture. In Figure 4.2, the arrow from the *Feature Model (as model)* to *Feature Model (as meta-model)* illustrates this promotion.

In industrial automation management systems, only using feature modeling cannot satisfy the needs of variability representation, since there are two domain-specific variability types, topology and processes, reported previously in Section 2.2.2 of Chapter 2. Three stakeholders' roles – requirement engineers, hardware-oriented engineers and software engineers – participate in derivation tasks. They have different variability concerns and perspectives [FLE+15]. The system complexities cause ineffective understanding and communication among them, due to lack of mutual understanding from others' perspectives [BWR11]. This leads to difficulties of tracing requirements to both software implementation and hardware configuration, which impedes the efficiency of development teams.

Since more than a decade, researchers and practitioners have explored and applied model-based engineering approaches in linking the software problem space and the solution space [LIA10]. Model-based techniques, such as domain-specific modeling, specify software systems at the level of abstraction beyond code, using a notation closer to the problem space [Val10]. Domain-specific models help domain stakeholders to express their knowledge in a clear and more concise way, so that the creation of the models brings the benefits of supporting communication among stakeholders and of standardizing specification [TK05, LKT04]. Additionally, subsequent tools, such as model transformation or code generators, enable the automatic configuration of software artifacts, to help software development teams to achieve better efficiency and productivity [TK05]. Therefore, the modeling framework in this chapter integrates two domain-specific models of topology and processes as variability representation mechanisms.

At the M2 level in Figure 4.2, the modeling framework contains feature, topology and process meta-models. For each of these variability types, the principles of choosing meta-elements and the procedures of developing meta-models are introduced, followed by the developed meta-models, and examples presented in the corresponding editors. The principles and the procedures can be used as guidelines to develop domain-specific meta-models to enable the multi-variability modeling for other software domains for software derivation.

4.2. Feature Variability

It is necessary to provide a common understanding of the notion of elements in feature models for this thesis, although many existing approaches have already proposed feature meta-models. This section starts the description of feature meta-modeling.

Feature modeling serves as a tree-like “language” to express the common and variable features of a software product line. Many researchers proposed “dialects” of feature modeling in academia. Kang et al. propose to use feature models in Feature-Oriented Domain Analysis (FOAD) [KCH+90]. There are also existing extensions and formalization of feature modeling based on feature relationships [KLD02, FFB02, VGBS01], and cardinality [QRD13, CHE04], just to name a few. Learning from these existing feature definitions, in this thesis the feature model is defined as follows:

Feature Model:
A feature model is a tree structure that consists of a set of feature nodes and a set of relations among the feature nodes.

The intention of this thesis is not to re-define a completely new feature meta-model. The feature meta-model shown in Figure 4.3 is established mainly based on existing works [KCH+90, CHE04]. It presents the feature meta-model and the possible relation types among feature nodes. The goal of proposing this feature meta-model in Figure 4.3 is actually to provide a (semi-)formalization and a common understanding of the notion of elements in feature models. A feature model can have arbitrary numbers of feature nodes, but only one root feature node is allowed. A feature node can have a *name* and a *description*. It also refers to a *TypedValue*, in which *Integer*, *String*, or *Float* values are possibly supported. Extension to check constraints to values may involve mechanisms, such as the Object Constraint Language (OCL) [OMG14] to describe rules for values.

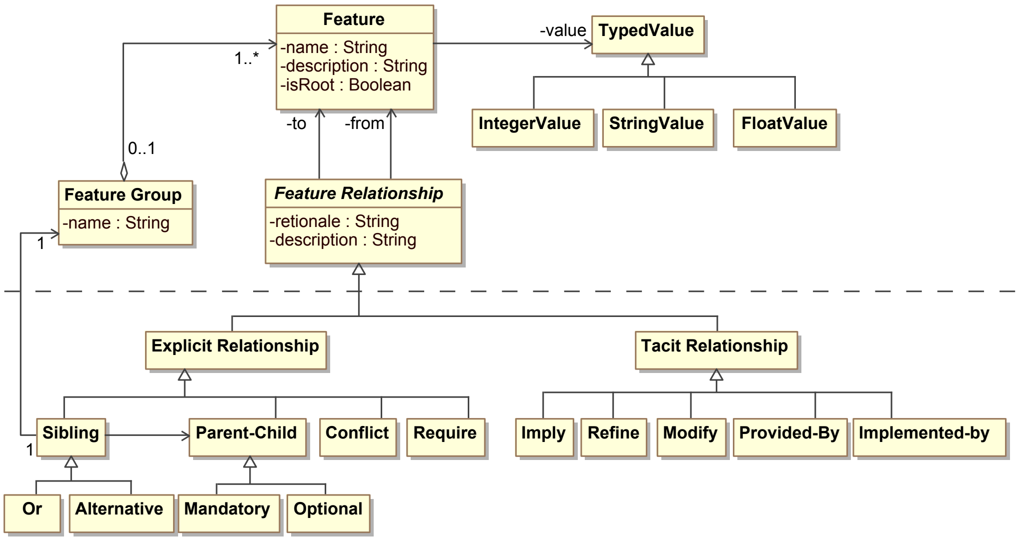


Figure 4.3.: A Feature Meta-Model

Feature Relationship in Figure 4.3 has *Explicit Relationship* and *Tacit Relationship*, which is defined according to several other feature meta-models [KCH^{+}90, CHE04, QRD13]. This meta-model does not differentiate features with types, but how features relate to other features. For example, an optional feature in this meta-model is expressed as a child feature that is in an optional relation with its parent feature. Both the *Or* and the *Alternative* relationships relate to the *Feature Groups*, in which several features or a single feature can be selected at configuration time. Two other relationships in the *Explicit* relationships are *Conflict* and *Require*, so that a feature can be either mutually exclusive to one another, or definitely needed by another feature. The *Tacit Relationship* is difficult to give absolute semantics during family engineering time. Fey et al. propose these relationships to improve understandability and usefulness of feature modeling [FFB02]. For example, the *Imply* relationship indicates a weak suggestion from one feature to anther one. The *Refine* relationship demands a more detailed description of (the services of) another feature [FFB02]. The *Modify* relationship may trigger some changes to other decisions.

4.2.1. Principles of Modeling Features

Lee et al. have proposed [LKL02] guidelines when modeling features. The subsection collects the relevant principles with regard to identification, organization and refinement of features, defined by Lee et al. in their work [LKL02]:

- **Principle 1:** Use standard terms in the product line domain to identify features.

- **Principle 2:** Do not include commonalities of implementation details as features.
- **Principle 3:** Identify features to be mapped into architectural elements, and if necessary refine them to enable such a mapping.
- **Principle 4:** Do not organize features according to their functional dependencies.

4.2.2. Procedures of Modeling Features

With the feature meta-model, feature models can be created, which are commonly essential outcomes of family engineering activities. This sub-section describes how to develop a feature model for a software family as guidelines to identify features and establishing feature models.

Input. Two types of artifacts are the input of this procedure, including requirement specifications and reusable software product line architecture.

Procedure. Based on existing guidelines suggested by [DGRN10, AC06, LKL02], Figure 4.4 shows the guideline for modeling features:

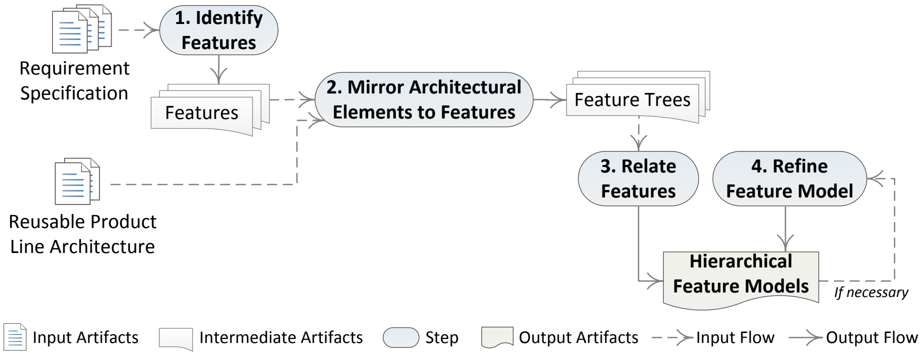


Figure 4.4.: The Procedure of Modeling Features

1. *Identify features.* The input artifacts of this step are the existing requirement specifications. Most features identified in this step are business related features, which can be categorized to essential features. For each identified feature, a unique identifier is assigned.
2. *Mirror architectural elements to features.* This step takes the reusable architecture as the input. The architectural elements such as components are mirrored as features to indicate their commonalities and variability. At this step, detailed feature-like variability inside components is identified as well. For those components with process and topology variability, it is then important to differentiate their influencing variability types, so that further process models and/or topology models can be attached later on (see Section 4.5). This step results in mostly technical features.

3. *Relate features.* Features are hierarchically structured in feature models. Commonly, essential features are at a higher abstraction level than technical features; therefore tend to be put into upper levels of the hierarchy.
4. *Refine the feature model.* The last step of establishing feature models is to refine and ensure that they are aligned with the consideration of marketing and crosscutting concerns.

Output. The output of the execution of this procedure is a feature model for the target software product line.

Role. The execution of this procedure requires deep domain knowledge and experiences. Ideally, representatives from each of the three stakeholder roles are expected. For example product managers or software architects can be good candidates to perform the tasks.

Example. Figure 4.5 shows an excerpt of the feature model developed for the running example of warehouse management systems in this thesis. The developed tooling for feature modeling is also shown in this figure. The vertical toolbar on the left side allows creating new feature nodes and modeling relationships among features.

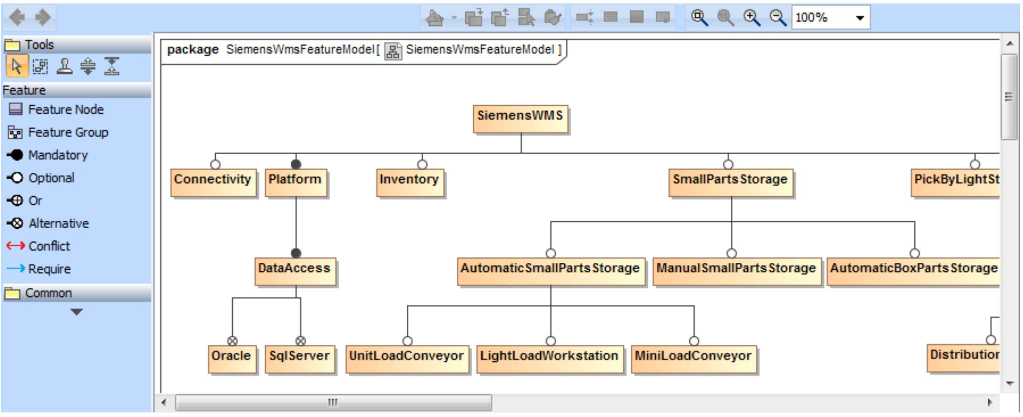
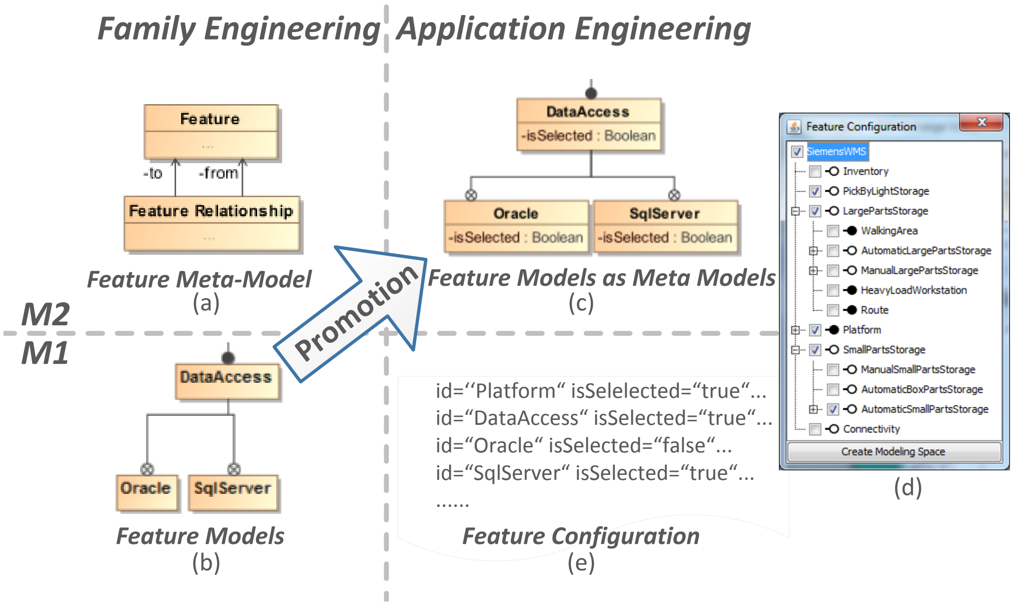


Figure 4.5.: Feature Notations with Examples

By following the principles of modeling features in Section 4.2.1 and the procedure proposed in this section, a feature model of a scoped software product line can be developed.

4.2.3. Promotion of Feature Models to Meta-Models

Feature models play different roles during family engineering (or domain engineering) and application engineering. During family engineering, feature models are the outcomes, whereas during application engineering, feature models serve as meta-models for support application-specific configuration. Figure 4.6 presents the promotion of feature models to meta-models. The models are separated into M2 and M1 levels, aligned with MOF levels in Figure 4.2.



id="Platform" isSelelected="true"...
id="DataAccess" isSelected="true"...
id="Oracle" isSelected="false"...
id="SqlServer" isSelected="true"...
......

☑ SiemensWMS
- □ Inventory
- ☑ PickByLightStorage
- ☑ LargePartsStorage
  - □ WalkingArea
  - □ AutomaticLargePartsStorage
  - □ ManualLargePartsStorage
  - □ HeavyLoadWorkstation
  - □ Route
- ☑ Platform
- ☑ SmallPartsStorage
  - □ ManualSmallPartsStorage
  - □ AutomaticBoxPartsStorage
  - ☑ AutomaticSmallPartsStorage
- □ Connectivity

Figure 4.6.: Promoting Feature Models as Meta Models for Configuration

During family engineering time:

- *(a) Feature meta-model*: Figure 4.3 has already presented the complete feature meta-model used in this thesis. This model is generic for the configuration tooling of feature modeling. Figure 4.6 (a) only includes a small excerpt of the feature meta-model to illustrate the concept.
- *(b) Feature models as instances*: Feature models are the instances of the feature meta-model. They are usually the important outcomes of domain engineering or family engineering activities. To illustrate the concept, Figure 4.6 (b) presents the surface syntax (the notations) of three features and their relationships. *DataAccess* is the parent feature of *SqlServer* and *Oracle*. *SqlServer* and *Oracle* are sibling features, that they are alternative to each other.

During application engineering time:

- *(c) Promotion of feature models to meta-models*: For configuration and derivation, feature models are no longer instances. Instead, they are transformed to meta-models for configuration during application engineering. In other words, the Feature Models in (b) and Feature Configuration in (e) are both at the M1 in MOF architecture, which means that during application engineering time, the Feature Models are pushed up to M2 level. The arrow from Figure 4.6 (b) to (c) represents the promotion transformation [dLGC13], which adds a new attribute *isSelected* to each feature node. A possible alternative to avoid having the promotion is to allow the direct instantiation of feature models as feature configuration. This would require adding some attributes of application engineering into feature meta-model. For example, the *isSelected* attribute, is not necessary to be explicit in feature meta-model tagged in (a), since whether a feature is included or not is only relevant during configuration in application engineering.

- *(d) Tool support for feature configuration*: Tool support has been developed in this thesis for feature configuration. It traverses the transformed feature models and displays them as feature trees in the Feature Editor. At run-time, derivation stakeholders use the editor to do feature configuration to fulfill customers' need. The Feature Editor in Figure 4.6 (d) is the screen-shot of the implementation as a MagicDraw Plugin.
- *(e) Feature Configuration*: The feature configuration is the list of selected and configuration features. They are the output of the Feature Editor.

4.3. Topology Variability

A topology, in the context of this thesis, describes the configuration upon a set of distributed real-world elements in a particular zone in a targeting manufacturing environment. These real-world elements include software components, hardware devices and mechanics tools that are interacting with IAM systems. With the running example shown in Figure 4.7 (introduced already in Chapter 2), *Customer 1* has only two conveyors in the receiving zone, whereas *Customer 2* has at least four conveyors to perform the same tasks. There are many constraints affecting the decision of selection and configuration, such as temperature, frequency interference, cost, budget, etc. For example, in some manufacturing domains, there are regulations to separate the imported components and locally produced components for storage. Many of these constraints in topology configuration are not software constraints, however, they may restrict software development and derivation.

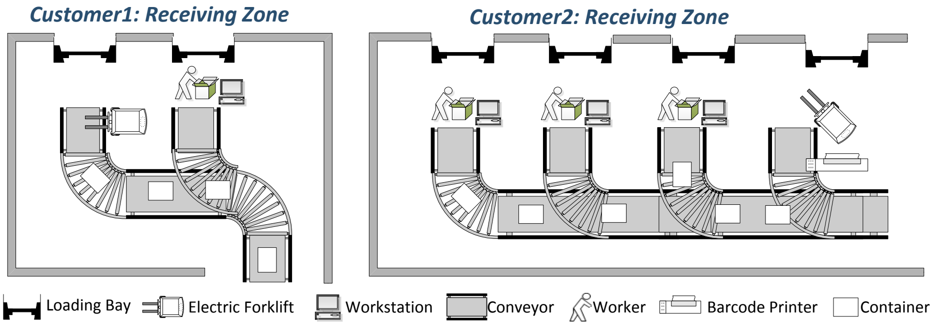


Figure 4.7.: Two Examples of Topology Configuration from Running Example

A topology model describes a set of real-world elements and their relationships within a given manufacturing zone, as defined in Chapter 2 (see section 2.2.1). In IAM system, using topology models to capture the configuration of the elements and their relationships enables the representation of topology variability.

It is important to emphasize that not all information that can be extracted into a topological model are critical for software derivation. Therefore, this section introduces the principles and modeling procedures, which result in a topology meta-model to represent the topology configuration for the systems in industrial automation management to support software derivation.

4.3.1. Principles of Modeling Topology

Many existing SPLE techniques apply class models of the Unified Modeling Language (UML) to express topology variability in various software domains, such as integrated control software [BYBS13], fire alarm systems [BSØ+14], or industrial metal technology [DSL+14]. Although some solutions for modeling and configuring topological variability are available, there is no well-established method can be used to develop topology meta-models, to the best of the author's knowledge. Especially, no ready-to-use principles available clearly define how to define the topology meta-elements and their relationships. Therefore, this section proposes design principles of meta-modeling, aiming at filling this gap.

To collect the principles for practitioners, who use the proposed approach in this thesis, it is important to use a systematic method to identify entities in real world and reflect them into meta-elements in meta-models. Isoda [Iso01] differentiates two object-oriented modeling methods: **natural** and **pseudo** real-world modeling.

According to the suggestions of Isoda [Iso01], natural (or genuine) real-world modeling identifies classes based on entities in real world and relationships among them. It assigns attributes and operations to classes according to the natural properties and functions of the entities. Figure 4.8 uses the running example introduced previously in Chapter 2 (see Section 2.2.1).

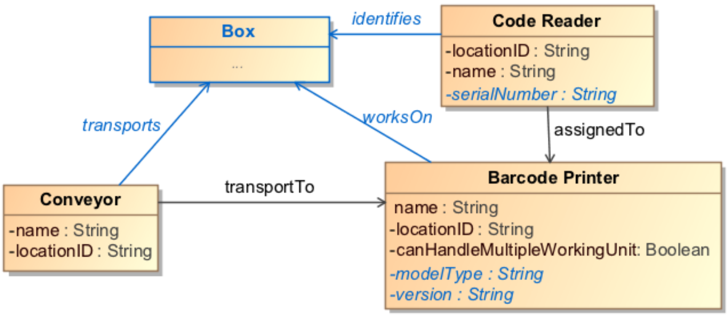


Figure 4.8.: The Differences of Natural and Pseudo Real-World Modeling

In this running example shown in Figure 4.8, a *Conveyor* carries *Boxes*, *Code Reader* detects the boxes, and the *Barcode Printer* prints a new matrix barcode for the boxes, etc. Following this method, the meta-model includes attributes, such as *version* for *Barcode Printer* or *serialNumber* for *Code Reader*.

However, pseudo real-world modeling identifies the entities selectively, according to a particular purpose [Iso01]. In the context of this thesis, it should focus on software development and implementation information. From this perspective, attributes, such as *serialNumber* of *Code Reader* in Figure 4.8, are no more relevant to software derivation and configuration, even though they are important during hardware purchase time and installation.

> **Pseudo Real-World Modeling:**
> Pseudo real-world modeling represents the real world in a class diagram by way of identifying classes, their attributes, operations and relationships, which are relevant for the software derivation purpose [Iso01].

Furthermore, an IAM system has actually no direct communication and information exchange with boxes. Instead, IAM systems use either barcodes or possibly radio-frequency identification (RFID) chips to identify each of the boxes. In this case, they do not need to appear in the topology meta model according to the pseudo real-world modeling approach, since the *Box* is a run-time notion, which is out of the scope of topological variability.

In contrast, the three "long-living" classes, the *Code Reader*, the *BarcodePrinter*, and the *Conveyor*, have *locationID* attributes according to the pseudo real-world modeling approach, which carry the location information for IAM systems. In other words, IAM systems recognize the topology elements by their values of *locationID*. The actual real-world entities are more important to be understood by derivation stakeholders or modelers, compared to IAM systems. It is easy to neglect such attributes without applying pseudo real-world modeling, which requires to take the information viewed by the management systems into consideration.

According to the modeling rules of pseudo real-world modeling, this thesis proposes the following principles for designing meta-elements for topology configuration in IAM systems:

- **Principle 1**: *Identify the classes that correspond to the entities whose information is dealt with by the IAM systems.* The goal of developing topology meta-model is to model and configure topology variability to support the software derivation in the context of this thesis. Therefore, the modeled elements should have information dealt with by IAM systems to make sure that they are relevant during software derivation.
- **Principle 2**: *Choose "long-living" entities in real world to be in the meta-model.* This principle helps on looking for the topological related entities. For example, the *Box* identified in Figure 4.8 based on natural real-world modeling is not a

"long-living" entity in the scope of the running example, even if it is a natural real world element in factories.

- **Principle 3**: *Exclude entities that are only relevant at run-time.* This principle is complementary to Principle 2, to limit the complexity of the topology meta-model. For the purpose of on human understanding, some run-time entities may still be modeled. However, after configuration, the model interpreter will neglect these instances of elements and their attributes, since they are not relevant to software product derivation of IAM systems.
- **Principle 4**: *Assign the attributes to the modeled elements that are dealt with or serve for interaction with the IAM systems.* This principle ensures that the attributes assigned to the elements in meta-models are relevant with respect to the modeled pseudo real world.
- **Principle 5**: *Do not model operation in the topology meta-models as for the purpose of IAM derivation.* Isoda points out that even when an entity has a function, we do not necessarily have to assign it to the class corresponding to the entity, because the function itself is merely information to IAM systems [Iso01]. In IAM systems, the functions are also behaviors of individual hardware devices. These are not important information to derivation, as IAM systems are at the higher management level to coordinate these devices.

4.3.2. Procedure of Topology Meta-Model Development

A procedure is important to guide the development of the topology meta-model, for example, to filter out unnecessary meta-elements, and to avoid unnecessary data to be modeled in topology models. The goal is actually to harness the complexity of topological configuration and to focus on the derivation-critical topological elements.

Role. A domain expert with certain modeling experience is expected to guide the execution of this development procedure. Besides, hardware-oriented engineers need to also participate especially during Step 3 and 4 for re-factoring and relating meta-elements. In practical settings, it is rare that a modeling expert has solid domain knowledge in industrial automation management, and vice versa. Thus, the participation of these two roles is both important to the meta-model development.

Input. Explicit input artifacts are the domain models and factory layout in requirement specifications of the targeted product lines. Typically, a domain model captures the real-would concepts, which represents also the common vocabulary in the domain. A UML class diagram is often used to represent domain models, for example, the natural real-world modeling shown in Figure 4.8 without pseudo real-world analysis.

Procedure. Figure 4.9 shows the procedure of developing the topology meta-model. It follows a bottom-up approach to develop domain-specific models, which extends the procedure suggested in [SCDLG12]:

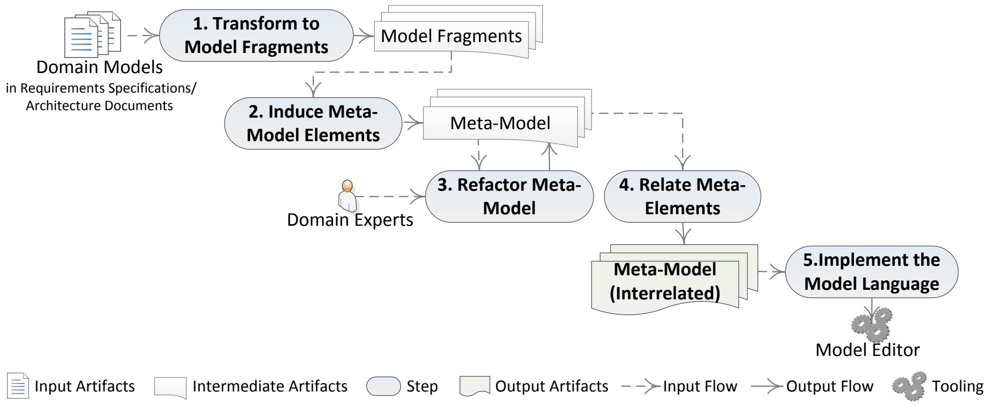


Figure 4.9.: A Bottom-Up Procedure of Modeling Topology

1. Domain experts transform existing domain models as model fragments. Usually, such domain models are in the format of UML class diagrams in the requirement or architecture documentations. Real-world elements such as conveyors or racks are identified in this step.

2. Based on the domain model fragments, the meta-elements of the topological elements are further induced and determined. For example, the *Conveyor* can be further induced as a transporter type.

3. At this re-factoring step, the domain experts should be involved, as they are the potential users of the domain-specific models. They provide the information that cannot be identified by "mining" the legacy documents.

4. Domain experts relate the meta-elements, according to their sub-types, and identify the relationships among the elements, which also would be an important part of the meta-model.

5. At this step, a concrete modeling platform must be decided to develop the domain-specific model. Each meta-element needs a shape and possibly an icon for representation purposes. Each relation is usually represented as a connector between the elements. The idea is to make the developed domain-specific models to be intuitive to the model users.

Output. The output is the developed meta-modeling elements for topology. In the next subsection (see section 4.3.3 as an example), the developed topology meta-model will be described along with the supporting model editor. More details about the final topology meta-model with the modeling language editor will be introduced as well in the next subsection.

4.3.3. Topology Meta-Model with Configuration Tooling

This section presents the developed topology meta-model of IAM systems. The meta-model has two parts: core meta-model in Figure 4.10 and a supplementary meta-model in Figure 4.11. The core meta-model includes the abstract elements, to present the relationships among the core meta-elements. In Figure 4.10, a topology model may have an arbitrary number of topology elements. A *Topology Element* can be assigned to another topology element. *Transporter Element* and *Stationary Element* are the two sub-types of *Topology Element*. They are specific to the IAM domain. The *Transporter Element* refers to the mobile elements in factories, such as staplers or conveyors, which bring the working units from one location to another. The *Stationary Element* represents fastened work places or devices that they actually perform the concrete manufacturing tasks in factories.

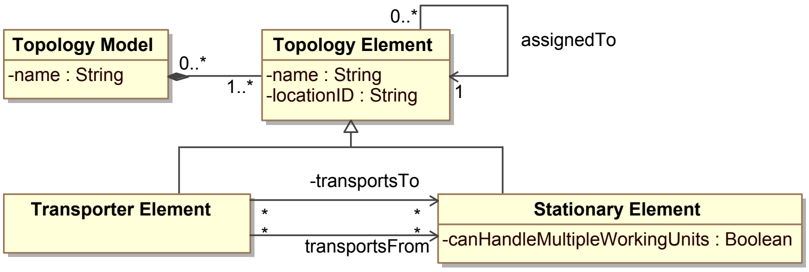


Figure 4.10.: The Core Topology Meta-Model

During the development of the IAM topology model, the supplementary meta-model in Figure 4.11 is separated from core topology meta-elements to reduce the complexity. The reason is that the number of individual types of topological elements increases dramatically, when having an exact one-to-one mapping from hardware devices to topological elements. For example, within one factory, there could be 60 heavy-load staplers with 5 different load limitations, purchased from different manufacturers. These staplers with different versions and providers serve the same functions for IAM systems to transport materials to support manufacturing tasks. Therefore, they can be still abstracted with the core meta-element as a transporter element.

Figure 4.11 presents the supplementary topology meta-model, which includes the sub-types of *Transporter Element* and the *Stationary Element*. As can be seen, in IAM systems, there are different types of conveyors, which automatically transport raw materials or working units inside manufacturing plants; staplers serve for transporting heavy materials, which are manually driven by human workers. Stationary elements include various devices that are fixed at a particular location in factories.

It is notable that these supplementary elements shown in Figure 4.11 are not meant to be complete for every sub-domain of IAM systems. A physical devices library [BSØ$^+$14] is ideally necessary to allow flexibly adding more topological elements even at run-time to have a full adoption in practice.

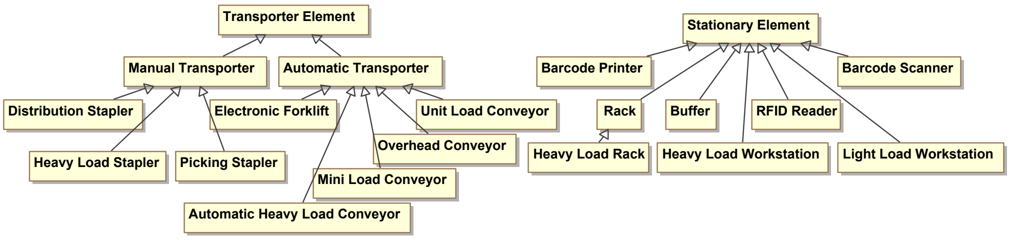


Figure 4.11.: The Supplementary Topology Meta-Model

Figure 4.12 presents the tooling for topology configuration, according to the proposed meta-model. On the left side, it shows the vertical toolbar of different topological meta-elements. On the right side, it shows an excerpt of topology configuration. As can be seen, three instances of light load workstations are connected by the mini-load conveyors to a buffer for temporarily storage with corresponding locationIDs.

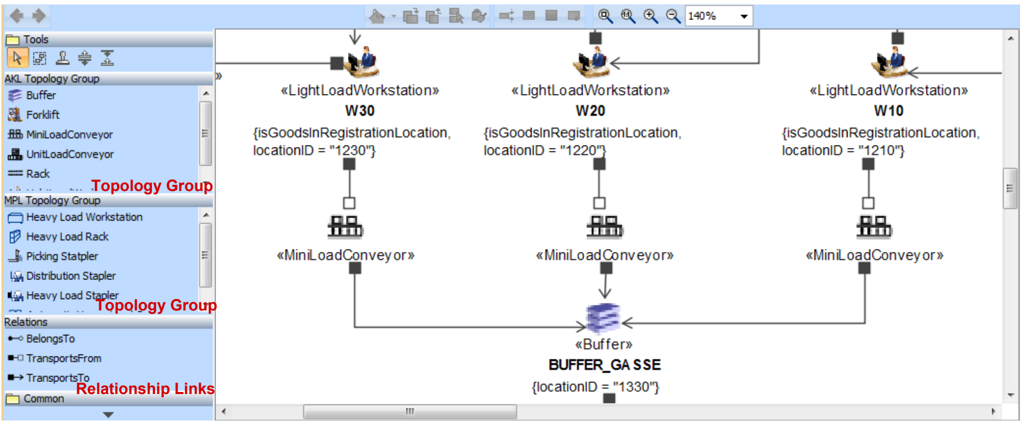


Figure 4.12.: Topology Editor – The Tool Support for Topology Configuration

4.4. Process Variability

A process describes a behavioral sequence of a system function. A process model contains a set of action nodes and edges among the action nodes. The actions in this context are usually self-contained that can be combined to form a process.

> **Process Model:**
> A process model contains a number of actions and describes their execution sequence [RPB99], which captures a behavioral flow of a system function.

At the modeling time of processes in the context of this thesis, processes can be configured by changing the sequence of execution, adding new actions, or deleting some actions from process models. This section presents the principles, the development procedure, and the developed process meta-model for the target domain.

4.4.1. Principles of Modeling Processes

The development of a useful domain-specific model for processes is non-trivial. The decision of developing such a model is as an enabler of reuse of software artifacts. This subsection clarifies the principles of developing the meta-model.

- **Principle 1**: *Focus on the declarative aspect of the modeling notations.* The developed process meta-model is with a declarative and expressive purpose for expressing behaviors of IAM systems as processes. The modeled process is one part of the input of the derivation infrastructure during derivation, instead of workflow engines with execution purposes [MHS05].
- **Principle 2**: *Base on existing process meta-modeling where possible.* In contrast to topology meta-modeling where practitioners commonly use class diagrams in UML [BSØ$^+$14], process meta-modeling has much more variations such as Event-Driven Process Chain [vdA97], BPMN [OMG11], UML Activity Diagram extensions [TZD08], etc. Considering the costs on development and maintenance of meta-modeling, it is more reasonable to "reuse" existing process meta-elements as much as possible. The development of a process meta-model should base on existing process modeling notations, where possible. After all, the investment on developing such models should economically pay-off [Wil03, MHS05]. For example, in the context of this thesis, the meta-elements were inspired by the TORE approach for information systems [ADEG09, Ada12, Doe10].
- **Principle 3**: *Develop the process meta-elements in an iterative manner.* With the decision of enabling reuse. The analysis of the IAM domain results in an abstraction of behavioral sequences. For this, the participation of domain experts is definitely necessary [SCDLG12], even some technical experiences can be critical to the success of the developed meta-model for processes. The development and refinement require usually an iterative manner.

4.4.2. Procedures of Process Meta-Model Development

The expectation of proposing the process meta-model is to represent the process models in IAM domain. Similar to the procedure of topology meta-modeling, the procedure of process meta-model development follows a bottom-up approach.

Input. The input artifacts are the process descriptions existing in requirement specifications or architectural documentations of the targeted product lines.

Procedure. Figure 4.13 shows the procedure of developing the process meta-model. Accordingly, it follows a bottom-up approach to develop domain-specific models, proposed by Cuadrado et. al [SCDLG12]:

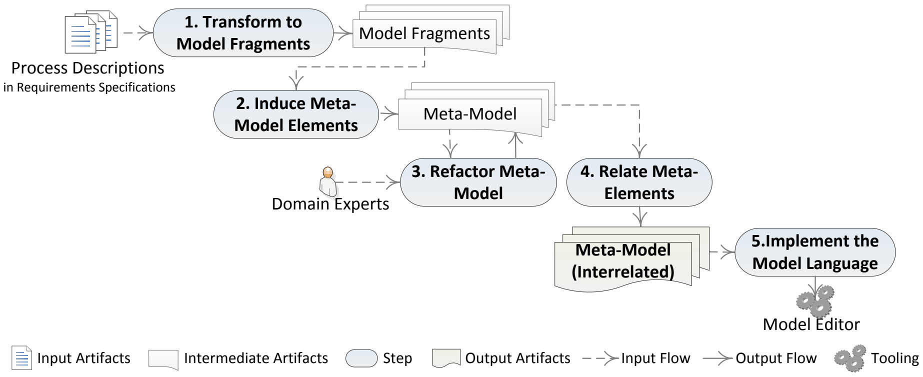


Figure 4.13.: A Bottom-Up Procedure of Modeling Processes

1. The domain expert analyzes existing requirement documentations, especially process specifications. He or she extracts the types of process steps from the description and transforms them into model fragments.
2. Based on the model fragments, the domain expert derives and determines the meta-models of the process elements. For process descriptions, subjects and verbs usually indicate important information for extracting process meta-elements [NE08, MBK91]. Here, using technologies in natural language processing may significantly improve the identification of process meta-element types, because verbs and verb-directed objects may indicate information for functional requirements[1] [NE08]. How to better utilize the techniques to automate the meta-element finding, however, is out of the core scope of this thesis, and should be considered as future work.
3. At the third step, the domain experts should be involved to support the decisions of meta-element creation. Sometimes, there can be hidden types from technical viewpoints, which requires domain knowledge.
4. The meta-elements should be interrelated. Dependencies and constraints for process models can be identified as well. For example, usually user-system-interaction has a weak dependency to a software display action, so that a user can interact with the displayed user interface.
5. At this step, a concrete modeling platform must be selected to develop the domain-specific model. Each meta-element needs a shape and possibly an icon, as well as the connectors among the elements.

[1] The author of this thesis uses two real process specifications of legacy IAM systems. A process specification in German contains 499 sentences in which 95 verbs are used. Typical "verb-object" pairs found in the specification test are for example:"schickt Auftrag" appears 7 times or "aktiviert Auftrag" appears 2 times. Both of them indicate user-system-interactions. A process specification in English contains 366 sentences, in which 57 verbs are used. Typical "verb-object" pairs found in this specification are for example: "creates delivery" appears 25 times; "selects order" appears 19 times; "changes delivery" appears 16 times. These corpora also indicate user-system-interactions.

Output. The developed meta-modeling elements for process will be introduced in the next sub-section.

Role. Similar to the development of the topology meta-model, domain experts need to participate in this procedure. The ideal role who participates in this development procedure should have both domain knowledge and technical modeling background.

4.4.3. Process Meta-Model with Configuration Tooling

Based on the three principles and the previously defined procedure, this section presents the process meta-model. Figure 4.14 presents two parts of the meta-model regarding the actions and the edges of the IAM domain. Each process consists of multiple *Actions* and *Edges*. An *Action* represents a single atomic step within the MES processes. An *Edge* represents the execution sequence of actions, which has usually an incoming action and an outgoing action.

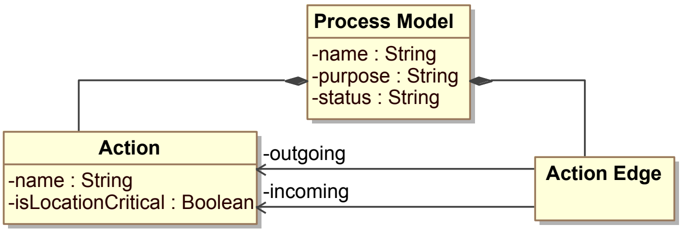


Figure 4.14.: The Process Meta-Model

Figure 4.15 shows the part of meta-model of actions. There are typical actions for information systems, such as *Software Action* and *Human-System Interaction*. There are also several domain-specific actions. A *Manual Action* can be used to model manual steps performed by human workers. This is typical for the non-automated tasks in manufacturing factories. The *Automated Action* is defined for representing the automated tasks, commonly performed by hardware devices. The *Periodic Action* has usually a loop time to re-execute.

- A *Human-System Interaction* is an interaction between users and the IAM systems, which usually requires a user interface.
- A *Manual Action* is a task that is performed by human workers. A task description is needed to guide the workflow. Although manual tasks are not actually performed by software, they are still potentially coordinated by IAM systems. For representation purpose, it was necessary to include this type in the meta-model.
- A *Software Action* is an abstract type of a software task.
- A *Display Action* is a sub-type of the software action for showing information on a user interface.

- A *Computational Action* is a sub-type of the software action for computing or calculating tasks.
- A *Periodic Action* is an IAM-specific task to be triggered periodically by an internal timer, or by external monitoring components.
- A *Control Action* is an abstract action defined for controlling the processes.
- An *Automated Action* is IAM-specific. It represents the automated functions in the factory, which are not actually performed by an IAM system, but coordinated, demanded, and triggered by it. These actions are actually executed and performed by hardware devices.

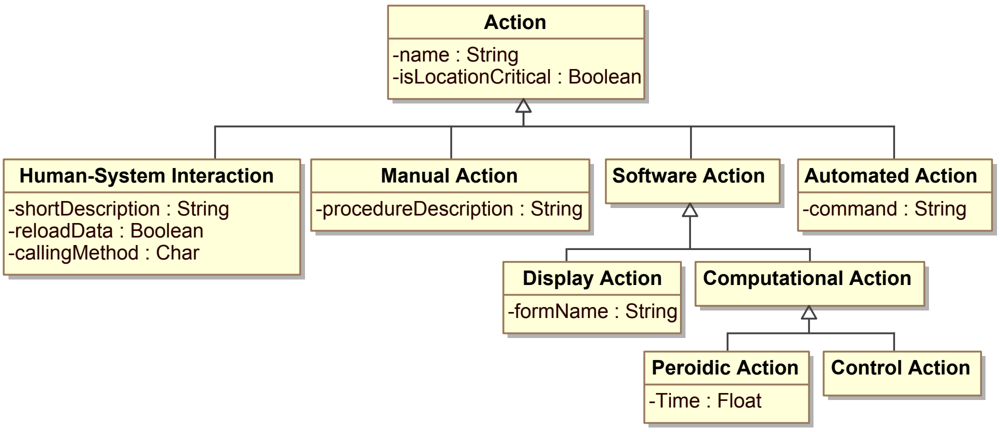


Figure 4.15.: The Action Types in the Process Meta-Model

Figure 4.16 shows the further types of control actions in IAM systems.

- A *Decision* is a typical control action in IAM processes to check a condition.
- A *Merge* action usually joins parallel execution of actions together.
- A *Thread Fork* is defined as a group of tasks to be performed in parallel.
- A *Thread Join* represents a join action after thread fork.
- A *Sync Fork* is defined for a group of tasks, in which each task should be performed one after another.
- A *Sync Join* represents a join action after sync fork.

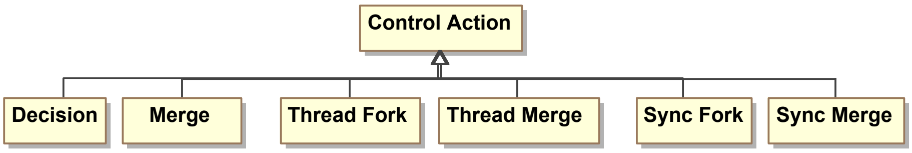


Figure 4.16.: The Control Actions in the Process Meta-Model

Figure 4.17 shows the types of edges in the process meta-model. Three out of the four types of edges in Figure 4.17 are also IAM-specific, except the control flow. The *Material Flow* represents the movement or location changes of working units in real world (see Figure 2.6 in Chapter 2). The *Manual Flow* represents the sequences of human-worker workflows. The *Message Flow* represents the messages sent to hardware systems outside IAM system implementation, or a received signal from external systems observed by IAM systems.

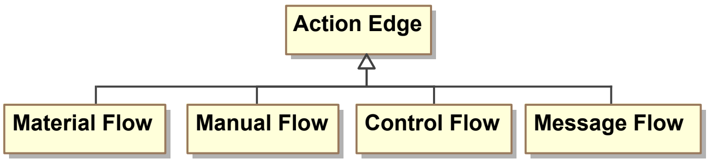


Figure 4.17.: The Edges in the Process Meta-Model

The tooling for process configuration has been also implemented in MagicDraw. Figure 4.18 presents the developed process editor based on the proposed meta-model. The vertical toolbar on the left hand side collects the meta elements of different process-action types. The lower part of the vertical toolbar includes the four action edges in Figure 4.17. The canvas on the right hand side of the screen-shot shows an example of a process, in which all elements are periodic actions.

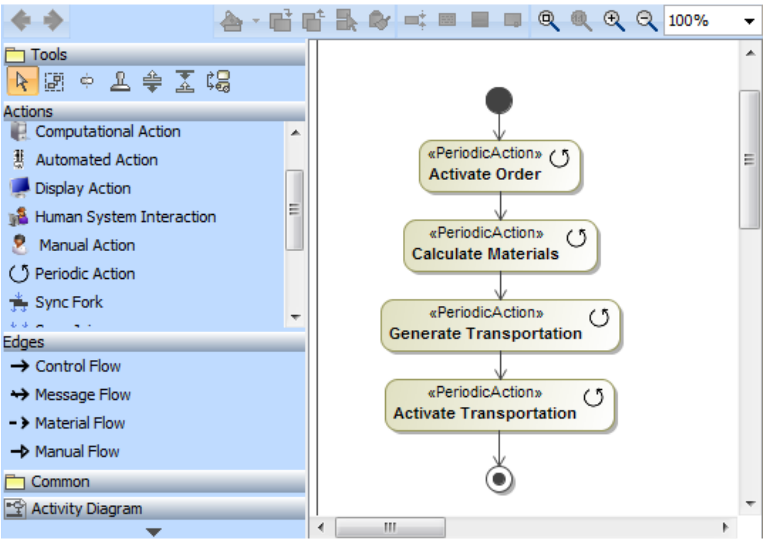


Figure 4.18.: Process Editor – The Tool Support for Process Configuration

4.5. Hierarchical Multi-Variability Modeling

It is possibly to link features to architectural building blocks [SGEL09], while linking features to all kinds of core assets in different abstraction levels is difficult [SGEL09, FLE+15]. The association between the *Feature* and the *Component* in Figure 4.19 shows this situation.

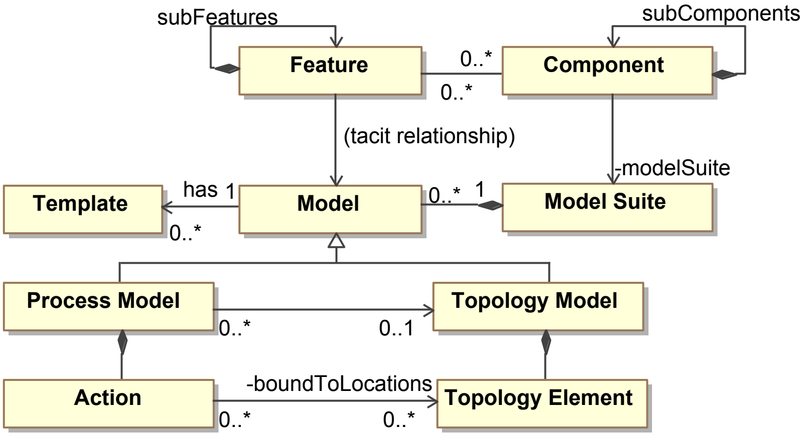


Figure 4.19.: Hierarchical Multi-Variability Associations

Establishing multiple (domain-)variability models for representation purposes is not the ultimate goal of this approach. The associations among them are necessary to have, so that they can be systematically instantiated and integrated in a derivation process. Figure 4.19 shows how the feature, process and topology (meta-)elements are associated in this thesis.

As presented in Figure 4.19, features may contain sub-features hierarchically. They are related to components and sub-components, which describe the software product line architecture. When a component has further process or topology variability, the corresponding variability models are added to a model suite. Such a model suite contains the information about the names, types and paths of the variability models. Each involved model can have model templates, which are extracted during family engineering. During application engineering these models are instantiated with appropriate templates, to serve as base configurations to help derivation stakeholders to achieve better configuration productivity.

There can also be tacit relations and dependencies among *Features* and domain-specific variability *Models*. For example, a feature may imply a weak suggestion of another feature—or may require a more detailed description in a process model configuration. Existing works already propose solutions to solve these types of relations [WMW11, GBPG13]. In our target systems, we identify domain-specific relations between the process and topology models. In process models, there can be actions that are bound to certain topological locations (see the running example in Figure 2.9

in Section 2.2). To describe this situation in models, we need the *boundToLocations* relation between the *Action* and the *Topology Element* as shown in Figure 4.19.

4.6. Discussions and Limitation Analysis

This section discusses the lessons learned and limitations of the proposed *Hierarchical Multi-Variability Modeling* approach. The design of a domain-specific model is a difficult task because different stakeholders as the users of the models have various perceptions of what a “good modeling language” actually is.

- **The influence of different modeling tools on the users and their perceived acceptance of the developed approach should not be underestimated.** The same meta-model implemented on top of different modeling platforms may significantly influence the users' acceptances of the approach. Potential users should be familiar with the chosen modeling platform, since each modeling platform provides a different user experience. It would potentially increase the probability that they accept the proposed modeling notations, when the implementation is based on a modeling tool that they use in their requirements or architectural design. For this reason, we have evaluated several modeling platforms, such as Enterprise Architect [ea], Eclipse-based modeling tools [ecl], MagicDraw [mag] and Microsoft Visio [vis]. MagicDraw was chosen for the implementation of modeling platform. However, this aspect may not be critical from a scientific perspective. This “favor” of modeling platforms can be individual to organizations, which is not considered critical to this thesis.

- **The support to ensure dependency and consistency during configuration.** This chapter includes feature, topology, and process meta-models to represent the variability in IAM systems. For each for the modeling type, the author of this thesis has developed a configuration tooling. During configuration, when feature is selected or de-selected, the impacted topology and process models are instantiated. By doing so, the dependency among configurations and decisions are proactively addressed to a certain extent. More details about configuration process will be in Chapter 6. To enhance the dependency and consistency checking in the approach, other configuration support could be possibly integrated to the configuration tooling. For example, Kowal and Schaefer present an incremental method to achieve consistency checking among several UML-based variability models, in which intra-, inter-perspective and cross-variant rules are defined and added in to a tool chain [KS16]. Mendonca et al. propose to use interactive modeling and reasoning services for consistency diagnosis [MBC09]. White et al. report on their method to automatically diagnose errors in configuration and repair invalid feature selections [WBS$^+$10]. These functions are not yet integrated in the tooling presented in this chapter, but have been considered as a part of future work (c.f. Section 8.2).

- **The generalizability of the proposed approach and the meta-models.** This chapter presents the two domain-specific models of topology and processes for IAM. The developed meta-models might have their limitations to be generalized to every sub-domain of IAM systems. Proper adaptations and extensions are necessary, when the approach is adopted within a concrete context and environment. Still, the overall modeling framework, together with the principles and development procedures are meant to be generic. The proposed modeling framework aims at the whole IAM domain.

4.7. Comparison with Related Work

Chapter 3 has summarized and discussed the state of the art. This section aims at analyzing the differences of the proposed modeling framework proposed in this Chapter.

The overall modeling framework shown in Figure 4.2 follows the MOF architecture. It includes the feature meta-model, the promotion of feature models as meta-models, the domain topology meta-model and the domain process meta-model. The work of this thesis identifies the two domain-specific variability types and then integrates them into a variability model framework, which is the first uniqueness of the contribution. In addition, for each variability type the principles and procedures to develop the meta-models are also proposed in this chapter.

The rest of this section further compares each of the individual variability types in this thesis and their modeling techniques with related work.

4.7.1. Related Work in Feature Modeling

Since the initial proposal of feature modeling by Kang et al. [KCH$^+$90], many researchers and authors have proposed extensions of feature modeling to be part of their own methods. Several authors propose remarkably approaches to adopt formal semantics to feature diagrams [HST$^+$08], such as a textual feature description language [VDK02] and free feature diagrams [SHTB07]. Heymans et al. point out that formal semantics bring the benefits to avoid ambiguities, which is critical to support further validity checking [HST$^+$08]. The feature meta-modeling presented in this thesis is not intended to re-define a completely new feature semantics, with model checking purposes. It is established mainly based on existing works [KCH$^+$90, CHE04], to fit to the MOF architecture, and more importantly to be integrated to the configuration process with the other two variability types.

In addition, the feature meta-model used in this thesis does not include all possible tacit relationships, based on existing works [KCH$^+$90, KLD02, FFB02]. The goal of having the feature meta-model in Figure 4.3 is to clarify the understanding of elements in feature models in the context of this thesis, as well as to enable the integration with other variability modeling types. The tacit relationships may expand the modeling space, but are difficult to interpret by the derivation infrastructure. Therefore, the tacit relationships are intentionally not modeled, because the goal of this thesis is to support software product derivation.

There are commercial and open source feature modeling tools, such as *pure::vairants* [pur] and *FeatureIDE* [KTS$^+$09]. The intention to develop tool support shown in Figure 4.5 and the feature editor in Figure 4.6 (d) is however to ease the integration with the variability modeling approach with the other variability types in this thesis. Comparing to these existing tools, the developed tooling may not provide the functions as good as commercial tools, for example as pure::vairants. But it is sufficient for modeling and configuring features, and it is flexible to integrate to the remaining part of the modeling framework.

4.7.2. Related Work in Topology Modeling

Compared to existing topology modeling approaches, such as Behjati et al. [BYBS13] or Berger et al. [BSØ$^+$14], the author of this thesis emphasizes that the completeness of topological elements existing in real world should not be the goal of topology modeling. For example in [BYBS13], the authors propose that "...the hardware model should provide a containment hierarchy that is complete with respect to the following criteria...the hierarchy should contain all the hardware computing resources that have a configurable software class deployed to them [BYBS13]...", that is to find a complete set of all possible hardware devices that have a piece of configurable software running on them. This is neither necessary nor possible to be fully modeled in case of the IAM domain. In a real automation factory, there can be thousands of different sensors, controllers and actuators distributed in different functioning zones. Not every of them is critical for IAM software derivation.

In contrast to existing work, one important challenge of topology modeling is to identify the relevant elements in the meta-model and harness the complexity. The principles in Section 4.3.1 give clear criteria to limit the numbers of elements to be modeled in topology. Therefore, in this thesis, the proposed principles and the core topology meta-model provides the opportunities to overcome the challenge.

The domain experts with knowledge especially on technical aspects of the targeted IAM product families participate into the design and development activities can be critical to the creation of the topology meta-model. This is closely related to one of the assumptions (see Section 1.3.2 in Chapter 1), that organizations have already accumulated legacy artifacts of the software produce line. The approach of topology meta-modeling is more suitable for extractive or re-active product line adoption.

4.7.3. Related Work in Process Modeling

The idea is to develop and integrate the process meta-model into the approach in order to provide a customized notation of process modeling in the target domain. Compared to non-tailored process modeling approaches, using the proposed notation actually restricts process configuration during derivation time.

Brown et al. propose to attach behavioral models to feature nodes to compensate the representation limitation of feature modeling, and to represent weak sequential dependencies among features [BGB^{+}06]. However, their approach cannot fundamentally solve the challenge in IAM systems, which requires linking features to more fine-grained assets. Heinzemann and Becker propose to use adapted concepts of the UML, named MechatronicUML, to enable hierarchical re-configuration of complex software components for self-adaptive mechatronic systems [HB13], which was inspiring for this study. In MechatronicUML, however, topology variability is not considered. There are also approaches aiming at generic business process families [GBPG13, WMW11, SP06].

Business process modeling notations (BPMN) [OAS] are developed to represent business process models. These works are originated mainly from research areas such as business process management. Business processes are usually meant to be executable, which can be run directly on top of a Business Process Management System, defined by La Rosa [LRVdADM17]. However, having executable processes requires the software families to have very high maturity level, i.e. at the configurable level. At the same time, the software domain must be relatively stable. For some software domains, this could be almost impossible to achieve, especially for IAM systems in this thesis. In contrast, the process model presented in this chapter is conceptual. The configuration of the process models is given to the developed product line infrastructure and the code generator to support derivation. With the proposed process-meta model, it is possible to express the IAM-specific relationships, for example the bound-to-locations relations between process steps and locations.

4.8. Validation Objectives

The evaluation of feasibility (H1) by using the developed multi-variability models and editors is important to understand whether they can be applied in an industrial environment.

> **H1: Feasibility.** It is feasible to apply the approach in family engineering.

The expectation of using a variability modeling framework covering multiple variability types is that more assets, especially fine grained assets should be included into the core asset base of software product line. This is therefore related to Hypothesis 2 (H2).

> **H2: (To characterize) the composition of multiple variability models.** The multi-variability modeling, by introducing process and topology models, can capture significantly more variability (>50%), compared to feature modeling.

Furthermore, the validation needs to prove that the developed modeling framework can be easily accepted by derivation stakeholders (H4). For instance, having the approach should improve the satisfaction of them during derivation tasks comparing to their current working situation without the approach in practice.

H4: Users' perceived usefulness and ease of use. The approach in application engineering can be easily accepted by users for their derivation tasks.

4.9. Chapter Summary

This chapter provides the answer to RQ1 about the representation of multiple variability types. Figure 4.20 shows a summary of the core meta-models and their associations presented in this chapter.

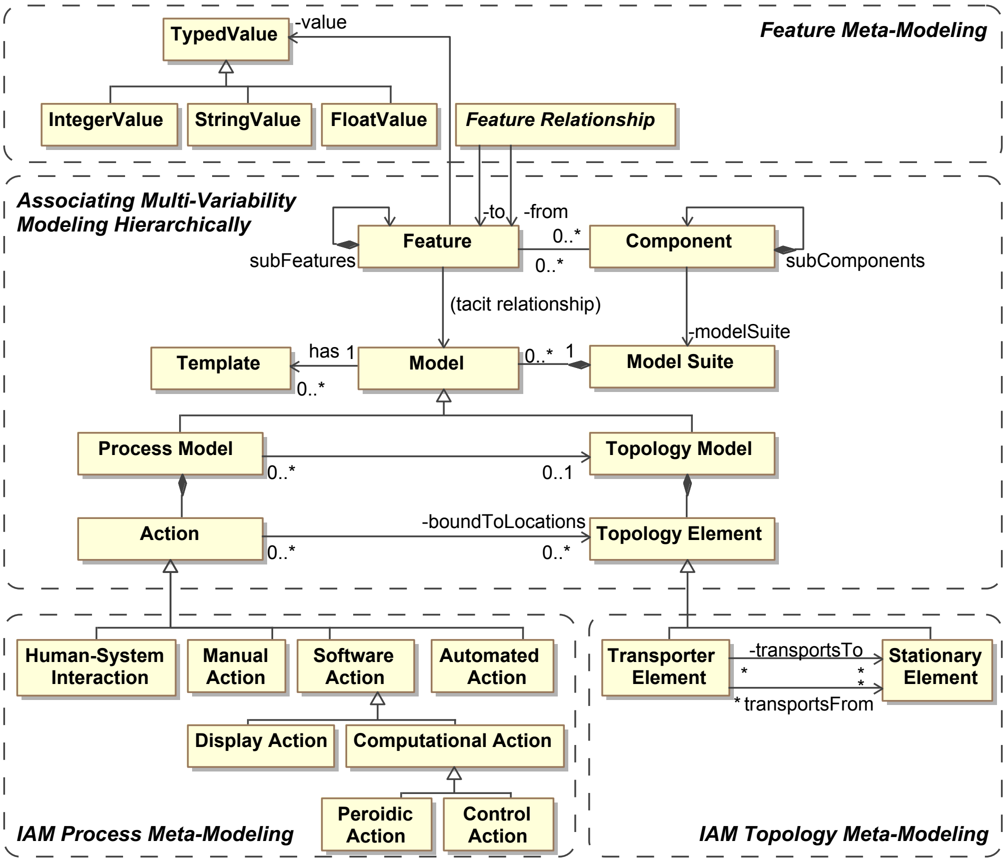


Figure 4.20.: Summary of Meta-Models and Their Associations

At the top of Figure 4.20, the feature meta-modeling includes features, typed values and feature relationships. At the lower part of this figure, the core elements within the process and topology meta-models are presented. In the middle of Figure 4.20, features, components and IAM-specific meta-elements are associated hierarchically.

By proposing the meta-models and their associations, this chapter answers the question "RQ1: How can we structure the variability modeling space to support the representation of multiple variability types?" The chapter presents a four-level modeling framework according to MOF architecture. The meta-modeling level contains four meta-models, including the feature meta-model, the promoted feature models as meta-models, the domain topology meta-model and the domain process meta-model. The principles and procedures of the design and development of the IAM-specific meta-modeling are reported as well. For the validation of the approach, the feasibility, the improvement on size of reusable asset base and the users' acceptance of the approach should be tested.

To achieve the mapping from variability modeling elements to assets, a multi-variability realization should be supported. The next chapter will present the variability realization and implementation to enable the mapping.

5 Multi-Variability Realization

"Architecture begins where the engineering ends."

–Walter Gropius

In software product line engineering, researchers commonly separate reusable core assets into the problem space and the solution space [CE00, BBM05]. The problem space refers to the assets for system specifications during requirement engineering, and the solution space refers to the technical artifacts related to architecture and implementation [BBM05]. With regard to the derivation approach presented in Figure 5.1, the previous chapter has presented the multi-variability modeling, which belongs to the problem space. Based on the multi-variability modeling framework, this chapter presents the realization mechanisms for the multi-variability models in the solution space. Both of the multi-variability modeling and realization are core components of the derivation approach in family engineering.

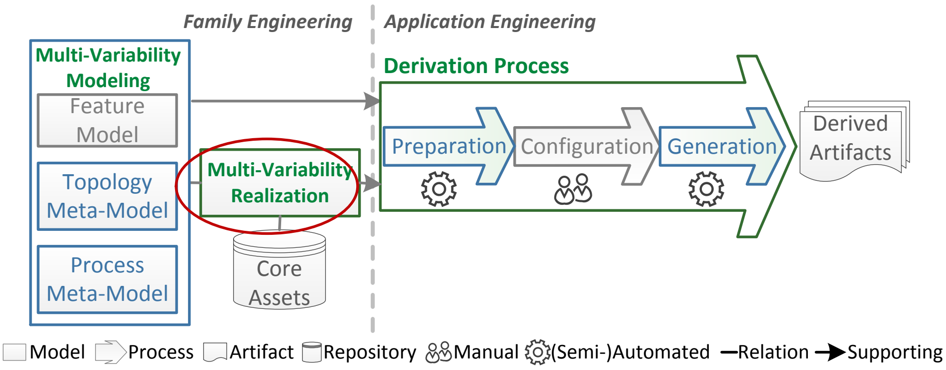


Figure 5.1.: Multi-Variability Realization in the Solution Overview

To enable the multi-variability realization, it is necessary to map and trace the multi-variability (meta)-models to their corresponding core assets. With the variability realization mechanisms, reusable software artifacts can be finally selected and configured by instantiating and configuring variability models. Figure 5.1 shows the contribution of this chapter to the overall approach of this thesis.

This chapter answers the research question, *RQ2: How can we establish the mapping between the different variability modeling elements with their associated core assets?* To answer RQ2, this chapter contributes to:

- A taxonomy of multi-variability realization: Feature, topology, and process variability should be implemented during derivation time.

The remainder of this chapter is organized as follows: As the foundation of variability realization, Section 5.1 introduces the background of variability realization including derivation life cycle and abstraction levels. With this section, the variability realization in this thesis is scoped to two phases: architecture derivation and detailed derivation. Accordingly, Section 5.2 presents how the variability is realized usually during architecture derivation; and Section 5.3 proposes taxonomy of multiple variability realization at the detailed derivation level. Section 5.4 gives the discussion about the limitations and lessons learned. Section 5.5 compares the proposed variability realization taxonomy with the prior approaches. Section 5.6 reflects variability realization to the validation goals. Lastly, Section 5.7 provides the summary of this chapter.

5.1. Foundation of Variability Realization

Variability can be realized at different phases of application engineering. According to Svahnberg et al., these phases are derivation, application-specific development, compilation, linking, or even at run-time as shown in Figure 5.2 [SVGB05].

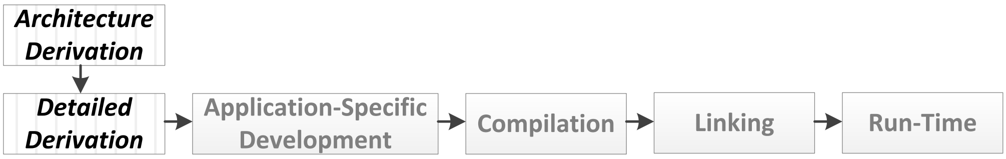


Figure 5.2.: The Variability Realization Life Cycle

Various variability realization techniques summarized previously in Chapter 3 are developed to apply at different realization phases. For example, the conditional compilation mechanism resolves variability at compilation phase. However, with the scope of this thesis, the variability realization in derivation is the focus.

Variability Realization:
Variability realization during derivation refers to the implementation of variants within variability models in the software core assets, so that selecting and configuring of variants can derive the customer-specific artifacts.

Variability realization during software product derivation can appear at two different levels of abstraction [VG07]:

- Architectural level: The typical reusable assets include the product line architecture, components, and frameworks. The variants are possibly optional components or architectural reorganizations [SVGB05].
- Code (detailed) level: The commonalities are sets of classes in software product lines. The variable parts are "embedded" in code fragments [VG07, SVGB05], which need to be composed to realize their variability.

5.2. Variability Realization at the Architectural Level

At the architectural level, the reusable architectural units are actually components. The variability realization of the components in IAM systems is similar to configurable software product lines. Feature-like variability at this level plays the most important role for component inclusion and exclusion, as suggested in many existing approaches [SGEL09, VG07].

5.2.1. Components as Architectural Elements

Components are the architectural units for the inclusion or exclusion at the architecture level of derivation.

Component:
A component in software systems is an architectural unit, which is an encapsulation of a set of software artifacts to implement certain functions or features.

Figure 5.3 presents the meta-types of components in the multi-variability derivation approach of this thesis. The atomic components have no sub-components, whereas the composite components contain other components.

The boolean attribute *isRootComponent* of the *Component* indicates whether the component is the root component of the software product line. The root component is at the top of the component hierarchy of a software product line, which is usually a composite component that contains all the other components. A software product line can only have one root component.

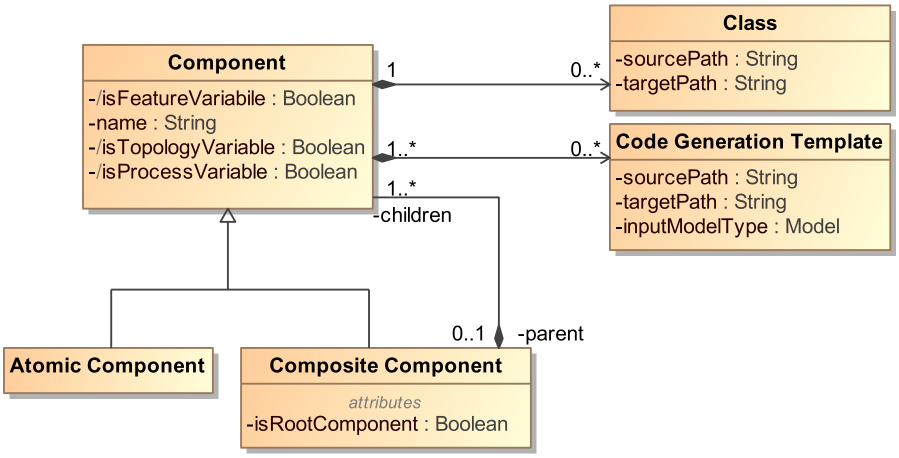


Figure 5.3.: Component Types

Root Component:
A root component is a composite component that hierarchically contains all components of the software product line.

The *Class* in Figure 5.3 refers to the source code classes and files in software implementation without variability. These classes are directly reusable among applications.

A *Component* can have any of the three variability types, which is realize by the *Code Generation Templates*, as shown in Figure 5.3. The three boolean attributes *isFeatureVariable*, *isProcessVariable*, and *isTopologyVariable* are derived based on their models in the *Model Suite* (c.f. Section 4.5 in Chapter 4) as well as their sub-components.

At the architectural level, only the inclusion of the application-specific *Code Generation Templates* is decided. The list of these templates is only relevant to a specific customer. The variability in each *Code Generation Template* is not resolved at the architectural level, but at the code derivation level.

5.2.2. Variability Realization Techniques at the Architectural Level

Based on the variability realization techniques collected by Svahnberg et al. [SVGB05], several variability realization patterns are available to apply during derivation at the architectural level [SVGB05]:

- **Architecture reorganization** supports several application-specific architecture by reorganizing the overall product line architecture.
- **Optional architecture component** supports a component, which is optional to appear in the application-specific architecture.

- **Variant architecture component** supports several parallel components implementing similar conceptual functionality, which can be alternative or replaceable to one another.

To implement these variability realization patterns at the architectural level, it is essential to resolve the variability between caller components and callee components. Figure 5.4 shows a small example how to resolve the variability by using the configuration of features. *Pick by Light* is an optional component. When the corresponding feature is selected in the configuration, the derivation infrastructure includes the source of *Pick by Light* during architecture derivation, and wires up its instance to the appropriate instance of the *Pick Logic* component. In this example, the parameters of components, such as the *mode* of the *Pick Logic* can be configured via selecting and configuring the value of the corresponding feature as well.

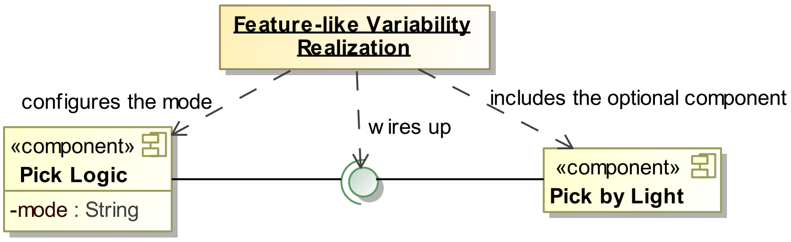


Figure 5.4.: Variability Realization at the Architectural Level

Svahnberg et al. point out that existing solutions to resolve the variability at the architectural level commonly use architecture description languages or external tooling to link and bind components [SVGB05]. Existing approaches support architectural configurations and composition to enable the variability realization. For example, Koala is an architecture description language developed for a family of television products in Philips. The goal to develop Koala is to wire up components, and to enable the interaction among them [ASM04, VOVDLKM00]. Several other architecture description languages allow tracing the specification languages to lower-level artifacts, such as Rapide and Darwin [MT00, LKA$^+$95, MDEK95].

5.3. Multi-Variability Realization at the Detailed Level

According to Schwanninger et al. [SGEL09], it is not sufficient to only establish links from features to architectural building blocks. All variations should be linked to the concrete variation implementations [SGEL09]. This subsection presents how to achieve this by mapping the process and topology variability to code generation templates for IAM systems.

At the code level, the variability realization of IAM systems requires to resolve process and topology variability. The upper part of Figure 5.5 shows the involved variability modeling constructs. (Section 4.5 in the previous chapter has presented some of the concepts.) The lower part shows the variability realization mechanism in terms of code generation. The implementation of the realization arrows in Figure 5.5 uses the common construction in template languages, such as loops or conditions. For example, to realize $LogicalGroup$ with $TypeTraverse$ in Eclipse Xpand [xpa], the loop ($foreach$) and the type selection ($typeSelect$) need to be used. These are commonly supported by other template languages, such as in T4 [t4].

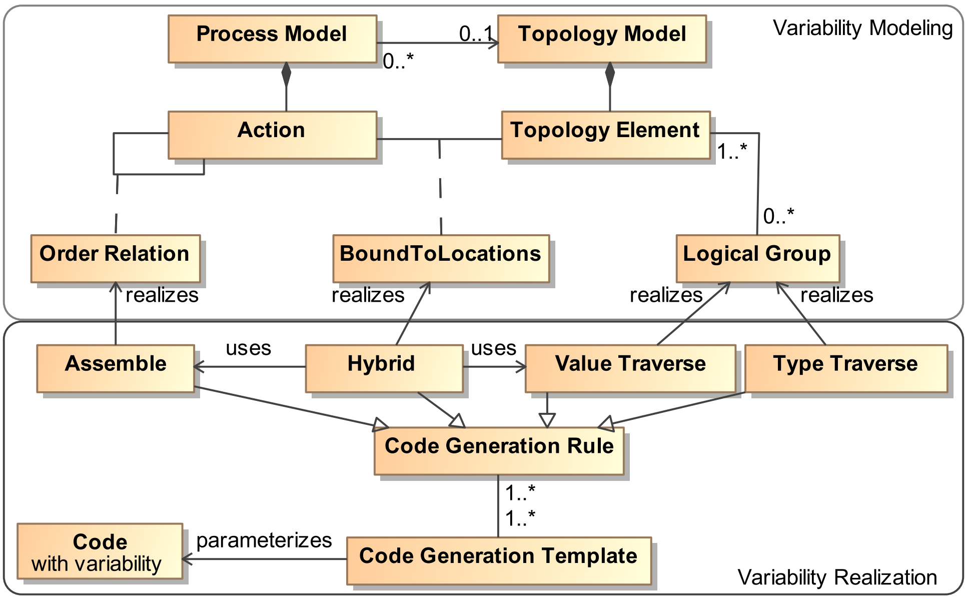


Figure 5.5.: Variability Realization at the Detailed (Code) Level

The intention of proposing the taxonomy for multi-variability realization at the detailed level is independent from any template languages. The concrete implementation in programming depends on the grammar of chosen template languages that is not the core focus of this chapter.

Section 5.3.1 focuses on interpreting the variability types and the associations among them, which is the upper part of Figure 5.5. Section 5.3.2 proposes a taxonomy of realizing the multi-variability models, which covers the lower part of Figure 5.5.

5.3.1. The Needs of Multi-Variability Realization in IAM Systems

At the detailed derivation level, the variability realization of IAM systems requires to resolve process and topology variability. The upper part of Figure 5.5 shows the involved variability modeling constructs that require variability realization in IAM systems. The meta-elements, such as actions and topology elements have already been introduced in Section 4.5 in the previous chapter.

Order Relation. The derivation stakeholders or modelers specify behavioral sequences in process models. The actions within the process models imply order relations among them [WMW11]. The reusable artifacts are the individual implementations of each action. The realization of this relation requires to *assemble* the reusable code of actions according to their order in the process configurations as shown in Figure 5.5. This assembly is implemented by generating the "glue-code" for invocation.

Order Relation:
Let P_{A} be the set of all actions in a process model with a, b and $c \in P_{\mathsf{A}}$; The order relation R_{Order}, denoted by $\rightsquigarrow$, is a relation, where holds: if a $\rightsquigarrow$ b and b $\rightsquigarrow$ c, then a $\rightsquigarrow$ c.

Logical Group: The *logical group* is introduced to model a logical association between software services and their corresponding topological elements. The reason to model such a group is that software services need to manage and observe different topological elements. Figure 5.6 shows two examples of logical groups. For group 1, the two mini-load conveyors and two buffers belong to the logical group of the location-tracking component, in which the transportation progress of boxes is monitored. For group 2, the component of buffer management coordinates only the two buffers in this example. The cardinality of topological elements within each logical group causes the variability. The realization of this variability is by *type traversing* or *value traversing* of the model instances within the variability models.

Logical Group:
Let E be the set of all topological elements in a topology model; A logical group is a set, where $G_{\mathsf{LogicalGroup}} \subseteq E$ with $e \in G_{\mathsf{LogicalGroup}}$, if e perform or interact with the same software components.

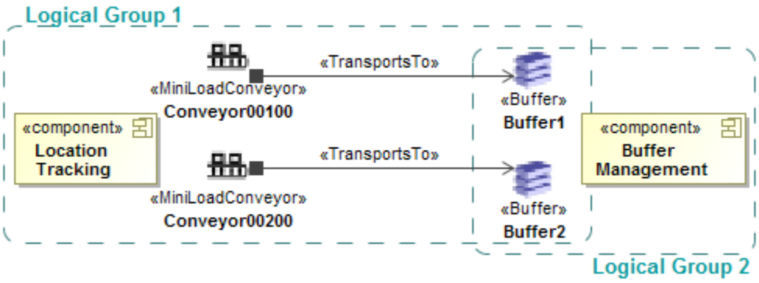


Figure 5.6.: Examples of Logical Groups

Bound-to-Locations Relation. This relation has been introduced already in Figure 4.19. It is defined to express that an action is to be executed at certain locations within the factories. A *hybrid* realization as presented in Figure 5.5 is necessary for this relation, based on both *assembly* and *traversing* realization techniques.

Bound-to-Locations relation:
The bound-to-locations, $R_{\text{Bound-to-Locations}}$ is a relation, where the execution of actions in process models is bound to locations of certain topological elements in the topological models.

The lower part of Figure 5.5 shows the variability realization mechanisms in terms of code generation, in order to automate software derivation. The next subsection introduces each of the realization techniques with more details.

5.3.2. Taxonomy of Multi-Variability Realization

This subsection defines taxonomy of variability realization in IAM systems. As already depicted at the lower part of Figure 5.5, the detailed level derivation requires to parameterize the code artifacts, and to come up with the code templates for generation. To fulfill the need of software product derivation of IAM systems mentioned in the previous subsection, further variability realization techniques are needed, as shown in Figure 5.5. To implement the variability in a complex artifact with multiple variability, the combination of these realization techniques can be necessary.

The realization techniques in other phases of the derivation life cycle have been summarized in the work of Svahnberg et al. [SVGB05]. The taxonomy presented in this thesis is an extension of this previous work focusing at the detailed derivation level, which is not covered in the prior work. To keep the consistency with the taxonomy of Svahnberg et al. [SVGB05], this extended taxonomy follows the design-pattern like form [BMR+96, Gam95] explained in Table 5.1.

Items	***Rationales***
Intent	A short description of the realization technique
Motivation	An explanation of the problems solved by the realization technique
Solution	A technical solution to resolve the variability
Consequences	The consequences of applying the realization technique in solution space
Solution	The analysis of the consequences by introducing the realization techniques to the derivation infrastructure
Example	A concrete example with its implementation of using the technique

Table 5.1.: The Taxonomy Form of Variability Realization Techniques

The motivation to follow the design-pattern like form is also that these are reusable techniques to realize the re-occurring IAM variability that are independent from implementation techniques. Collecting and validating variability realization patterns, however, is not considered as the core contribution of this thesis. The collection is sufficient for variability realization at the detailed realization level in the context of IAM systems, which would not be complete for all software domains. Looking for more re-occurring cases and the validation of these variability realization patterns are considered as a part of the future work.

5.3.2.1. Template-Based Derivation

Intent. Support the reuse of source code, classes, or textual artifacts with variability.

Motivation. The derivation at the detailed level can be realized by generating application-specific artifacts based on their existing implementation. The potential artifacts to apply template-based derivation include classes, configuration files, or some other textual fragments.

Solution. A code generation template can be developed or extracted based on existing implementation. Even for highly variable code artifacts, having a template may still pay-off, since templates contain the commonalities and regularly occurring variability, which allow the developers to focus on the customer-individual parts during application-specific development. Usually, the template-based derivation needs to combine with other derivation techniques that will be explained in the following realization techniques of this taxonomy.

Consequence. Compared to runnable source code, the maintenance of code templates is more expensive. To test or to debug code templates require sometimes a valid configuration of variability models, as only the derived code is actually runnable and testable. This determines that the creation of code generation templates should be based on stable and mature architectural decisions.

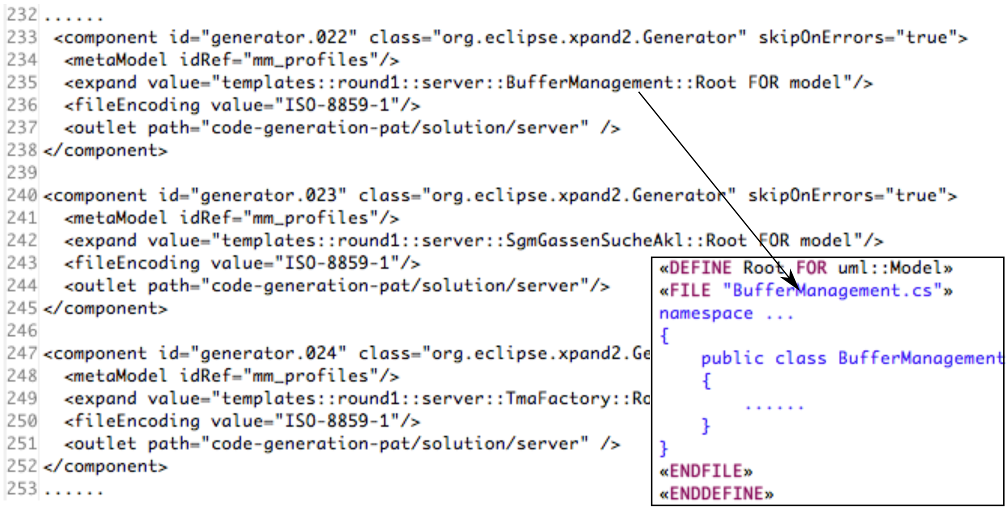
Example. Figure 5.7 shows an example of using Eclipse Xpand to include code templates and implement a class template. The script on the left hand side of Figure 5.7 shows how to include the class template with its path (i.e. at line 235) and the expected output path (at line 237) of the generated class. In line 235, the *value* is the path of the template, named *BufferManagement*. The *Root for model* indicates that the code template will receive the root of the variability models. In line 237, the *outlet* defines to which path the class should be generated. The template fragment shown on the right hand side illustrates the template of the *BufferManagement* class to be derived.

5.3.2.2. Parameterization

Intent. Support usually the configuration of values, such as numbers, strings, or boolean values of variants.

Motivation. This technique allows the configuration of detailed attributes, especially values of variability elements.

Solution. At the first place, the variable values within the artifacts should be identified. At the second place, the type of the variable values should be decided. For instance, in case of a number or an integer, it is important to understand the range of the valid values. The decision requires domain knowledge to judge what should be the valid values of the parameters. Finally, the variables in the artifact are parameterized and



```
......
 <component id="generator.022" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
  <metaModel idRef="mm_profiles"/>
  <expand value="templates::round1::server::BufferManagement::Root FOR model"/>
  <fileEncoding value="ISO-8859-1"/>
  <outlet path="code-generation-pat/solution/server" />
</component>

<component id="generator.023" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
  <metaModel idRef="mm_profiles"/>
  <expand value="templates::round1::server::SgmGassenSucheAkl::Root FOR model"/>
  <fileEncoding value="ISO-8859-1"/>
  <outlet path="code-generation-pat/solution/server"/>
</component>

<component id="generator.024" class="org.eclipse.xpand2.Ge
  <metaModel idRef="mm_profiles"/>
  <expand value="templates::round1::server::TmaFactory::Rc
  <fileEncoding value="ISO-8859-1"/>
  <outlet path="code-generation-pat/solution/server" />
</component>
......
```

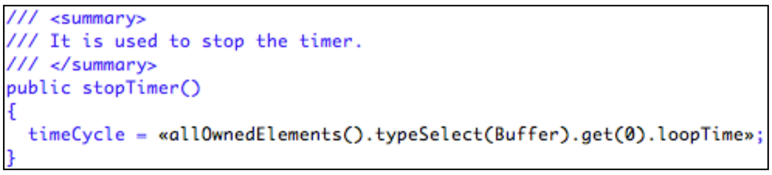
```
«DEFINE Root FOR uml::Model»
«FILE "BufferManagement.cs"»
namespace ...
{
    public class BufferManagement
    {
        ......
    }
}
«ENDFILE»
«ENDDEFINE»
```

Figure 5.7.: A List of Code Templates and a Class Template Example

associated with variability meta-models, so that the configurations of variability can be used to bind variability during derivation.

Consequence. During parameterization, it is possible to identify some detailed parameters as attributes to be added in variability (meta-)models. This might lead to some minor refinements to the meta-models. Furthermore, the derivation infrastructure should support a value checking mechanism to ensure the correctness and validity of the configured values, to avoid that the derivation infrastructure uses invalid values to do code generation.

Example. Figure 5.8 shows an implementation of the parameterization technique. As can be seen, the loop time of a buffer is variable among customers. In this example, the value is configured in the topological element *Buffer*. A *loopTime* attribute of the *Buffer* is used in this case to decide the parameter in the code. During configuration time, the loop time is decided by stakeholders, and given to the variability models. During derivation time, the loop time value is retrieved and put at the desired location in the code.



```
/// <summary>
/// It is used to stop the timer.
/// </summary>
public stopTimer()
{
  timeCycle = «allOwnedElements().typeSelect(Buffer).get(0).loopTime»;
}
```

Figure 5.8.: An Example of Parameterizing a Loop Time

5.3.2.3. Assembling

Intent. Support the realization of order relation in process variability configurations.

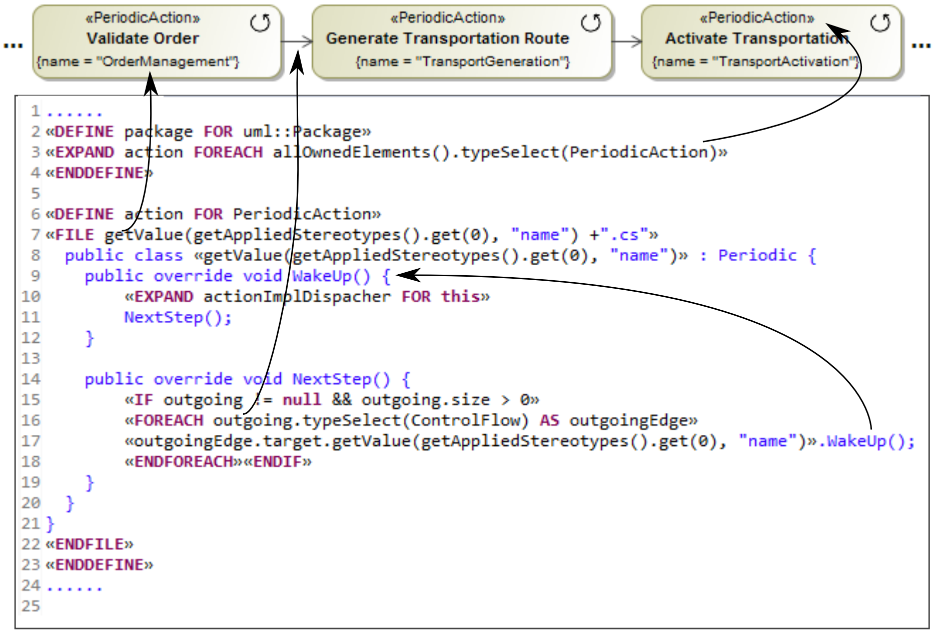
Motivation. As analyzed in Section 5.3.1, the actions in process models imply an order relation, which is one of the variability types in IAM systems. The reusable artifacts in process models are the implementation of each of the actions in processes.

Solution. The solution idea is to collect the corresponding artifacts of configured actions and to derive glue-code to allow the assembly of the reusable artifacts. The realizations of the glue-code of the four types of IAM flows defined in Section 4.4 Chapter 4 are slightly different:

- *Message flows* represent the communication among software and distributed hardware components. The implementation of this type of flows requires information such as the destination of messages, status, commands, or errors.
- *Control flows* handle the software actions and their execution sequences. The generation-oriented derivation solution is to generate the glue-code to enable the assembly of the action code.
- *Manual flows* guide human workers, to let them follow a manual procedure to perform their tasks. The derivation of processes following manual flows are related to the order of the display actions. Compared to the other types of flows, human workers who manually perform the actions take care of the orders of the task execution.
- *Material flows* represent the movement of raw materials in factories. It is usually used to describe high-level processes. For software derivation, it is important to further break down the material flows, for example to differentiate whether the movement is done by automation of by manual workers, so that material flows can be refined by the processes with the other three types of flows accordingly.

Consequence. One of the consequences to implement the assembly is the refactoring of the existing code artifacts. Suggested by Lopez et al. [LHMME11], the refactoring patterns to implement this technique are for example adding hook methods, relocating business logic, or addition at the end of methods (to invoke the next process step).

Example. Figure 5.9 presents an example of a code template to assemble classes with a hook method as the realization of the order relations for process variability. At the top part of Figure 5.9, the process includes several $PeriodicActions$ and their $ControlFlow$ edges, as defined in the process meta model in Section 4.4.3 (see Figure 4.15 and Figure 4.16). At the lower part of Figure 5.9, the code template selects all actions of the type $PeriodicAction$ from its targeted package. It generates a class for each action (line 7). Inside the generated classes, the $WakeUp$ and $NextStep$ methods separate the business logic of the current action and the glue code to invoke the next action. The reusable business logic is included via a dedicated template (at line 10). At line 17, the code template searches for outgoing edges of actions and generates the code to "wake up" the next $PeriodicAction$ in the process.


...
«PeriodicAction»
Validate Order
{name = "OrderManagement"}
«PeriodicAction»
Generate Transportation Route
{name = "TransportGeneration"}
«PeriodicAction»
Activate Transportation
{name = "TransportActivation"}
...

```
......
«DEFINE package FOR uml::Package»
«EXPAND action FOREACH allOwnedElements().typeSelect(PeriodicAction)»
«ENDDEFINE»

«DEFINE action FOR PeriodicAction»
«FILE getValue(getAppliedStereotypes().get(0), "name") +".cs"»
  public class «getValue(getAppliedStereotypes().get(0), "name")» : Periodic {
    public override void WakeUp() {
        «EXPAND actionImplDispacher FOR this»
        NextStep();
    }

    public override void NextStep() {
        «IF outgoing != null && outgoing.size > 0»
        «FOREACH outgoing.typeSelect(ControlFlow) AS outgoingEdge»
        «outgoingEdge.target.getValue(getAppliedStereotypes().get(0), "name")».WakeUp();
        «ENDFOREACH»«ENDIF»
    }
  }
}
«ENDFILE»
«ENDDEFINE»
......
```

Figure 5.9.: Realization of a Hook Method

5.3.2.4. Value Traversing

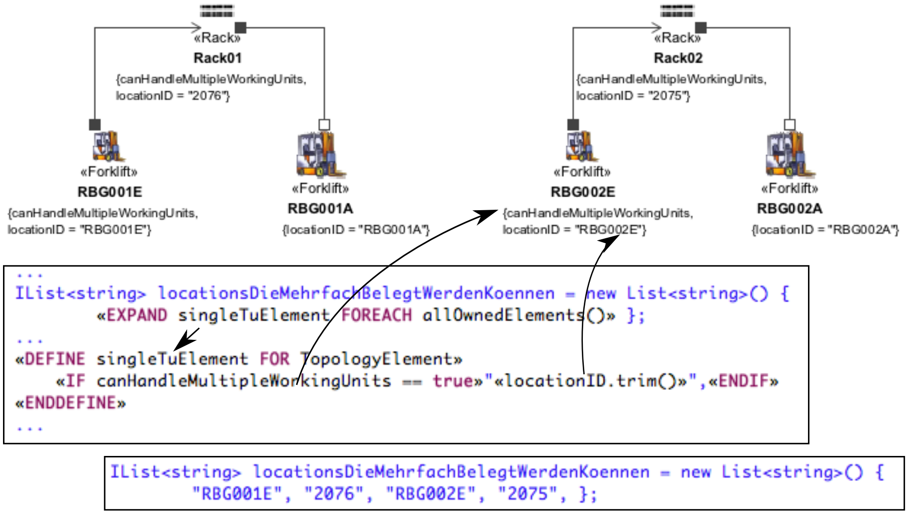
Intent. Group elements in the configured topology models by their meta-types.

Motivation. As motivated already by the needs of modeling logical groups in the previous section, some elements in variability models need to be grouped by their certain attributes, in order to support software derivation. This is especially important for topological elements in modeling industrial automation systems.

Solution. To realize value traversing, the derivation infrastructure needs to parse the multi-variability meta-elements and use certain attribute values to traverse the given configuration models.

Consequence. One of the consequences of using this technique can be that it also requires having parameterization of variable values. The candidate refactoring patterns to enable the implementation of this technique are moving fields, or renaming declarations of field modifiers [LHMME11].

Example. The top part of Figure 5.10 is a topology model. To collect all the elements that can handle several working units at the same time (*canHandleMultipleWorkingUnits*) in the given configured model, the code template in the middle goes through all the allowed elements and looks for those who have *canHandleMultipleWorkingUnits* as true. The lowest part of Figure 5.10 is the outcome of code generation according to this code template fragment.



```
...
IList<string> locationsDieMehrfachBelegtWerdenKoennen = new List<string>() {
        «EXPAND singleTuElement FOREACH allOwnedElements()» };
...
«DEFINE singleTuElement FOR TopologyElement»
    «IF canHandleMultipleWorkingUnits == true»"«locationID.trim()»",«ENDIF»
«ENDDEFINE»
...
```

```
IList<string> locationsDieMehrfachBelegtWerdenKoennen = new List<string>() {
        "RBG001E", "2076", "RBG002E", "2075", };
```

Figure 5.10.: An Example of Value Traversing

5.3.2.5. Type Traversing

Intent. Group elements in the configured variability models by their meta-types.

Motivation. The configured elements in the variability models sometimes need to be grouped, according to their meta-types.

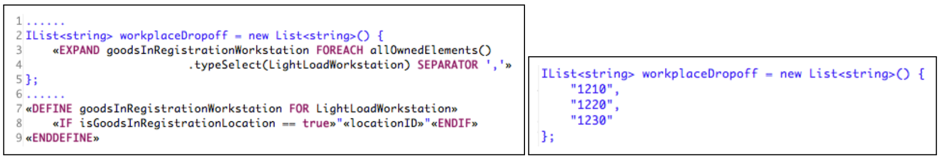
Solution. To realize type traversing, the derivation infrastructure needs to receive the proper meta model, and use the corresponding meta-type to traverse the given configuration models.

Consequence. Similar to value traversing, the candidate refactoring patterns to enable the implementation of this technique are moving fields, or renaming declarations of field modifiers in source code [LHMME11].

Example. In code templates, the keyword *typeSelect* has been already presented in the examples of the previously introduced realization techniques. Figure 5.11a shows a combined example of type- and value traversing in a code template. In Line 3-4, the *typeSelect* filters all the elements in input models to look for only those with *LightLoadWorkstation* as the meta-type. In between line 7-9, the value *isGoodsInRegistrationLocation* is evaluated in order to include the target *locationID* into the list of *workplaceDropoff* in Line 2. Figure 5.11b shows an example outcome of code generation, where three location IDs are included in the source code.

5.3.2.6. Hybrid Usage of Variability Realization

Intent. This taxonomy has already introduced five variability realization techniques at the code derivation level: template-based derivation, parameterization, assembling,



```
......
IList<string> workplaceDropoff = new List<string>() {
    «EXPAND goodsInRegistrationWorkstation FOREACH allOwnedElements()
                    .typeSelect(LightLoadWorkstation) SEPARATOR ','»
};
......
«DEFINE goodsInRegistrationWorkstation FOR LightLoadWorkstation»
    «IF isGoodsInRegistrationLocation == true»"«locationID»"«ENDIF»
«ENDDEFINE»
```

(a) An Example of Type- and Value Traversing

```
IList<string> workplaceDropoff = new List<string>() {
    "1210",
    "1220",
    "1230"
};
```

(b) The Results

Figure 5.11.: An Example of Type- and Value Traversing for Code Generation

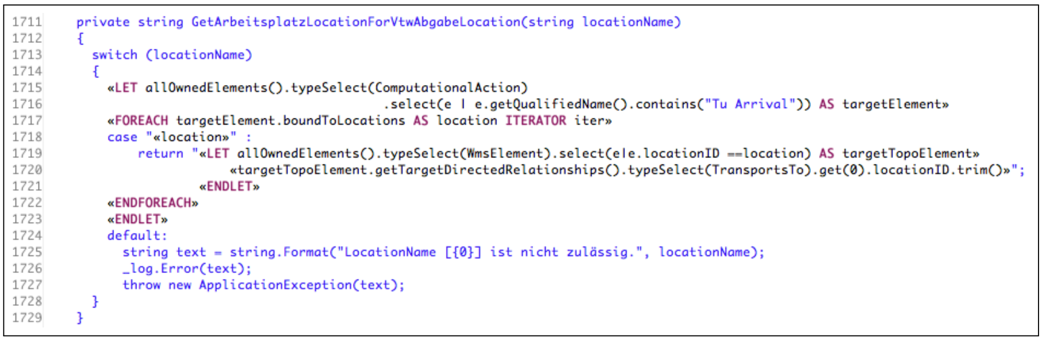
value traversing, and type traversing. In complex code templates, these techniques are combined and hybrid, to enable the generation.

Motivation. Some legacy code during development has no consideration of variability realization, so that they contain high variability of different types. To enable the derivation of such artifacts, hybrid usage of variability realization techniques are necessary.

Solution. Combine several of variability realization techniques.

Consequence. At the time the code is developed, developers did not take code generation into consideration. This may create difficulties to extract, parameterize and develop code templates. Completely clear separation of variability at code derivation level requires very high re-engineering effort and should be avoided.

Example. Figure 5.12 shows a relatively complex code template. The action named "Tu Arrival" is bound to topological locations and the relations among them. The code generation requires both the process and topology variability configuration.



```
    private string GetArbeitsplatzLocationForVtwAbgabeLocation(string locationName)
    {
      switch (locationName)
      {
        «LET allOwnedElements().typeSelect(ComputationalAction)
                                        .select(e | e.getQualifiedName().contains("Tu Arrival")) AS targetElement»
        «FOREACH targetElement.boundToLocations AS location ITERATOR iter»
        case "«location»" :
            return "«LET allOwnedElements().typeSelect(WmsElement).select(ele.locationID ==location) AS targetTopoElement»
                        «targetTopoElement.getTargetDirectedRelationships().typeSelect(TransportsTo).get(0).locationID.trim()»";
                    «ENDLET»
        «ENDFOREACH»
        «ENDLET»
        default:
          string text = string.Format("LocationName [{0}] ist nicht zulässig.", locationName);
          _log.Error(text);
          throw new ApplicationException(text);
      }
    }
```

Figure 5.12.: An Example of Hybrid Variability Realization

5.4. Discussions and Limitation Analysis

In the first place, legacy software artifacts usually cannot be used directly for software derivation in a software product line approach. It is necessary to conduct re-engineering and re-factoring to the IAM architectural and code artifacts [FLE^{+}15], which would be critical to enable the variability realization techniques discussed in this chapter. Lopez et al. propose re-factoring patterns to be applied during feature-oriented re-engineering processes [LHMME11]. Liebig et al. develop an approach for C code with pre-processor directives to support variability-aware code refactoring [LJG$^+$15]. To the best of the author's knowledge, there is limited automation support in the state of the art to achieve the re-engineering.

In the second place, the motivation of proposing the detailed level variability techniques is that, in IAM systems, process and topology variability types lead to fine-grained reusable artifacts. These artifacts are code fragments, which determines that the reuse of them is template-based. For this reason, following the extractive manner of SPLE adoption is encouraged when applying this approach (cf. Section 1.3.2 Chapter 1).

Furthermore, the examples in this taxonomy are shown in the template language in Eclipse Xpand. However, the techniques are implementation independent, regardless of template languages or tooling. Similar toolkit could be Text Template Transformation Toolkit (T4), or MOF Model to Text Transformation Language.

Last but not least, this chapter introduces the taxonomy of variability realization techniques with a design-pattern-like format. It is an extension on the detailed derivation phase of the existing taxonomy reported by Svahnberg et al. [SVGB05]. This taxonomy is with the variability concerns specifically to process and topology variability at derivation time. It may have its restrictions, since the generation is based on template-based derivation. To enrich this taxonomy, further case study is necessary to conduct, which is considered as a part of future work.

5.5. Comparison with Related Work

This section provides the discussion about related work of variability realization and the comparison to the variability realization techniques presented in this chapter.

5.5.1. Related Work in Variability Realization Taxonomy

The variability realization taxonomy presented in this chapter is an extension of a previous variability realization taxonomy proposed by Svahnberg et al. [SVGB05]. The previous taxonomy on variability realization techniques reported by Svahnberg et al. covers the variability realization over the life cycle of application engineering, such as architecture derivation, linking, complication and run-time. To differentiate from the previous taxonomy reported by Svahnberg et al., the taxonomy presented in this chapter extends the derivation phase at the detailed derivation level. Especially, it focuses on the realization of process and topology variability and their relationships for IAM systems.

5.5.2. Related Work in Architecture Description Languages

The other related approaches of variability modeling and realization techniques in solution space are architecture description languages. Chapter 3 has already reported some of these approaches. For example, Koala is an architecture description language, developed at Philips to reuse components of a family of television products [ASM04]. In Koala, the components are developed independently, and these components are wired and interacted by interfaces. The binding of variability in components is at configuration time, by using a compiler [VOVDLKM00]. As an extension based on Koala, Koalish combines the architecture description language with explicit variation modeling mechanisms [ASM04] to enable automatic configuration of products in the software family. Since Koala was developed for a family of television products in Philips [ASM04], it was not suitable to be used in IAM systems without the consideration of topology and process variability.

5.5.3. Related Work in Domain-Specific Modeling

Heinzemann and Becker propose a model-based approach, named MechatronicUML, to enable hierarchical re-configuration of complex software components for self-adaptive mechatronic systems [HB13]. In MechatronicUML, component models specify the static software architecture. Different information is modeled with MechatronicUML including the behavior specification of components, or the ports and interfaces to control the physical systems. MechatronicUML is customized to the mechatronic domain, with the main focus on requirement and design verification. MechatronicUML is inspiring to the development of approach presented in this thesis, but it is not applicable for the IAM systems.

5.6. Validation Objectives

Software product line engineering involves the activities in family engineering to develop the variability models and reusable artifacts, which benefits the application engineering to reuse existing artifacts, shorten time-to-delivery, and achieve better quality. This chapter presents the variability realization techniques, which implement feature, topology and process-variability models in the solution space. The effort and cost spent in the variability models and realization are the foundations to support the derivation in application engineering. Considering the effort spent on re-engineering of existing software artifacts and establishing the core assets in family engineering, the first validation objective for this chapter should be:

H1: Feasibility. It is feasible to apply the approach in family engineering.

The core topic of this chapter is to propose the variability realization techniques, in order to enable the architecture derivation and code generation. By having the variability realization techniques, it should be possible to automatically derive more source code, compared with the derivation without using the approach. For this reason, the following hypothesis should be tested:

H3: Automated level of derivation. The approach improves the code generation rate significantly (>35%), compared with the derivation without using the approach.

5.7. Chapter Summary

The taxonomy presented in this chapter is an extension of the existing variability realization techniques proposed by Svahnberg et al. [SVGB05]. This taxonomy is scoped at the derivation time of IAM systems with two abstraction levels: architecture derivation and code derivation.

Figure 5.13 provides the summary of these techniques. The techniques at the architecture derivation level include architecture re-organization, optional components and variant components. At the code level, more detailed derivation techniques are necessary, such as template-based derivation and parameterization. These variability realization techniques answer RQ2, since they enable the mapping between the different variability modeling elements to their associated core assets.

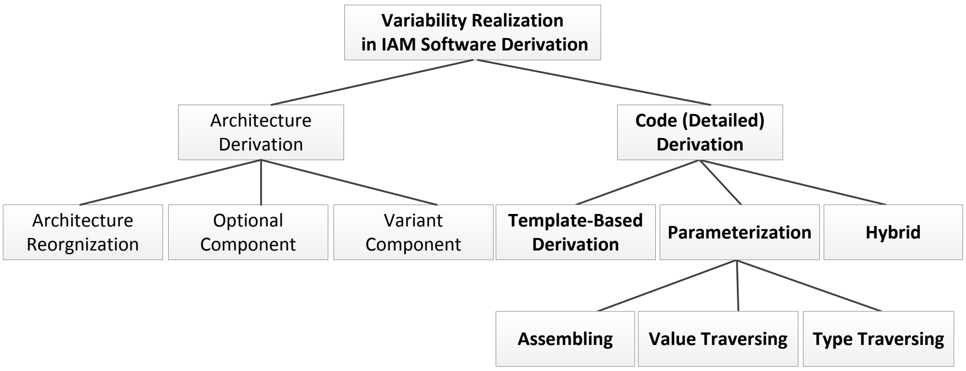

Variability Realization in IAM Software Derivation
Architecture Derivation
Code (Detailed) Derivation
Architecture Reorgnization
Optional Component
Variant Component
Template-Based Derivation
Parameterization
Hybrid
Assembling
Value Traversing
Type Traversing

Figure 5.13.: Summary of Variability Realization Techniques During IAM Derivation

Combining the multi-variability modeling presented in the previous chapter and the multi-variability realization in this chapter, it is then possible to achieve a semi-automated derivation process. The next chapter will introduce the derivation approach.

6 Automation of Model and Code Derivation

"All problems in computer science can be solved by another level of indirection."

–David Wheeler

Chapter 4 has introduced the hierarchical multi-variability modeling framework for variability in industrial automation, including the meta-models for features, topology and processes. Chapter 5 presents how the multiple types of variability modeling elements can be realized in solution space. As shown in Figure 6.1, Chapter 4 and Chapter 5 have introduced the family engineering parts of the derivation approach, which established the foundation to support the product derivation during application engineering in this chapter.

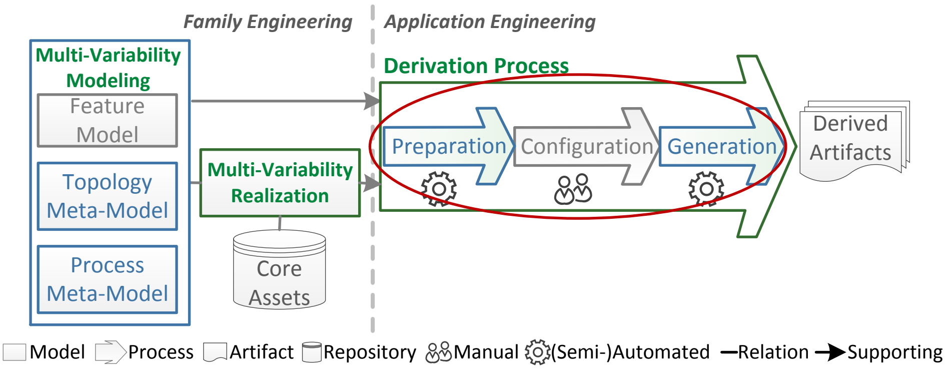


Figure 6.1.: The Derivation Process in the Overall Solution

The research question of this chapter is *RQ3: How to integrate the multiple variability types during the derivation process?* To answer this question, this chapter contributes to a Multi-vAriability Derivation (MADE) approach, which integrates the configurations of multiple variability models into a derivation process for IAM systems. The approach has three major phases: preparation, configuration and generation, aligned with three derivation activities in Chapter 2 (see Section 2.1.4). The unique contributions of the approach are twofold:

- The preparation activity creates process and topology model instances according to feature configuration, which allows the reuse of model templates.
- The approach includes the preparation and generation as two (semi-)automated activities, aiming at saving the time and effort of derivation stakeholders during application engineering.

To allow the configuration of the derivation stakeholders – requirement engineers, hardware-oriented engineers, and software developers – with their variability concerns, a sub-question is *RQ3.1: What should be the order of configuration steps when multiple stakeholders are involved with their variability concerns?* Furthermore, for derivation, the previously proposed multi-variability modeling and realization should be integrated into the derivation process, which requires answering *RQ3.2: How to use the configuration of multiple variability types as input to improve derivation efficiency?*

The remainder of this chapter is organized as follows: Section 6.1 shortly introduces the overview of the derivation activities of the MADE approach. Section 6.2, Section 6.3 and Section 6.4 define the input and output data of each derivation (sub-)activity, the procedures of execution with supporting examples. Section 6.5 discusses the lessons learned during the development of the approach and its limitations. Section 6.6 relates and compares the proposed approach with the state-of-the-art approaches. Section 6.7 reflects the contribution of this chapter to the evaluation goals. Finally, Section 6.8 summarizes this chapter.

6.1. Overview of the Derivation Process

Figure 6.2 shows the overview of the derivation approach. The left column includes the input artifacts, which are the core assets developed during family engineering. The right column contains the application-specific output artifacts supporting the derivation. The derivation activities are presented in the middle, which utilize and produce the application-specific artifacts.

The highlighted artifacts, the hierarchical multi-variability model and the multi-variability realization, are the core contributions presented previously in Chapter 4 and Chapter 5 respectively. They both also serve as the foundation to support the automation of the corresponding derivation activities in this chapter.

A short explanation of each derivation activity is given as follows. More details of each activity will be given in the remaining parts of this chapter.

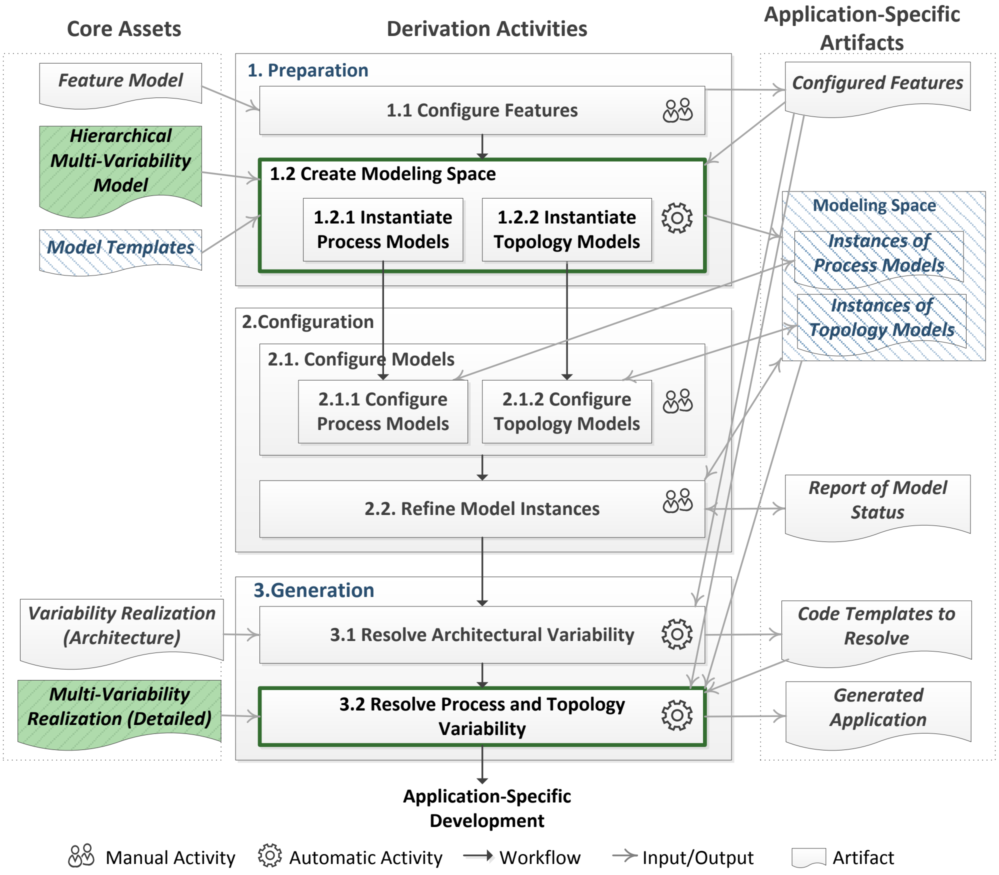

Core Assets
Derivation Activities
Application-Specific Artifacts
Feature Model
Hierarchical Multi-Variability Model
Model Templates
1. Preparation
1.1 Configure Features
1.2 Create Modeling Space
1.2.1 Instantiate Process Models
1.2.2 Instantiate Topology Models
Configured Features
Modeling Space
Instances of Process Models
Instances of Topology Models
2.Configuration
2.1. Configure Models
2.1.1 Configure Process Models
2.1.2 Configure Topology Models
2.2. Refine Model Instances
Report of Model Status
3.Generation
Variability Realization (Architecture)
3.1 Resolve Architectural Variability
Code Templates to Resolve
Multi-Variability Realization (Detailed)
3.2 Resolve Process and Topology Variability
Generated Application
Application-Specific Development
Manual Activity
Automatic Activity
Workflow
Input/Output
Artifact

Figure 6.2.: The Derivation Process

1.Preparation: This is a semi-automated activity. The overall goal of this step is to prepare a base configuration with reusable model templates and model fragments from core assets to help derivation stakeholders, so that they do not need to start a configuration of a complex system from scratch.

- **1.1:** As the first configuration step, derivation stakeholders can already decide the feature variability, especially the global features in higher abstraction levels.
- **1.2:** The derivation infrastructure interprets configured features as input, and automatically generates a modeling space, which contains process and topology model instances with corresponding model templates.

2.Configuration: This is a manual step for derivation stakeholders to configure the desired models. Nevertheless, the configuration of different variability types is non-trivial in industrial automation management systems, especially when process and topology variability are related in an intermingled way. The configuration order of the three derivation stakeholders is aligned with the staged derivation process of industrial automation management systems.

- **2.1.1:** The requirement engineers configure the process model instances to specify the functional requirements.
- **2.1.2:** The hardware-oriented engineers decide and configure the topology of hardware and their relationships.
- **2.2:** The software developers refine the configured models, by associating process elements and topological elements, adding more details and correcting errors when necessary.

3.Generation: Given the configured features, processes and topology models, this derivation activity automatically generates the application-specific architecture and source code as the (partially) derived applications.

- **3.1:** This step resolves the architectural variability mainly by inclusion and exclusion of components. The output of this step is an intermediate script including the relevant code generation templates for the final derivation step.
- **3.2:** The code generator uses the outcome of Activity 3.1 and the configured variability models as input. It finally derives the application.

Application-Specific Development. As motivated already in Chapter 2, industrial automation management systems belong to the type of flexible software product lines, full generation of customer-specific application is not realistic to achieve. Application-specific development is always expected to finally fulfill all customer requirements. Software architects, developers and testers potentially need to participate in the application-specific development activity.

6.2. Derivation Activity 1: Preparation

The configuration of large and complex software systems is usually not from scratch (see Section 2.1.4.1 in Chapter 2). For the derivation stakeholders, it is helpful to have a base configuration as a starting point of their tasks. The goal of having the *Activity 1: Preparation* is to automatically instantiate the relevant variability models with potential model templates or fragments to save the manual configuration effort for the derivation stakeholders.

6.2.1. Derivation Activity 1.1: Configure Features

Purpose. This is the first activity of the overall derivation process. In the context of this thesis, a feature in a feature model represents an abstraction of a functionality of a software product line. With this definition, a feature can be an abstraction of a concept, such as a representation of a composite component with their corresponding process and topology variability. Such features are defined as abstract features, whereas concrete features directly decide configuration at the programming level [TKES11]. In case, some decisions of features cannot be made in this activity, the approach support, to certain extent, iterative configuration among variability models. Section 6.5 will discuss more details.

Input. The input of this activity is a feature model of the target product line.

Procedure. The procedure starts by derivation stakeholders who decide to use a feature editor to configure features. The configuration of features starts with opening the feature editor. A feature model is displayed as a feature tree for configuration. Figure 6.3 presents the pseudo code to initialize the feature editor. The root feature is represented as the root tree node. For each feature node in feature tree, each feature is rendered according to the corresponding feature type (Line 4). A check box is added for feature selection (Line 5). When a feature is mandatory, and all of its predecessors in the feature hierarchy are mandatory, the feature should be set as selected (Line 7). Using the initialized feature editor, the manual procedure to configure features is straightforward. The stakeholders who perform this activity shall do the following tasks. They follow the hierarchy of the feature model, starting from the root feature node. They also assign values to features, such as integers, boolean or strings. It is important to ensure the validity of the values and avoid conflicts among features. Ideally, this manual configuration procedure can be optimized with various tooling or algorithms, such as automated planning [SAG+12] or propagation [UBFC14].

```
1  Create a root feature tree node based on the root feature in the feature model;
2  Build the feature tree recursively, with the root node and root feature, begin
3      for each sub-feature of the root feature do
4          Create a new child tree node according to the sub-feature;
5          Add a check box to select or de-select the tree node;
6          if the sub-feature is mandatory and all predecessors are mandatory then
7              Set the check box as selected;
8          Append the new child node to its parent; if the sub-feature has children then
9              Build the feature tree recursively, with the child node and the sub-feature;
10 Display the feature editor;
```

Figure 6.3.: The Pseudo Code of to Initialize a Feature Tree

Role. The derivation stakeholders who perform this activity shall have good domain knowledge, understand requirements and have at least certain technical background. Senior requirement engineers and product managers are good candidates to perform this activity, as well as software architects, who participate in the negotiation with customers. Notice that these roles are their job titles. They still belong to the three main derivation stakeholders, as requirement engineers, hardware-oriented engineers, or software developers.

Output. The output is a list of configured features for a specific application.

Rationales. The input of this step is a feature model with a tree structure, but the output uses a list as the data structure for all the configured features. The reason is that during feature configuration, the feature tree is beneficial for users to configure features following a top-down manner. However, during the following derivation steps, the features to other architectural elements may have many-to-many relationships. Especially for the automated steps, such as Activity 1.2 and Activity 3, the binding of architectural elements primarily defines where the configured features can be realized. For this reason, a list structure is chosen to collect the configured features to reduce the complexity of further derivation activities.

Tool Support. Figure 6.4 shows the architecture of the *Feature Editor*. On the left hand side, the input model is the feature model as the meta-model (See Section 4.2.3 in Chapter 4). In the middle of Figure 6.4, the *Feature Editor* contains the components supporting the *Configure Features* activity. The *Feature Editor View* contains and displays the feature trees with the values and status of feature nodes and their configuration. The *Feature Editor Controller* calls the *Feature Tree Render* and the *Feature Tree Node Render* to enable the configuration and manipulations of feature nodes. The controller also configures the feature configuration, when requests arrive from *Feature Editor View*. On the right hand side of Figure 6.4, it shows the output of configured features. Furthermore, Figure 6.5 shows a screen-shot of the feature editor seen and used by the users, which is actually the *Feature Editor View* component of the tool support.

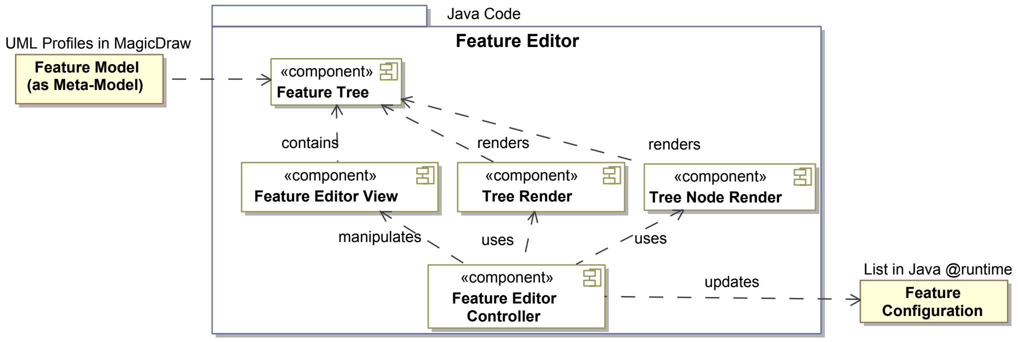
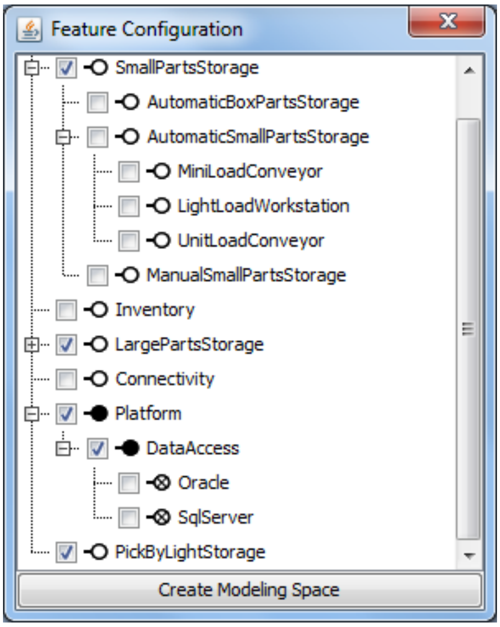


Figure 6.4.: Design of the Feature Editor

Example. A requirement engineer starts to understand the customer requirements, with the received business proposal and the factory layout. Afterwards, the requirement engineer starts the feature editor. The mandatory features are automatically selected, according to the algorithm in Figure 6.3. For example, the mandatory feature *Platform* and *Data Access* are automatically selected shown in Figure 6.5. The requirement engineer further configures the other features in the feature editor.



- ☑ SmallPartsStorage
  - □ AutomaticBoxPartsStorage
  - □ AutomaticSmallPartsStorage
    - □ MiniLoadConveyor
    - □ LightLoadWorkstation
    - □ UnitLoadConveyor
  - □ ManualSmallPartsStorage
- □ Inventory
- ☑ LargePartsStorage
- □ Connectivity
- ☑ Platform
  - ☑ DataAccess
    - □ Oracle
    - □ SqlServer
- ☑ PickByLightStorage

Figure 6.5.: A Screenshot of Feature Editor

6.2.2. Derivation Activity 1.2: Create Modeling Space

Purpose. The purpose of having this automated activity is to help derivation stakeholders to instantiate relevant models as a base configuration for their further derivation tasks. During family engineering, the reusable architecture has already been developed and managed as a part of the core assets. Complex business-oriented components require process and topology models to clearly describe their variability. These models are already maintained as templates or model fragments to be parts of core assets as well. During application engineering, this activity, *Derivation Activity 1.2: Create Modeling Space* helps automatically instantiating these models, aiming at saving the time and effort of derivation stakeholders.

Input. The input of this activity includes: the list of configured features for a specific application, the hierarchical multi-variability models of the software product line and the model templates in the core asset base.

Procedure. The derivation infrastructure executes this activity automatically. By clicking the button *Create Modeling Space* at the lowest part of the *Feature Editor* (see Figure 6.5) this procedure is triggered. Figure 6.6 shows the procedure.

Roles. The roles of derivation stakeholders who decide to create the modeling space (by using the feature editor) are the same as the roles who perform the *Derivation Activity 1.1*, including the requirement engineers, the product managers, or the architects who as technical roles may also participate in early pre-sales phase or negotiation with customers.

```
Let $ConfiguedFeatures$ be the list of configured features;
Let $RootComponent$ be the root component in the multi-variability hierarchy;
Let $RootModelPackage$ be the root package of the targeted modeling space;
CreateVariabilityModel($RootComponent$);
CreateModelPackage($RootModelPackage$, $RootComponent$);
Function CreateVariabilityModel($Component$)
    for each $Feature \in ConfiguedFeatures$ do
        if $Feature$ can decide variants then
            Bind variants in $Component$ according to $Feature$;
    $CompositeComponents \leftarrow Component.CompositeComponents$;
    for each $CompositeComponent \in CompositeComponents$ do
        CreateVariabilityModel($CompositeComponent$);
Function CreateModelPackage($Package$, $Component$)
    $Models \leftarrow$ ModelSuite of $Component$;
    for each $Model \in Models$ do
        if $Model.type$ = Process then
            Create $Instance$ with ProcessModelBuilder;
        if $Model.type$ = Topology then
            Create $Instance$ with TopologyModelBuilder;
        Add $Instance$ to $Package$;
    $Children \leftarrow$ all child components of $Component$;
    for each $Child \in Children$ do
        Create $SubModelPackage$;
        Attach $SubModelPackage$ to $Package$;
        CreateModelPackage($SubModelPackage$, $Child$);
```

Figure 6.6.: The Pseudo Code of the Model Instance Generator

Output. The output is the modeling space of a specific application. It is a package, which hierarchically contains the instances of process and topology models for the specific application to be further configured and used in the next derivation activities.

Tool Support. The procedure of this derivation activity is implemented in Java as a plugin of the chosen modeling platform [mag]. Figure 6.7 shows the design of the *Modeling Space Generator* with three types of input on the left, the modeling space as output on the right. Figure 6.7 also presents the internal components of the implementation in the middle. To instantiate different types of models, the corresponding model builder is used to create the available elements and model templates. The details of the *Process Model Builder* and the *Topology Model Builder* with the model templates and their input data to the derivation activity will be explained in the next two subsections as Activity 1.2.1 and Activity 1.2.2 respectively.

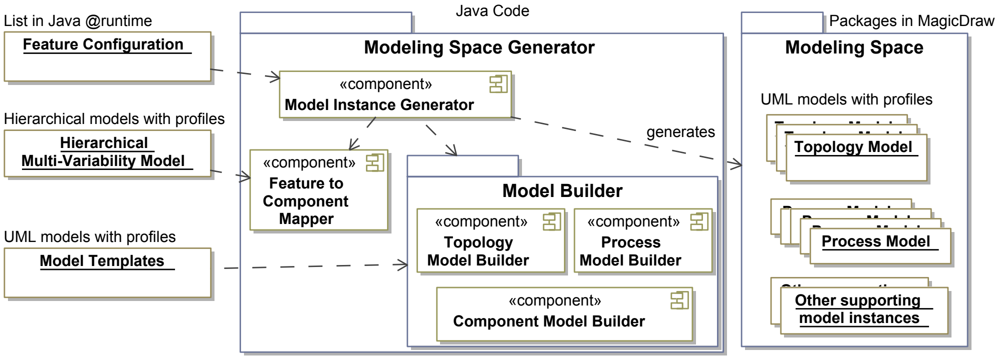

List in Java @runtime
Feature Configuration
Hierarchical models with profiles
Hierarchical Multi-Variability Model
UML models with profiles
Model Templates
Java Code
Modeling Space Generator
«component» Model Instance Generator
«component» Feature to Component Mapper
Model Builder
«component» Topology Model Builder
«component» Process Model Builder
«component» Component Model Builder
generates
Packages in MagicDraw
Modeling Space
UML models with profiles
Topology Model
Process Model
Other supporting model instances

Figure 6.7.: The Design of Modeling Space Generator

Examples. In the previous example of Activity 1.1, the requirement engineer selects and configures features with the feature editor. Then, he or she decides to create the modeling space by clicking the button at the lower part of the feature editor (see Figure 6.5). This button triggers the system to interpret the list of configured features and generate the packages of models for further configuration. As the output of this example, the generated packages of model instances are within the model explorer of MagicDraw as presented in Figure 6.8.

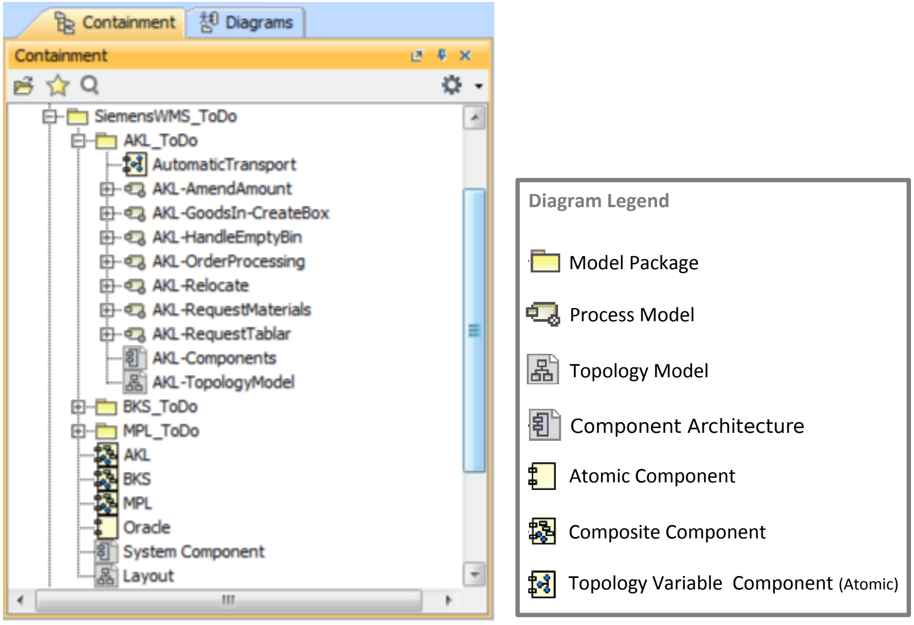


Figure 6.8.: The Generated Modeling Space with Model Instances

6.2.3. Derivation Activity 1.2.1: Instantiate Process Models

Purpose. This step automatically instantiates relevant process models with corresponding process templates to save the effort of derivation stakeholders.

Input. As the sub-step of activity 1.2, this activity requires one more input data, which are the process templates managed in core assets.

Procedure. Figure 6.9 shows the pseudo algorithm of this automated activity, which is performed by the Process Model Builder (see the design of the modeling space generator in Figure 6.7).

```
Function CreateProecssInstance
    Let $ProcessModel$ be the desired process model;
    Instantiate $Instance$ as an instance of $ProcessModel$;
    if $ProcessModel$ has a $Template$ in core assets then
        for each $Action \in$ all actions of $Template$ do
            Instantiate a new $Action$ and include it into $Instance$;
        for each $Edge \in$ all edges of $Template$ do
            Instantiate a new $Edge$;
            Associate the instantiated incoming $Action$ to the $Edge$;
            Associate the instantiated outgoing $Action$ to the $Edge$;
            Include the $Edge$ into $Instance$;
    Return $Instance$;
```

Figure 6.9.: Create Process Instances

Roles. As an automated activity, the derivation infrastructure instantiates the processes. The roles of derivation stakeholders who make the decisions to perform this activity can be possibly the requirement engineers, product managers, or the software architects.

Output. The output is the instantiated processes with the process templates.

Example. Figure 6.10 shows examples of the automatically generated processes. On the left-hand side, there is the model explorer for navigating the instantiated models. On the right-hand side, there are two examples of generated processes. The top one is created for the order processing for manual storage of large parts. The lower one is also an order processing, but for storage of automatic small parts. There can be processes sharing similar actions. As shown in Figure 6.10, the two processes share *Activate Order* and *Calculate Materials*. But the implementation of them for manual and automatic processes and the invocations among the actions can be different.

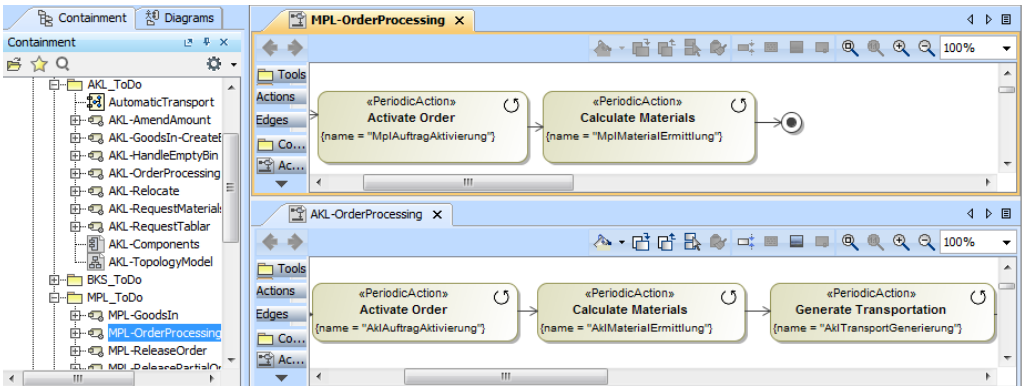


Figure 6.10.: Examples of Derived Processes

6.2.4. Derivation Activity 1.2.2: Instantiate Topology Models

Purpose. Similar to *Derivation Activity 1.2.1*, the purpose of having this activity is to help derivation stakeholders to automatically instantiate relevant topology models as foundations for their further configuration tasks.

Input. The input of this activity includes the configured features and the topological model fragments maintained in core assets.

Procedure. Figure 6.11 shows the pseudo code of this automated activity, which is the implementation of topology model instantiation.

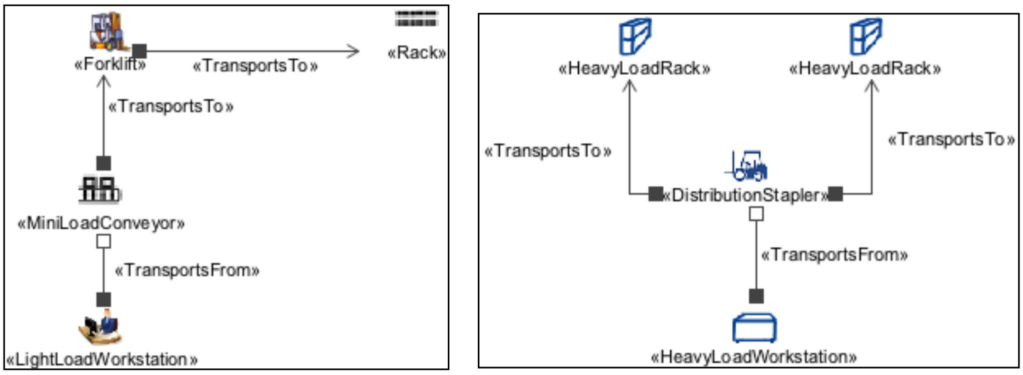
```
Function CreateTopologyModelInstance
    Let $TopologyModel$ be the desired topology model;
    Instantiate $Instance$ as an instance of $TopologyModel$;

    if $TopologyModel$ has a $Template$ in core assets then
        for each $StationaryElement \in Template$ do
            Instantiate a new instance of $StationaryElement$ and include it into
             $Instance$;

        for each $TransporterElement \in Template$ do
            Instantiate a new instance of $TransporterElement$;
            Associate the instantiated transportsTo $StationaryElement$ to the
             $TransporterElement$;
            Associate the instantiated transportsFrom $StationaryElement$ to the
             $TransporterElement$;
            Include the $TransporterElement$ into $Instance$;

    Return $Instance$;
```

Figure 6.11.: Create Topology Model Instances

Roles. The execution of this activity is automated by the derivation infrastructure. The roles who make the decisions to perform this activity can be the requirement engineers, the product managers or architects, which are similar to the previous activities.

Output. The output of this activity is the instantiated topological models with possibly topological model fragments.

Examples. Figure 6.12 shows two examples of topological model templates. The variability in topological models are the number of elements, the relationships among them and their logical groups. The information is highly variable among customers. A topology model template cannot foresee all these possible combinations, but some basic elements with their common relationships. The full configuration and multiplicities of the instances must involve manual configuration based on the understanding of customer-specific topology requirements.



(a) Topology Instance 1

(b) Topology Instance 2

Figure 6.12.: Examples of Topology Model Fragments

6.3. Derivation Activity 2: Configuration

As discussed in the Section 2.2.2 in Chapter 2, the typical development process of IAM systems is staged involving requirement engineers, hardware-oriented engineers and software developers. For this reason, the order of configuring the generated variability models is non-trivial. The configuration of the variability models should be aligned with the natural configuration stages. Although the derivation approach is supported by tooling during application engineering, the decisions are made following this sequence, which is important to answer RQ3.1 about the order of variability configuration.

6.3.1. Derivation Activity 2.1: Configure models

Purpose. This is a manual activity to configure the already instantiated variability models of the output of *Activity 1*.

Input. The generated processes and topology from *Activity 1* are the input.

Manual procedure. According to the derivation process presented in Figure 6.2, process models are configured at the first place, followed by the topology configuration. This is aligned with the natural decision making stages in practice as well.

Roles. Requirement engineers and hardware-oriented engineers are the two roles who perform the two sub-activities, *Derivation Activity 2.1.1* and *Derivation Activity 2.1.2* respectively.

Output. The configured process and topology models according to customer-specific requirements are the output of this activity.

Example. Examples are presented in following sub-activity sections.

6.3.2. Derivation Activity 2.1.1: Configure process models

Purpose. The system functions are usually process-like, due to the material flows in the domain of IAM systems as motivated in Chapter 2. Requirement engineers use many process specifications to describe the requirements. In practice, defining these processes is the first step of software development of IAM systems.

Input. The input to this step includes the automatically generated process models in the modeling space with model templates or model fragments.

Manual procedure. Figure 6.13 provides the manual procedure to configure processes.

```
for each instantiated process in the modeling space do
    Remove unnecessary actions;
    Add new actions into the process, if necessary;
    Re-order actions according to the needs of customers, if necessary;
    for each $Action$ in the process do
        Specify the types and attributes of $action$ according to customer requirements;
    for each $R_{\text{Order}}$ in the process do
        Specify the type of $R_{\text{Order}}$;
        Specify the attributes of the $R_{\text{Order}}$;
```

Figure 6.13.: The Manual Procedure of Configuring Process Models

Output. The configured process models are the output of this manual activity.

Tool Support. The process editor is developed to support the configuration.

Roles. The main stakeholders to perform this task are requirement engineers.

Example. A requirement engineer opens the process with the process editor shown in Figure 6.14. He or she describes the human systems interactions, the manual action to be performed by human workers, as well as the computational action.

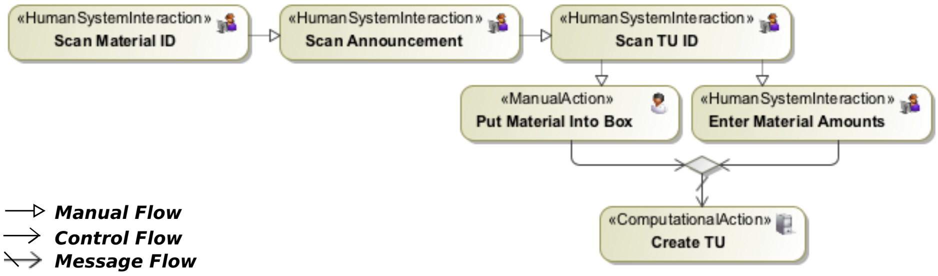


Figure 6.14.: An Example of a Configured Process

6.3.3. Derivation Activity 2.1.2: Configure topology models

Purpose. In the IAM domain, each customer requires a specific topological configuration. IAM systems manage and coordinate these hardware components. The topological configuration is necessary to be addressed in the software derivation.

Input. The input is the instantiated topology models in the modeling space.

Manual procedure. Figure 6.15 describes the manual procedure to guide this activity.

```
for each instantiated topology model in the modeling space do
    Understand the design of hardware distribution of the corresponding
      manufacturing area in the factory;
    Filter and select the software critical topological elements, since not all hardware
      components are relevant to software derivation;
    Specify the stationary elements;
    Specify the relationships among stationary elements with transporter elements;
    for each element in the topology model do
        Assign attributes, such as the $name$ and the $locationID$;
    for each stationary element in the topology model do
        Configure whether the element can handle multiple working units;
```

Figure 6.15.: The Manual Procedure of Configuring Topology Models

Output. The configured topology models are the output of this manual activity.

Tool Support. The topology editor has been developed to support the configuration.

Roles. The hardware-oriented engineers should perform this activity.

Example. A hardware-oriented engineer opens an instantiated topology model, for example the one shown in Figure 6.12b, and configures the elements in the model to describe topology. Figure 6.16 presents a screen-shot of a configured topology model.

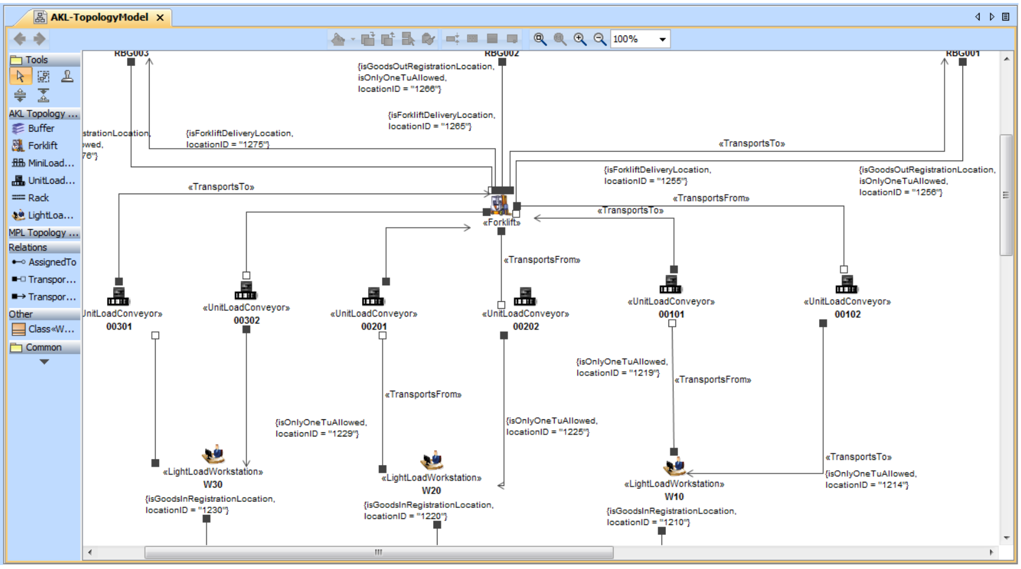


Figure 6.16.: A Screenshot of a Detailed Topology Model

6.3.4. Derivation Activity 2.2: Refine models

Purpose. The ultimate goal to configure all the models is to use them for the IAM software derivation. After the requirement engineers and hardware-oriented engineers have configured the variability models, the models may still not at the appropriate level of details for direct code generation. It is necessary to refine the models to add more details, to remove inconsistencies where necessary, in order to support code generation. Knowing the technical details and the input needs of the generator is important to perform this activity. The reason to differentiate this activity from the previous Activity 2.1, is that the stakeholder roles who supposed to perform this task is different.

Input. The processes and topology models in the modeling space.

Manual procedure. Figure 6.17 presents the manual procedure to guide this activity.

Output. The refined configuration of models in the modeling space is the output.

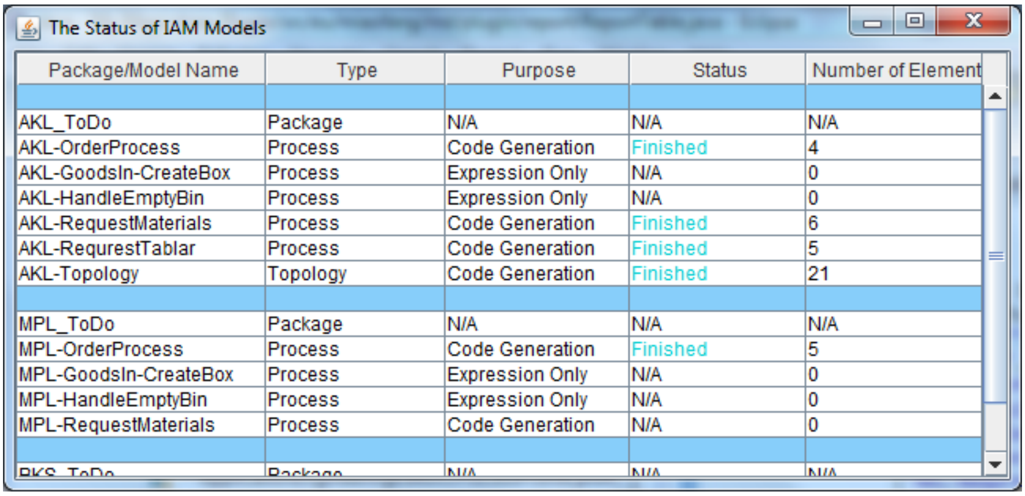
Roles. The software developers are responsible for this activity.

Tool Support. Figure 6.18 shows the tooling to help developers to understand the status of models during configuration. As presented in Figure 6.18, two types of models are differentiated. The purpose column shows the creation of models for different purposes: "Expression Only" and "Code Generation" purposes. The models with "Expression Only" allow the creation of models for communication purpose among the stakeholders by means of the domain-specific models. They are not taken as input for the next code generation activity, but they are useful to support the application-specific

```
for each instantiated process in the modeling space do
    for each $Action$ in the process do
        if $Action$ is location critical then
            Identify corresponding topological elements in the topology model;
            Relate action to the corresponding topological elements;
        Ensure the configurations of each attribute are configured;
        Ensure the configurations of each attribute are valid;
    Ensure the validity of process configurations;
for each instantiated topology model in the modeling space do
    for each element in topology model do
        Ensure the configurations of each attribute are configured;
        Ensure the configurations of each attribute are valid;
Finalize the models for code generation;
```

Figure 6.17.: The Manual Procedure of Model Refinement

development. Only those models with “Code Generation” purpose will be used as the input for the next activity for generation. The status column presents the status of each model, to show that whether the models are considered finished or not, which provides a overview of the progress of the model configuration. After all the models with the “Code Generation” purpose are changed to finished, the configurations are ready to be given to the next activity.



The Status of IAM Models

Package/Model Name	Type	Purpose	Status	Number of Element
AKL_ToDo	Package	N/A	N/A	N/A
AKL-OrderProcess	Process	Code Generation	Finished	4
AKL-GoodsIn-CreateBox	Process	Expression Only	N/A	0
AKL-HandleEmptyBin	Process	Expression Only	N/A	0
AKL-RequestMaterials	Process	Code Generation	Finished	6
AKL-RequrestTablar	Process	Code Generation	Finished	5
AKL-Topology	Topology	Code Generation	Finished	21
MPL_ToDo	Package	N/A	N/A	N/A
MPL-OrderProcess	Process	Code Generation	Finished	5
MPL-GoodsIn-CreateBox	Process	Expression Only	N/A	0
MPL-HandleEmptyBin	Process	Expression Only	N/A	0
MPL-RequestMaterials	Process	Code Generation	N/A	0
PKS_ToDo	Package	N/A	N/A	N/A

Figure 6.18.: The Report of Model Status

Example. A software developer opens a process model, as shown on the left-hand side of Figure 6.19, and realizes that the action *Receive Tu Arrival Notification* is location critical. By checking the corresponding topological model, the developer finds out that the action should be bound to the three *LightLoadWorkstations*. Accordingly, he or she adds the three *locationIDs* to its *boundToLocations* collection. With these detailed information, this model is ready for code generation.

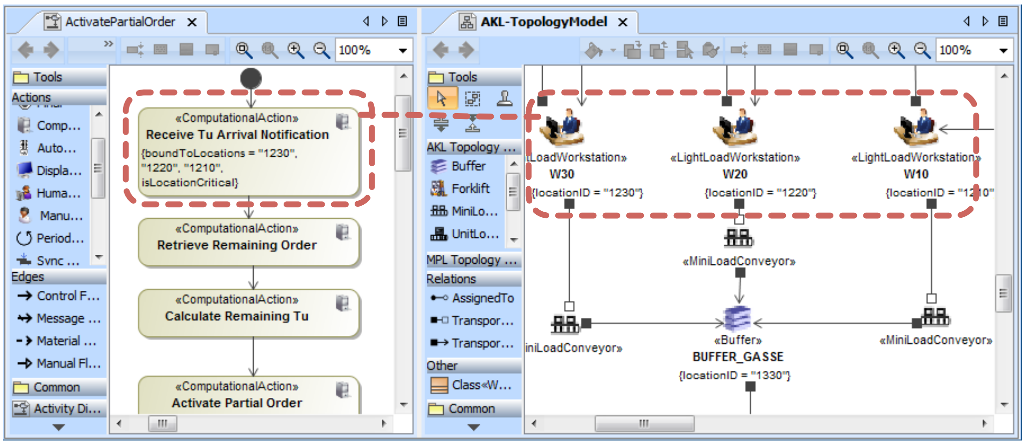


Figure 6.19.: An Example of Linking Processes and Topology Models

6.4. Derivation Activity 3: Generation

This derivation activity answers *RQ3.2: How to use the configuration of multiple variability types as input to improve derivation efficiency?* The aim of this activity is to use the configuration of multiple variability models as input to finally derive the software application. Activity 3.1 and Activity 3.2 are aligned with the derivation at the two levels of abstraction as introduced in Section 5.1 in the previous chapter: the architecture derivation and the detailed derivation.

6.4.1. Derivation Activity 3.1: Resolve Architectural Variability

Purpose. The purpose of Activity 3.1 is to resolve architectural level variability. The variability realization techniques are architecture re-organization, optional components and variant components (see Section 5.2 in Chapter 5). Therefore, at this level, the derivation infrastructure mainly has to decide the inclusion and exclusion of components. During the development of feature models, as suggested in Section 4.2.2 of Chapter 4, the components as architectural elements are mirrored as features in feature models. When the decision to include a component is made, the derivation infrastructure should include the corresponding code templates of the component to prepare for the final code generation activity.

Input. The input of this activity includes configured features, the root component of the targeted software product line and its hierarchical variability models.

Procedure. Figure 6.20 shows the pseudo code of this activity to include the application-specific artifacts, starting with the root component.

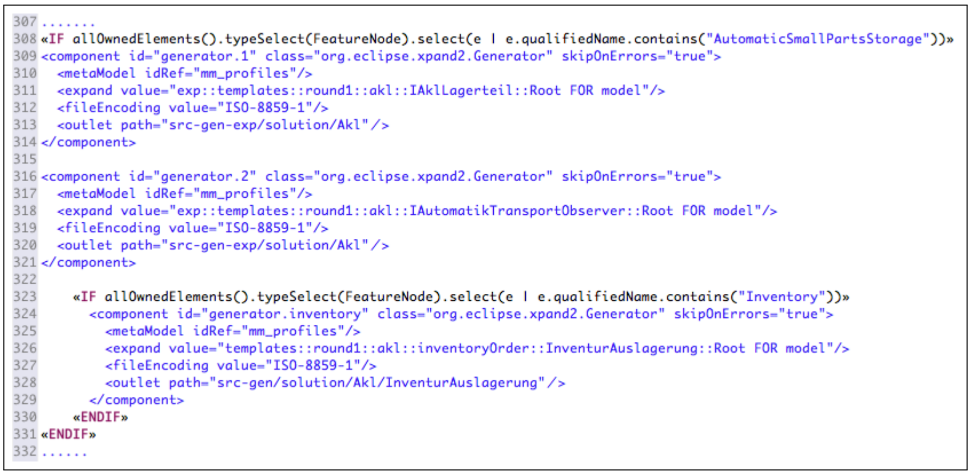
```
Let $RootComponent$ be the root component of the product line;
Let $ConfiguedFeatures$ be the list of configured features;
Let $List_{\text{codeTemplate}}$ be the list of code templates for code generation;
DecideComponentInclusion($RootComponent$) ;
Function DecideComponentInclusion($Component$)
    if the corresponding feature of $Component$ is in $ConfiguedFeatures$ then
        Attach all code templates of $Component$ to $List_{\text{codeTemplate}}$;
    for each $Sub\text{-}Component \in Components$ do
        DecideComponentInclusion($Sub\text{-}Component$);
```

Figure 6.20.: Derivation at the Architecture Level

Output. A list of code templates is the output of this activity.

Roles. Since this is an automated activity, the derivation infrastructure fully supports the execution. The candidate users of the derivation infrastructure for this activity can be, for example, software developers or architects.

Example. Figure 6.21 shows a fragment of an architecture derivation script. The *AutomaticSmallPartsStorage* is a composite component (line 308), which contains a sub-component named *Inventory* (line 323). As can be seen, the script checks whether the features are selected or not, to decide the inclusion and exclusion of their code templates. The code generator includes the code templates for all the components and sub-components for which the corresponding features are selected.



```
307 .......
308 «IF allOwnedElements().typeSelect(FeatureNode).select(e | e.qualifiedName.contains("AutomaticSmallPartsStorage"))»
309 <component id="generator.1" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
310   <metaModel idRef="mm_profiles"/>
311   <expand value="exp::templates::round1::akl::IAklLagerteil::Root FOR model"/>
312   <fileEncoding value="ISO-8859-1"/>
313   <outlet path="src-gen-exp/solution/Akl"/>
314 </component>
315
316 <component id="generator.2" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
317   <metaModel idRef="mm_profiles"/>
318   <expand value="exp::templates::round1::akl::IAutomatikTransportObserver::Root FOR model"/>
319   <fileEncoding value="ISO-8859-1"/>
320   <outlet path="src-gen-exp/solution/Akl"/>
321 </component>
322
323     «IF allOwnedElements().typeSelect(FeatureNode).select(e | e.qualifiedName.contains("Inventory"))»
324       <component id="generator.inventory" class="org.eclipse.xpand2.Generator" skipOnErrors="true">
325         <metaModel idRef="mm_profiles"/>
326         <expand value="templates::round1::akl::inventoryOrder::InventurAuslagerung::Root FOR model"/>
327         <fileEncoding value="ISO-8859-1"/>
328         <outlet path="src-gen/solution/Akl/InventurAuslagerung"/>
329       </component>
330     «ENDIF»
331 «ENDIF»
332 ......
```

Figure 6.21.: A Fragment of an Architecture Derivation Script

6.4.2. Derivation Activity 3.2: Resolve Process and Topology Variability

Purpose. As the last derivation step of the MADE approach, Activity 3.2 finally generates the source code of specific applications. Especially, it resolves the process and topology variability finally at the detailed code derivation level.

Input. The input data of this activity includes the configured and refined process and topology models within the modeling space and the list of code templates as the output of Activity 3.1.

Procedure. The procedure of resolving each code template is specific depending on its function and implementation. Some very simple code templates may have only simple parameters to be resolved. Some others may realize multiple variability types, based on the configuration of multi-variability models. Figure 6.22 illustrates a situation of a derivation procedure with multiple variability types to resolve in a code template.

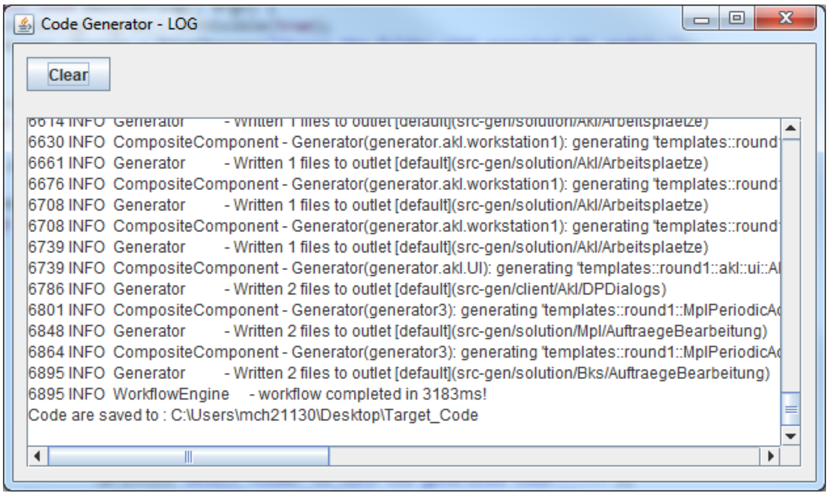
```
Let $List_{\text{codeTemplate}}$ be the list of code templates for code generation;
Let $P_{\text{A}}$ be the set of all actions in a process model;
for each $codeTemplate \in List_{codeTemplate}$ do
    for each $action \in P_A$ do
        Include corresponding reusable business logic code;
        if $action$ is location critical then
            Let $G_{\text{LogicalGroup}}$ be the logical group of $action$ ;
            Traverse the value of the elements in the $G_{\text{LogicalGroup}}$ of $action$;
            Collect the elements in $G_{\text{LogicalGroup}}$;
        for each $R_{Order}$ in $action$ do
            if $action \rightsquigarrow nextAction$ then
                Generate the invocation method to $nextAction$;
```

Figure 6.22.: The Detailed Derivation

Roles. Similar to Activity 3.1, the candidate users of the derivation infrastructure to finally derive the applications can be key software developers or architects.

Output. The generated (partial) application is the output of this activity.

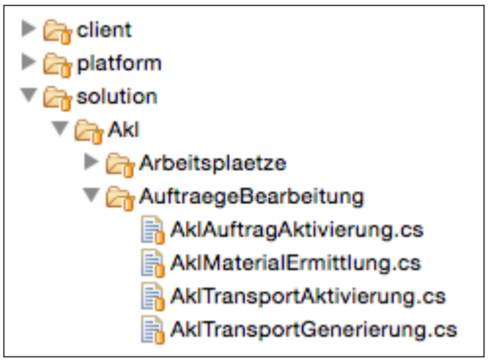
Tool Support. The code generator is a part of the derivation infrastructure. It includes an importer of the configured variability models in the modeling space. Then, it generates the source code. Figure 6.23 shows a screen-shot of the log output of the developed tooling for code generation. It reports the number of files generated to the target folders in the code structure. As can be seen, the time spent on this automated code generation is quite fast. It takes usually some seconds.



```
6614 INFO  Generator        - Written 1 files to outlet [default](src-gen/solution/Akl/Arbeitsplaetze)
6630 INFO  CompositeComponent - Generator(generator.akl.workstation1): generating 'templates::round
6661 INFO  Generator        - Written 1 files to outlet [default](src-gen/solution/Akl/Arbeitsplaetze)
6676 INFO  CompositeComponent - Generator(generator.akl.workstation1): generating 'templates::round
6708 INFO  Generator        - Written 1 files to outlet [default](src-gen/solution/Akl/Arbeitsplaetze)
6708 INFO  CompositeComponent - Generator(generator.akl.workstation1): generating 'templates::round
6739 INFO  Generator        - Written 1 files to outlet [default](src-gen/solution/Akl/Arbeitsplaetze)
6739 INFO  CompositeComponent - Generator(generator.akl.UI): generating 'templates::round1::akl::ui::Al
6786 INFO  Generator        - Written 2 files to outlet [default](src-gen/client/Akl/DPDialogs)
6801 INFO  CompositeComponent - Generator(generator3): generating 'templates::round1::MplPeriodicAc
6848 INFO  Generator        - Written 2 files to outlet [default](src-gen/solution/Mpl/AuftraegeBearbeitung)
6864 INFO  CompositeComponent - Generator(generator3): generating 'templates::round1::MplPeriodicAc
6895 INFO  Generator        - Written 2 files to outlet [default](src-gen/solution/Bks/AuftraegeBearbeitung)
6895 INFO  WorkflowEngine    - workflow completed in 3183ms!
Code are saved to : C:\Users\mch21130\Desktop\Target_Code
```

Figure 6.23.: Tool Support for Code Generation

Example. Figure 6.24 shows a screen-shot of the generated package structure and classes. It is the output of this derivation activity.



```
▶ client
▶ platform
▼ solution
    ▼ Akl
        ▶ Arbeitsplaetze
        ▼ AuftraegeBearbeitung
              AklAuftragAktivierung.cs
              AklMaterialErmittlung.cs
              AklTransportAktivierung.cs
              AklTransportGenerierung.cs
```

Figure 6.24.: An Example of Generated Source Code

6.5. Discussions and Limitation Analysis

The derivation approach presented in this chapter combines the configuration of three different variability models into a semi-automated derivation process, aiming at helping derivation stakeholders and improving their efficiency. To fully adopt the approach in practice, it might have certain limitations. This section discusses several of these aspects.

6.5.1. Model Consistency and Interaction during Configuration

During configuration time, the consistency among the configured modeling elements is important to be ensured. For verification of one variability type, many existing approaches in the state of the art report related functions, such as consistency checking [WBS$^+$10, TBG13], error fixing [ELSP11], completeness checking [AHH11], etc. For the verification of configuration among several variability types, Kowal and Schaefer propose an incremental consistency checking approach [KS16], in which they point out that the consistency checking has several scopes and levels. For example, consistency checking can be isolated within a single variability type, among variability types, or even cross the application-specific development phases. These functions are not yet integrated in the MADE approach. As future work, integrating these approaches for configuration verification can be important for industrial adoption to improve the confidence of stakeholders when using the approach in their day-to-day work.

Besides the consistency aspect, the interaction among the three different variability models during configuration time is also challenging. The selection of a feature can determine the inclusion of several other topology and process models. In complex industrial, it is possible that a configuration in process elements demands certain changes backwards to feature modeling. Currently, in the proposed approach, the support of model interaction is still limited. The first derivation activity requires configuring features. Based on the feature configuration, the derivation tooling instantiates the corresponding process and topology models. By doing so, the interaction between features and the other two types of models are partially solved in a proactive way. Only the required topology models and processes are instantiated in the modeling space. When adopting the approach in practice, it is necessary to integrate the propagation support into derivation tooling to automatically navigate the users through the impacted decisions and configurations [CNCJ14, UBFC14].

6.5.2. Evolution of Model and Generated Artifacts

Tracking changes in models after modification. One of the consequences when allowing the configuration of features, processes and topology in different stages is to handle the model modification across the stages. The derivation infrastructure ideally needs to support the modification of models during model-to-model transformation. For example, at Activity 2.2, the software developers may realize some misconfigurations and need to modify the configurations of feature models, which are already done in Activity 1.1. The derivation approach should allow the modification of re-configuration the feature model and re-instantiation the process and topology models without affecting the already configured ones. At the moment, the implementation of the approach partially supports it. After the re-configuration and re-instantiation of features, the derivation infrastructure updates the modeling space, by adding newly instantiated process and topology models. In case, the newly instantiated models have conflicts with the existing models in the modeling space, the derivation infrastructure keeps both for manual comparisons and decisions.

Tracking changes in generated code after modification. The derivation infrastructure should ideally support the modification of models during model-to-code transformation. For example, after the code derivation of Activity 3.2, during application-specific development, the derivation stakeholders might find that a modification must be made in the already configured multi-variability models. Even with the state-of-the art approaches, this is still difficult to achieve, due to the complexity of model-to-code traceability, to the extent of the author's knowledge. There are existing program generation patterns can be used to alleviate the impact of modification in models [Voe03].

6.6. Comparison with Related Work

This section provides the discussion about related works, which could be possibly the alternatives, as well as the rationales why they are not taken as part of the approach.

6.6.1. Related Work in Derivation Processes

Rabiser et al. propose key activities during product derivation [ROR11]. The three main activities are: Preparing for Derivation, Product Derivation/Configuration, and Additional Development/Testing [ROR11], which was inspiring the design and development of the derivation approach in this chapter. Especially, preparation for and configuration becomes the Activity 1 and Activity 2 (c.f. Section 6.2.1 and Section 6.3). Activity 3, generation, is explicitly separated as a main activity, because it is fully automated in the derivation approach.

In addition, Chapter 3 has already reported several generic derivation processes. For example, O'Leary et al. propose Pro-PD as a reference process for software product derivation. Pro-PD contains six steps covering the whole life-cycle of application engineering, including Initiate Project, Identify and Refine Requirements, Derive the Product, Develop the Product, Test the Product, and Management and Assessment [ODAR12]. Compared to Pro-PD, the derivation process presented in this chapter provides technical support and the design of the derivation infrastructure. This approach does not aim at covering the whole application engineering practices as proposed in Pro-PD. Instead, it focuses on the *Derive the Product* step in Pro-PD, with technical automation support for the IAM systems.

However, the order of configuration of the different variability types is also important to the staged derivation of IAM systems, which is the key to answer the sub-question RQ3.1 of this chapter. In IAM system development, hardware topology and configurations usually are decided earlier than software solutions, which are driven by space in factories and the budget of construction. Software solutions is decided later than hardware configuration, and sometimes need to compensate some of the limitations. To the best of the author's knowledge, existing approaches do not support the configuration of multiple variability types in the staged product derivation process. Especially, no existing approaches explicitly addresses the process, topology variability and their interrelations in the same derivation process. Besides, the proposed approach also enables the model-level reuse, by automatically instantiating reusable model templates.

6.6.2. Related Work in Integration of Variability Models

Chapter 3 has also included several approaches involving several variability model types. The variability modeling techniques could be both graphical and textual. For example, Behjati et al. propose a method named SimPL [BYBS13], which provides multiple views to the users to specify integrated control systems. In this approach, the variability of software, hardware and their inter-dependencies are modeled in different views. The derivation process in SimPL is focused on interactive user guidance and consistency checking of variability configuration. EASy-Producer supports also the modeling of several variability types, such as the textual variability modeling language and topology modeling [ES15]. Voelter and Visser propose to use domain-specific languages in the context of SPLE [VV11]. On the one hand, using multiple variability modeling techniques brings the benefits of more flexible variability expression, which means more variability could be modeled. On the other hand, how to integrate the configuration of them into a derivation process becomes a challenge.

To differ from these approaches, the derivation activities presented in this chapter focuses on automation of the derivation process, such as the model instantiation and code generation. The order of configuring feature, topology and process-models is aligned with the natural decision making process in the IAM development (see Section 2.2.2). These two aspects are not well supported by the existing approaches.

6.7. Validation Objectives

The derivation approach presented in this chapter integrates three types of variability into a staged derivation process. The primary goal of having such an approach is to help the derivation stakeholders to improve their efficiency at derivation time. Automating the derivation process and generating code artifacts can significantly save their time and manual work. Therefore, the first validation objective of this chapter, in terms of hypotheses, should be:

> **H3: Automated level of derivation.** The approach improves the code generation rate significantly (>35%), compared with the derivation without using the approach.

Besides the automation level of derivation, whether the users of the approach can easily accept the approach should be tested:

H4: Users' perceived usefulness and ease of use. The approach can be easily accepted by users for their derivation tasks.

By modeling the multiple variability types, realizing variability and automating the derivation process, the ultimate goal is to help the stakeholders in the target domain, to shorten the time-to-delivery of software systems to customers. Therefore, the Hypothesis 5 should be evaluated:

H5: Time saving during application engineering. By applying the approach, the stakeholders can shorten the time spent during application engineering.

6.8. Chapter Summary

The derivation approach presented in this chapter has three main activities during application engineering:

1. The **preparation** activity is a semi-automated activity to prepare a base configuration with instantiated models and reusable model templates to help derivation stakeholders to start their configuration tasks. It enables the model-level reuse, compared to other state-of-the-art approaches.
2. The **configuration** activity, as a manual activity, allows the stakeholders to configure the models according to customer needs. Requirement engineers, hardware-oriented engineers and software developers participate in the sub-activities, with their own variability concerns. These manual activities are supported by tooling in the proposed approach.
3. The **generation** activity resolves the variability both on architecture and code derivation levels to finally derive the customer-specific application. It enables fine-grained reuse of code templates in the core asset base.

This semi-automated derivation process describes the order of configuration steps when multiple stakeholders are involved with their variability concerns, which becomes the answer to RQ3.1.

The **generation** activity resolves the variability at two levels of abstraction. Firstly, architectural variability is solved by component inclusion and exclusion. It generates an intermediate list of all the relevant code generation templates. Secondly, it uses the intermediate list to finally derive the application. With the configured multi-variability models and code generation techniques, the approach potentially reduces the amount of manual tasks of stakeholders during derivation, which becomes the answer to RQ3.2.

Figure 6.25 shows the overview of the MADE approach. Chapter 4 has introduced a multi-variability modeling framework with meta-models. Chapter 5 has introduced the taxonomy for multi-variability realization, especially on resolving the process and topology variability of IAM at the detailed derivation level. In this chapter, the semi-automated derivation approach integrates the multiple variability types during the derivation process, by involving the configuration of multi-variability at the three derivation stages, which becomes the answer to RQ3.

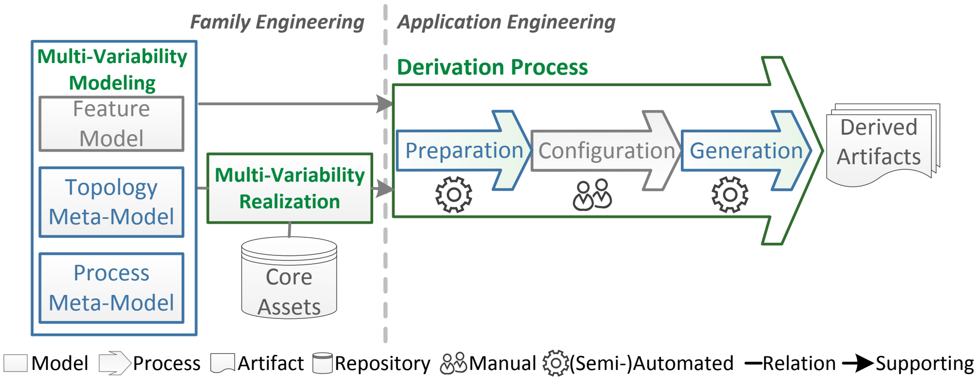


Figure 6.25.: The Multi-vAriability Derivation (MADE) approach

Chapter 4, Chapter 5 and this chapter have already shown the complete approach for IAM software derivation of this thesis. The next chapter will report on the evaluation of the approach from both research and practical perspectives.

7 Validation

"For the things we have to learn before we can do them,

we learn by doing them."

–Aristotle

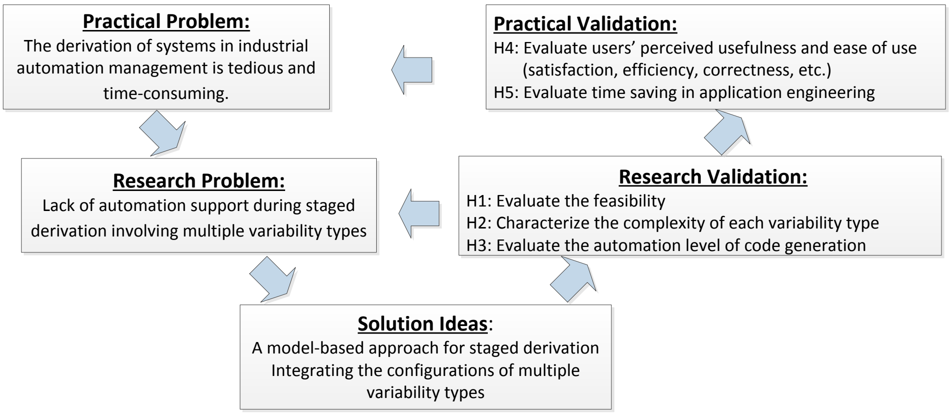
This thesis aims at developing an approach to improve the software product derivation of IAM systems. Chapter 4 has presented the hierarchical multi-variability modeling framework involving feature-, process- and topology models for variability representation. Chapter 5 has proposed the taxonomy for variability realization techniques to implement the multiple variability types in core assets. Chapter 6 has proposed the Multi-vAriability Derivation (MADE) approach, which integrates the configuration of multi-variability into a semi-automated derivation process.

This chapter reports on the validation results of the proposed derivation approach. With this chapter, the research question, *"RQ4. How good is the proposed approach on improving IAM product derivation?"* will be answered.

The remainder of this chapter is structured as follows: Section 7.1 defines the validation goals, the questions and their corresponding metrics to be used for measurements. Section 7.2 motivates the chosen research methodologies for validation, in which the rationales of using a case study and semi-structured interviews in this thesis are explained. Section 7.3 and Section 7.4 report on the case study and the semi-structured interviews respectively. Finally, Section 7.5 provides the summary of the validation results.

7.1. Validation Goals, Questions, and Metrics

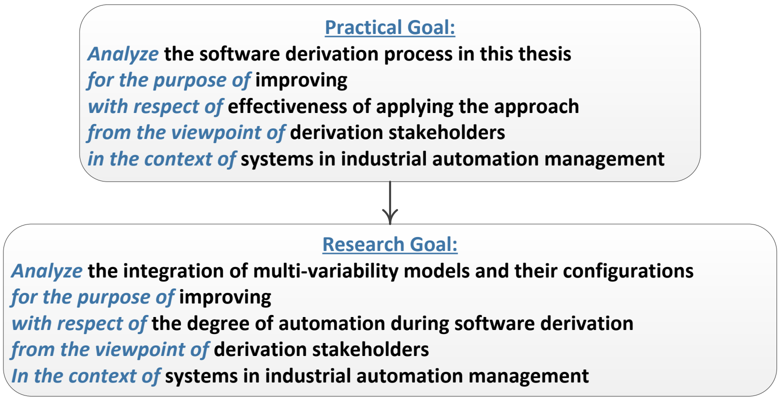
Figure 7.1 shows the research process of this thesis, which has been already introduced in Section 1.3.1. At the beginning, the practical problem was formulated to motivate this thesis. Afterwards, the research problem was defined based on the limitation analysis of the state-of-the-art approaches. Then, the MADE approach as the solution has been presented. By following this process, this chapter presents the validation of the proposed approach at both the research level and at the practical level.



Practical Problem:
The derivation of systems in industrial automation management is tedious and time-consuming.
Practical Validation:
H4: Evaluate users' perceived usefulness and ease of use (satisfaction, efficiency, correctness, etc.)
H5: Evaluate time saving in application engineering
Research Problem:
Lack of automation support during staged derivation involving multiple variability types
Research Validation:
H1: Evaluate the feasibility
H2: Characterize the complexity of each variability type
H3: Evaluate the automation level of code generation
Solution Ideas:
A model-based approach for staged derivation Integrating the configurations of multiple variability types

Figure 7.1.: The Research Process – Re-Visit

Figure 7.2 presents the two goals. At the research level, the validation aims at testing the improvement on automating the derivation activities. At the practical level, the validation aims at testing the effectiveness of using the approach for software derivation in practice. The next two subsections further break down the goals into their questions and corresponding measures at both the research level and the practical level, guided by the goals, questions and metrics (GQM) approach [VSB99, BCR94, VSBCR02].



Practical Goal:
Analyze the software derivation process in this thesis
for the purpose of improving
with respect of effectiveness of applying the approach
from the viewpoint of derivation stakeholders
in the context of systems in industrial automation management
Research Goal:
Analyze the integration of multi-variability models and their configurations
for the purpose of improving
with respect of the degree of automation during software derivation
from the viewpoint of derivation stakeholders
In the context of systems in industrial automation management

Figure 7.2.: The Main Validation Goals

7.1.1. GQM and Hypotheses at the Research Goal Level

The GQM measurement model at the research level is presented in Figure 7.3, based on the guideline proposed by Solingen and Berghout [VSB99]. Three sub-research-goals (SRG) are derived as can be seen in Figure 7.3: evaluate the feasibility (SRG1), characterize the composition of each variability type in the variability models of the target systems (SRG2), and evaluate the automation level of code generation (SRG3). To achieve these goals, the questions and metrics are further derived.

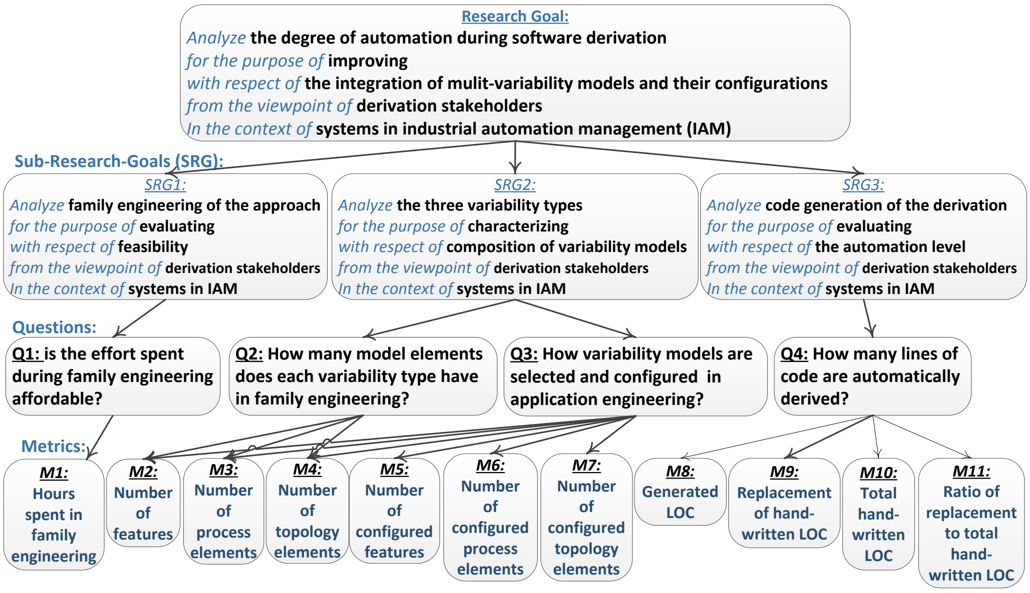

Research Goal:
Analyze the degree of automation during software derivation
for the purpose of improving
with respect of the integration of mulit-variability models and their configurations
from the viewpoint of derivation stakeholders
In the context of systems in industrial automation management (IAM)
Sub-Research-Goals (SRG):
SRG1:
Analyze family engineering of the approach
for the purpose of evaluating
with respect of feasibility
from the viewpoint of derivation stakeholders
In the context of systems in IAM
SRG2:
Analyze the three variability types
for the purpose of characterizing
with respect of composition of variability models
from the viewpoint of derivation stakeholders
In the context of systems in IAM
SRG3:
Analyze code generation of the derivation
for the purpose of evaluating
with respect of the automation level
from the viewpoint of derivation stakeholders
In the context of systems in IAM
Questions:
Q1: is the effort spent during family engineering affordable?
Q2: How many model elements does each variability type have in family engineering?
Q3: How variability models are selected and configured in application engineering?
Q4: How many lines of code are automatically derived?
Metrics:
M1: Hours spent in family engineering
M2: Number of features
M3: Number of process elements
M4: Number of topology elements
M5: Number of configured features
M6: Number of configured process elements
M7: Number of configured topology elements
M8: Generated LOC
M9: Replacement of hand-written LOC
M10: Total hand-written LOC
M11: Ratio of replacement to total hand-written LOC

Figure 7.3.: Goals, Questions and Metrics at the Research Level

SRG1: It is important to evaluate whether it is feasible to apply the approach, especially in family engineering. If the cost to come up with the software product line infrastructure is too high, it would be difficult to pay off when using the product line infrastructure in application engineering. For this reason, Q1 is derived to evaluate the effort, which should be affordable by the development team. The corresponding metric is the time spent on enabling the approach in terms of re-engineering and developing the reusable artifacts in family engineering. The hypothesis to address SRG1 and the metric are defined as follows.

Hypothesis 1 (H1): feasibility (addresses SRG1)

- H1: It is feasible to apply the approach in family engineering.

Metrics:

- M1: The time spent on enabling the approach during family engineering.

SRG2: The approach presented in this thesis integrates process and topology variability models besides feature modeling, to enhance the variability modeling and expression.

To validate the approach, it is necessary to characterize how the three variability types are selected and configured to support software derivation as the second sub-goal shown in Figure 7.3. To accomplish this goal, the GQM tree includes Q2 about the numbers of feature, process and topology variability elements in family engineering. Additionally, the size of each variability type, in terms of lines of code (LOC) is important to characterize the composition of them in the modeling space. Q3 is derived to understand how the three variability models are used in application engineering. The hypothesis to address SRG2 of characterization of variability types and the metrics are defined as follows.

Hypothesis 2 (H2): Composition of variability models (addresses SRG2)

- H2: The multi-variability modeling, by introducing process and topology models, can capture significantly more variability (>50%), compared to feature modeling.

Metrics:

- M2: The number of features.
- M3: The number of process elements.
- M4: The number of topological elements.
- M5: The number of configured features.
- M6: The number of configured process elements.
- M7: The number of configured topological elements.

SRG3: The third sub-goal is with regard of effectiveness of automated code generation. The question of SRG3 is the generated lines of code by applying the approach in derivation. However, the generated code is based on product line architecture after re-engineering, in stead of the original code based on legacy software architecture. As a result, the comparison of the automatically derived lines of code requires the analysis of how many hand-written code can be replaced. Furthermore, the total lines of code from the original hand-written code are included in the metrics, in order to understand how effectiveness of the generated lines of code. Automation level of derivation (H3) and its metrics are defined as:

Hypothesis 3 (H3): Automation level of derivation (addresses SRG3)

- H3: The approach improves the code generation rate significantly (>35%), compared with the derivation without using the approach.

Metrics:

- M8: The generated LOC with the proposed approach.
- M9: The replacement of hand-written LOC without the proposed approach.
- M10: The total of hand-written LOC without the proposed approach.
- M11: The ratio of the replacement of hand-written LOC and the total of hand-written LOC without the proposed approach.

7.1.2. GQM and Hypotheses at the Practical Goal Level

The GQM measurement model at the practical level is proposed in Figure 7.3. In order to achieve the overall practical goal, two questions are derived.

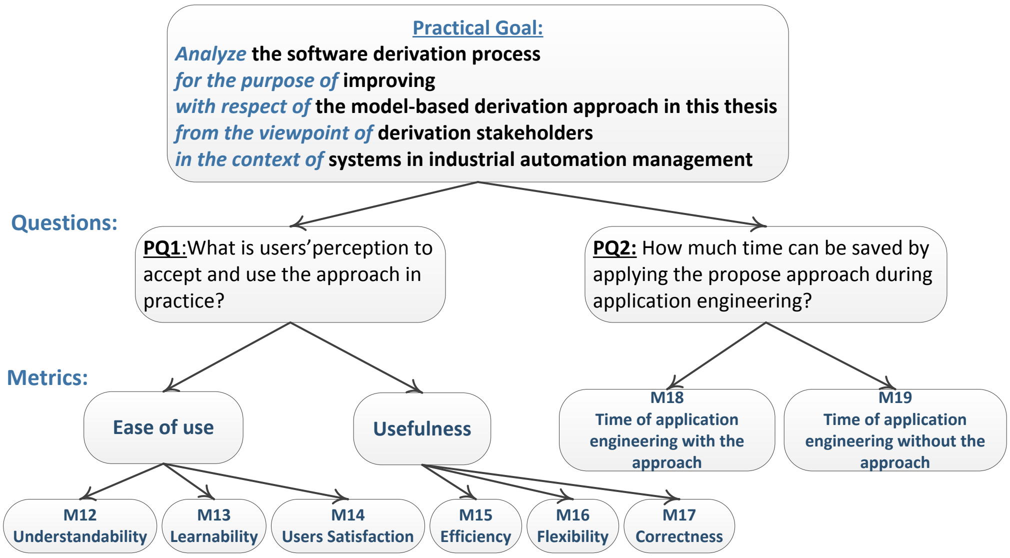

Practical Goal:
Analyze the software derivation process
for the purpose of improving
with respect of the model-based derivation approach in this thesis
from the viewpoint of derivation stakeholders
in the context of systems in industrial automation management
Questions:
PQ1:What is users'perception to accept and use the approach in practice?
PQ2: How much time can be saved by applying the propose approach during application engineering?
Metrics:
Ease of use
Usefulness
M18 Time of application engineering with the approach
M19 Time of application engineering without the approach
M12 Understandability
M13 Learnability
M14 Users Satisfaction
M15 Efficiency
M16 Flexibility
M17 Correctness

Figure 7.4.: Goal, Questions and Metrics at the Practical Level

PQ1 aims at evaluating the derivation stakeholders' perception of using the proposed approach from a practical viewpoint. These stakeholders are potentially the users of the approach in the real industrial context. The hypothesis, users' acceptance of the approach (H4), is defined according to the technology acceptance model [Chu09, Dav89]. The assessment is based on the users' perceptions regarding the approach to be applied in their working context, compared to their state of practice without a systematic derivation approach.

Hypothesis 4 (H4): Users' perceived usefulness and ease of use (addresses PG1)

- H4: The approach in application engineering can be easily accepted by users for their derivation tasks.

Metrics:

- Subjective assessment on perceived usefulness and perceived ease-of-use, such as M12 understandability, M13 learnability, M14 satisfaction, M15 efficiency, M16 flexibility and M17 correctness.

PQ2 aims at evaluating the overall improvement of efficiency during application engineering. The suitable metric to be tested is the time saving during application engineering.

Hypothesis 5 (H5): efficiency during application engineering

- H5: Applying derivation approach in application engineering can shorten 1/3 of time spent on application engineering.

Metrics:

- M18: The time spent during application engineering by applying the approach.
- M19: The time spent during application engineering without the approach.

7.1.3. Interrelation of Hypotheses and Derivation Approach

Figure 7.5 inter-relates the hypotheses to the proposed multi-variability derivation approach. As can be seen, the hypotheses have different focus. For family engineering, the H1 evaluates whether it is feasible to establish the core assets to come up with the foundation of the approach. The H2, the evaluation of the complexity of multi-variability types is aimed at understanding IAM systems and how they are influenced by the different variability types. The H3 is aimed at evaluating how good is the proposed approach on automating code-level derivation. Both H4 and H5 focus mainly on application engineering. The H4 tests the users' perceived usefulness and ease of use during derivation. H5 is defined to further evaluate the time saving by applying the approach, to understand after how many application-specific derivations and development the approach would pay-off.

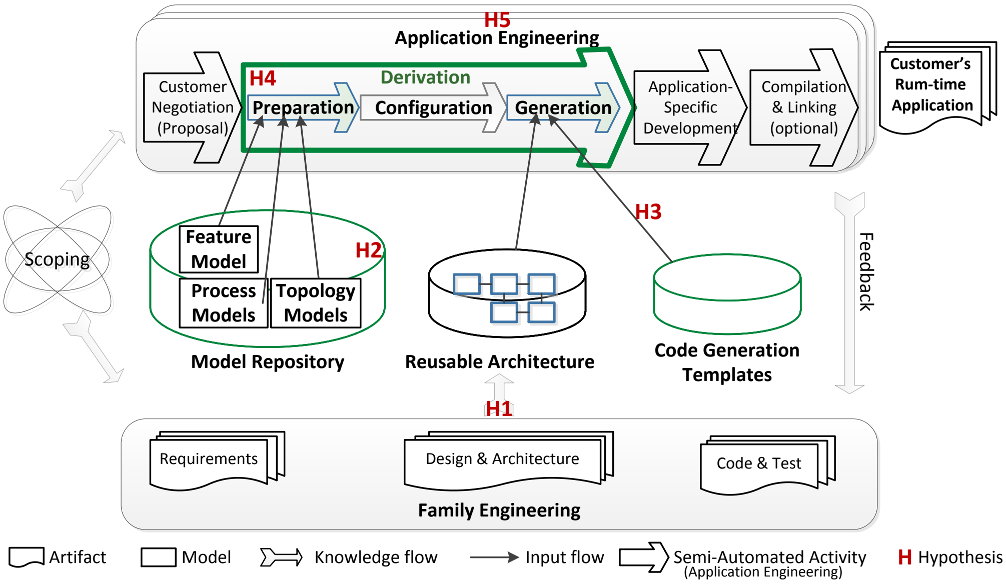


Figure 7.5.: The Coverage of Validation Goals

7.2. Rationales of Choosing Research Methodologies

According to the GQM approach, the (sub-)goals for validation at the research and practical levels were defined. Then, it is important to carefully choose and plan the validation with reasonable and feasible research methodologies.

Software engineering researchers and practitioners use various qualitative and quantitative validation methods to show evidence to support their research results. This section presents the analysis and the rationales of the validation methods chosen in this thesis.

According to Shaw [Sha02], the tension of software engineering research is between:

- "narrow truths proved convincingly by statistically sound experiments", and
- "broad 'truths', generally applicable, but supported only by possibly unrepresentative observations" [Bro88, Sha02]

Shaw further points out that the former satisfies the gold standard of scientific research, but are few and narrow compared to the decision makers in practice daily. The latter provides pragmatic guidance that are closer to decision makers, but at risk of overgeneralization [Sha02]. Choosing suitable methods to obtain the validation results should be based on the type of the proposed approaches. For example, a formal model should be supported by rigorous derivation and proof, not by one or two simplified examples, whereas, a representative example derived from a practical system may play a major role in validating a new type of development method [Sha03].

Shaw examined the accepted papers at International Conference on Software Engineering (ICSE) in 2002, and concluded that among different validation types, the most successful kinds of validation were based on the analysis of real-world experiences [Sha02, Sha03]. In particular, a "slice-of-life" example based on a real system is most likely to be convincing, given the explanation of why the simplified example is representative to the essence of the problem being solved [Sha03].

Reflecting to the context of this thesis, the practical problem in Section 2.2 stems from the problems observed in the state of practice, in particular that the current development and derivation process of IAM is tedious and time-consuming. The characteristics of IAM systems lead to difficulties when trying to apply the prior approaches to support the variability modeling and derivation, as pointed out in the work of the author of this thesis [FLED13b]. On the one hand, the process and topology variability causes the difficulties when using general-purpose variability models, such as feature modeling, for variability representation. On the other hand, three roles of stakeholders participate in the derivation process of IAM systems with different variability concerns. This thesis presents an approach of developing software product derivation infrastructure for IAM systems.

Thus, to understand the improvement brought by using such an approach, a "slice-of-life" is suitable to be applied. A complex and representative component was carefully chosen as a **case study** in the Siemens warehouse management system, in which both qualitative and quantitative data are collected. In addition, novel findings regarding the approach can possibly be captured by such in-depth case studies in a real-life context [Yin13, Eis89]. The hypotheses at the research level, including H1, H2 and H3 in Section 7.1.1, are tested with the case study. Section 7.3 reports the case study with the results and experiences.

The second part of validation for this thesis includes **semi-structured interviews** as the research method. The goal of conducting the semi-structured interviews was to understand the perceptions and acceptance of the approach by the potential users in practice. To mitigate the risks of over-generalization, the invited interviewees have practical experiences from different sub-domains of industrial automation management. The H4 at the practical level defined in Section 7.1.2 should be tested. Section 7.4 reports on the semi-structured interviews with systematic analysis about the benefits and potential risks of full-adoption.

To test the hypothesis 5 (H5), the ideal research method would be to do an experiment with a treatment group and a control group of software engineers. Since the development of industrial automation management requires already domain knowledge, such an experiment requires engineers with certain domain background. It is not feasible to achieve the validation of H5 within the limited time and resources of this thesis, but it is considered to be a part of future work.

7.3. Case Study

The objective of conducting this case study is to fulfill the validation goals defined at the research level. Setting up a clear scope is critical for validation, since there are many existing product derivation approaches in the state of the art as analyzed in Chapter 3. In fact, it is not possible to compare the effectiveness and improvement of the presented MADE approach with each of the existing approaches. As analyzed in Chapter 3, some of the approaches in the state of the art have been already tailored to fit to a particular software product line, especially the approaches focusing on the solution space, such as architecture description languages. These approaches are not applicable to IAM systems. In addition, in the software product line community, many derivation approaches are with general-purpose variability models. Some researchers and practitioners have mentioned that feature modeling is de facto standard method of variability representation [HLHE11, ACLF13]. For this reason, the scope of this case study is based on the analysis, validation and discussion compared with derivation approaches with general-purpose variability models, in particular feature models.

This case study was planned according to case study guidelines [Yin13, RH09]. It comprises the tasks of planning, executing, and finally reporting the results, which have been published in [FLDE16]. The remainder of this section is organized to follow these tasks.

7.3.1. Planning

This case study is based on a complex component of a warehouse management system developed by Siemens in the Nürnberg area, Germany. Figure 7.6 shows the simplified layer architecture (without technical details) of this software family. The architecture of this software family contains a platform layer and a solution layer. At the design time of this software family, most of the components in the platform layer were developed as commonalities, and they are well reusable across different solutions. As presented in Figure 7.6, the platform layer comprises mostly utility components, such as *Timer*, *Logging* and *Data Access*. The components in the solution layer are business related and customer-specific. They were not reused, due to the complex characteristics of the IAM variability types.

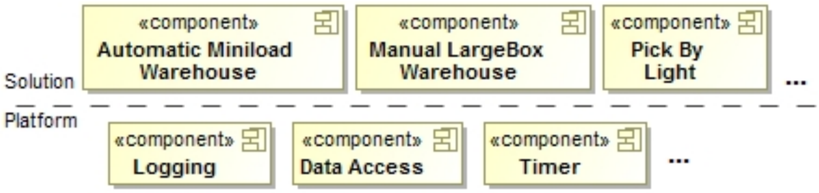


Figure 7.6.: The Simplified Layered Architecture

Subject. According to several discussions with domain experts and experienced researchers, the *Automatic Miniload Warehouse* component in Figure 7.6 was carefully chosen as the subject of this case study, since it is a typical representative with regard to domain complexities, size and widespread usage. It has seven sub-components; two of them are optional components. According to the experiences of domain experts and analysis of legacy projects, the size of this component is usually around 8000-12000 lines of code (LOC). Currently, for each project, the code of this chosen component is completely hand-written.

Hypotheses and Metrics: The three hypotheses defined for research level validation should be tested:

- *H1: Evaluate feasibility:* The approach should be applied in an industrial-grade context and setting. The time, measured by man-hours of using the approach, should be affordable and reasonable especially during family engineering.
- *H2: Characterize the composition of multiple variability models:* Besides feature models, the MADE approach integrates two domain-specific models for handling processes and topology variability. The expectation of using the multi-variability modeling and realization is that more variability of the component can be modeled and expressed. The aspired outcome is to characterize to which extent the chosen component is influenced by the three variability types, by measuring the lines of code (LOC) of variability models of each variability types. In other words, the purpose to test H2 is to understand the influences of the three variability types to the chosen component.

- *H3: Evaluate the automation level of derivation:* The approach provides a semi-automated software derivation process, supporting model instantiation and code generation. The expectation of using the approach is to improve the efficiency of derivation. In particular, the LOC that can be generated by using the approach are the metrics. For this purpose, the H3 should be tested.

7.3.2. Execution Procedure

Foundation. This case study is based on the foundation of the proposed approach, which consists of the multi-variability modeling in Chapter 4, multi-variability realization in Chapter 5, and the developed derivation infrastructure presented in Chapter 6. The effort spent on developing the whole approach aims at a broader scope of IAM system, not specifically to the domain of warehouse management systems. Especially, the developed domain-specific models for process and topology variability have been validated in both warehouse management systems, and external automotive manufacturing in BMW with the participation of 7 domain experts. The development was distributed over a 15-month time frame. A reasonable estimation of the effort spent of these works was approximately 7-10 person months, which is not included in this case study.

This sub-section describes the procedure of conducting the case study on the chosen subject. Following the family and application engineering processes in SPLE, the case study procedure has also two phases.

7.3.2.1. Family Engineering Procedure

One researcher experienced in model-based development and one architect of the warehouse management systems from Siemens participated in the execution of this case study:

- *FE1. Establish the reusable product line architecture.* Several legacy projects of different customers have variants of the target component. There are subtle structural differences, even though the same team developed them. The developed reusable architecture is based on the most mature component variant among them.
- *FE2. Identify the types of components and locate the influential variability in code.* During the execution of this step, it is important to identify and establish the traces between variability in the (meta-)models and in the technical solution space. For example, the optional sub-components were associated with features. Within sub-components, the pieces of source code influenced by the different variability types were located. The corresponding templates of process models are created and added as parts of the core assets.

- *FE3. Extract and manage the reusable assets in repositories.* Classes without variability can be directly added to the core asset. For classes with variability, a re-factoring step was necessary to separate common parts from variable parts. Variable implementation parts were, in simple cases, included via feature-like variability realization (cf. Section 5.2 in Chapter 5). For process and topology variability, code templates were extracted, following the realization techniques of assembling, value traversing, type traversing and hybrid composition (cf. Section 5.3.2 in Chapter 5).

The development and extraction of reusable code assets at step FE3 applied several re-factoring patterns suggested in [LHMME11], such as hook methods, relocate classes, addition at the end of methods. The developed code templates are included as core assets for derivation.

7.3.2.2. Application Engineering Procedure

In application engineering, feature, process and topology models are used to configure the systems and derive the products, according to the proposed derivation process in Chapter 6. An ideal validation would be to compare the development time with and without the proposed approach with entirely new customer requirements. As this is not practically feasible, the case study instead replayed the derivation of previously executed projects based on their original requirements. Following the proposed approach, the three derivation activities are:

- *AE1.* Preparation: Feature configuration is the first step of the preparation. According to the configured features, the derivation infrastructure instantiated the process and topology models that need to be further configured.

- *AE2.* Configuration: The two participants involved in this case study played the roles of derivation stakeholders and performed the configuration tasks with the support of the developed process and topology editors.

- *AE3.* Generation: The derivation infrastructure took all the configured models to generate the application-specific architecture and source code. The generated code was validated with the applicable unit tests from the original projects (with adaptations, where necessary) and with code reviews.

7.3.3. Reporting the Case Study

By following the execution procedures in the previous section, the data of the case study were collected to gather evidence for characterization, evaluation and discussion. This section reports on the results of the case study.

7.3.3.1. Evaluate Feasibility

The time spent during family engineering for this particular case study was slightly less than 100 hours (M1). This includes the activities of (FE1) deciding the architecture, (FE2) identifying and locating the variability, and (FE3) extracting code templates (see Section 7.3.2). The involved software architect reported that implementing such a component usually took 2 to 3 person months per project. He confirmed that the case study actually shows the feasibility of applying the approach in a practical setting. Considering the benefits brought by domain-specific modeling and automated code generation, the approach has a high potential to be further applied to other complex components within this software family.

Hypothesis 1 (H1) aims at evaluating the feasibility of applying the approach. The execution of the case study has already shown that it is possible to apply the approach in an industrial grade setting. The effort of applying the approach, in terms of time spent on family engineering, is reasonable and affordable in the scope of this case study, based on the feedback from the domain architect. Using a single case study has its limitations of subjectivity and external validity threats. Further replication case studies can be important to improve the credibility of the results. Thus, we come up with the conclusion that H1 can be accepted with certain limitations.

Discussion: Several factors have turned out to be critical to the feasibility of such an SPLE approach:

- Maturity of reference architecture: The decision making for coming up with the reusable component's architecture was rather quick in our case study. The effort was around 1 day of work. The reason is that the existing reference architecture of the component was already stable and quite satisfactory for reuse. The level of maturity of the component was critical in this regard. Each software family, each component, or even each artifact within the same systems, may come in a different level of maturity [Bos02]. Improving the maturity of target systems or components may require significant effort, which would increase the re-factoring work, and in the worst case, makes the approach economically unfeasible.
- How mature is the domain: Bosch [Bos02] points out that not all systems need to or can improve their maturity levels. It depends on how variable the domain is, and how good the variability models can capture the variability in both requirements and implementation concisely. Warehouse management is a relatively mature domain of Siemens business. The engineers working for this product family have accumulated good domain and technical knowledge.
- The accessible resources during the development of family engineering: It turned out that the participation of the architect from the warehouse management systems was critical to the decision making and variability mining steps. Besides, the expertise of modeling and code generation is also very important. This can lead to another risk, because not all development teams have such expertise on a regular basis.

7.3.3.2. Characterize the composition of multiple variability types

The subsection presents the results of the evaluation of H2. Table 7.1 shows the identified reusable variability element types for the chosen complex component. For the chosen subject, 16 features were identified, including both architecture-relevant features that decide component inclusion and code-level features. For processes, 8 reusable process models with process templates were identified with a total of 31 reusable actions that can be instantiated. Among these actions, seven of them are location critical, which means that they require the bound-to-location relations to be configured. For topology, one topology model is needed for this chosen component that may comprise 8 different types of topology elements. Having these results, we can see that the two domain-specific models significantly enrich the modeling space of variability representation.

	Feature (M2)	**Process (M3)**		**Topology (M4)**		**Total**
Number	**16**	**Models**	**Actions**	**Models**	**Elements**	-
		8	**31**[1]	**1**	**8**	
Size	388 LOC	922 LOC		313 LOC		1632 LOC
Ratio	23.9%	56.8%		19.3%		100%

Table 7.1.: The Number and Size of Variability Elements

Among the seven sub-components, two of them are intensively process variable, and two of them are highly topological variable. The other three sub-components have more or less scattered types of variability. Some code parts comprise two or three influential variability types in a tangled way. The reason is that at the development time of the software family, the software architects and developers didn't take variability consideration into account for architectural decisions and source code implementation. Therefore, at re-factoring time of this case study, reducing the tangling of variability types for a piece of code makes extraction of code templates easier, while it increases the effort of re-factoring.

Hypothesis 2 (H2) aims at characterizing each variability types and their influences to the target component. The expectation of using process and topology variability is that more variability can be captured and expressed by models. With the results shown in Table 7.1, H2 can be accepted.

Discussion: The modeled topological elements in factories have different importance to software derivation. For example, the *Stationary Elements* often occur in several different *logical groups*, which means that they are observed by or interact with software components more often than *Transporter Elements*. The reason is that their events are more likely to trigger processes or other business logic. In contrast, *Transporter Elements* are usually tracked by the monitoring services. For IAM derivation, they have less effect on the business logic.

[1] Among these actions, seven of them are location critical, which means that they require the bound-to-location relations to be configured.

7.3.3.3. Evaluate the Automated Level of Derivation

Table 7.2 shows the types and size of the developed reusable asset base. The code-relevant assets in our context include Xpand templates for variable code and C# classes without variability. There are still four classes with very low reuse possibility, where only some stub methods could be used. Besides, for the derivation infrastructure, it was necessary to develop the required workflows and script files for model instantiation and code generation (see the automated derivation steps in Chapter 6), which serves as input of the derivation infrastructure supporting the automation.

Asset Types	Formats	LOC
Code with Variability	Xpand Code Template	5130
Directly Reusable Classes	C#	2399
Inputs to Infrastructure	MWE Work-Flow and Builds	513
	Total	8042

Table 7.2.: Reusable Asset Types and Sizes

Following the case study procedure for application engineering, the corresponding features are selected; The instantiated processes and topology models are configured; and finally code is derived, according to the known requirements of two legacy projects. Table 7.3 shows the configured features, the reused process actions, and the instantiated topological elements of the target component in Project 1 (P1) and Project 2 (P2).

	Configured Features (M5)	Process (M6)	Topology (M7)
		Reused Actions	Instances
P1	16	29	51
P2	12	24	67

Table 7.3.: The Results of Variability Modeling

Table 7.4 shows the results of the automatically derived source code of the two projects, compared to the original size of hand written source code. It is important to emphasize that the automatic generated LOC represent to certain extent the replacement of hand-written code, but they are not directly replaceable or interchangeable. The reason is that the reusable architecture of the chosen component was based on the reference architecture with re-factoring, whereas each existing variants of this component still have subtle differences. As can be seen from Table 7.4, the generated LOC are larger than the replacement LOC from the original code base. To analyze and understand the correctness and quality of the generated applications, the test cases are used, and a code review was performed to analyze and understand the correctness and quality of the generated applications. Based on the analysis, both of the two projects reached more than 60% code generation rate.

Hypothesis 3 (H3) aims at testing the effectiveness of code generation, to automate the derivation process for more than 35%. The results of code generation rate for

	Code Generation			**Comparison with Hand-Written Code**		
	Common LOC	Specific LOC	Sum (M8)	Replacement (M9)	Original LOC (M10)	Rate (M11)
P1	2399	5027	**7426**	**7224**	11530	**62.7%**
P2		4194	**6593**	**6149**	9869	**61.5%**

Table 7.4.: The Results and Analysis of Code Generation

both P1 and P2 are based on the comparison of their original hand-written source of the chosen subject of the case study. It is reasonable to conclude that H3 can be accepted.

Discussion: Influence of multi-variability on code generation. The results of automatic code generation show that the components with topological variability can reach nearly full-generation, whereas the process variable components cannot reach the same automation level. The feedback from the involved architect reveals that this situation is actually aligned with his expectation. The realization of topological variability was based on type or value traversing, and these are well supported in the template language. The realization of processes requires the assembly of reusable pieces of code in assets. Thus, for process variable artifacts, he expected to get more “glue-code” generated, especially when specified actions in processes do not yet have reusable implementation code in the core assets. In these cases the developers have to implement the business logic manually.

7.3.4. Implications and Lessons Learned

Applying the approach enforces more explicit and more reusable architecture. The finalized reusable architecture has differences with the documented reference architecture of this chosen component. The documented reference architecture had not enough details to facilitate code generation. In particular, the re-factoring enforces to collect variable code at certain places, e.g. in a code template, which “highlights” the non-variable code in the architecture. Because of this, It was possible to find the directly reusable classes shown in Table 7.2, which were previous hand-written, although they were actually already reusable. It is important to emphasize that the architecture developed for reuse is not the optimized solution for reducing code complexity. The code generation templates in core assets have the goal of covering a significant amount of variants, so that to a certain extent the understandability of the software architecture and code need to be sacrificed to gain generatability.

Using the domain-specific models improves the willingness of to provide high-quality model specifications. The domain-specific models enable fine-grained associations between the requirements in problem space and the technical implementation in solution space. This becomes a motivation for “modelers” during manual configuration time to specify processes and topology with more effort and attention. With such an approach the outcomes of modeling have an impact on the final systems. This is a significant improvement over the current state of practice, where requirement engineers specify the processes only informally and hand it over to hardware-oriented engineers and software engineers. The processes are only considered as input to be understood. Compared to this scenario, the approach presented in this thesis actually encourages the stakeholders to produce better quality of models.

The highest possible rate of code generation should not be the goal. The initial goal of conducting this case study was to strictly separate all generatable code from the hand-written code, to optimize the code generation rate. During re-factoring, it turned out that there are significant trade-offs to make. While there are several known patterns, such as hook methods, override mechanisms, or partial classes, they commonly only pay off for code blocks of significant size. Therefore, we only pursued to achieve reuse to a certain extent, avoiding too fine-grained reuse and to avoid increasing the complexity of the architecture.

Strict separation of variability impacts is difficult. The 5130 LOC of code templates (see Table 7.2) were investigated with variability, to try to further differentiate the impacts of different variability types to these code templates. However, the different variability models actually cover variability cross different abstraction levels. For instance, an abstract feature may decide an inclusion of a whole component, whereas a concrete features decides a loop time, which influences a single line. The template-based code generation for realizing process and topology variability increases the difficulty to analyze the impact, since a small piece of code templates with iterations can possibly bring a big impact to the generated code.

7.3.5. Threats to Validity

The case study was chosen as one part of the validation in this thesis, due to its suitability and applicability to understand the complex phenomena of product derivation in the industrial automation domain, motivated already in Section 7.2. This section discusses the threats to the credibility of the results of this case study, and what has been done to minimize the risks of threats to the credibility of the results.

7.3.5.1. Construct Validity

One of the threats to construct validity can be a poor construct definition to the case study. To alleviate this risk, at the planning phase of this case study, the objectives of the case study were clearly defined; the hypotheses, metrics used for validation and data collection methods were planned, as well as the execution procedure.

The researcher bias can have impact to lower construct validity. The execution of the case study involved one architect from the chosen software family and the author of this thesis. It is very difficult to completely avoid this threat, since introducing the approach into real industrial settings requires both domain knowledge and the approach knowledge. Due to the domain complexity and expertise required by the approach, it is recommended that as future work, further replications of case studies should be executed by external researchers or assistants to alleviate the risks of this threat.

7.3.5.2. Internal Validity

The subject of this case study comes from a real industrial software family of the warehouse management domain. When acquiring the candidate industrial projects, we actually encountered difficulties to get abundant resources, especially source code. Although the subject was carefully chosen to be representative to the characteristics of industrial automation domain, it is not a random sample, which may cause threats to internal validity.

7.3.5.3. Conclusion Validity

Low reliability of measures is a typical threat to conclusion validity. The essential idea of using a software product line approach is to facilitate software reuse, shorten time-to-market, and improve software quality. As the proposed derivation approach in this thesis is a product line approach, the ideal measurements would be the reuse rate, saved time and quality, by applying the approach with new customer requirements and implementation of development teams in practical settings. However, to fully validate all these aspects is not feasible to achieve, since only the requirement-engineering phase of IAM projects may last up to several months for a whole requirement engineering team. The case study reported in this thesis was based on scientific guidelines with close supervision of experienced researchers. The resources used in this case study are the original requirements, architectural documents and source code from legacy projects, which can be reflected to a “slice of life” example defined by Shaw [Sha03].

7.3.5.4. External Validity

A basic threat to external validity is generalization [Max92], whereas Flyvbjerg pointed out that a critical case could be defined as having strategic importance in relation to the general problem [Fly06]. During the planning phase of this case study, choosing the subject was carefully decided involving domain experts, to ensure that the case is representative to the domain problems. Meanwhile, the size and complexity of the case study should be within the time constraint and within the manageable scope of this thesis.

To understand the generalizability of this approach, we planned and executed the second part of validation (cf. the next Section 7.4) targeting the validation of the approach outside Siemens and other IAM systems. We on purpose involved stakeholders who are experienced across different sub-domains of IAM systems. By doing so, generalizability risks of the approach can be better understood by collecting the stakeholders perceptions.

7.4. Semi-Structured Interviews

The goal of conducting the semi-structured interviews is to evaluate the stakeholders' perception regarding the derivation process as already described in Section 7.1.2. The expected benefits of applying such an approach are to ease derivation stakeholder tasks, and improve their working efficiency during derivation. This section presents the planning, the execution and the results of the semi-structured interviews.

7.4.1. Planning

The objective of the interviews is to collect the stakeholders' perceptions regarding the proposed derivation approach and their expectation for its further improvement, so that the H4, the users' acceptance of the approach, can be tested.

According to the GQM model proposed in Section 7.1.2, six metrics should be tested. To plan the interviews, the author of this thesis had several discussions with experienced researchers in the area of technology acceptance, since there are existing methods with questionnaires to be used for testing the users' acceptance of a certain technology [Chu09]. However, the feedback from the researchers suggested that the existing questionnaires (with defined ordinal scales for ranking) are designed for getting statistical data. With such questionnaires, it would not be possible to receive valuable comments in details. To evaluate the approach presented in this thesis, it is more suitable to collect narrative data by open questions. During interviews, narrative data can bring more information. With open questions, it is possible to ask follow up questions regarding the deeper rationales of validation results and judgment by the interviewees. Therefore, based on several of existing studies both from the SPLE community [ORRT09, RGL12, RGD10] and the technology acceptance community [Chu09, Dav89], the definitions of these metrics are given as follows:

- *Understandability (M12):* The approach is comprehensible and the users can easily understand it.
- *Learnability (M13):* The user can quickly learn and utilize the approach.
- *User Satisfaction (M14):* Using the approach is pleasant and satisfying.
- *Efficiency (M15):* Using the approach improves the productivity and performance.
- *Flexibility (M16):* The approach is flexible, and does not restrict users' activities.
- *Correctness (M17):* Using the approach helps on producing the outcome correctly and properly.

7.4.2. Execution Procedure

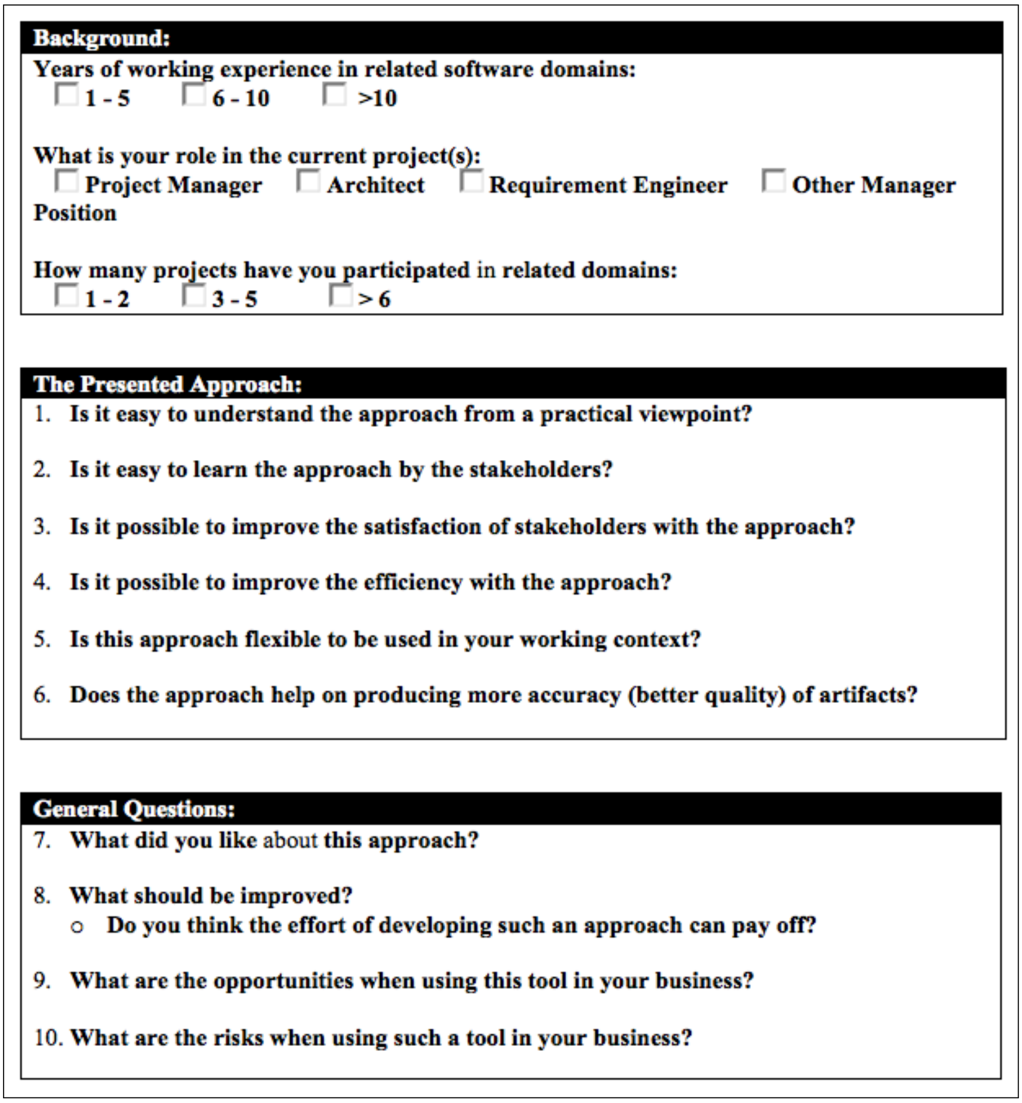
The execution procedure follows the research process recommended by Runeson and Höst [RH09]. This section presents the steps of execution and the next section reports the results.

- *Preparation:* Figure 7.7 shows the planned questions in three groups. These questions were planned in several iterations and refinements under the guidance of experienced researchers. The first part of questions includes fixed questions about the interviewees' background and experiences in the industrial automation domain, followed by open-ended questions targeting the six quality attributes as the second part. The third part of the questions includes also open-ended questions aiming at understanding the opportunities and risks of the derivation approach in practical settings. A presentation and a tool demo of the approach are prepared to present the approach.
- *Data collection:* The semi-structured interviews were conducted individually. The 10 chosen interviewees cover different roles in their current projects to receive opinions from different perspectives. The answers to the questions were noted down during the interviews as raw data.
- *Data Analysis:* The noted data were transcribed into tables for examination and analysis purposes. To ensure correct data interpretation, we also sent the transcribed data to a second researcher who also participated in the interviews to ensure the correctness of data interpretation.

7.4.3. Reporting the Semi-Structured Interviews

The semi-structured interviews involved the participation of 10 domain experts who perform different roles in their teams. Some of them have worked in their domains for many years, and also have performed other roles during their careers. The 10 interviews took in between 50 to 120 minutes, including 15-20 minutes at the beginning to explain the purpose of this study, to present the proposed approach and to demonstrate the tooling. Table 7.5 summarizes the roles, working years, and experiences of the interviewees. In the following paragraphs, we report the results regarding each of the six quality attributes.

Understandability (M12): One of the architects from Siemens reported that the graphical notations are easier to understand compared to textual process specifications. The graphical notations provide a better overview of the functionality of the processes, which is closer to the material flows in the industrial automation domain. Other standard notations (e.g. BPMN, UML) do not support material flows. Tailoring is important to the domain; Otherwise the notations are not understandable. Additionally, he said that the approach integrates also the topology models, but the topology models, from his viewpoint, work closer to the technical space compared to process models. Topology models can be used to express the linkages of IAM systems to hardware



Background:

Years of working experience in related software domains:
□ **1 - 5** □ **6 - 10** □ **>10**

What is your role in the current project(s):
□ **Project Manager** □ **Architect** □ **Requirement Engineer** □ **Other Manager Position**

How many projects have you participated in related domains:
□ **1 - 2** □ **3 - 5** □ **> 6**

The Presented Approach:

1. **Is it easy to understand the approach from a practical viewpoint?**
2. **Is it easy to learn the approach by the stakeholders?**
3. **Is it possible to improve the satisfaction of stakeholders with the approach?**
4. **Is it possible to improve the efficiency with the approach?**
5. **Is this approach flexible to be used in your working context?**
6. **Does the approach help on producing more accuracy (better quality) of artifacts?**

General Questions:

7. **What did you like about this approach?**
8. **What should be improved?**
   - **Do you think the effort of developing such an approach can pay off?**
9. **What are the opportunities when using this tool in your business?**
10. **What are the risks when using such a tool in your business?**

Figure 7.7.: The Questions Prepared for the Semi-Structured Interviews

distribution. The steering manager reported both of the pros and cons of using such an approach in the target domain. On the one hand, to standardize the notations of IAM processes is very helpful to improve understandability. He mentioned that the development teams and requirement teams have different goals. The current practice has very heave communication overhead. Having the customized notations can improve on the communication efficiency. On the other hand, he also pointed out that developing a generic standardization of modeling notations in all IAM domains is challenging. Each sub-domain may require small adaptation to ensure that the stakeholders can understand the modeling notations well. Highly automated IAM sub-domains are easier to standardize (e.g. metal or machining), compared to the

ID	Current Role	Location	Company	Experiences	
				Years	Projects
1	Requirement Engineer	Shenyang, China	BMW	1-5	3-5
2	Architect	Erlangen, Germany	Siemens	6-10	3-5
3	Maintenance Engineer	Shenyang, China	BMW	9	>6
4	Solution Manager	Shenyang, China	BMW	>10	>6
5	Steering Manager	Shenyang, China	BMW	>20	>20
6	Requirement Engineer	Shenyang, China	BMW	6-10	>6
7	Requirement Manager	Shenyang, China	BMW	6-10	>6
8	Architect	Munich, Germany	Siemens	>10	1-2
9	Architect	Munich, Germany	Siemens	6-10	3-5
10	Architect	Erlangen, Germany	Siemens	>10	>6

Table 7.5.: Interviewees and their background

domains with many manual manufacturing processes (e.g. logistics). For the same domain, among different countries or cultures, the automation levels of production can be highly variable, which makes the development of modeling notations challenging.

Learnability (M13): Two different learnability aspects need to be considered in our proposed derivation approach. The first aspect is the learnability of the domain-specific models for the topology and process configuration. All interviewees confirmed that the developed two domain-specific variability models are easy to learn compared to their currently used, non-tailored modeling tools. Further investigations regarding the quality of the domain-specific models should be conducted, such as the completeness of meta-elements. The second aspect is the learnability of the derivation approach to the users. Our interviewees reported that the conveyed ideas from the proposed approach is convincing and promising. However, many other factors may also impede learnability, such as tool maturity.

User Satisfaction (M14): All interviewees reported that applying the approach would improve their satisfaction at work. One architect reported that the derivation infrastructure creates the SPL architecture in a systematic way, which can improve the confidence of development work. The automatic code generator can be used at the starting phase of development to derive the architecture, which makes his job easier. One of the requirement engineers was concerned more with the modeling part, especially the process editor. It was obvious that for her the process editor was considered as a modeling tool to standardize her process descriptions; the effectiveness of the code generator was not in her main focus. She said that having a domain-specific terminology and standardized notations may significantly ease the communication between the requirement team and the development team, so that the satisfaction of both teams can be improved. The suggested improvement is to support the synchronization between configured models and derived code. With this function, the development team may react to changes faster by only manipulating models.

Efficiency (M15): Generally, all interviewees reported that the approach can significantly improve their work efficiency, when it is applied in their settings. By using it, the number of negotiation iterations among customers, requirements, and development can be potentially reduced. However, it is not possible to fully avoid any of the previous derivation stages (cf. the staged development process of IAM systems in Section 2.2.2 of Chapter 2), reported by the solution manager. He further said, a configuration tool should focus on helping users to improve their performance in each derivation stage, and it is not possible to replace any derivation steps with fully automated tools.

Flexibility (M16): On the one hand, the flexibility of the derivation tool should be limited, reported by one of the architects, the maintenance engineer, and the solution manager. The maintenance engineer said that according to her experiences, products of one product family in the target domain share 70%-80% functional requirements. For her, the derivation tool should focus on maximizing reuse of the shared requirements. In this case, the configuration activities should be under certain restriction and control, for example to limit available elements or to hide invalid elements to users. On the other hand, the steering manager and a requirement engineer reported that the flexibility should be also balanced, so that the configuration and specification of processes and topology are not totally restricted. For instance, when facing changes or reacting to application-specific requirements, the expectation of flexibility to the derivation infrastructure becomes higher, so that the users get certain "buffers" to specify new requirements at modeling time.

Correctness (M17): The correctness in our context refers to the quality of configured feature, topology, and process models. Incorrect configuration may lead to invalid or incomplete code generation. The solution manger and the maintenance engineer reported correctness as not important, whereas several other interviewees reported it as critical. The solution manger said that, when such a tool is used during pre-sales phase for presentation to customers or negotiation, the correctness is not important. The maintenance engineer pointed out that the correctness checking should be transparent, so that the users do not directly see it. She said that developers should be able to compensate incorrect configurations at later development stages. The derivation infrastructure should be able to tolerance users' misbehavior when it becomes more mature.

Three independent researchers conducted the evaluation of the narrative data collected in the semi-structured interviews individually. The symbol "+" represents positive evaluation; the "–" represents negative evaluation opinions. Correspondingly, the "++" shows a very positive opinion, and "– –" shows a very negative opinion. As interviewees often mentioned both positive and negative perceptions, the "+/–" represents positive answers with potential negative concerns, and vice versa. Table 7.6 summarizes the validation results, which represents the average of the evaluation from the three researchers who reviewed the narrative data .

The results show especially positive perceptions and feedback regarding understandability, learnability, users' satisfaction, and efficiency. Considering the difficulties of

ID	Understandability (M12)	Learnability (M13)	Satisfaction (M14)	Efficiency (M15)	Flexibility (M16)	Correctness (M17)
1	+	+	+	+	+/−	−
2	+	−/+	++	++	+/−	−
3	+	+	+	+/−	+/−	+/−
4	+	+	+/−	+/−	−	+/−
5	+	NA	+	−/+	+/−	NA
6	+/−	+	+	+/−	NA	−
7	−	−	NA	+/−	NA	NA
8	−	+/−	+	−	+	+
9	+	+/−	++	++	+	−
10	+/−	+/−	+/−	+/−	−	+/-
Sum	+	+	+	+	+/−	−

+: Positive −: Negative NA: Not Answered

Table 7.6.: The Summary of the Interview Results

approach adoption in practical settings, the learnability of the approach is considered to be challenging, which depends on the maturity of tooling. The feedback regarding flexibility is actually also interrelated with the tool support, but some interviewees mentioned the opportunity to flexibly use the approach with minor adaption. Further discussions will be in the next section. To achieve better accuracy of modeling and code, algorithms or mechanisms should support correctness checking during configuration time. Currently, such functions are not included in the approach.

7.4.4. Implication and Lessons Learned

The third part of the planned questions includes open questions about general aspects that were not bound to any specific quality attributes. For example, other opportunities or risks were explicitly asked during the interviewees.

7.4.4.1. Opportunities

Utilization in the pre-sales phase: An architect and the solution manager both have experiences in the pre-sales phase of projects in the target domain. At this phase, customers usually cannot explicitly express and describe precisely what they want to have. Furthermore, stakeholders of a manufacturing factory usually have highly varying backgrounds and knowledge, such as machining, business, software, or even chemical engineering. The communication among them is difficult. Meanwhile, the budget, the time line, and the resources of the whole project life cycle must be pre-planned and decided. These factors make the pre-sales phase very challenging. Both of these two interviewees reported that the proposed derivation infrastructure may support the creation of presentable software systems to customers, even before payment, instead of letting them wait for months or years to see the final delivery. The positive impact of using the derivation infrastructure during pre-sales phase may be on communicating requirements and estimating the development complexity. To utilize the approach in this proposed way was quite surprising and not planned. To accomplish this goal, the derivation infrastructure should be more presentable and attractive, especially for non-technical users.

Model level reuse with templates: Almost all interviewees confirmed the importance of the proposed *Model Instance Generator*, which allows the reuse of model templates. For the interviewees, it seems that model templates are considered as important as code templates in assets, especially for those who work in the problem space. We analyze the reason to be that for large and complex software systems, it is more realistic to think and adopt the first four derivation steps of the proposed approach as short-term goals (cf. Section 6.1 in Chapter 6), without the *Derivation Activity 3: Generation*. In the target domain, it is common to have several hundreds of processes to be specified. Having them automatically instantiated with model templates would significantly improve modeling productivity.

Generalizability: The chosen interviewees have very rich experiences in software development and some of them have experiences outside the IAM domain. During the interviews, the interviewees reported that the approach has very high potential to be applied in other related domains, not only in IAM sub-domains, but also other domains such as power plant automation in the area of distributed control systems, trains and rail management systems, where similar topological variability and process-like variability exist.

7.4.4.2. Risks

Tool maturity: A general remark mentioned by all interviewees was the importance of tool maturity. For users, the derivation tool should be more interactive and powerful than the currently developed prototype, for example on providing configuration guidance. For automated derivation, the generated code must have good quality and should be able to save developer's effort, so that the total development effort can be reduced.

Impact to stakeholders' way of working: Some of the opinions among interviewees are contradicting. The reason is probably that in our approach, domain-specific modeling for topology and processes is added to bridge the gap between the problem space and the solution space. Experienced requirement engineers may be able to use the process editor to model code-level behaviors, which generates source code. In this case, the boundary between the requirement team and the development team becomes unclear. The changes to stakeholders' responsibilities by adopting the approach are not yet clear and explicit. In addition, learning and adopting new approaches is always challenging, because some team members may be reluctant to change. The adoption requires careful plans and continuous training. The decision is definitely not easy and cheap. It is necessary to get support and commitment from higher management.

Higher modularity required by process-intensive artifacts: The interviewees who have development background mentioned the necessity of modularity of process-intensive code, which was also a challenge encountered during re-engineering. To improve the percentage of code generation, more re-factoring work needs to be done, for example to enable the generation of components comprising both process and topology variability in a very intermingled way. To the best of the author's knowledge, there are no existing methods or patterns directly targeting this purpose. It seems that there is no state of the art about how to re-factor existing code, and develop proper generators, when multiple variability types need to be taken into account in parallel. The state-of-the-art re-factoring techniques were applied to the best of the author's experiences. It is necessary to further investigate and to develop guidelines about how to migrate existing code, and how to design a system from scratch with architectural patterns for code generation purposes.

7.4.5. Threats to Validity

During the planning, execution, data collecting, and reporting phases, several aspects were considered to mitigate the validity threats, and to ensure the credibility of the results of the interviews.

7.4.5.1. Construct Validity

Mono-operation bias is one of the threats to construct validity. To minimize it, a pre-study was conducted involving experienced researchers in software product line engineering, in order to optimize the design of questionnaire being used in the semi-structured interviews. Two iterations were performed to ensure the quality of the questionnaire.

Hypothesis guessing is potentially another validity threat to construct validity. The interviewees may have the potential to know the expected improvements of the proposed approach. To minimize this validity threat, the potential expectation and benefits of applying the approach were not disclosed to the candidate interviewees, when contacting them, however, this might not be able to fully avoid the threats to the construct validity.

7.4.5.2. Internal Validity

When creating the questionnaire and conducting the semi-structured interview, the author of this thesis as one of the involved researchers may introduce bias to the questions. During the interviews, it might happen that the expected data of the researchers are gathered, and information that should be collected from the interviews is neglected. To reduce the effects of personal bias, we used "member-checking" [Cre03] to involve a second observer during the whole process of planning and execution of interviews, which helped us to determine whether the data we collected, presented the accurate and correct feedback from industry.

7.4.5.3. Conclusion Validity

A typical threat to conclusion validity is low statistical power. In the context of this study, the proposed approach is meant to apply to the industrial automation management or related domains. Having practical experiences was the most important prerequisite of looking for candidate interviewees. As presented in Section 7.4.3 (Table 7.5), the interviews involved very experienced experts, and some of them have more than 10 years experiences in various sub-domains of IAM systems. Still it was challenging to find appropriate interviewees. As part of future work, more replications of such interviews should try to involve domain experts as many as possible.

Another possible threat to conclusion validity could be the heterogeneity of interviewees. The invited interviewees of this study are all experienced engineers in industry. We on purpose have chosen the interviewees, who are working on or had worked on the jobs of the three stakeholder roles: requirement engineers, hardware-oriented engineers, or software developers, especially the interviewees in BMW, Shenyang, China (see Table 7.5). Their evaluation and feedback regarding the approach is based on practical and realistic viewpoints. The observation shows that stakeholders working in problem space and solution space have different expectations, when seeing the proposed model-based approach. The heterogeneity of stakeholders' perspectives is actually a risk of full adoption of the approach in practice.

7.4.5.4. External Validity

Generalizability is the main threat to external validity in this study. Triangulation [Gui02] is a method used by qualitative researchers to check and establish validity in their studies by analyzing a research question from multiple perspectives. To minimize this validity threat, both data and environmental triangulation were considered in the design of the study.

- For data triangulation, we tried to diverse the background of the interviewed stakeholders with respect to their current roles in their projects. As can be seen from the Table 7.5, the interviewed stakeholders include requirements engineers working in the problem space, architects working in technical space, a maintenance engineer, and managers covering different hierarchical levels.

- For environmental triangulation, although with relatively low statistical power, we could involve interviewees from three different sites in two countries. The interviews have also experiences in different sub-domains of IAM systems. Thus, it was possible to reduce the external validity threats.

7.5. Summary

This chapter started with the practical and research goals for validating the intended benefits brought by the proposed approach. Following the GQM paradigm, the measurement models at the two levels were created, and the corresponding hypotheses are defined. Afterwards, the research methodologies to validate the hypotheses were motivated. The validation has two parts: a case study based on a "slice-of-life" example of an industrial IAM system and semi-structured interviews involving domain experts from practical viewpoints.

The results of the case study have shown that integrating process and topology variability into the derivation process can significantly enhance the variability modeling and configuration. More variability in the core assets can be expressed. This leads to a higher automation level of software product derivation. Furthermore, the time spent on enabling the approach for the case study could show the feasibility of applying the approach in a practical and realistic setting.

The semi-structured interviews were planned and designed to understand the perceptions of stakeholders from the IAM domain, because they are the potential users of the proposed derivation approach. The results show that the most important improvement is on user satisfaction and derivation efficiency perceived by the interviewees. The process and topology modeling improves the variability representation, and the overall derivation approach could show good understandability, reported by the involved interviewees. However, learning and adopting new approaches in practical settings is always challenging, which requires the maturity of tooling and the readiness throughout different organizational levels. To support full adoption of the approach, additional mechanisms for consistency checking during configuration should be integrated into the approach.

Table 7.7 summarizes the results of testing each of the hypotheses. The feasibility of the approach (H1) was demonstrated in the setting of this case study. Further replications of such studies are necessary to fully test the feasibility of the approach in a broader scope. H2 can be accepted, as the process and topology variability models significantly enrich the variability modeling space. With the configuration of these two variability types, a higher automation level of derivation can be achieved. The code generation analysis shows that using the approach can generate more than 60% of the source code of the chosen complex component in the case study, so that H3 can be accepted. For testing the perceived users' acceptance (H4), the semi-structured interviews were chosen as the validation methodology. A questionnaire with open questions with regard to usefulness, ease-of-use, opportunities and risks is prepared for the interviews. H4 is considered to be accepted with certain limitations, based on the inductive analysis of the narrative data collected during the interviews. An ideal setting to test H5 would be an industrial experiment with two development groups. One group uses the proposed approach and one control group develops the software without the proposed approach. The comparison should be based on the time difference of application engineering of the two groups. Due to the limited resources, H5 could not be tested within the scope of this thesis.

Hypotheses	Results
H1: Feasibility	Accepted with limitation
H2: Characterization of multiple variability models	Accepted
H3: Automation level of derivation	Accepted
H4: Users' perceived usefulness and ease of use	Accepted with limitation
H5: Time saving in application engineering	Not tested

Table 7.7.: A Summary of Validation Results

To mitigate the risks of validity threats during the planning, execution, data collecting, and reporting phases of the case study and semi-structured interviews, certain techniques were applied, such as data triangulation and member checking, as described in the planning and execution phases of the two research methodologies, for example in Section 7.4.2. However, the author could not fully avoid all the validity threats. Figure 7.8 shows a summary of these validity threats.

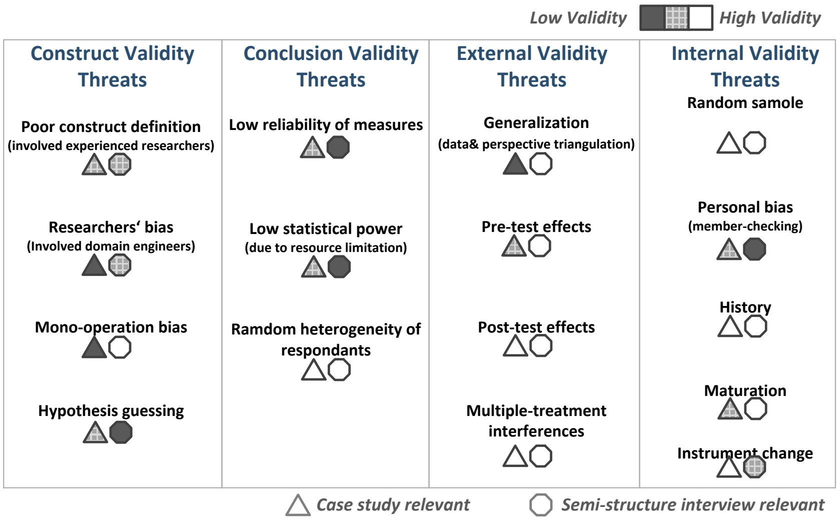

Low Validity
High Validity
Construct Validity Threats
Poor construct definition
(involved experienced researchers)
Researchers' bias
(Involved domain engineers)
Mono-operation bias
Hypothesis guessing
Conclusion Validity Threats
Low reliability of measures
Low statistical power
(due to resource limitation)
Ramdom heterogeneity of respondants
External Validity Threats
Generalization
(data& perspective triangulation)
Pre-test effects
Post-test effects
Multiple-treatment interferences
Internal Validity Threats
Random samole
Personal bias
(member-checking)
History
Maturation
Instrument change
Case study relevant
Semi-structure interview relevant

Figure 7.8.: Validity Threats

With the validation results, RQ4 can be answered. The approach shows significant improvement on the software product derivation process for systems in industrial automation management, where different variability types are concerned in different derivation stages. The analysis of the results also shows the potential risks, limitation, and future work for full adoption of the approach in practice.

8 Conclusions and Further Research Directions

"Success is not final, failure is not fatal:"

it is the courage to continue that counts."

–Winston S. Churchill

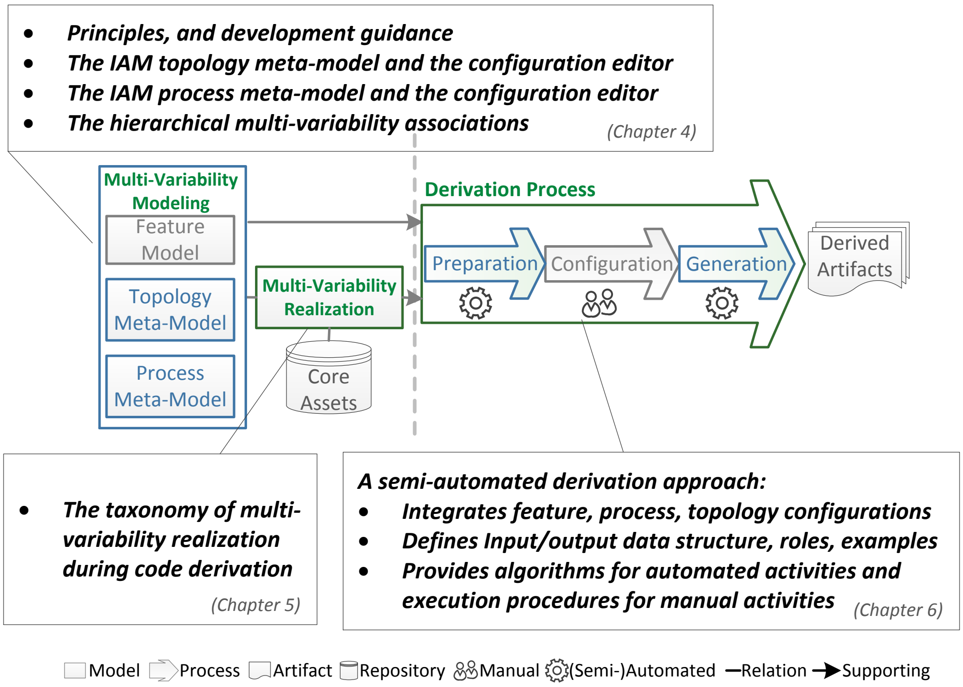
This thesis has presented the MADE approach, which integrates the configuration of feature, process and topology variability into a semi-automated derivation process for IAM systems. This chapter provides the summary of the contributions of this thesis. It also provides the outline of future research directions based on the lessons learned from the work this thesis.

8.1. Summary of Contributions

The original motivation of conducting the research in this thesis came from the practical problems during the software development and derivation of industrial automation management systems. Besides feature variability, process and topology variability are the two main characteristics of product families in the target domain. They lead to additional complexity when attempting to use general-purpose variability modeling techniques to establish a software produce line. Furthermore, requirement engineers, hardware-oriented engineers, and software developers need to participate in the development process, in which these stakeholders have particular concerns. Therefore, a systematic derivation approach is desired to have the capabilities to support modeling of feature, process and topology variability; automating the derivation in terms of model-level reuse, the staged derivation and the integration of the multi-variability configuration.

This thesis has included a systematic literature review aiming at collecting the latest advances of derivation approaches in the state of the art. Based on the analysis of the prior approaches, this thesis has reported the limitations as: lack of modeling support of the two domain-specific variability types, lack of support on model-level reuse, and lack of support on integrating the configuration of multiple variability during derivation.

Approach. As illustrated in Figure 8.1, this thesis has presented an approach, which consists of three major parts:



- Principles, and development guidance
- The IAM topology meta-model and the configuration editor
- The IAM process meta-model and the configuration editor
- The hierarchical multi-variability associations
(Chapter 4)
Multi-Variability Modeling
Feature Model
Topology Meta-Model
Process Meta-Model
Multi-Variability Realization
Core Assets
Derivation Process
Preparation
Configuration
Generation
Derived Artifacts
- The taxonomy of multi-variability realization during code derivation
(Chapter 5)
A semi-automated derivation approach:
- Integrates feature, process, topology configurations
- Defines Input/output data structure, roles, examples
- Provides algorithms for automated activities and execution procedures for manual activities
(Chapter 6)
Model
Process
Artifact
Repository
Manual
(Semi-)Automated
Relation
Supporting

Figure 8.1.: Summary of Main Contributions of the Thesis

Multi-variability modeling. For feature, topology and process variability, the principles and development guidance are presented at the first place, with which the meta-models of modeling topology and processes in IAM systems are proposed. Furthermore, these multiple variability types are hierarchically interrelated for representing the variability in the core assets, in particular the components in the reusable software product line architecture.

Multi-variability realization. To implement the process and topology variability, this thesis presents an extension of variability realization taxonomy. The taxonomy consists of template-based derivation techniques, including parameterization, assembly, value and type traversing.

The semi-automated derivation process. As presented in Figure 8.1, the proposed MADE approach consists of three main derivation activities. (1) The *Preparation Activity* is a semi-automated activity. By configuration of features, the derivation infrastructure instantiates the corresponding process and topology models with model templates or model fragments. (2) The *Configuration Activity* is a manual activity supported by editors to allow derivation stakeholders to configure the process, topology models and their relations. (3) The *Generation Activity* is an automated activity. The first sub-activity uses feature configuration as input and decides the inclusion of code generation templates. The second sub-activity finally uses the included code templates to derive the application-specific code. The MADE approach defines also the input and output data of these activities. For automated activities, the approach provides pseudo-algorithms. For manual activities, the approach provides guidelines to support configuration procedures.

Validation. At the research validation level, the thesis has presented a case study to apply the proposed approach in an industrial-grade complex component from the target domain. The case study demonstrates the feasibility of the approach. The results of the case study show also that using the process and topology models significantly improves the expression and representation of variability. Within the settings of this case study, the automated generated code could replace more than 60% of the hand-written code in the chosen complex component. As the practical validation, the thesis has presented semi-structure interviews with experienced engineers to evaluate the users' perceived efficiency, satisfaction, learnability, understandability, flexibility and correctness. The results show high potential on improving efficiency and user satisfaction, whereas the flexibility and correctness may still rely on a better maturity of tooling.

8.2. Open Issues and Further Research Directions

Re-engineering with automation support during family engineering. Developing the reusable software product line architecture and core assets is non-trivial. Up to now, only limited numbers of state-of-the-art approaches address this issue. For example, Lopez et al. propose feature-oriented refactoring patterns [LHMME11]. As one direction of future work, manual procedures of re-engineering and refactoring can be potentially supported by automated tooling, for example, to combine with information retrieval techniques [NE08, ASBZ16], or variability-aware code refactoring [LJG$^+$15].

Support of manual configuration tasks. Improving the manual configuration support, in particular the second derivation activity "Configuration", is not the main goal of this thesis. As already pointed out in Chapter 2, there is a space to further improve the configuration efficiency with existing techniques. There are approaches aiming at optimizing configuration, such as automating selection [GWW+11], or prioritizing variability to be configured [CE11]. Adding mechanisms to perform error detection, to diagnose constraint violations or to ensure consistency can improve the confidence of configured multiple variability models [TBG13, AHH11]. Integrating such functions into the derivation tooling can be a part of future work of this thesis, especially when applying the approach in practice.

Extension of the approach to early derivation phases. During the semi-structured interviews, the interviewees have pointed out a potential usage of the approach, which is to use it in the pre-sales phase. The potential benefits are tow-fold. On the one hand, during the pre-sales phase, domain-specific models can ease the communication among different stakeholders. The customers of a manufacturing factory have completely different backgrounds. The tailored and customized modeling languages enhance the expression of the requirements. On the other hand, such an approach may help the development team to better estimate the complexity of customer-specific development, in terms of effort and resources of the whole project life cycle. It helps the derivation stakeholders to foresee what can be reused and what must be newly developed already during the negotiation with customers. To fulfill this goal, the derivation infrastructure should be more presentable and attractive, especially for non-technical users. With reasonable effort, this is possible to achieve as future work.

References

[ABM00] Colin Atkinson, Joachim Bayer, and Dirk Muthig. Component-based product line development: the kobra approach. In *Software Product Lines Conference (SPLC)*, pages 289–309. Springer, Denver, USA, 2000.
(Cited on page 41.)

[AC06] Sangim Ahn and Kiwon Chong. A feature-oriented requirements tracing method: A study of cost-benefit analysis. In *Hybrid Information Technology*, volume 2, pages 611–616, Cheju Island, Korea, November 2006. IEEE.
(Cited on page 58.)

[ACG$^+$12] Mathieu Acher, Philippe Collet, Alban Gaignard, Philippe Lahire, Johan Montagnat, and Robert B France. Composing multiple variability artifacts to assemble coherent workflows. *Software Quality Journal*, 20(3-4):689–734, 2012.
(Cited on page 19.)

[ACLF13] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B France. Familiar: A domain-specific language for large scale management of feature models. *Science of Computer Programming*, 78(6):657–681, 2013.
(Cited on page 132.)

[Ada12] Sebastian Adam. *Incorporating Software Product Line Knowledge into Requirements Processes*. PhD thesis, Fraunhofer IRB, 2012.
(Cited on pages 13, 41, and 68.)

[ADEG09] Sebastian Adam, Joerg Doerr, Michael Eisenbarth, and Anne Gross. Using task-oriented requirements engineering in different domains–experiences with application in research and industry. In *International Requirements Engineering Conference (RE)*, pages 267–272, Atlanta, U.S.A., August-September 2009. IEEE.
(Cited on page 68.)

[AHH11] Ebrahim Khalil Abbasi, Arnaud Hubaux, and Patrick Heymans. A toolset for feature-based configuration workflows. In *Software Product Line Conference (SPLC)*, pages 65–69, Munich, Germany, August 2011. IEEE.
(Cited on pages 19, 119, and 156.)

[AKL09] Sven Apel, Christian Kastner, and Christian Lengauer. Featurehouse: Language-independent, automated software composition. In *International Conference on Software Engineering (ICSE)*, pages 221–231, Vancouver, Canada, May 2009. IEEE.
(Cited on page 39.)

[AS13] Sebastian Adam and Klaus Schmid. Effective requirements elicitation in product line application engineering–an experiment. In *International Working Conference on Requirements Engineering: Foundation for Software Quality*, pages 362–378. Springer, 2013.
(Cited on page 41.)

[ASBZ16] Chetan Arora, Mehrdad Sabetzadeh, Lionel Briand, and Frank Zimmer. Extracting domain models from natural-language requirements: approach and industrial evaluation. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, pages 250–260. ACM, 2016.
(Cited on page 155.)

[ASM04] Timo Asikainen, Timo Soininen, and Tomi Männistö. A koala-based approach for modelling and deploying configurable software product families. In *Software Product-Family Engineering*, pages 225–249. Springer, 2004.
(Cited on pages 44, 85, and 96.)

[BBM05] Kathrin Berg, Judith Bishop, and Dirk Muthig. Tracing software product line variability: from problem to solution space. In *Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 182–191. South African Institute for Computer Scientists and Information Technologists, 2005.
(Cited on page 81.)

[BCR94] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal questin metric approach, 1994.
(Cited on page 126.)

[BFK$^+$99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. Pulse: a methodology to develop software product lines. In *Symposium on Software Reusability*, pages 122–131, Los Angeles, USA, 1999. ACM.
(Cited on page 41.)

[BGB$^+$06] T John Brown, Rachel Gawley, Rabih Bashroush, Ivor Spence, Peter Kilpatrick, and Charles Gillan. Weaving behavior into feature models for embedded system families. In *Software Product Line Conference (SPLC)*, pages 52–61, Baltimore, USA, August 2006. IEEE.
(Cited on pages 46 and 77.)

[BGMW00] Joachim Bayer, Cristina Gacek, Dirk Muthig, and Tanya Widen. Pulse-i: Deriving instances from a product line infrastructure. In *Engineering of Computer Based Systems Workshop (ECBS)*, pages 237–245, Edinburgh, UK, 2000. IEEE.
(Cited on page 41.)

[BLR$^+$15] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. What is a feature?: a qualitative study of features in industrial software product lines. In *Software Product Line Conference (SPLC)*, pages 16–25, Nashville, USA, July 2015. ACM.
(Cited on page 12.)

[BMR$^+$96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. A system of patterns: Pattern-oriented software architecture. 1996.
(Cited on page 88.)

[Bos00] Jan Bosch. *Design and use of software architectures: adopting and evolving a product-line approach*. Pearson Education, 2000.
(Cited on page 12.)

[Bos02] Jan Bosch. Maturity and evolution in software product lines: Approaches, artefacts and organization. In *Software Product Lines Conference (SPLC)*, pages 257–271. Springer, San Diego, USA, August 2002.
(Cited on pages 1, 12, 13, 13, and 136.)

[Bro88] Frederick P Brooks. Grasping reality through illusion—interactive graphics serving science. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1–11. ACM, 1988.
(Cited on page 131.)

[BSØ$^+$14] Thorsten Berger, Ștefan Stănciulescu, Ommund Øgård, Øystein Haugen, Bo Larsen, and Andrzej Wąsowski. To connect or not to connect: experiences from modeling topological variability. In *Software Product Line Conference (SPLC)*, pages 330–339, Florence, Italy, September 2014. ACM.
(Cited on pages 22, 45, 62, 66, 68, and 76.)

[BSR04] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling step-wise refinement. *IEEE Transactions on Software Engineering*, 30(6):355–371, 2004.
(Cited on pages 20 and 39.)

[BWR11] Elizabeth Bjarnason, Krzysztof Wnuk, and Björn Regnell. Requirements are slipping through the gaps—a case study on causes & effects of communication gaps in large-scale software development. In *Requirements Engineering Conference (RE)*, pages 37–46, Trento, Italy, August-September 2011. IEEE.

(Cited on page 55.)

[BYBS13] Razieh Behjati, Tao Yue, Lionel Briand, and Bran Selic. Simpl: a product-line modeling methodology for families of integrated control systems. *Information and Software Technology*, 55(3):607–629, 2013.
(Cited on pages 44, 62, 76, and 121.)

[CABA09] Lianping Chen, Muhammad Ali Babar, and Nour Ali. Variability management in software product lines: a systematic review. In *Software Product Line Conference (SPLC)*, pages 81–90, San Francisco, USA, August 2009. Carnegie Mellon University.
(Cited on pages 2 and 14.)

[CBH11] Andreas Classen, Quentin Boucher, and Patrick Heymans. A text-based approach to feature modelling: Syntax and semantics of tvl. *Science of Computer Programming*, 76(12):1130–1143, 2011.
(Cited on page 54.)

[CE00] Krzysztof Czarnecki and Ulrich W Eisenecker. Generative programming. *Edited by G. Goos, J. Hartmanis, and J. van Leeuwen*, page 15, 2000.
(Cited on page 81.)

[CE11] Sheng Chen and Martin Erwig. Optimizing the product derivation process. In *Software Product Line Conference (SPLC)*, pages 35–44, Munich, Germany, August 2011. IEEE.
(Cited on pages 19 and 156.)

[CGR$^+$12] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. Cool features and tough decisions: a comparison of variability modeling approaches. In *workshop on variability modeling of software-intensive systems*, pages 173–182, Leipzig, Germany, January 2012. ACM.
(Cited on pages 2, 14, and 54.)

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration using feature models. In *Software Product Lines Conference (SPLC)*, pages 266–283. Springer, Boston, USA, Auguest-September 2004.
(Cited on pages 19, 24, 38, 40, 56, and 75.)

[Chu09] Mohammad Y Chuttur. Overview of the technology acceptance model: Origins, developments and future directions. *Working Papers on Information Systems*, 9(37):9–37, 2009.
(Cited on pages 129 and 142.)

[CKSW13] Paul Clements, Charles Krueger, James Shepherd, and Andrew Winkler. A ple-based auditing method for protecting restricted content in derived products. In *Software Product Line Conference (SPLC)*, pages 218–226, Tokyo, Japan, August 2013. ACM.
(Cited on pages 37 and 39.)

[CN02] Paul Clements and Linda Northrop. *Software product lines: practices and patterns*. Addison-Wesley, 2002.
(Cited on page 11.)

[CNCJ14] Jaime Chavarriaga, Carlos Noguera, Rubby Casallas, and Viviane Jonckers. Propagating decisions to detect and explain conflicts in a multi-step configuration process. In *Model-Driven Engineering Languages and Systems*, pages 337–352. Springer, Valencia, Spain, September-October 2014.
(Cited on pages 19 and 119.)

[CNKL12] Elder Cirilo, Ingrid Nunes, Uirá Kulesza, and Carlos Lucena. Automating the product derivation process of multi-agent systems product lines. *Journal of Systems and Software*, 85(2):258–276, 2012.
(Cited on page 45.)

[CNP$^+$08] Ciarán Cawley, Daren Nestor, André PreuBner, Goetz Botterweck, and Steffen Thiel. Interactive visualisation to support product configuration in software product lines. In *the Second International Workshop on Variability Modeling of Software-Intensive Systems*, Essen, Germany, May 2008. ICB-Research Report.
(Cited on page 41.)

[Cre03] J Creswell. Mixed methods procedures. *Research design: Qualitative, quantitative, and mixed methods approaches*, pages 208–227, 2003.
(Cited on page 149.)

[cvl12] Common variability language – revised submission, 2012.
(Cited on page 14.)

[Dav89] Fred D Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS quarterly*, pages 319–340, 1989.
(Cited on pages 129 and 142.)

[DGR11] Deepak Dhungana, Paul Grünbacher, and Rick Rabiser. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Automated Software Engineering (ASE)*, 18(1):77–114, 2011.
(Cited on page 40.)

[DGRN10] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.
(Cited on pages 40 and 58.)

[Dij76] Edsger Wybe Dijkstra. *A discipline of programming*, volume 1. prentice-hall Englewood Cliffs, 1976.
(Cited on page 39.)

[dLGC13] Juan de Lara, Esther Guerra, and Jesús Sánchez Cuadrado. Model-driven engineering with domain-specific meta-modelling languages. *Software & Systems Modeling*, 14(1):429–459, 2013.
(Cited on page 60.)

[Doe10] Joerg Doerr. *Elicitation of a complete set of non-functional requirements*. Fraunhofer-Verlag, 2010.
(Cited on page 68.)

[DPFSG13] Jessica Diaz, JM Perez, Cesar Fernandez-Sanchez, and Juan Garbajosa. Model-to-code transformation from product-line architecture models to aspectj. In *Software Engineering and Advanced Applications (SEAA)*, pages 98–105, Santander, Spain, September 2013. IEEE.
(Cited on pages 20 and 45.)

[DS99] Jean-Marc DeBaud and Klaus Schmid. A systematic approach to derive the scope of software product lines. In *International Conference on Software Engineering (ICSE)*, pages 34–43, Los Angeles, USA, May 1999. IEEE.
(Cited on page 2.)

[DS11] Ferruccio Damiani and Ina Schaefer. Dynamic delta-oriented programming. In *Software Product Line Conference (SPLC)*, Munich, Germany, August 2011. ACM.
(Cited on page 39.)

[DSB05] Sybren Deelstra, Marco Sinnema, and Jan Bosch. Product derivation in software product families: a case study. *Journal of Systems and Software*, 74(2):173–194, 2005.
(Cited on page 16.)

[DSL^{+}14] Deepak Dhungana, Herwig Schreiner, Martin Lehofer, Michael Vierhauser, Rick Rabiser, and Paul Grünbacher. Modeling multiplicity and hierarchy in product line architectures: Extending a decision-oriented approach. In *WICSA*, page 11, Sydney, Australia, April 2014. ACM.
(Cited on pages 40, 42, and 62.)

[dSOdAdLM15] Leandro Oliveira de Souza, Pádraig O'Leary, Eduardo Santana de Almeida, and Sílvio Romero de Lemos Meira. Product derivation in practice. *Information and Software Technology*, 58:319–337, 2015.
(Cited on page 36.)

[DTA05] Oscar Díaz, Salvador Trujillo, and Felipe I Anfurrutia. Supporting production strategies as refinements of the production process. In *Software Product Lines Conference (SPLC)*, Rennes, France, September 2005. Springer.
(Cited on page 39.)

[ea] Enterprise Architect. `http://www.sparxsystems.de/`.
(Cited on page 74.)

[eas] EASy-Producer. `https://sse.uni-hildesheim.de/forschung/projekte/easy-producer/`.
(Cited on page 40.)

[ecl] Eclipse Modeling Tools. `https://eclipse.org/modeling`.
(Cited on page 74.)

[EESKS14] Holger Eichelberger, Sascha El-Sharkawy, Christian Kröher, and Klaus Schmid. Easy-producer: product line development for variant-rich ecosystems. In *Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, pages 133–137, Florence, Italy, September 2014. ACM.
(Cited on page 40.)

[Eic] Holger Eichelberger. Topological Configuration. `https://sse.uni-hildesheim.de/en/research/projects/qualimaster/topological-configuration/`.
(Cited on page 42.)

[Eis89] Kathleen M Eisenhardt. Building theories from case study research. *Academy of management review*, 14(4):532–550, 1989.
(Cited on page 132.)

[Els11] Christoph Elsner. *Automating Staged Product Derivation for Heterogeneous Multi–Product-Lines*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, 2011.
(Cited on pages 14, 16, and 18.)

[Els12] Christoph Elsner. Light-weight tool support for staged product derivation. In *Software Product Line Conference (SPLC)*, pages 146–155, Salvador, Brazil, September 2012. ACM.
(Cited on pages 19, 26, and 40.)

[ELSP11] Christoph Elsner, Daniel Lohmann, and Wolfgang Schröder-Preikschat. Fixing configuration inconsistencies across file type boundaries. In *Software Engineering and Advanced Applications (SEAA)*, pages 116–123, Oulu, Finland, August-September 2011. IEEE.
(Cited on pages 20 and 119.)

[EQSS16] Holger Eichelberger, Cui Qin, Roman Sizonenko, and Klaus Schmid. Using ivml to model the topology of big data processing pipelines. In *Proceedings of the 20th International Systems and Software Product Line Conference*, pages 204–208. ACM, 2016.
(Cited on page 42.)

[ES15] Holger Eichelberger and Klaus Schmid. Ivml: a dsl for configuration in variability-rich software ecosystems. In *Conference on Software Product Line*, pages 365–369, Nashville, USA, July 2015. ACM.
(Cited on pages 40, 42, and 121.)

[Fan13] Alessandro Fantechi. Topologically configurable systems as product families. In *Software Product Line Conference (SPLC)*, pages 151–156, Tokyo, Japan, August 2013. ACM.
(Cited on page 45.)

[Fau01] Stuart R Faulk. Product-line requirements specification (prs): An approach and case study. In *Symposium on Requirements Engineering (REFSQ)*, pages 48–55, Interlaken, Switzerland, June 2001. IEEE.
(Cited on page 41.)

[FFB02] Dániel Fey, Róbert Fajta, and András Boros. Feature modeling: A meta-model to enhance usability and usefulness. In *Software Product Lines*, pages 198–216. Springer, San Diego, USA, August 2002.
(Cited on pages 24, 43, 56, and 75.)

[FK05] William B Frakes and Kyo Kang. Software reuse research: Status and future. *IEEE transactions on Software Engineering*, 31(7):529–536, 2005.
(Cited on page 1.)

[FLDE16] Miao Fang, Georg Leyh, Joerg Doerr, and Christoph Elsner. Multi-variability modeling and realization for software derivation in industrial automation management. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pages 2–12, Saint Malo, France, October 2016. ACM.
(Cited on page 132.)

[FLE$^+$15] Miao Fang, Georg Leyh, Christoph Elsner, Joerg Doerr, and Jingjing Zhao. Towards model-based derivation of systems in the industrial automation domain. In *Software Product Line Conference (SPLC)*, pages 283–292, Nashville, USA, July 2015. ACM.
(Cited on pages 5, 21, 25, 37, 55, 73, 94, and 177.)

[FLED13a] Miao Fang, Georg Leyh, Christoph Elsner, and Joerg Doerr. Challenges in managing behavior variability of production control software. In *International Workshop on Product Line Approaches in Software Engineering (PLEASE)*, pages 21–24, San Francisco, USA, May 2013. IEEE.
(Cited on pages 21 and 26.)

[FLED13b] Miao Fang, Georg Leyh, Christoph Elsner, and Joerg Doerr. Experiences during extraction of variability models for warehouse management systems. In *Software Engineering Conference (APSEC)*, pages 111–116, Bangkok, Thailand, December 2013. IEEE.
(Cited on pages 4, 21, 24, 25, 29, 43, and 131.)

[Fly06] Bent Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245, 2006.
(Cited on page 141.)

[Gam95] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.
(Cited on page 88.)

[GBPG13] Gerd Gröner, Marko BošKović, Fernando Silva Parreiras, and Dragan GašEvić. Modeling and validation of business process families. *Information Systems*, 38(5):709–726, 2013.
(Cited on pages 45, 73, and 77.)

[Gom04] Hassan Gomaa. Designing software product lines with uml: From use cases to pattern-based software architectures., 2004.
(Cited on page 44.)

[GSC15] Susan P Gregg, Rick Scharadin, and Paul Clements. The more you do, the more you save: the superlinear cost avoidance effect of systems product line engineering. In *Software Product Line Conference (SPLC)*, pages 303–310, Nashville, USA, July 2015. ACM.
(Cited on page 14.)

[Gui02] Lisa Ann Guion. *Triangulation: Establishing the validity of qualitative studies*. University of Florida Cooperative Extension Service, Institute of Food and Agricultural Sciences, EDIS, 2002.
(Cited on page 150.)

[GWW$^+$11] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208–2221, 2011.
(Cited on pages 19 and 156.)

[HB13] Christian Heinzemann and Steffen Becker. Executing reconfigurations in hierarchical component architectures. In *International ACM Sigsoft symposium on Component-based software engineering (CBSE)*, pages 3–12. ACM, July 2013.
(Cited on pages 46, 77, and 96.)

[HEGV13] Gerald Holl, Christoph Elsner, Paul Grünbacher, and Michael Vierhauser. An infrastructure for the life cycle management of multi product lines. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 1742–1749. ACM, 2013.
(Cited on page 40.)

[HGE$^+$13] Gerald Holl, Paul Grunbacher, Christoph Elsner, Thomas Klambauer, and Michael Vierhauser. Constraint checking in distributed product configuration of multi product lines. In *Asia-Pacific Software Engineering Conference (APSEC)*, volume 1, pages 347–354, Bangkok, Thailand, December 2013. IEEE.
(Cited on page 40.)

[HGR12] Gerald Holl, Paul Grünbacher, and Rick Rabiser. A systematic review and an expert survey on capabilities supporting multi product lines. *Information and Software Technology*, 54(8):828–852, 2012.
(Cited on page 36.)

[HHS+13] Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens, Dirk Deridder, and Ebrahim Khalil Abbasi. Supporting multiple perspectives in feature-based configuration. *Software & Systems Modeling*, 12(3):641–663, 2013.
(Cited on page 18.)

[HHSD10] Arnaud Hubaux, Patrick Heymans, Pierre-Yves Schobbens, and Dirk Deridder. Towards multi-view feature-based configuration. In *Requirements Engineering: Foundation for Software Quality*, pages 106–112. Springer, Essen, Germany, June-July 2010.
(Cited on page 18.)

[HLHE11] Evelyn Nicole Haslinger, Roberto E Lopez-Herrejon, and Alexander Egyed. Reverse engineering feature models from programs' feature sets. In *Working Conference on Reverse Engineering (WCRE)*, pages 308–312, limerick, Ireland, October 2011. IEEE.
(Cited on page 132.)

[Hol11] Gerald Holl. Product line bundles to support product derivation in multi product lines. In *Software Product Line Conference (SPLC)*, page 41, Munich, Germany, August 2011. ACM.
(Cited on pages 24 and 40.)

[HP03] Günter Halmans and Klaus Pohl. Communicating the variability of a software-product family to customers. *Software and Systems Modeling*, 2(1):15–36, 2003.
(Cited on pages ix, 1, 2, 14, 16, 24, 26, and 41.)

[HST+08] Patrick Heymans, P-Y Schobbens, J-C Trigaux, Yves Bontemps, R Matulevičius, and Andreas Classen. Evaluating formal properties of feature diagram languages. *IET software*, 2(3):281–302, 2008.
(Cited on page 75.)

[IEC] Enterprise-control system integration. International Electrotechnical Commission.
(Cited on page 3.)

[ISA] ANSI/ISA-95.00.03-2005, enterprise-control system integration, part 3: Models of manufacturing operations management. The International Society of Automation.
(Cited on page 3.)

[Iso01] Sadahiro Isoda. Object-oriented real-world modeling revisited. *Journal of Systems and Software*, 59(2):153–162, 2001.
(Cited on page 62.)

[KBR12] Vinay Kulkarni, Souvik Barat, and Suman Roychoudhury. Towards business application product lines. In *workshop on variability modeling of software-intensive systems*. Springer, September-October 2012.
(Cited on page 46.)

[KCH$^+$90] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document, 1990.
(Cited on pages 2, 12, 14, 56, and 75.)

[Kee07] Staffs Keele. Guidelines for performing systematic literature reviews in software engineering. In *Technical report, Ver. 2.3 EBSE Technical Report. EBSE*. Keele University, 2007.
(Cited on pages 33 and 36.)

[Kit04] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 2004.
(Cited on pages 34 and 35.)

[KLD02] Kyo C Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE software*, (4):58–65, 2002.
(Cited on pages 56 and 75.)

[KLM$^+$97] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. *Aspect-oriented programming*. Springer, 1997.
(Cited on page 45.)

[Kru02] CharlesW Krueger. Easing the transition to software mass customization. In *Software Product-Family Engineering*, pages 282–293. Springer, 2002.
(Cited on page 7.)

[Kru07a] Charles W Krueger. The 3-tiered methodology: Pragmatic insights from new generation software product lines. In *Software Product Line Conference (SPLC)*, pages 97–106, Kyoto, Japan, 2007. IEEE.
(Cited on pages 37 and 39.)

[Kru07b] Charles W Krueger. Biglever software gears and the 3-tiered spl methodology. In *Object-oriented programming systems and applications (OOPSLA)*, pages 844–845, Montreal, Canada, October 2007. ACM.
(Cited on pages 14 and 39.)

[Kru13] Charles W Krueger. Multistage configuration trees for managing product family trees. In *Software Product Line Conference (SPLC)*, pages 188–197, Tokyo, Japan, August 2013. ACM.
(Cited on pages 37 and 39.)

[KS16] Matthias Kowal and Ina Schaefer. Incremental consistency checking in delta-oriented uml-models for automation systems. *arXiv preprint arXiv:1604.00348*, 2016.
(Cited on pages 74 and 119.)

[KSRB13] Dean Kramer, Christian Sauer, and Thomas Roth-Berghofer. Towards explanation generation using feature models in software product lines. *Knowledge Engineering and Software Engineering (KESE)*, page 13, 2013.
(Cited on page 54.)

[KTS$^+$09] Christian Kästner, Thomas Thüm, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, and Sven Apel. Featureide: A tool framework for feature-oriented software development. In *International Conference on Software Engineering (ICSE)*, pages 611–614, Vancouver, Canada, May 2009. IEEE.
(Cited on pages 14, 39, and 76.)

[LBT09] Kwanwoo Lee, Goetz Botterweck, and Steffen Thiel. Feature-modeling and aspect-oriented programming: Integration and automation. In *Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing*, pages 186–191, Daegu, Korea, May 2009. IEEE.
(Cited on page 45.)

[LHMME11] Roberto E Lopez-Herrejon, Leticia Montalvillo-Mendizabal, and Alexander Egyed. From requirements to features: An exploratory study of feature-oriented refactoring. In *International Software Product Line Conference (SPLC)*, pages 181–190, Munich, Germany, August 2011. IEEE.
(Cited on pages 91, 92, 93, 95, 135, and 155.)

[LIA10] Grzegorz Loniewski, Emilio Insfran, and Silvia Abrahão. A systematic review of the use of requirements engineering techniques in model-driven development. In *Model driven engineering languages and systems (MoDELS)*, pages 213–227. Springer, Oslo, Norway, October 2010.
(Cited on page 55.)

[LJG$^+$15] Jörg Liebig, Andreas Janker, Florian Garbe, Sven Apel, and Christian Lengauer. Morpheus: Variability-aware refactoring in the wild. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, pages 380–391, Florence, Italy, May 2015. IEEE Press.
(Cited on pages 95 and 155.)

[LKA$^+$95] David C. Luckham, John J. Kenney, Larry M. Augustin, James Vera, Doug Bryan, and Walter Mann. Specification and analysis of system

architecture using rapide. *IEEE Transactions on Software Engineering*, 21(4):336–354, 1995.
(Cited on page 85.)

[LKL02] Kwanwoo Lee, Kyo C Kang, and Jaejoon Lee. Concepts and guidelines of feature modeling for product line software engineering. In *International Conference on Software Reuse*, pages 62–77. Springer, 2002.
(Cited on pages 57 and 58.)

[LKT04] Janne Luoma, Steven Kelly, and Juha-Pekka Tolvanen. Defining domain-specific modeling languages: Collected experiences. In *The 4th Workshop on Domain-Specific Modeling*, Vancouver, Canada, 2004.
(Cited on page 56.)

[LRVdADM17] Marcello La Rosa, Wil MP Van der Aalst, Marlon Dumas, and Fredrik P Milani. Business process variability modeling: A survey. *ACM Computing Surveys*, 50(1):2, 2017.
(Cited on page 77.)

[mag] MagicDraw. `http://www.nomagic.com/products/magicdraw.html`.
(Cited on pages 74 and 106.)

[Max92] Joseph Maxwell. Understanding and validity in qualitative research. *Harvard educational review*, 62(3):279–301, 1992.
(Cited on page 141.)

[MBC09] Marcilio Mendonca, Moises Branco, and Donald Cowan. Splot: software product lines online tools. In *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, pages 761–762. ACM, 2009.
(Cited on page 74.)

[MBK91] Yoëlle S Maarek, Daniel M Berry, and Gail E Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17(8):800–813, 1991.
(Cited on page 69.)

[McC97] Michael McClellan. *Applying manufacturing execution systems*. CRC Press, 1997.
(Cited on page 3.)

[McG05] John D McGregor. Preparing for automated derivation of products in a software product line. 2005.
(Cited on page 47.)

[MDEK95] Jeff Magee, Naranker Dulay, Susan Eisenbach, and Jeff Kramer. Specifying distributed software architectures. *Software Engineering—ESEC'95*, pages 137–153, 1995.
(Cited on page 85.)

[MHS05] Marjan Mernik, Jan Heering, and Anthony M Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.
(Cited on page 68.)

[MSA09] Mike Mannion, Juha Savolainen, and Timo Asikainen. Viewpoint-oriented variability modeling. In *Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 67–72, Seattle, USA, July 2009. IEEE.
(Cited on page 19.)

[MT00] Nenad Medvidovic and Richard N Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on software engineering*, 26(1):70–93, 2000.
(Cited on page 85.)

[Mut02] Dirk Muthig. *A light-weight approach facilitating an evolutionary transition towards software product lines*. PhD thesis, Fraunhofer IRB, 2002.
(Cited on page 2.)

[NE08] Nan Niu and Steve Easterbrook. Extracting and modeling product line functional requirements. In *IEEE International Requirements Engineering Conference (RE)*, pages 155–164, Catalunya, Spain, September 2008. IEEE.
(Cited on pages 15, 69, and 155.)

[OAS] OASIS. Business Process Model And Notation (BPMN) Version 2.0. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.
(Cited on page 77.)

[ODAR12] Pádraig O'Leary, Eduardo Santana De Almeida, and Ita Richardson. The pro-pd process model for product derivation within software product lines. *Information and Software Technology*, 54(9):1014–1028, 2012.
(Cited on pages 48 and 120.)

[OMG11] OMG. Business Process Model And Notation (BPMN) Version 2.0. `http://www.omg.org/spec/BPMN/2.0/`, 2011.
(Cited on page 68.)

[OMG14] OMG. OObject Constraint Language (OCL). `http://www.omg.org/spec/OCL/2.4/`, 2014.
(Cited on page 57.)

[OMG15] OMG. OMG Meta Object Facility (MOF) Core Specification. `http://www.omg.org/spec/MOF/2.5/PDF/`, 2015.
(Cited on page 54.)

[ORRT09] Pádraig O'Leary, Rick Rabiser, Ita Richardson, and Steffen Thiel. Important issues and key activities in product derivation: experiences from two independent research projects. In *Software Product Line Conference (SPLC)*, pages 121–130, San Francisco, USA, August 2009. ACM.
(Cited on pages 3, 16, 17, 18, 29, 38, 48, and 142.)

[OTBR08] Pádraig O'Leary, Steffen Thiel, Goetz Botterweck, and Ita Richardson. Towards a product derivation process framework. In *Central and East European Conference on Software Engineering Techniques (CEE-SET)*, Brno, Czech Republic, October 2008.
(Cited on page 47.)

[PFS09] Chitralekha Pillai, Ronald Fabel, and Lou Somers. *Model Based Control Software Synthesis for Paper Handling in Printers*. Eindhoven University of Technology, 2009.
(Cited on page 46.)

[pur] pure::variants. `http://www.pure-systems.com/pure_variants.49.0.html`.
(Cited on pages 14 and 76.)

[QRD13] Clément Quinton, Daniel Romero, and Laurence Duchien. Cardinality-based feature models with constraints: a pragmatic approach. In *Software Product Line Conference*, pages 162–166, Tokyo, Japan, August 2013. ACM.
(Cited on page 56.)

[RGD07] Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. Supporting product derivation by adapting and augmenting variability models. In *Software Product Line Conference (SPLC)*, pages 141–150, Kyoto, Japan, 2007. IEEE.
(Cited on pages 14 and 40.)

[RGD10] Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, 52(3):324–346, 2010.
(Cited on pages 29, 34, 34, 35, 36, 36, 44, 44, 49, and 142.)

[RGL12] Rick Rabiser, Paul Grünbacher, and Martin Lehofer. A qualitative study on user guidance capabilities in product configuration tools. In *Automated Software Engineering (ASE)*, pages 110–119, Essen, Germany, September 2012. IEEE.
(Cited on pages 19, 40, and 142.)

[RH09] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.
(Cited on pages 132 and 143.)

[RHE+10] Rick Rabiser, Wolfgang Heider, Christoph Elsner, Martin Lehofer, Paul Grünbacher, and Christa Schwanninger. A flexible approach for generating product-specific documents in product lines. In *Software Product Lines Conference (SPLC)*, pages 47–61. Springer, Jeju Island, South Korea, September 2010.
(Cited on page 41.)

[ROR11] Rick Rabiser, Pádraig O'Leary, and Ita Richardson. Key activities for product derivation in software product lines. *Journal of Systems and Software*, 84(2):285–300, 2011.
(Cited on pages 3, 16, 17, 40, 48, and 120.)

[RPB99] Colette Rolland, Naveen Prakash, and Adolphe Benjamen. A multi-model view of process modelling. *Requirements Engineering*, 4(4):169–187, 1999.
(Cited on page 67.)

[SAG+12] Samaneh Soltani, Mohsen Asadi, Dragan Gašević, Marek Hatala, and Ebrahim Bagheri. Automated planning for feature model configuration based on functional and non-functional requirements. In *Software Product Line Conference*, pages 56–65, Salvador, Brazil, September 2012. ACM.
(Cited on page 103.)

[SBB+10] Ina Schaefer, Lorenzo Bettini, Viviana Bono, Ferruccio Damiani, and Nico Tanzarella. Delta-oriented programming of software product lines. In *Software Product Lines Conference (SPLC)*, pages 77–91. Springer, Jeju Island, South Korea, September 2010.
(Cited on pages 20 and 39.)

[SCDLG12] Jesús Sánchez-Cuadrado, Juan De Lara, and Esther Guerra. *Bottom-up meta-modelling: An interactive approach*. Springer, September-October 2012.
(Cited on pages 64, 68, and 68.)

[SD08] Marco Sinnema and Sybren Deelstra. Industrial validation of covamof. *Journal of Systems and Software*, 81(4):584–600, 2008.
(Cited on page 39.)

[SDH06] Marco Sinnema, Sybren Deelstra, and Piter Hoekstra. The covamof derivation process. In *Reuse of Off-the-Shelf Components*, pages 101–114. Springer, 2006.
(Cited on page 39.)

[SDNB04] Marco Sinnema, Sybren Deelstra, Jos Nijhuis, and Jan Bosch. Covamof: A framework for modeling variability in software product families. In *Software Product Lines Conference (SPLC)*, pages 197–213. Springer, Boston, USA, Auguest-September 2004.
(Cited on page 38.)

[SE15] Klaus Schmid and Holger Eichelberger. Easy-producer: from product lines to variability-rich software ecosystems. In *Software Product Line Conference (SPLC)*, pages 390–391, Nashville, USA, July 2015. ACM.
(Cited on page 40.)

[SGEL09] Christa Schwanninger, Iris Groher, Christoph Elsner, and Martin Lehofer. Variability modelling throughout the product line lifecycle. In *Model Driven Engineering Languages and Systems (MODELS)*, pages 685–689. Springer, Denver, USA, October 2009.
(Cited on pages 73, 83, and 85.)

[SGF$^+$10] Mikael Svahnberg, Tony Gorschek, Robert Feldt, Richard Torkar, Saad Bin Saleem, and Muhammad Usman Shafique. A systematic review on strategic release planning models. *Information and software technology*, 52(3):237–248, 2010.
(Cited on pages 35 and 35.)

[Sha02] Mary Shaw. What makes good research in software engineering? *International Journal on Software Tools for Technology Transfer*, 4(1):1–7, 2002.
(Cited on page 131.)

[Sha03] Mary Shaw. Writing good software engineering research papers: minitutorial. In *International Conference on Software Engineering (ICSE)*, pages 726–736, Portland, USA, May 2003. IEEE Computer Society.
(Cited on pages 49, 131, and 141.)

[SHTB07] Pierre-Yves Schobbens, Patrick Heymans, Jean-Christophe Trigaux, and Yves Bontemps. Generic semantics of feature diagrams. *Computer Networks*, 51(2):456–479, 2007.
(Cited on page 75.)

[SIE] Siemens Picture Database. `http://www.automation.siemens.com/bilddb/`.
(Cited on page 4.)

[SK13] Hamideh Sabouri and Ramtin Khosravi. Delta modeling and model checking of product families. In *Fundamentals of Software Engineering*, pages 51–65. Springer, Tehran, Iran, April 2013.
(Cited on page 40.)

[SP06] Arnd Schnieders and Frank Puhlmann. Variability mechanisms in e-business process families. *BIS*, 85:583–601, 2006.
(Cited on page 77.)

[SVGB05] Mikael Svahnberg, Jilles Van Gurp, and Jan Bosch. A taxonomy of variability realization techniques. *Software: Practice and Experience*, 35(8):705–754, 2005.
(Cited on pages 12, 14, 82, 84, 88, 95, 95, and 97.)

[t4] T4 - Microsoft. `https://msdn.microsoft.com/en-us/library/dd820703.aspx`.
(Cited on page 86.)

[TBG13] Leopoldo Teixeira, Paulo Borba, and Rohit Gheyi. Safe composition of configuration knowledge-based software product lines. *Journal of Systems and Software*, 86(4):1038–1053, 2013.
(Cited on pages 20, 119, and 156.)

[TK05] Juha-Pekka Tolvanen and Steven Kelly. Defining domain-specific modeling languages to automate product derivation: Collected experiences. In *Software Produce Line Concerence (SPLC)*, pages 198–209. Springer, Rennes, France, September 2005.
(Cited on pages 51 and 56.)

[TKES11] Thomas Thum, Christian Kastner, Sebastian Erdweg, and Norbert Siegmund. Abstract features in feature modeling. In *Software Product Lines Conference (SPLC)*, pages 191–200, Munich, Germany, August 2011. IEEE.
(Cited on page 102.)

[TPRB06] Bruce Trask, Dominick Paniscotti, Angel Roman, and Vikram Bhanot. Using model-driven engineering to complement software product line engineering in developing software defined radio components and applications. In *Software Product Line Conference (SPLC)*, pages 846–853, Baltimore, USA, August 2006. ACM.
(Cited on page 46.)

[TZD08] Huy Tran, Uwe Zdun, and Schahram Dustdar. View-based integration of process-driven soa models at various abstraction levels. In *Model-Based Software and Data Integration*, pages 55–66. Springer, 2008.
(Cited on page 68.)

[UBFC14] Simon Urli, Mireille Blay-Fornarino, and Philippe Collet. Handling complex configurations in software product lines: a tooled approach. In *Software Product Line Conference (SPLC)*, pages 112–121, Florence, Italy, September 2014. ACM.
(Cited on pages 19, 45, 103, and 119.)

[Val10] Antonio Vallecillo. On the combination of domain specific modeling languages. In *European Conference on Modelling Foundations and Applications (ECMFA)*, pages 305–320. Springer, Paris, France, June 2010.
(Cited on pages 51 and 55.)

[VBMD11] Valentino Vranić, Michal Bebjak, Radoslav Menkyna, and Peter Dolog. Developing applications with aspect-oriented change realization. In *Central and East European Conference(CEE-SET)*, pages 192–206. Springer, Brno, Czech Republic, October 2011.
(Cited on page 20.)

[vdA97] W van der Aalst. On the verification of interorganizational workflows. *Computing Science Reports*, 97:16, 1997.
(Cited on page 68.)

[vDATHKB03] Wil MP van Der Aalst, Arthur HM Ter Hofstede, Bartek Kiepuszewski, and Alistair P Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.
(Cited on page 45.)

[VDK02] Arie Van Deursen and Paul Klint. Domain-specific language design requires feature descriptions. *CIT. Journal of computing and information technology*, 10(1):1–17, 2002.
(Cited on page 75.)

[VG07] Markus Voelter and Iris Groher. Product line implementation using aspect-oriented and model-driven software development. In *Software Product Line Conference (SPLC)*, pages 233–242, Tokyo, Japan, September 2007. IEEE.
(Cited on pages 83 and 83.)

[VGBS01] Jilles Van Gurp, Jan Bosch, and Mikael Svahnberg. On the notion of variability in software product lines. In *2001 Working IEEE / IFIP Conference on Software Architecture (WICSA)*, pages 45–54, Amsterdam, The Netherlands, August 2001. IEEE.
(Cited on page 56.)

[vis] Visio. `https://products.office.com/de-de/visio`.
(Cited on page 74.)

[Voe03] Markus Voelter. A catalog of patterns for program generation. In *EuroPLoP*, pages 285–320, 2003.
(Cited on page 120.)

[VOVDLKM00] Rob Van Ommering, Frank Van Der Linden, Jeff Kramer, and Jeff Magee. The koala component model for consumer electronics software. *Computer*, 33(3):78–85, 2000.
(Cited on pages 44, 85, and 96.)

[VSB99] Rini Van Solingen and Egon Berghout. *The Goal/Question/Metric Method: a practical guide for quality improvement of software development*. McGraw-Hill, 1999.
(Cited on pages 126 and 127.)

[VSBCR02] Rini Van Solingen, Vic Basili, Gianluigi Caldiera, and H Dieter Rombach. Goal question metric (gqm) approach. *Encyclopedia of software engineering*, 2002.
(Cited on page 126.)

[VV11] Markus Voelter and Eelco Visser. Product line engineering using domain-specific languages. In *Software Product Line Conference (SPLC)*, pages 70–79, Munich, Germany, August 2011. IEEE.
(Cited on pages 44 and 121.)

[WBS$^+$10] Jules White, David Benavides, Douglas C Schmidt, Pablo Trinidad, Brian Dougherty, and A Ruiz-Cortes. Automated diagnosis of feature model configurations. *Journal of Systems and Software*, 83(7):1094–1107, 2010.
(Cited on pages 20, 74, and 119.)

[WC15] Len Wozniak and Paul Clements. How automotive engineering is taking product line engineering to the extreme. In *Proceedings of the 19th International Conference on Software Product Line*, pages 327–336, Nashville, USA, July 2015. ACM.
(Cited on page 14.)

[Wil03] David Wile. Lessons learned from real dsl experiments. In *Hawaii International Conference on System Sciences (HICSS)*, pages 10–pp. IEEE, Janurary 2003.
(Cited on page 68.)

[WL99] David M Weiss and Chi Tau Robert Lai. *Software product-line engineering: a family-based software development process*. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA, 1999.
(Cited on pages 2, 14, 36, and 47.)

[WMW11] Matthias Weidlich, Jan Mendling, and Mathias Weske. A foundational approach for managing process variability. In *Advanced Information Systems Engineering*, pages 267–282. Springer, 2011.
(Cited on pages 45, 73, 77, and 87.)

[xpa] Xpand - Eclipse. `https://eclipse.org/modeling/m2t/?project=xpand`.
(Cited on page 86.)

[Yin13] Robert K Yin. *Case study research: Design and methods*. Sage publications, 2013.
(Cited on pages 132 and 132.)

[YZZJ14] Wenjing Yu, Wei Zhang, Haiyan Zhao, and Zhi Jin. Tdl: a transformation description language from feature model to use case for automated use case derivation. In *Software Product Line Conference (SPLC)*, pages 187–196, Florance, Italy, September 2014. ACM.
(Cited on page 41.)

A

Appendix: The Studies in Systematic Literature Review

In Chapter 3, the systematic literature review resulted in 109 studies. With inclusion criteria, exclusion criteria, quality assessment, and snowball-searching, 43 studies published between 2008 and 2015 become the outcome of the literature review, including the one of the author's own work [FLE^{+}15]. This appendix collects these studies in Table A.1, with the information about the approach name, the year of publication, the title, the first author and the publisher.

Approach	**Year**	**Title**	**First Author**	**Publisher**
COVAMOF	2008	Industrial validation of COVAMOF	Sinnema, Marco	Elsevier
Three-tiers methodology	2013	Multistage configuration trees for managing product family trees	Krueger, Charles	ACM
	2013	A PLE-based auditing method for protecting restricted content in derived products	Clements, Paul	ACM
Feature-oriented programming	2009	FEATUREHOUSE: Language-independent, automated software composition	Apel, Sven	IEEE
	2009	FeatureIDE: A tool framework for feature-oriented software development	Kästner, Christian	IEEE
Delta-oriented programming	2010	Delta-oriented programming of software product lines	Schaefer, Ina	Springer
	2010	Dynamic delta-oriented programming	Damiani, Ferruccio	ACM
	2013	Delta modeling and model checking of product families	Sabouri, Hamideh	Springer
SimPL	2014	SimPL: a product-line modeling methodology for families of integrated control systems	Behjati, Razieh	Elsevier

DOPLER	2010	Structuring the modeling space and supporting evolution in software product line engineering	Dhungana, Deepak	Elsevier
	2011	The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study	Dhungana, Deepak	Springer
	2011	A systematic review and an expert survey on capabilities supporting multi product lines	Rabiser, Rick	Elsevier
	2012	A qualitative study on user guidance capabilities in product configuration tools	Rabiser, Rick	IEEE
	2014	Modeling multiplicity and hierarchy in product line architectures: Extending a decision-oriented approach	Dhungana, Deepak	ACM
Multi-product-line	2011	Product line bundles to support product derivation in multi product lines	Holl, Gerald	ACM
	2012	Light-weight tool support for staged product derivation	Elsner, Christoph	ACM
	2013	Constraint Checking in Distributed Product Configuration of Multi Product Lines	Holl, Gerald	IEEE
	2013	An infrastructure for the life cycle management of multi product lines	Holl, Gerald	ACM
Product-Line Requirements Specification	2010	A flexible approach for generating product-specific documents in product lines	Rabiser, Rick	Springer
	2013	Effective Requirements Elicitation in Product Line Application Engineering–An Experiment	Adam, Sebastian	Springer
	2014	TDL: a transformation description language from feature model to use case for automated use case derivation	Yu, Wenjing	ACM
DSL-based methodology	2011	Product line engineering using domain-specific languages	Voelter, Markus	IEEE
EASy-Producer	2014	EASy-producer: product line development for variant-rich ecosystems	Eichelberger, Holger	ACM
	2015	EASy-Producer: from product lines to variability-rich software ecosystems	Schmid, Klaus	ACM
	2015	IVML: a DSL for configuration in variability-rich software ecosystems	Eichelberger, Holger	ACM
Topology-oriented methodologies	2014	Topologically configurable systems as product families	Fantechi, Alessandro	ACM
	2014	Handling complex configurations in software product lines: a tooled approach	Urli, Simon	ACM
	2014	To connect or not to connect: experiences from modeling topological variability	Berger, Thorsten	ACM
FAST	1999	Software product-line engineering: a family-based software development process	Weiss, David	Addison-Wesley

Domain-specific modeling	2012	Towards business application product lines	Kulkarni, Vinay	Springer
	2013	Executing reconfigurations in hierarchical component architectures	Heinzemann, Christian	ACM
Process-oriented methodologies	2011	A foundational approach for managing process variability	Weidlich, Matthias	Springer
	2013	Modeling and validation of business process families	Groener, Gerd	Elsevier
Aspect-oriented programming	2009	Feature-modeling and aspect-oriented programming: Integration and automation	Lee, Kwanwoo	IEEE
	2012	Automating the product derivation process of multi-agent systems product lines	Cirilo, Elder	Elsevier
	2013	Model-to-Code transformation from product-line architecture models to aspectJ	Diaz, Jessica	IEEE
Pro-PD	2008	Towards a product derivation process framework	O'Leary, Pádraig	ifip
	2012	The Pro-PD process model for product derivation within software product lines	O'Leary, Pádraig	Elsevier
Reference process of product derivation	2009	Important issues and key activities in product derivation: experiences from two independent research projects	O'Leary, Pádraig	ACM
	2011	Key activities for product derivation in software product lines	Rabiser, Rick	Elsevier
Literature Reviews and General Case Studies				
	2010	Requirements for product derivation support: Results from a systematic literature review and an expert survey	Rabiser, Rick	Elsevier
	2012	A systematic review and an expert survey on capabilities supporting multi product lines	Holl, Geral	Elsevier
	2015	Product derivation in practice	de Souza, Leandro Oliveira	Elsevier

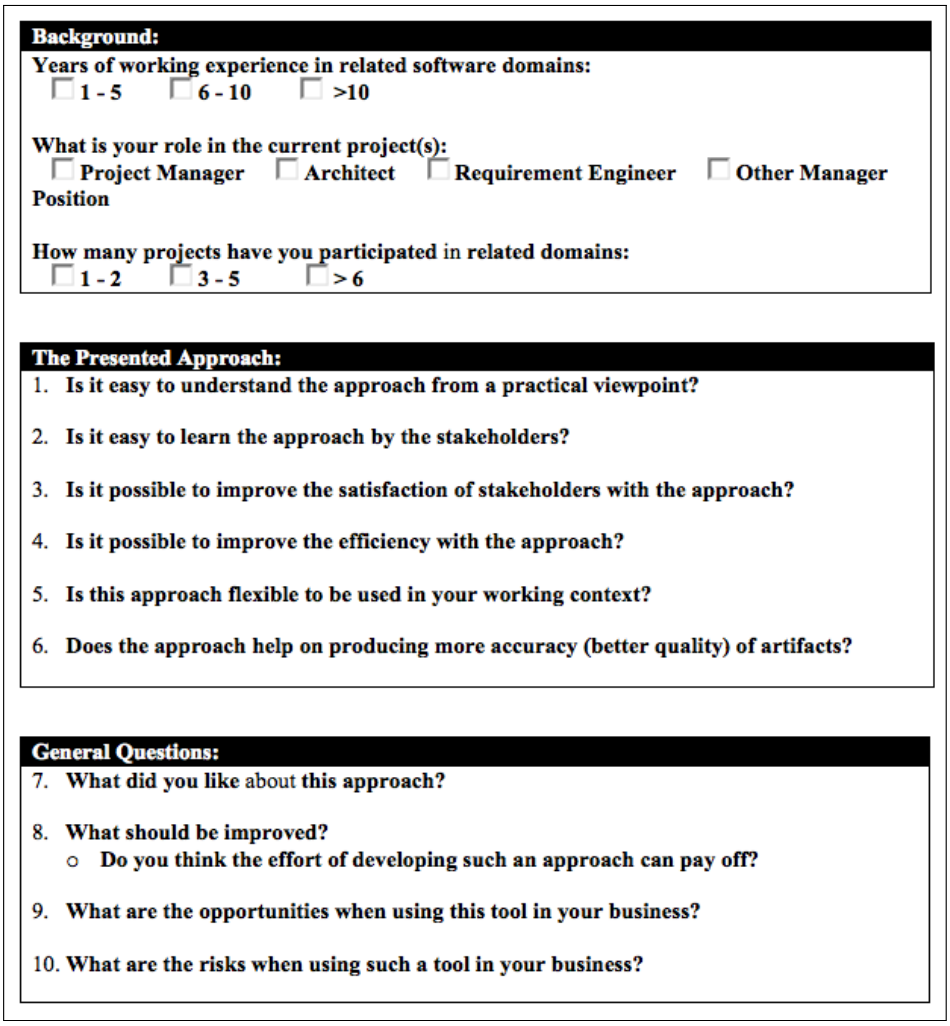
Table A.1.: Paper Matrix

B
Appendix: Interview Scripts

The answers collected in the ten semi-structured interviews during the evaluation of the proposed approach in this thesis are in the format of raw scripts. Figure B.1 presents the questionnaire, which has been already introduced in Chapter 7. This appendix reports on these data.

Interview 1:

- **Current role of the interviewee:** Requirement Engineer
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** 1-5
- **Number of project experience:** 3-5
- **Interview date:** February 16, 2015
- **Interview duration:** 50 minutes
- **Answer of Question 1:** Yes. It is very easy to understand the modeling notations. The understandability depends (also) on the quality of the tooling. The meta-elements may still need to be further tailored to specific product families.
- **Answer of Question 2:** Yes. For me, an individual person, it is easy to learn. The adoption of the approach in the team requires support from management.
- **Answer of Question 3:** Yes, for both requirement engineers and software developers. The most important benefit is to ease the communication between them.
- **Answer of Question 4:** Yes. When the adoption is successful, the approach provides each role of stakeholders a shared understanding to the problem.



Background:

Years of working experience in related software domains:
□ **1 - 5** □ **6 - 10** □ **>10**

What is your role in the current project(s):
□ **Project Manager** □ **Architect** □ **Requirement Engineer** □ **Other Manager Position**

How many projects have you participated in related domains:
□ **1 - 2** □ **3 - 5** □ **> 6**

The Presented Approach:

1. **Is it easy to understand the approach from a practical viewpoint?**
2. **Is it easy to learn the approach by the stakeholders?**
3. **Is it possible to improve the satisfaction of stakeholders with the approach?**
4. **Is it possible to improve the efficiency with the approach?**
5. **Is this approach flexible to be used in your working context?**
6. **Does the approach help on producing more accuracy (better quality) of artifacts?**

General Questions:

7. **What did you like about this approach?**
8. **What should be improved?**
   - **Do you think the effort of developing such an approach can pay off?**
9. **What are the opportunities when using this tool in your business?**
10. **What are the risks when using such a tool in your business?**

Figure B.1.: The Questionnaire Used in the Semi-Structured Interviews

- **Answer of Question 5:** The flexibility should be balanced.
- **Answer of Question 6:** It is very necessary to have correctness checking. It should be applied before the submission of requirement specification. This would further lead to the extraction of business rules within processes.
- **Answer of Question 7:** The topology and process editors standardize the requirement documentation. The standardization improves the communication efficiency and work efficiency.
- **Answer of Question 8:** (The answer to this question has be included in the answers to the previous questions.)
- **Answer of Question 9:** It would be nice to have the approach implemented in Visio. Using MagicDraw could be difficult for the adoption.

- **Answer of Question 10:** The knowledge transform is a risk. Training should be performed regularly to synchronize the understanding of standardized notations, especially when new people join the team, or after new "meta-elements" are added. Otherwise, people turn to revert to their previous working style.

Interview 2:

- **Current role of the interviewee:** Architect
- **Location of the interview:** Erlangen, Germany
- **Organization:** Siemens
- **Years of working experience:** 6-10
- **Number of project experience:** 3-5
- **Interview date:** February 11, 2015
- **Interview duration:** 57 minutes
- **Answer of Question 1:** Before, they (the stakeholders) have only textual specification for the processes; the graphical notations are easier to understand compared to before. The graphical notations give better overview of the functionality of the "processes", the topology are not for understanding but as linkages to hardware devices or controllers. Other standard notations, e.g. Business Process Modeling Notations, or Unified Modeling Languages, don't have the material flow. Therefore, tailoring is important to the domain. Otherwise the notations are not understandable.
- **Answer of Question 2:** In the topology model, the "Transport To/Transport From" are bad for learning. Other parts are easy to learn. The risks could be, for example, the projects used the SPL family engineering are not enough, more notations may be needed, when enlarging the scope. But currently it is okay.
- **Answer of Question 3:** Yes. The reason is also related to self-confidence about the result. The code generator can be used at the starting phase of development to derive the architecture and the structure of the code for the complex components. It make my job easier. Synchronization of model and code would make the change faster. The approach creates common understanding between requirement engineers and software developers. The graphical notation gives customers a better chance to understand systems during pre-sale phase. Topology model helps the hardware development to communicate to software team easier.
- **Answer of Question 4:** Yes. The current development requires the understanding among customers, requirement engineers, and developers. Later on, it requires the negotiation back with customers. The approach helps on reducing the number of these loops. The code generator enables the generation of architecture faster. The abstract topology layout can be linked to components, code structure, and the setup of the project directly. The process templates save the time of modeling, and are linked to reusable artifacts.

- **Answer of Question 5:** The usage of the editors should be limited to avoid misconfiguration, and to restrict accessible elements, so that the manual configuration is under-controlled.
- **Answer of Question 6:** Model validation is important during modeling time. To use the approach in practice, it is necessary to have it.
- **Answer of Question 7:** The Model Instance Generator is a very good idea, because it guarantees the generated code structure trustworthy, since the component architecture was even variability. The code generator is easy to use.
- **Answer of Question 8:** Choosing the modeling platform matters. Versioning and merging of models need to be considered in real industrial settings. The tooling should be gradually extended. The code generator should be more powerful, to generator more code.
- **Answer of Question 9:** For sales, the tool needs to be restricted. The approach helps on quickly generating a prototype to show to customers. It gives a better "image" to customers, and makes the requirement easier to understand by customers. For the modeling part, the development of the models should be iterative, possibly for different purpose, i.e code generation or communication.
- **Answer of Question 10:** The business can be limited because of the tool. The evolution of SPL infrastructure may slow down the evolution of business. The learning curve to adopt the approach could be high. The people who don't know about modeling can be scared away.

Interview 3:

- **Current role of the interviewee:** Maintenance Engineer
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** 9
- **Number of project experience:** >6
- **Interview date:** February 16, 2015
- **Interview duration:** 55 minutes
- **Answer of Question 1:** The approach is very easy to understand. Since the tool does generation, it is important to improve modularity of the code, and extract configurable process steps.
- **Answer of Question 2:** Yes. It is not necessary to give the tool to requirement people. Project managers are decision makers, but they do not need to understand it. The tool is important for function designer sand developers. They must completely understand the tool and each process action.
- **Answer of Question 3:** Yes, because of the improvement of efficiency.

- **Answer of Question 4:** Yes. The idea is good, but it requires to have tool maturity. The improvement is also at the reuse of SPL architecture. The generated code must have good quality, so that they can be the replacement of developers' own coding.
- **Answer of Question 5:** The flexibility should be limited, and restricted to the SPL family. The tooling should be used to communicate with customers and users. Process commonalities are around 70%-80% among customers. The focus of the reuse should be at the common parts.
- **Answer of Question 6:** There is not need to do model validation in the approach. The refinement and correctness should be done by developers. Testing the code should also be done by developers, when the reuse of process steps is actually at the plug-reuse level. When we expect the tool does too many different tasks, we would not be able to use it at all.
- **Answer of Question 7:** (The answer to this question has be included in the answers to the previous questions.)
- **Answer of Question 8:** (The answer has been included in answers to Question 4, and question 5.
- **Answer of Question 9:** Considering the commonalities of processes among customers, the reuse can be quite beneficial.
- **Answer of Question 10:** It depends on how good the experience of team members is. The risk is high, but after several iterations, the situation can be definitely improved.

Interview 4:

- **Current role of the interviewee:** Solution Manager
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** >10
- **Number of project experience:** >6
- **Interview date:** February 16, 2015
- **Interview duration:** 60 minutes
- **Answer of Question 1:** I think the notations, especially the process notations are easy to understand. I would like to have an additional notation to attach detailed workflows at certain workstations as the standard manual operation processes. Process templates are important to extract action instances with function points and details.
- **Answer of Question 2:** It is easy for me. The adoption of the approach in practice requires key users to give training.

- **Answer of Question 3:** Yes. It depends on the communication purposes and targeting people. For ERP systems, customers only have ideas. For MES, it requires to communicate detailed workflows.
- **Answer of Question 4:** It is not possible to avoid natural derivation steps, and it is not possible to replace all manual activities with fully automated derivation. The approach can speed up the "performance". The quality of tool support is critical.
- **Answer of Question 5:** The tool should tolerance some configuration mistakes. It is related to IT reuse, not related to requirements. It also depends on the "quality" of family engineering results.
- **Answer of Question 6:** Correctness is not important for requirement reuse, but important for IT reuse.
- **Answer of Question 7:** Model templates are very helpful, but the number of modularized process steps is critical to the configuration. Currently in practice, software development life-cycle is very long. The budget, time-line and resources are pre-planned. Customers don't know what they exactly want; and stakeholders come from different background. The approach supports the generation of "presentable" software systems to customers, even before "payment". The approach simplifies the operations.
- **Answer of Question 8:** Tool maturity is very important to achieve. The coverage of core assets from family engineering should be good enough.
- **Answer of Question 9:** The potential benefit is on "money", which is basically saving. It is important to have a decision maker (from the management level) to support the technology adoption.
- **Answer of Question 10:** Risks when applying the tool in practice could be, for example, when facing change, the approach should provide certain flexibility. Using the tool(s) can improve the efficiency; but, without the tool, the whole development process should still work. It should not totally depend and rely on the tool.

Interview 5:

- **Current role of the interviewee:** Steering Manager
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** >20
- **Number of project experience:** >20
- **Interview date:** February 16, 2015
- **Interview duration:** 98 minutes

- **Answer of Question 1:** To standardize the notations is very helpful on improving the communication. The development team and the requirement team have different goals (when using the proposed notations). To ensure the willingness of using the new approach and notations is probably the first step.
- **Answer of Question 2:** (Not answered)
- **Answer of Question 3:** Yes. To achieve it, it is necessary to reconstruct the SPL to achieve standardization and modularity.
- **Answer of Question 4:** Yes. But it requires the maturity of the tool chain. The generated lines of code doesn't directly reflect to effort saving. Sometimes, writing 100 lines of critical code can take longer than 1000 lines.
- **Answer of Question 5:** It depends on the purposes (on the individual profits). For reuse, the notations should be limited. To react to changes or IT data processing, it would be nice to give certain "buffers".
- **Answer of Question 6:** (Not answered)
- **Answer of Question 7:** The process templates, but it requires high degree of process standardization.
- **Answer of Question 8:** The sub-domains in MES can be different. Not every of the sub-domains can be standardized. Logistic is difficult to achieve standardization, because of human participants and manual activities. Machining is completely possible. Pure warehouse is maybe also possible.
- **Answer of Question 9:** (Not answered)
- **Answer of Question 10:** It is difficult to test the logic behind the tool. Understandability of generated source code can be decreased, when applying the approach. The approach cannot react to change very well. It is applied only once at the starting phase. For example, database cannot be removed and recreated with the current implementation.

Interview 6:

- **Current role of the interviewee:** Requirement Engineer
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** 6-10
- **Number of project experience:** >6
- **Interview date:** February 16, 2015
- **Interview duration:** 50 minutes
- **Answer of Question 1:** The adoption of such an approach would anyway takes some time. The scalability of the size of the domain-specific process model can be challenging, if we want to apply it in practice. (Too many actions in processes.)

- **Answer of Question 2:** There should not be any problem to learn it.
- **Answer of Question 3:** Yes.
- **Answer of Question 4:** When the number of available process steps increases to very high, the usability decreases. It is necessary to provide an index of processes and structure the reusable actions for this purpose.
- **Answer of Question 5:** (Not answered)
- **Answer of Question 6:** The tooling should include a correctness checking function. The more accurate, the better.
- **Answer of Question 7:** The benefits of using the approach come with the improved efficiency during process specification and requirement communication.
- **Answer of Question 8:** (Not answered)
- **Answer of Question 9:** (Not answered)
- **Answer of Question 10:** The adoption of the approach requires training, especially for new team members. The adoption should be initialized by the management.

Interview 7:

- **Current role of the interviewee:** Requirement Manager
- **Location of the interview:** Shenyang, China
- **Organization:** BMW
- **Years of working experience:** 6-10
- **Number of project experience:** >6
- **Interview date:** February 16, 2015
- **Interview duration:** 40 minutes
- **Answer of Question 1:** Topology is not important, should be removed from the approach. And it is important to formulate the interaction of material flow and information flow, the interchange points.
- **Answer of Question 2:** The development of derivation infrastructure should involve domain experts more.
- **Answer of Question 3:** (Not answered)
- **Answer of Question 4:** Efficiency can be improved, but it is not the most important consideration. Spending longer time on requirements may finally speed up the projects and avoid problems later on.
- **Answer of Question 5:** (Not answered)
- **Answer of Question 6:** (Not answered)
- **Answer of Question 7:** (Not answered)

- **Answer of Question 8:** The prototype of processes and workflows would be better to allow the description of graphical user interface. It would be nice to add an interaction diagram, e.g. UML sequence diagram, to describe data exchange between client and server. Furthermore, the integration to project management, for example, to assign specification tasks to team members or monitor work progress, can be very helpful, when using such a tool in practice.
- **Answer of Question 9:** (Not answered)
- **Answer of Question 10:** To understand and define process actions is critical to the success of the development of such derivation infrastructure and tool support.

Interview 8:

- **Current role of the interviewee:** Architect
- **Location of the interview:** Munich, Germany
- **Organization:** Siemens
- **Years of working experience:** >10
- **Number of project experience:** 1-2
- **Interview date:** January 7, 2016
- **Interview duration:** 107 minutes
- **Answer of Question 1:** The approach can be understand with some software product line engineering experiences. It is not so easy to understand from completely scratch.
- **Answer of Question 2:** It is easy to learn for technical people, but not easy for non-technical people.
- **Answer of Question 3:** Yes, it is possible to improve the satisfaction.
- **Answer of Question 4:** It is skeptical. It needs a very mature product line. Maybe the value brought by such an approach is more on communication efficiency. Improvement sometimes is not the core of solution to customers.
- **Answer of Question 5:** The idea of approach is flexible. I can imagine integrating other domain-specific models, for example to be used in rail and train domain.
- **Answer of Question 6:** For the narrow domain, yes. Adding DSMLs is helpful on improving the accuracy and correctness, compared to the models in other modeling tools, such as Visio, I am not sure.
- **Answer of Question 7:** The approach enables the creation of high-level models early. The expectation of an early requirement tooling is different from the development tooling at later phase. These should be clearly separated.
- **Answer of Question 8:** (It has been mentioned in the answer to Question 4.) SPLE approaches usually cannot pay-off.

- **Answer of Question 9:** (Not directly answered. It has been mentioned in the previous questions.)
- **Answer of Question 10:** To find the proper domain scope, the shown DSMLs are domain-specific for warehouse domain. If the scope is too small, the approach is not reusable to other MES sub-domains. And if the scope is too big, investment (to the family engineering) is too high. It is important to make a careful decision to invest on such SPLE solution. Saving effort is generally too difficult to tell. Return on investment is usually difficult to get (based on the past SPLE projects). Every user should feel like beneficial. Developing code generation tooling is very expensive.

Interview 9:

- **Current role of the interviewee:** Architect
- **Location of the interview:** Munich, Germany
- **Organization:** Siemens
- **Years of working experience:** 6-10
- **Number of project experience:** 3-5
- **Interview date:** January 7, 2016
- **Interview duration:** 90 minutes
- **Answer of Question 1:** Yes. It is very clear to understand the need of having such an approach. Understanding the impact of a particular configuration is not easy. This approach can improve on this situation.
- **Answer of Question 2:** It is not easy for non-technical users to learn the meta-models. Both domain and language experts are often needed to develop the DSMLs.
- **Answer of Question 3:** The improvement of satisfaction comes from the "end-results" as models are directly reusable for down-stream activities. I think the stakeholders will be more satisfied, compared to other drawing tools, e.g. Visio.
- **Answer of Question 4:** It is very clear for development and derivation. For variability representation, the variability modeling is more expressive.
- **Answer of Question 5:** It is flexible to extend the approach by integrating other domain-specific variability types.
- **Answer of Question 6:** Both of the models and code should be checked to avoid errors during configuration and derivation, which is not included in the tooling yet.
- **Answer of Question 7:** The approach helps on improving the traceability from models to code. It can reduce the number of errors. With the approach it is possible to control over the configuration and the end products.

- **Answer of Question 8:** It is flexible and has quite high potential to generalize to other domains.
- **Answer of Question 9:** (It is mentioned already in the answer questions 6.)
- **Answer of Question 10:** In general, the adoption of the approach is cumbersome, because of the underlining tooling,"MagicDraw". There are other modeling options, such as Enterprise Architect or Visio. When the targeted users of the approaches have already other modeling tools, it is difficult to let them change. It would be helpful to do the adoption "step-by-step". It changes people's way of working. Training cost would be high. People should have the confidence on the generated code.

Interview 10:

- **Current role of the interviewee:** Architect
- **Location of the interview:** Erlangen, Germany
- **Organization:** Siemens
- **Years of working experience:** >10
- **Number of project experience:** >6
- **Interview date:** April 10, 2016
- **Interview duration:** 120 minutes
- **Answer of Question 1:** Yes, with SPLE experiences it is easy to understand the approach. For normal engineers without the experiences, some training is needed.
- **Answer of Question 2:** It requires deep learning curve to reach full adoption. The modeling tasks require certain qualification, not for every one.
- **Answer of Question 3:** At initial phase, the answer is yes. The satisfaction could be improved. With the time, more questions and more requirements may raise to enhance the tooling. The tooling requires incremental development and improvement.
- **Answer of Question 4:** For modeling, the answer is "yes". It is very efficient. For development, after several iterations, maybe 5 times, till the architecture gets more mature, the efficiency can be better. The code templates cannot support debug. In this case, it is very important to have the acceptance and quality of generated code.
- **Answer of Question 5:** It depends on the target domain and how mature the domain is. With the shown one, can be sufficient.
- **Answer of Question 6:** For the model, yes, the approach helps on improving the model correctness. The code correctness is not checked and guaranteed by the current tooling. The approach helps on improving the self-confidence to the outcome.

- **Answer of Question 7:** The approach can be very helpful at the pre-sale phase during requirement negotiation.
- **Answer of Question 8:** For the domain-specific modeling, the adoption should try and start small to model the textual requirements in documents. Make the approach agile. Fit to the team and stakeholders' need.
- **Answer of Question 9:** The approach has high potential to be generalized to other domains, such as power generation, train, or building automation.
- **Answer of Question 10:** Using such an approach has the prerequisite to the flexibility of the domains, maturity of the domains, and experts during domain engineering. The difficulty of MDD and DSMLs, is also that when to stop modeling.

C
Appendix: List of Abbreviations

MADE	Multi-vAriability Derivation
SPLE	Software Produce Line Engineering
SPL	Software Produce Line
IAM	Industrial Automation Management
FE	Family Engineering
AE	Application Engineering
DSML(s)	Domain-Specific Modeling Language(s)
DSM	Domain-Specific Model(s)
DSL	Domain-Specific Language(s)
MOF	Meta-Object Facility
UML	Unified Modeling Language
BPMN	Business Process Modeling Notations
SLR	Systematic Literature Review
GQM	Goal, Question, Metrics
MDD	Model-Driven Development
ERP	Enterprise Resource Planning
QA	Quality Assessment
SRG	Sub-Research Goal
RQ	Research Question
WMS	Warehouse Management System(s)
BPMN	Business process modeling notations
LOC	Lines of Code
OCL	Object Constraint Language
RFID	Radio-frequency identification

Curriculum Vitae

Miao Fang

Nationality: Chinese

Place of Birth: Beijing, China

EDUCATION

Double European Master's Degree in Software Engineering	
University of Kaiserslautern, Kaiserslautern, Germany	08/2010 - 07/2012
Blekinge Institute of Technology, Karlskrona, Sweden	08/2009 - 07/2010
Bachelor's Degree in Software Engineering	
Beijing University of Technology, Beijing, China	09/2002 - 06/2006

WORK EXPERIENCE

Software Development	
DATEV, Nürnberg, Germany	04/2019 - Present
Research and Development	
Cosmino AG, Nürnberg, Germany	08/2016 - 03/2019
PhD Research	
Siemens, Corporate Technology, Erlangen, Germany	08/2012 - 04/2016
Master Thesis	
Siemens, Corporate Technology, Erlangen, Germany	12/2011 - 07/2012
Internship	
Siemens, Corporate Technology, Munich, Germany	05/2011 - 11/2011
Software Design	
Ivar Jacobson International, Beijing, China	11/2005 - 07/2009