Location of Features as Model Fragments and their Co-Evolution

Jaime Font Burdeus

September 18, 2017

Thesis submitted for the degree of Philosophiæ Doctor

Series of dissertations submitted to the
Faculty of Mathematics and Natural Sciences, University of Oslo
No. 1928

ISSN 1501-7710

Cover: Hanne Baadsgaard Utigard.
Print production: Reprosentralen, University of Oslo.

Abstract

SOFTWARE Product Lines (SPLs) exploit commonalities across a family of related products in order to increase quality and reduce time to market and costs. Most SPLs are built from a set of existing products, that needs to be re-engineered into reusable assets following feature location approaches. Traditional feature location approaches target program code, but less attention has been paid in the literature to other software artifacts such as the models.

In this dissertation we present an approach for Feature Location in Models that relies on an Evolutionary Algorithm (FLiMEA). FLiMEA capitalizes on experts domain knowledge to boost the feature location process and produce model fragments that properly capture the reusable units of the domain. The approach performs a search (guided by a fitness function) over alternative model fragment realizations of the feature being located (generated through genetic operations). As a result, variability and commonalities are formalized in the form of reusable model fragments. We have explored different genetic operations and fitness functions so the approach can be tailored to work under the different conditions present in industrial scenarios.

In addition, when the features have been located and formalized as reusable assets, there is a need for evolution of those elements. In this dissertation we focus on the co-evolution of the model fragments and the language used to create them. To address this challenge we propose Variable MetaModel (VMM), an approach that relies on variability modeling ideas applied at metamodel level to enable the co-evolution. The VMM expresses each evolution of the language in terms of commonalities and variabilities, to ensure the conformance of model fragments with the new version of the language.

The approaches have been validated and evaluated in our industrial partners (BSH, the biggest manufacturer of home appliances in Europe, and CAF, an international provider of railway solutions).

ACKNOWLEDGEMENTS

The work presented in this thesis were carried out at the Department of Informatics, University of Oslo, Norway, and the School of Architecture and Technology, Universidad San Jorge, Spain, during the period 2013-2017. Universidad San Jorge provided me with a research fellowship, for which I am deeply grateful, thank you Pedro Larraz and Luis Correas.

First of all I would like to thank my two supervisors Øystein Haugen and Carlos Cetina for guiding me in this *adventure*. Thank you for sharing your deep knowledge and experience with me, while giving me the freedom to follow my own *paths*. Thank you for keeping me motivated enough to become the researcher I needed to be to succeed in this *journey*.

Thanks to my parents for all the love and patience. Thank you for providing me with the best *weapons* for facing life: an education to overcome any challenge I may face and your continuous example of effort and perseverance (that still applies today). Both have been critical to *gain the experience needed to level up* and enabled me to *kill all the monsters* I have encountered so far. Thanks to my brother, life has been easier *playing in your party* and I have learnt a lot from you (even if you were unable to *win me at Tekken III* during our childhood).

Thanks to the SVIT Research Group and all the people from the School for all the time we have shared inside and outside the lab. Thank you, for giving me the opportunity of *raising my skills* and *prove my value as a hero* in the degree of Design and Development of *Videogames*. Thank you for all the calls for whatever, for all the paper parties and for the University Thursdays.

Special thanks to Lorena Arcega for her endless support during these years, specially during that cold winter in Norwegian lands. We have become a good team and together we have *cleared all the levels*. As you already know, "I am really good at videogames", but playing together allowed us to reach a *Hi-Score*.

Thanks to the people from BSH, thank you for giving me the opportunity to put into practice the approach and for sharing all your domain knowledge with us. I would also like to thank the people from CAF, having new challenges from your industrial scenarios helped in further developing the approach.

Thanks to my friends in Zaragoza and Huesca. Thank you for taking me out and helping me to *collect all the coins*. Thank you for celebrating my successes, thank you for encouraging me and for always being there, even if you did not see me a lot when work kept me busy and thank you for those evenings of *Easter Eggs*.

Finally, thanks to all the people I have meet in the conferences I attended. Thank you for your constructive criticism that has helped in improving the work. Thanks to the anonymous reviewers for all the excellent feedback provided and special thanks to the members of the adjudication committee.

Bilbo – *"Can you promise that I will come back?"*
Gandalf – *"No. And if you do, you will not be the same."*

The Hobbit: An Unexpected Journey

Jaime Font Burdeus
September 2017

CONTENTS

List of Figures

Part I

Introduction

1 INTRODUCTION

Contents

1.1 Motivation of the Dissertation

Software Product Lines (SPLs) aim at reducing development cost and time to market while improving quality of software systems by exploiting commonalities and managing variabilities across a set of software applications [1]. The SPL engineering paradigm separates two processes; domain engineering (where the commonalities are identified and realized as reusable assets) and application engineering (where specific software products are derived by reusing the variability of the SPL) [2]. Traditionally, a domain analysis is performed to build a feature model that captures the variability of the system in terms of features [3, 4]. The domain knowledge from the experts is captured and used to build the library of reusable assets.

A recent survey [5] reveals that most of the SPLs are built when there are already products; therefore, the set of existing products is re-engineered into an SPL [6]. This is known as the extractive approach to SPLs [6]; it capitalizes on existing systems to initiate a product line, formalizing variability among a set of similar products into a variability model. The resulting SPL is capable of generating the products used as input (among others) with the benefit of having the variability among the products formalized, enabling a systematic reuse.

Feature Location (FL) is known as the process of finding the set of software artifacts that realize a particular feature, and it has gained attention during recent years [7, 8]. However, most of the research on FL targets program code [7, 8] as the software artifacts that realize the feature, neglecting other software artifacts such as the models. Manually spotting the commonalities and variability among the set of product models may become cumbersome and error prone [9], especially as the number of models and its complexity increases.

Therefore, we can apply FL to automate the identification and extraction of the features existing among a family of product models and re-engineering them into a model-based SPL (an SPL whose final products are models) by establishing precisely the variability between the features. However, the challenge of locating features among a set of product models, while capitalizing on expert domain knowledge, has not been fully addressed in the literature. In this work, we refer to this challenge as the **Feature Location in Models or FLiM** (see Figure 1.1). To address this challenge we propose an approach that turns a set of similar but different product models with no variability specification into a set of product models with a formal variability definition that specifies the commonalities and variability among them.

In our work, features located over product models are formalized as model fragments, the subset of model elements (from a whole product model) that realize a particular feature. Therefore, the outcome of addressing the FLiM challenge is a model-based SPL (where the features are realized in the form of model fragments). However, those model fragments have to be evolved over time (to cope with changing requirements, enhancements or other events), which results in the second challenge addressed by this dissertation, the **evolution of the model fragments** (see Figure 1.1). To address this challenge we propose an approach that relies on variability management ideas applied at metamodel level to enable the co-evolution of the model fragments while at the same time enables the evolution of the language used to create the model fragments.

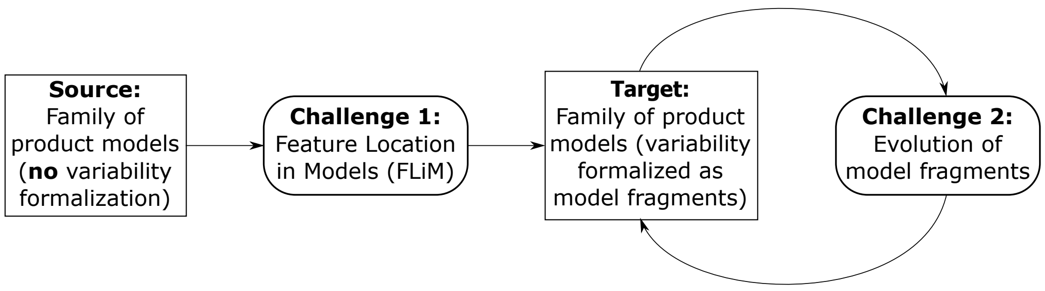

Source: Family of product models (**no** variability formalization)
Challenge 1: Feature Location in Models (FLiM)
Target: Family of product models (variability formalized as model fragments)
Challenge 2: Evolution of model fragments

Figure 1.1: Motivation of the Dissertation

1.2 Problem Statement

The extractive approach for building SPLs from products is being widely used in the industry [5]. However, there is a need for approaches that target models as the feature realization artifacts. In addition, evolving the features extracted in the form of model fragments is a must in industrial scenarios and needs to be properly addressed in order to have model-based SPL's approaches adopted by the industry. In this dissertation we move towards this direction addressing three Research Questions related to these challenges:

Research Question 1: How to identify and formalize the variability present among a set of product models in terms of features realized by model fragments?

Research Question 2: How to capitalize on expert domain knowledge to boost the process of feature location?

Research Question 3: How to co-evolve the model fragments that capture the features and the language used to create them?

1.3 Contribution

To address the Research Question 1, we present FLiMEA [10, 11, 12, 13, 14] (see Chapter 11): a software engineering approach for Feature Location in Models that relies on an Evolutionary Algorithm to locate features in product models and formalize them as model fragments. The FLiMEA performs a search (guided by a fitness function based on model fragment occurrences) over alternative model fragment realizations for the feature being located (generated through genetic operations).

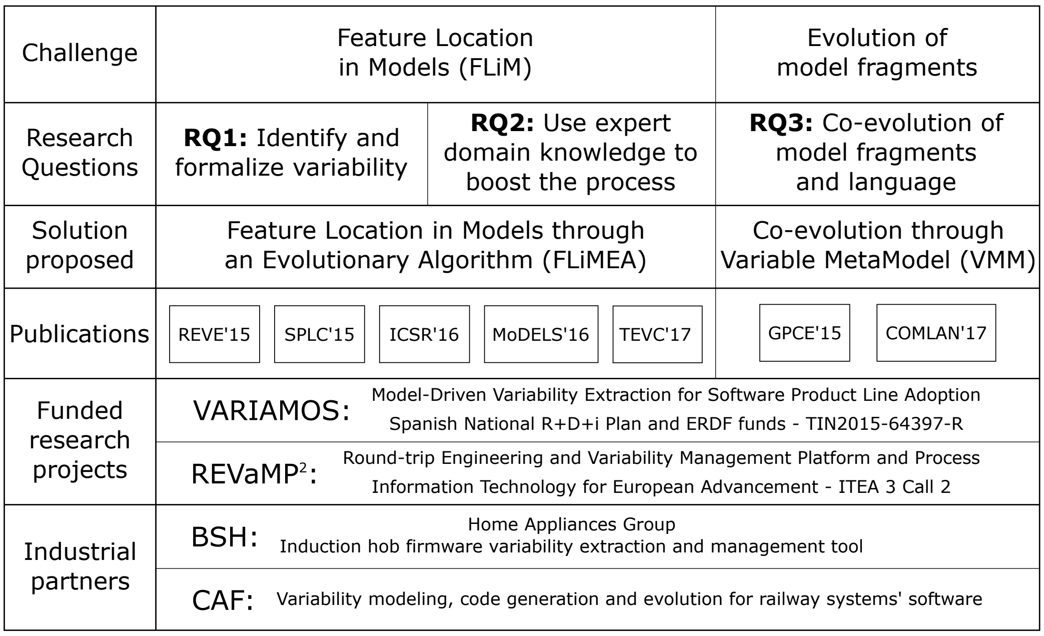
In response to the Research Question 2, FLiMEA can be tailored to work under different domains [11, 12, 13] (see Chapter 11). Particularly, FLiMEA provides different ways of embedding the domain knowledge from the engineers depending on the nature of the family of models and the type of information available. We added support to describe the feature to be located using natural language. Specifically, we have augmented FLiMEA with new genetic operations and fitness functions able to work with domain knowledge.

In response to the Research Question 3, we present the Variable MetaModel (VMM) [15, 16] (see Chapter 12, an approach for co-evolving the model fragments realizing the features and the language of the models. The VMM applies variability modeling ideas to express each evolution of the language in terms of commonalities and variabilities, ensuring the conformance of all model fragments (old fragments and new fragments) with the VMM.

In addition, we have evaluated the presented contributions with our industrial partners, applying them to industrial product models and using the domain knowledge from their domain experts. The contributions have been developed under National and International research projects aligned with the research performed in this dissertation. The contributions have been shared with the community in the form of conference and journal peer-reviewed publications. Finally, we have identified some challenges that remain unaddressed in this dissertation and that constitute our ongoing research.

1.4 Overview of the Work

Figure 1.2 shows an overview of the work performed as part of this dissertation. It is structured into size different rows: (row 1) identifies the challenge that is addressed; (row 2) shows the research questions about the challenge; (row 3) shows the solution proposed in this dissertation; (row 4) lists the scientific publications generated; (row 5) lists the research projects where the work has been contributed to; (row 6) lists the industrial partners where the solutions has been matured and evaluated.



Challenge	Feature Location in Models (FLiM)		Evolution of model fragments
Research Questions	**RQ1:** Identify and formalize variability	**RQ2:** Use expert domain knowledge to boost the process	**RQ3:** Co-evolution of model fragments and language
Solution proposed	Feature Location in Models through an Evolutionary Algorithm (FLiMEA)		Co-evolution through Variable MetaModel (VMM)
Publications	REVE'15, SPLC'15, ICSR'16, MoDELS'16, TEVC'17		GPCE'15, COMLAN'17
Funded research projects	VARIAMOS: Model-Driven Variability Extraction for Software Product Line Adoption. Spanish National R+D+i Plan and ERDF funds - TIN2015-64397-R		
	REVaMP^2: Round-trip Engineering and Variability Management Platform and Process. Information Technology for European Advancement - ITEA 3 Call 2		
Industrial partners	BSH: Home Appliances Group. Induction hob firmware variability extraction and management tool		
	CAF: Variability modeling, code generation and evolution for railway systems' software		

Figure 1.2: Overview of the work performed as part of the dissertation

For the first challenge (FLiM), two research questions are identified (RQ1 and RQ2), FLiMEA is proposed as our solution and five publications are presented in chronological order (REVE'15 [10], SPLC'15 [11], ICSR'16 [12], MoDELS'16 [13] and TEVC'17 [14]).

For the second challenge (Co-Evolution of model fragments and Language), one research question is identified (RQ3), VMM is proposed as our solution and two publications are presented in chronological order (GPCE'15[15], COMLAN'16 [15]).

There are two projects where the work presented in this dissertation was contributed: (VARIAMOS) a Spanish national research project whose objective is the extraction of variability in the form of model fragments to achieve the adoption of SPL approaches; (REVaMP[2]) an international ITEA 3 Call 2 project whose main objective is the creation of a holistic platform and process for variability extraction and management over time.

There are two industrial partners where the work presented in this dissertation was evaluated: (BSH) the leading manufacturer of home appliances in Europe, we have collaborated in the creation of a variability extraction and management tool for the induction hobs firmware; (CAF) a worldwide provider of railway solutions, we have collaborated in the creation of a solution for managing the variability of the software existing in the railway systems.

1.5 Research Methodology

In order to perform the work of this dissertation, we have applied a research project following the design science research methodology for performing research in information systems as described by [17] and [18]. Design research involves the analysis of the use and performance of designed artifacts to understand, explain and, very frequently, to improve the behaviour of aspects of Information Systems [18].

The design science research cycle consists of a five-phase process:

1 - Awareness: An awareness of an interesting research problem may come from multiple sources including new developments in industry or in a reference discipline. The output is a proposal for a new research effort.

2 - Suggestion: The suggestion phase follows the proposal and consists of the suggestion of a solution to the problem, and a comparison of this solution with already existing solutions. The output is a tentative design.

3 - Development: The tentative design is further developed and implemented in this phase. The implementation need not to involve novelty beyond the state-of-practice for the given artifact; the novelty is primarily in the design, not in the construction of the artifact. The output is the developed artifact.

4 - Evaluation: The artifact is evaluated according to criteria that are implicit. This phase includes a sub-phase in which hypotheses are made about the behaviour of the artifact. Then, deviations from expectations are gathered and the additional information gained in the construction and running of the artifact is used to another round of suggestions.

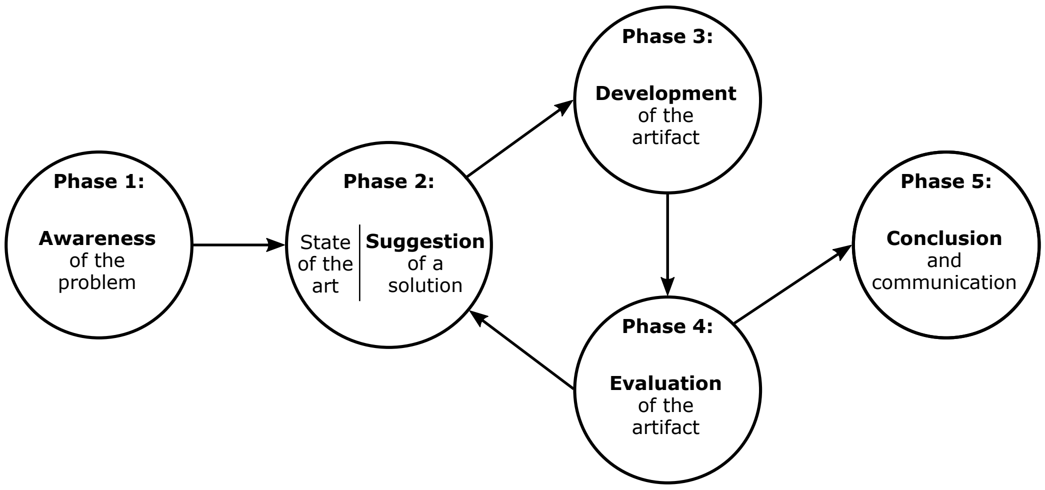


Figure 1.3: Research methodology followed in this Dissertation.

5 - Conclusion: This phase is the end of the research cycle, and is typically the result of an evaluation phase that is considered “good enough”. The results of the efforts are consolidated and communicated.

The design cycle is an iterative process; knowledge produced in the process by developing and evaluating artifacts is used as input for a better suggestion towards the solution of the problem. In this dissertation we have applied the cycle two times, one for each of the challenges identified.

Following the cycle defined in the design science research methodology, we started with the awareness of the problem (see Figure 1.3). In our case the awareness of the problem came from new developments for our industrial partners. We identified the problem to be resolved and we stated it as a proposal for a new research effort. Then, we performed the second phase, including the suggestion of a solution to the problem (see Sections 3.3.4 and 3.2.4) and its comparison with already existing solutions (see Chapter 3).

Next, we performed the third phase, further developing the tentative design and implementing it (see Chapters 5, 6 and 8). Then, we evaluated the artifacts as part of the fourth phase and extracted some conclusions as part of phase five (see Chapters 7, 9 and 10).

1.6 Quick Reference

Figure 1.4 shows a quick reference about the scope of the work done as part of this dissertation. It has been divided in order to establish clearly what elements constitute the background, what elements are part of the dissertation work and what elements are infrastructure for that work.

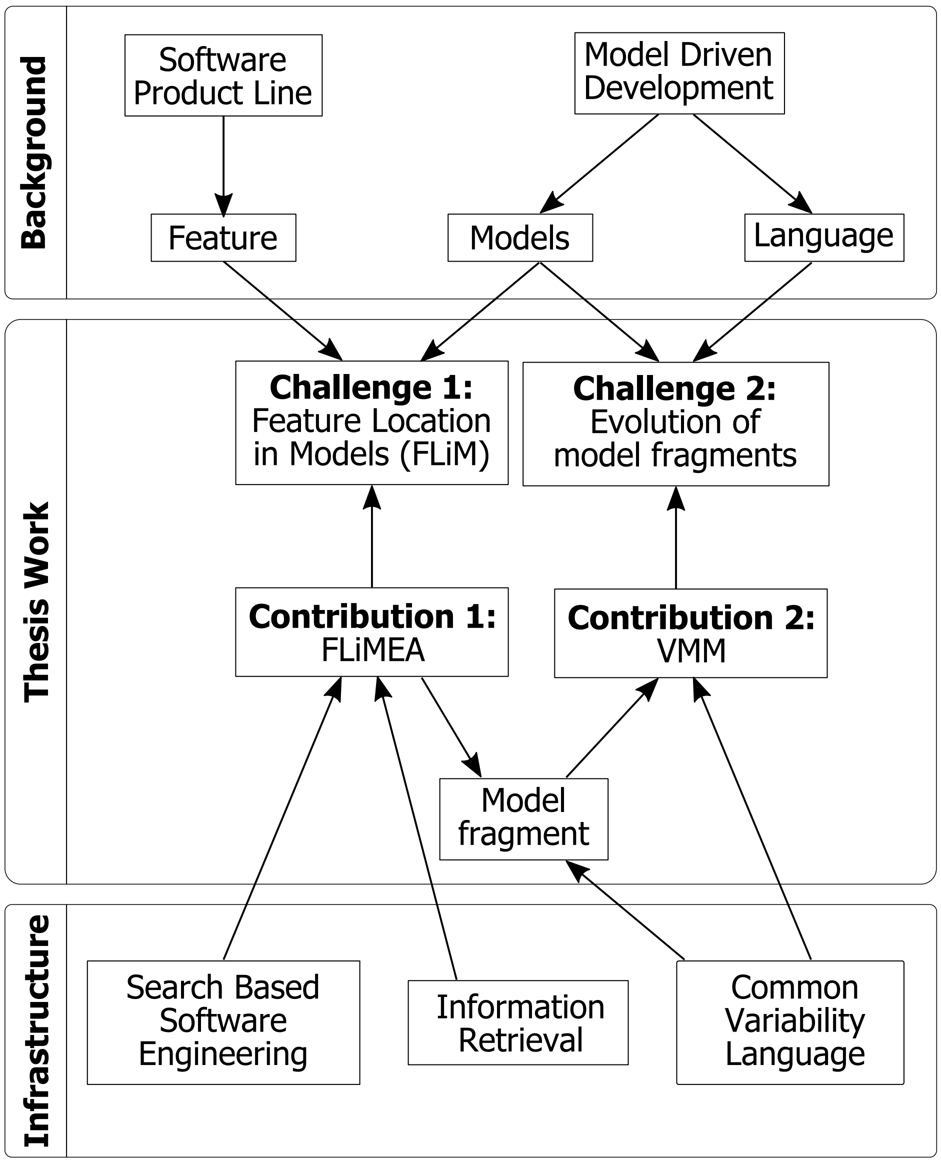


Figure 1.4: Cheat Sheet

1.7 Structure of the dissertation

This dissertation is structured into five parts:

Part I The first part is the introduction of the dissertation, later it presents some background and discusses the state of the art.

- **1 Introduction** This section introduces the motivation for the dissertation, the challenges that are addressed, the contribution, the overview of the work done, the methodology followed and the structure of the dissertation.
- **2 Background** This section presents some background related to the topics covered in the dissertation. Specifically, it presents Model Driven Development, SPLs and the Running Example extracted from one of our industrial partners that is used to illustrate the rest of the dissertation.
- **3 State of the Art** This section discusses the state of the art in relation to the two challenges addressed by this dissertation (FLiM and Co-evolution fo model fragments and Language) and motivates the two solutions presented (FLiMEA and VMM).

Part II The second part of the dissertation focuses on the Feature Location in Models (FLiM) challenge.

- **4 Feature Location in Models by an Evolutionary Algorithm (FLiMEA)** This chapter presents the overview of the Feature Location in Models by an Evolutionary Algorithm (FLiMEA), our approach to address the FLiM challenge.
- **5 FLiMEA as Model Fragments** This chapter presents the FLiMEA tailored to locate the features in the form of model fragments.
- **6 FLiMEA as Variation Points** This chapter presents the FLiMEA tailored to locate the features in the form of variation points.
- **7 Evaluation of FLiMEA** This chapter presents the details of the evaluations performed to validate the FLiMEA approach. It introduces our industrial partners' models where the features are located, explains how the results are measured and compared with an oracle and presents the results of the evaluations performed for each of the different configurations of the FLiMEA approach.

Part III The third part of the dissertation focuses on the challenge of the evolution of model fragments.

8 Variable MetaModel (VMM) This chapter presents the details of Variable metaModel (VMM), our approach to address the co-evolution of model fragments and language, including the different operations that compose it.

9 Evaluation of VMM This section presents the evaluation performed over the VMM approach, the results obtained and a set of lessons learned from its application on our industrial partner.

Part IV The fourth part of the dissertation presents the conclusion.

10 Conclusion This chapter includes the conclusion, the recapitulation of the research questions presented and their answers, the next steps in the research and the concluding remarks.

Part V The fifth part of the dissertation includes the seven papers selected for the dissertation.

11 Feature Location in Models Includes the five papers published in relation to the FLiM challenge.

12 Evolution of Model Fragments Includes the two papers published in relation to the evolution of model fragments challenge.

2

BACKGROUND

Contents

2.1 Overview of the Chapter

In this chapter the background of the dissertation is introduced. The background in this case is conformed by the approaches that are related to the objectives of this work: (1) locate the features existing among a set of similar but different product models; (2) enable the co-evolution of the features (realized as model fragments) and the language used by the model fragments. Therefore, this chapter provides a basic background for understanding the overall dissertation work. Specifically, we present Model Driven Development (MDD), Software Product Lines (SPLs) and the Running Example that will be used to illustrate the approaches included in the dissertation.

First, we present **Model Driven Development**, which is a paradigm where we can construct a model of a software system that we can then transform into the real thing. The goal of this paradigm is to automatically translate an abstract specification of the system into a fully functional software product.

Second, we present **Software Product Lines** engineering, which intends to produce a set of products that share a common set of assets in an specific domain. These techniques allow to adapt a product to the needs of the customer while its production costs and time to market are decreased. SPL promotes the shift from the development of stand-alone systems to the development of a family of systems.

Finally, we present our **Running Example** extracted from one of our industrial partners, BSH. We introduce the Common Variability Language and how it is applied to the models from our industrial partner in order to specify and manage the features as model fragments. Model fragments are central to this dissertation as it is the means used to formalize the features. Our FLiMEA approach (see Part II) locates features in the form of model fragments. Our VMM approach (see Part III) enables the co-evolution of the model fragments and the language used by them.

2.2 Model Driven Development

Model Driven Development (MDD) is a paradigm where models are central in the development. Model Driven Architecture (MDA) is a framework for software development proposed by the Object Management Group (OMG) in 2001 [19] (i.e., MDA is a concrete realization of MDD). The notion of Model Driven Engineering

(MDE) emerged later as a paradigm generalizing the MDA approach for software development [20].

2.2.1 Definition

The arrival of the MDD and MDA are changing the way of using models in the development of software. Model-driven is a paradigm where models are used to develop software. This process is driven by model specifications and by transformations among models. It is the ability to transform among different model representations that differentiates the use of models for sketching out a design from a more extensive model-driven software engineering process where models yield implementation artifacts. As stated by Agrawal et al. [21]:

> "the models are not merely artifacts of documentation, but living documents that are transformed into implementations. This view radically extends the current prevailing practice of using UML: UML is used for capturing some of the relevant aspects of the software, and some of the code (or its skeleton) is automatically generated, but the main bulk of the implementation is developed by hand. MDA, on the other hand, advocates the full application of models, in the entire life-cycle of the software product."

The goal of these approaches is to automatically translate an abstract specification of the system into a fully functional software product.

2.2.2 Model Driven Software Development Initiatives

Model-Driven Software Development (MDSD) is the notion that we can construct a model of a software system that can then be transformed into the real thing [22]. Models have been used for a long time in the software development field. From formal and executable specification languages (e.g., OBLOG [23], TROLL [24] or OASIS [25]), to the most accepted notations (like UML [26]) and processes (like RUP [27]) models are present in the software development area.

Stuart Kent [20] defines Model Driven Engineering (MDE) by extending MDA with the notion of software development process (that is, MDE emerged later as a generalization of the MDA for software development). MDE refers to the systematic use of models as primary engineering artifacts throughout the engineering lifecycle. Kurtev provides a discussion on existing MDE processes [28] (refer to [29, 30] for a specific approach). In general, these approaches introduce concepts, methods and tools [31]. All of them are based on the concept of model, meta-model, and model transformation.

Model Driven Architecture (MDA) is a concrete realization of MDD. MDA classifies models into two classes: Platform Independent Models (PIMs) and Platform Specific Models (PSMs) [19]. A PIM is a view of a system from a platform-independent viewpoint. Likewise, a PSM is a view of a system from a platform-dependent viewpoint [19]. Doing so, the definition of platform becomes fundamental.

Although the contribution of MDA has been critical, other initiatives under different descriptive terms have pushed in the direction of MDSD. These initiatives (or specific paradigms) highlight distinct aspects and/or follow specific strategies for applying MDSD. The following are remarkable examples of these initiatives.

Automatic programming: According to Balzer [32], who is considered the initiator of the modern automatic programming paradigm, automatic programming is based on the use of methods and tools which support the acquisition of high level of abstraction specifications, their validation and the generation of executable code. He was focused on the generation of efficient implementations, since the hardware resources (CPU power, memory size, etc.) were limited. Therefore, he proposes a semi-automated (interactive) translation approach which facilitates the specification of optimizations by human developers. It is important to note that he considers that the application of this paradigm to a narrower area (e.g., expert systems) allows an "attempt to eliminate the need for interactive translations".

Generative Programming: This paradigm was proposed by Czarnecki in his PhD Thesis [33] although the term was coined by Eisenecker in [34]. In Eisenecker words, Generative Programming "is a comprehensive software development paradigm to achieving high intentionality, reusability, and adaptability without the need to compromise the run-time performance and computing resources of the produced software". It is highly based on domain specific engineering and product line development, using techniques such as generic programming, domain-specific languages and aspect-oriented programming. Unlike other more general paradigms, Generative Programming suggests very specific techniques and steps for developing methods which follow this approach.

In general, MDSD initiatives promote a paradigm of reuse and automation. This emerges through the extensive use of models and model transformations, which replaces cumbersome (and usually repetitive) implementation activities. In this way, model-driven approaches improve development practices by accelerating them.

2.2.3 Domain Specific Languages

Domain specific languages play a key role in several of the MDSD approaches that have been presented above. According to [35], a domain specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power, focused on, and usually restricted to, a particular problem domain.

DSLs are not a new topic, but the current stress on MDSD has focused the interest of both academy and industry on this kind of languages. Examples of DSLs abound, including well-known and widely-used languages such as LATEX, YACC, Make, SQL, and HTML. As stated by [35], the older programming languages (Cobol, Fortran, Lisp) all came into existence as dedicated languages for solving problems in a certain area (respectively business processing, numeric computation and symbolic processing).

DSLs are tightly related to the Domain Engineering. In words of Tolvanen [36], the main focus of Domain Engineering is finding and extracting domain terminology, architecture and components. It is important to note that two points of view when dealing with the domain concept can be considered, as highlighted by Simos [37].

Conceptual domain: From this point of view, a domain is a set of interrelated real-world concepts. For instance, the health-care domain contains concepts such as medical center, patient, disease, medicament, etc. As another example, the industrial factory domain contains concepts such as stock, supplier, client, worker, etc.

Systems domain: From this point of view, a domain is characterized by a set of systems that share some common features [37]. These systems usually address a common problem area and conceivably share a common solution structure. In this case, we can talk about the expert systems domain, the database-based systems domain, the control/monitoring systems domain, the software games domain, etc.

Note that a software system can be seen as the combination of both a conceptual domain and a system domain. For instance, we can find expert systems for health-care and control/monitoring systems for industrial factories, but there are also expert systems for industrial factories and control/monitoring systems for health-care. Specific languages exist both for conceptual domains and systems domains.

Many benefits due to the use of DSLs can be found in the literature. For instance, according to [35].

- DSLs allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. Consequently, domain experts themselves can understand, validate, modify, and often even develop DSL programs.
- DSL programs are concise, self-documenting to a large extent, and can be reused for different purposes.
- DSLs enhance productivity, reliability, maintainability, and portability.
- DSLs can embody domain knowledge, and thus enable the conservation and reuse of this knowledge.
- DSLs allow validation and optimization at the domain level.

But some drawbacks have been also identified. These drawbacks are related to the associated costs (for designing, implementing and learning the DSL) and the specific nature of the language (possible lack of expressiveness and/or loss of efficiency).

Some researchers suggest that the success of visual notations as commonly used domain-specific languages is contingent on making similar tools and concepts for visual languages a commodity that can be readily used and understood by a wide audience, effectively lowering the initial hurdle to adoption [38]. Hopefully, the number and quality of tools for implementing DSLs is growing and, therefore, a wide use of DSLs is very probable.

2.3 Software Product Lines

Mass production was popularized by Henry Ford in the early 20th Century. McIlroy coined the term software mass production in 1968 [39]. It was the beginning of SPLs. In 1976, Parnas introduced the notion of software program families as a result of mass production [40]. The use of features (to drive mass production) was proposed by Kang in the early 1990s [4]. Shortly, the first conferences appeared turning SPL into a new body of research [41].

2.3.1 Definition

SPLs are defined as "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" [2, 42]. This definition can be redefined into five major issues:

1. **Products**. SPL shift the focus from single software system development to SPL development. The development processes are not intended to build one application, but a number of them (e.g., 10, 100, 10,000, or more). This forces a change in the engineering processes where a distinction between domain engineering and application engineering is introduced. Doing so, the construction of the reusable assets (platform) and their variability is separated from production of the product-line applications.

2. **Features**. Features are units (i.e., increments in application functionality) by which different products can be distinguished and defined within an SPL [43].

3. **Domain**. An SPL is created within the scope of a domain. A domain is a specialized body of knowledge, an area of expertise, or a collection of related functionality [44].

4. **Core Assets**. A core asset is an artifact or resource that is used in the production of more than one product in an SPL [2].

5. **Production Plan**. It states how each product is produced. The production plan is a description of how core assets are to be used to develop a product in a product line and specifies how to use the production plan to build the end product [45]. The production plan ties together all the reusable assets to assemble (and build) end products. Synthesis is a part of the production plan.

2.3.2 Software Product Line Processes

SPLs (or system families) provide a highly successful approach to strategic reuse of assets within an organization. A standard SPL consists of a product line architecture, a set of software components and a set of products. A product consists of a product architecture, derived from the product line architecture, a set of selected and configured product line components and product specific code.

Therefore, SPL engineering is about producing families of similar systems rather than the production of individual systems. SPL engineering consists of three main processes: domain engineering (also called core asset development), application engineering (also called product development) and management. These three processes are complementary and provide feedback to each other.

Domain Engineering is defined as "the activity of collecting, organizing and storing past experience in building systems or parts of systems in a particular domain in the form of reusable assets (e.g., architecture, "models, code, and so on), as well as providing an adequate means for reusing these assets (...) when building new systems" [3]. That is, Domain engineering is, among others, concerned with identifying the commonality and variability for the products in the product line and implementing the shared artefacts such that the commonality can be exploited while preserving the required variability.

Using a "design-for-reuse" approach, domain engineering (core asset development [2]) is on charge of determining the commonality and the variability among product family members. In general, domain engineering is divided into domain analysis, domain design and domain implementation.

Application Engineering is "the process of building a particular system in the domain" [3]. Application engineering (a.k.a., product Development [2]) is responsible for deriving a concrete product from the SPL using a "design-with reuse" approach. To achieve this, it reuses the reusable assets developed previously.

During application engineering, individual products are developed by selecting and configuring shared artifacts and, where necessary, adding product-specific extensions. This process is subdivided into application analysis, application design and application implementation.

Management is a separated process where organizational issues are handled specifically [2]. This process is responsible for giving resources, coordinating, and supervising domain and application engineering activities.

See [2, 1] for more details about the above processes. In SPL processes, variability is made explicit through variation points. A variation point represents a delayed design decision. When the architect or designer decides to delay the design decision, he or she has to design a variation point. The design of the variation point requires several steps: (1) the separation of the stable and variant behaviour, (2) the definition of an interface between these types of behaviour, (3) the design of a variant management mechanism and (4) the implementation of one or more variants. Given a variation point, it can be bound to a particular variant. For each variation point, the set of variants may be open, i.e. more variants can be added, or closed, i.e. no more variants can be added. Overall, during domain engineering new variation points are introduced, whereas during application engineering these variation points are bound to selected variants

Behind the SPL approach we can find the economies of scope principle. While economies of scale arise when multiple identical instances of a single design are produced collectively, economies of scale arise when multiple similar but distinct designs are produced collectively [46]. In this context, the same practices, processes, tools and materials are used to design and build similar unique products. This methodical reuse is responsible for an increase in productivity and quality.

2.4 Running Example

This section presents the Induction Hobs Domain, including the Domain Specific Language used by our industrial partner to specify their product models. It also presents how the Common Variability Language is applied to specify the variability among those product models. The language and graphical representations presented in this section will serve as the basis of the running example used to illustrate the rest of the dissertation.

2.4.1 The Induction Hobs Domain

Traditionally, stoves have a rectangular shape and feature four rounded areas that become hot when turned on. Therefore, the first Induction Hobs (IHs) created provided similar capabilities. However, the induction hobs domain is constantly evolving and, due to the possibilities provided by the induction phenomena and the electronic components present in the induction hobs, a new generation of IHs has emerged [1].

For instance, the newest IHs feature full cooking surfaces, where dynamic heating areas are automatically calculated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. There has been an increase in the type of feedback provided to the user while cooking, such as the exact temperature of the cookware, the temperature of the food being cooked, or even real-time measurements of the actual consumption of the IH. All of these changes are being possible at the cost of increasing the software complexity.

The Domain Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references among them and more than 180 properties. However, in order to gain legibility and due to intellectual property rights concerns, in this section we use a simplified subset of the IHDSL (see the top of Figure 2.1).

Inverters are in charge of converting the input electric supply to match the specific requirements of the induction hob. Specifically, the amplitude and frequency of the electric supply needs to be precisely modulated in order to improve the efficiency of the IH and to avoid resonance. Then, the energy is transferred to the hotplates through the channels. There can be several alternative channels, which enable different heating strategies depending on the cookware placed on top of the IH at runtime. The path followed by the energy through the channels is controlled by the power manager.

Inductors are the elements where energy is transformed into an electromagnetic field. Inductors are composed of a conductor that is usually wound into a coil. However, inductors vary in their shape and size, resulting in different power supply needs in order to achieve performance peaks. Inductors can be organized into groups in order to heat larger cookware while sharing the user interface controllers. Each group of inductors can have different particularities; for instance, some of them can be divided into independent zones or others can grow in size adapting to the size of the cookware being placed on top of them. Some of the groups of inductors are made at design time, while others can occur at runtime (depending on the cookware placed on top).

[1] freeInduction cooktop demo: https://www.youtube.com/watch?v=EZ8UAvt9paI

2.4.2 The Common Variability Language applied to Induction Hobs

The Common Variability Language (CVL) [47, 48, 49] was recommended for adoption as a standard by the Architectural Board of the Object Management Group and is our industrial partner's choice for specifying and resolving variability. CVL defines variants of a base model (conforming to MOF) by replacing variable parts of the base model by alternative model replacements found in a library model.

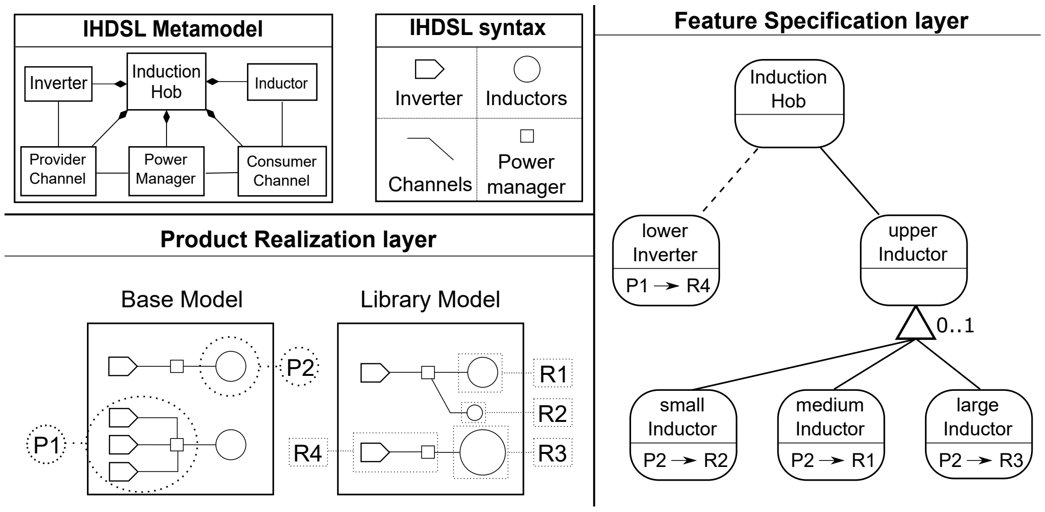


Figure 2.1: CVL applied to IHDSL

The variability specification through CVL is divided across two different layers: the feature specification layer (where variability can be specified following a feature model syntax) and the product realization layer (where variability specified in terms of features is linked to the actual models in terms of placements, replacements and substitutions).

The base model is a model described by a given DSL (here, IHDSL) which serves as a base for different variants defined over it. In CVL the elements of the base model that are subject to variations are the placement fragments (hereinafter placements). A placement can be any element or set of elements that is subject to variation. To define alternatives for a placement we use a replacement library, which is a model described in the same DSL as the base model that will serve as basis to define alternatives for a placement. Each one of the alternatives for a placement is a replacement fragment (hereinafter replacement). Similarly to placements, a replacement can be any element or set of elements that can be used as variation for a replacement.

CVL defines variants of the base model by means of fragment substitutions. Each substitution references to a placement and a replacement and includes the information necessary to substitute the placement by the replacement. In other words, each placement and replacement is defined along with its boundaries, which indicate what is inside or outside each fragment (placement or replacement) in terms of references among other elements of the model. Then, the substitution is defined with the information of how to link the boundaries of the placement with the boundaries of the replacement. When a substitution is materialized, the base model (with placements substituted by replacements) continues to conform to the same metamodel.

Figure 2.1 shows an example of variability specification of IH through CVL. In the product realization layer, two placements are defined over an IH base model (P1 and P2). Then, four replacements are defined over an IH library model (R1, R2, R3, and R4). In the feature specification layer, a Feature Model is defined that formalizes the variability among the IH based on the placements and replacements previously defined. For instance, P1 can only be substituted by R4 (which is optional), but P2 can be replaced by R1, R2, or R3. Note that each fragment has a signature, which is a set of references going from and towards that replacement. A placement can only be replaced by replacements that match the signature. For instance, the P2 signature has a reference from a power manager (outside the placement) to an inductor (inside the placement), while the R4 signature is a reference from a power manager (inside the replacement) to an inductor (outside the replacement). P2 cannot be substituted by R4 since their signatures do not match.

3

State of the Art

Contents

3.1 Overview of the Chapter

This chapter presents the state of the art for the two main challenges addressed in this dissertation: the Feature Location in Models (FLiM), and the evolution of model fragments. Both challenges are highly related, as the features located in the form of model fragments (first challenge) must be evolved and maintained over time (second challenge). However, the approaches proposed to address each of the challenges ground on different domains and therefore are presented separately. Next two sections focus on each of the challenges respectively.

3.2 Feature Location in Models

This section includes works from literature that are related to our Feature Location in Models following search based techniques. They are classified in three categories: (1) Feature Location (FL); (2) Search Based Software Engineering (SBSE); (3) Model Driven Engineering (MDE). Fig 3.1 shows an overview of the scope.

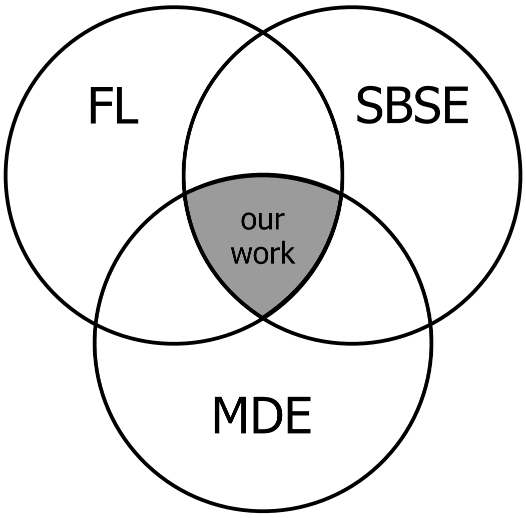


Figure 3.1: Overview of the scope of Feature Location in Models challenge

3.2.1 Feature Location

There are many feature location approaches that have been proposed to find relevant code for different tasks (e.g., maintenance) [8, 7]. The works from Feature Location that are related to this work can be divided into five categories: (1) Textual Similarity; (2) Trace Analysis; (3) Program Dependency Analysis; (4) Propositional Logic; (5) Type System (see Fig. 3.2).

Feature location techniques have been traditionally applied to the source code. According to the Extractive SPL Adoption catalog of case studies [50], more than three quarters of the case studies in the literature on the specific activity of feature location dealt with source code. In our work, the feature location is applied directly to the product models. In the mentioned catalog, including our Induction Hobs case study, the models only represents eight percent of the case studies in feature location.

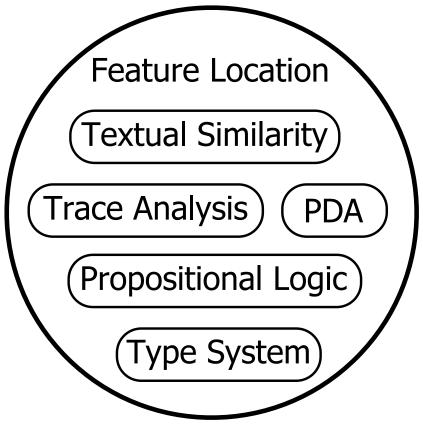


Figure 3.2: Overview of the scope of Feature Location in relation to FLiM challenge

Textual Similarity

Textual similarity techniques ground on mathematical and statistical methods to determine the similarity between different collections of texts. For instance, Latent Semantic Analysis (LSA) [51] takes into account the number of occurrences of a set of keywords (query) in large bodies of texts (documents). As a result, LSA can be used to determine the similarity between feature names or descriptions and the source code that realizes those features. Then the similarity between the feature description and the source code files can be represented in the form of vectors using Singular Value Decomposition (SVD) [52] and the Vector Space Model (VSM) [53].

For example, Marcus et al. [54] used IR techniques to map descriptions expressed in natural language (NL) to source code. Other approaches [55] apply the VSM to improve the results. Furthermore, some works combine the textual similarity techniques with dynamic analysis [56, 57, 58, 59]. Cavalcanti et al. [60] used IR techniques to assign change requests in software maintenance or evolution tasks based on context information. Kimmig et al. [61] proposed an approach for translating NL queries to concrete parameters of the Eclipse JDT code query engine.

Recently, several approaches have been proposed to improve the effectiveness of feature location. For example, Wang et al. [62] proposed a code search approach, which incorporates user feedback to refine the query. Hill et al. [63] proposed automatically extracting NL phrases to categorize them into a hierarchy in order to help developers to discriminate the relevance of results and to reformulate queries. Zou et al.[64] investigated the "answer style" of software questions with different interrogatives and proposed a re-ranking approach to refine search results.

Other approaches have been proposed to improve the effectiveness of feature location by getting information from public repositories [65] or expanding a user query with semantically similar words from websites [66]. For example, Dietrich et al. [67] improved the efficacy of future queries using feedback captured from a validated set of queries and traceability links. Lv et al. [68] enrich each API with its online documentation to match the query based on text similarity.

Trace Analysis

Trace Analysis is the main technique used at runtime to extract relevant information to build the variability model. When the system under study is executed, it generates traces that indicate which parts of the code have been executed. Usually, when a feature is exercised, the traces generated are compared with the traces when the feature is not executed to isolate the lines of code related to the feature.

Some approaches rely solely on trace analysis [69, 70, 71]. Other approaches combine the trace analysis with static analysis such as LSA [56, 57, 58, 59], PDA [72, 73] or VSM [74].

Program Dependency Analysis

Program Dependency Analysis (PDA) is a static analysis that takes advantage of the order of execution of each line of source code to establish restrictions among them. By doing so, the program can be represented as a Program Dependency

Graph (PDG) where the nodes are functions or global variables while the edges are calls to those functions or accesses to those variables.

PDA is central to feature location in source code and is used by multiple approaches [55, 75, 76, 77, 78, 79, 80]. Some approaches [72, 74, 73] combine PDA with other static analysis to improve the results.

Type System

Other works apply type systems to extract relevant information from the code. Typechef provides an infrastructure to analyse the #ifdef variability included in a C source code [81, 82, 83, 84, 85]. Typechef includes a variability-aware parser capable of parsing non pre-processed C code without applying heuristics (preserving the completeness of the results) in a reasonable time. Typechef [82] enables the extraction of information relevant for the formalization of the variability while detects compile-time errors. In [83] the authors extend Typechef to support variability defined across different modules, enabling the application of the approach to software ecosystems. In [84] the authors compare the application of heuristic-based strategies and Typechef. The comparison shows that Typechef outperforms many heuristic-based strategies while preserving the completeness of the results. In [81] type techniques are combined with textual analysis and PDA to perform feature location in source code. This work shows that the combination of different sources of information in the form of recommendation systems provides better results than its application separately.

Propositional Logic

Some works focus on building the feature model that represents the variability existing among a set of products, applying reverse engineering techniques [86, 87, 88]. In the one hand, there are works that propose to synthesise feature models applying logic formulas describing the dependencies among the features [88]. On the other hand, some works focus on extracting feature lists and descriptors to syntethize the feature models [87].

However, the combination of both techniques can produce better results. In [86] the authors combine the logic formulas and the feature list with descriptors extracted from the source code to obtain the hierarchy existing among the features of the feature model. Particularly, for each analysed feature, the approach proposes two lists of possible parents of the feature, enabling the user to make a decision without the need of analysing the whole list of features (which can grow over hundreds or thousands).

In [85] the authors propose an approach to extract constraints among the features based on the static analyses provided by Typechef [81, 82, 83]. The constraints are retrieved from the source code, parsing it with Typechef and analysing the errors produced and the conditions that raised them. To validate the approach, constraints retrieved are compared against trusted constraints obtained from the feature model of the system under study.

3.2.2 Search Based Software Engineering

The works from Search Based Software Engineering that are related to the Feature Location in Models can be divided into two categories: (1) Feature Model Configurations' Synthesis; (2) Feature Constraints Discovery (see Fig. 3.3).

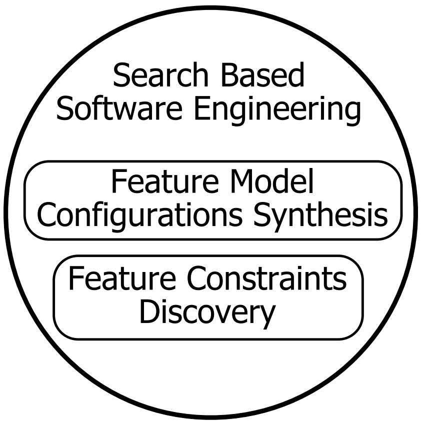


Figure 3.3: Overview of the scope of the Search Based Software Engineering in relation to FLiM challenge

Harman et al. [89] performed a survey on the topic of search-based software engineering applied to SPLs. They present an overview of recent articles classified according to themes such as configuration, testing, or architectural improvement. Lopez-Herrejon et al. [90] performed a preliminary systematic mapping study at the connection of search-based software engineering and SPL. They categorized the articles along a known framework for SPL development. These two surveys indicate that search-based software engineering techniques are being applied to SPLs. However, these surveys do not identify works that focus on finding model fragments that materialize the features of the SPL, as our work does.

Feature model Configurations' Synthesis

One common problem addressed by search-based software engineering related to SPLs is the synthesis of configurations from a feature model. Feature models can include constraints that must be fullfiled by the configurations of products obtained from them and search-based techniques can be applied to guarantee this.

White et al. [91] present an approach called Filtered Cartesian Flattening to create configurations from a feature model. The authors formulate the feature selection problem as a constrained single objective formulation and solve it applying Branch and Bound with Linear Programming (BBLP). The approach is evaluated on synthetic feature models of around 5000 features, suffering only a 7% loss of solution quality.

There are some research efforts that apply genetic algorithms to the SPLs domain. For instance, the authors in [92, 93] present GAFES, an approach for optimized feature selection in SPLs. The approach applies a repair operation to transform invalid configurations generated after crossover and then turn them into valid configurations of the feature model. They use a single objective for the optimization and report that their approach outperforms the Filtered Cartesian Flattening approach [91].

Sayyad et al. [94] provide a study of different metaheuristic algorithms for the multi-objective feature selection problem. Then the approach is further refined in [95] with a tuning of the parameters used by the genetic operations.

Wang and Pang [96] apply Ant Colony Optimization to the feature selection problem. The approach is compared to the Filtered Cartesian Flattening [91] and the GAFES approach [92, 93]. The authors report results balanced between the two compared approaches, achieveing a 6% less quality than the work from White et al. (but taking less time) and 10% better than GAFES (but taking more time).

Feature Constraints Discovery

Another common problem of SPLs that can be addressed by search-based techniques is the discovery of feature constraints among the features. Using these constraints, a feature model can be synthetized from a set of features and the constraints among them.

Chan et al. [97] address this problem applying a genetic programming approach that generates customer satisfaction models and their relationships. The application of the approach is illustrated with a digital camera SPL case study.

In [98, 99] the authors apply evolutionary algorithms to generate feature models from the sets of features that describe the product variants. They make use of repair operations to fix the invalid configurations generated by the crossover operation. The approach is extended in [100] to seek to learn feature models from instances applying genetic programming. Lopez-Herrejon et al. [101] evaluate three standard search-based techniques (evolutionary algorithm, hill climbing, and random search) in order to calculate the relationships of a feature model.

In addition, feature model generators are needed in order to evaluate those approaches. In the work of Segura et al. [102] they propose ETHOM, an approach to generate computationally hard feature models using an evolutionary algorithm. They apply it to search for feature models that fulfill some characteristics as the size or the number of constraints. The resulting feature models are considered 'hard' in the sense that analysing and processing them is computationally expensive in terms of time and memory.

3.2.3 Model Driven Engineering

The works from Model Driven Engineering that are related to the Feature Location in Models can be divided into three categories: (1) Feature Model Synthesis; (2) Mechanical Comparisons; (3) Manual guidelines (see Fig. 3.4). Model Driven Engineering techniques have been applied to locate features among product models using techniques based on model comparisons. In our work, we also take into account the domain knowledge and apply techniques based on this information such as textual based comparisons.

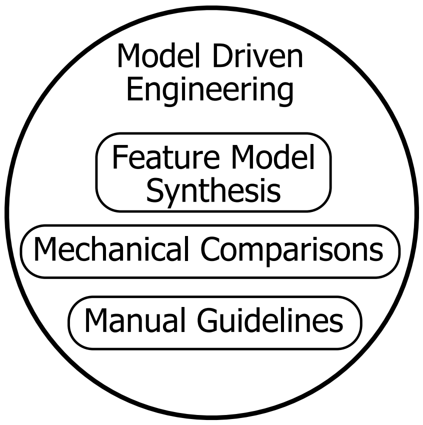


Figure 3.4: Overview of the scope of the Model Driven Engineering in relation to FLiM challenge

Feature Model Synthesis

Some research efforts focus on the source code of the products in order to extract the variability model. For instance, the authors in [86] present a tool-supported approach for reverse engineering feature models from different sources, such as makefiles, preprocessor declarations and documentation. They focus on the crucial point of identifying parents and combine logic formulas and descriptions as complementary sources of information. In addition, the authors in [103] propose an approach to identify features from the source code of products. They reduce the noise induced by spurious differences of various implementations of the same feature. Then, the process produces feature candidates that are manually pruned (to remove non-relevant candidates). The approach is further extended in [104] to introduce ExtractorPL, an automated technique that infers a full implementation of an SPL from the given code.

Manual Guideliness

There are several research efforts in existing literature towards the automation of the variability formalization among a set of products. However, most of them are focused on generating Feature Models (FMs) and not address CVL particularities. For instance, the authors in [9] present an approach to reverse engineering and evolve architectural FMs. In particular, they focus on plugin-based systems, projecting variability and technical constraints of plugin dependencies into an architectural FM. In [105], the authors present a reverse-engineering tool to extract variability data from web configurators and transform them into structured data (for instance, a feature model) in a semi-automated way. The tool incorporates a component that explores the configuration space simulating users' configuration actions in order to generate more variable data to be extracted.

Mechanical Comparisons

In [106], the authors propose the "CVL Compare process" a generic approach to automatically compare products and extract the variability among them in terms of a CVL variability model. The approach automatically turns identical elements into common parts of the product line, similar elements into alternative parts, and unmatched elements into optional parts. However, fully automating the decision of what should be reused and how it should be done reduces the flexibility of the approach. In [107], the approach is refined to automatically formalize the feature realizations of new product models that are added to the system.

In [108] the authors present an approach to mine family models from block-based models. The similarity between models is measured following an exchangeable metric, taking into account different attributes of the models and can be fine-tuned depending on the application. Then, the approach is further refined [109] to reduce the number of comparisons needed to mine the family model.

Martinez et al. [110] propose an extensible approach based on comparisons to extract the feature formalization over a family of models. In addition, they provide means to extend the approach to locate features over any kind of asset based on comparisons. The approach is further refined in [111], where the MoVaPL approach considers the identification of variability and commonality in model variants as well as the extraction of a Model-based SPL from the features identified on these variants. MoVaPL builds on a generic representation of models making it suitable for any MOF-based models.

Some works focus on transforming legacy products into Product Line assets. For instance, in [112], the authors present their experience in the Digital Audio & Video Domain. In [113], the authors explain their experience re-engineering the Image Memory Handler from Ricoh's products into an SPL. In [114], the authors report on their experience applying an extractive approach to a set-top box manufacturing company. These approaches extract variability from legacy products in industrial environments, but they focus on capturing guidelines and techniques for manual transformations. In contrast, our goal is to introduce automation into the process while taking advantage of the knowledge of the domain experts through a human-in-the-loop extractive approach.

In [115, 116], the authors propose the "merge-refactor" framework, a generic framework for mining legacy product lines and automating their refactor to contemporary feature-oriented SPLE approaches. They compare a set of UML variants with each other using the n-way merging [117], matching those whose similarity is above a certain threshold and merging them together. The focus in this work is the challenge of comparing and merging more than two model fragments at the same time.

However, all of these approaches are based on mechanical comparisons among the models, classifying the elements based on their similarity and identifying the dissimilar elements as the feature realizations. In contrast, our work does not rely on model comparisons to locate the features. Specifically, in our work, humans are involved in the search by means of an evolutionary algorithm. Domain experts and application engineers become part of the process, contributing their knowledge of the domain in order to tailor the approach with the feature description. The model fragments obtained mechanically are less recognizable by software engineers than those obtained with their participation [11].

3.2.4 Motivation of our Feature Location in Models Approach

Existing approaches for feature location in models presented above rely on mechanical comparisons to find model differences. First, several comparisons among the product models are performed. Then, a set of model fragments is extracted based on the differences and common parts spotted among the models. Identical elements are extracted as common parts of the product line, similar elements are extracted as variable alternative parts, and unmatched elements are extracted as variable optional parts. As a result, the variability existing among the set of similar product models is formalized.

However, we have detected an issue when applying this kind of reverse engineering approaches to extract and formalize the variability existing among the IH product models of our industrial partner. Specifically, the fragments obtained by these approaches do not represent logical units or concepts and therefore are difficult to grasp by domain experts [11].

Figure 3.5 illustrates the issue that we have experienced. The top part shows a representation of three of the IH models used by our industrial partner. To better illustrate the example, we only focus on the different inductors used by the IHs. Induction Hob 1 is the simplest IH; an inverter is connected to a power manager that connects with one standalone inductor (this construction is repeated two times in the IH). Induction Hob 2 is the next step in the evolution. An inverter is connected to a power manager that is connected to two inductors (one acts as the main inductor, and the other acts as a slave of the main; it is only activated if the main one is not able to heat the cookware placed on top by itself). Finally, Induction Hob 3 is composed by an inverter connected to a power manager that is connected to three inductors (they have different sizes and roles; one acts as main inductor, while the other two are auxiliary and are only activated when the size of the cookware is too large for the main inductor). It is important to note that the three IHs share a common point (an inverter connected to a power manager that is connected to the main inductor). However, each IH provides different functionalities and is driven by different software elements.

The bottom left part of Figure 3.5 represents the results obtained after applying

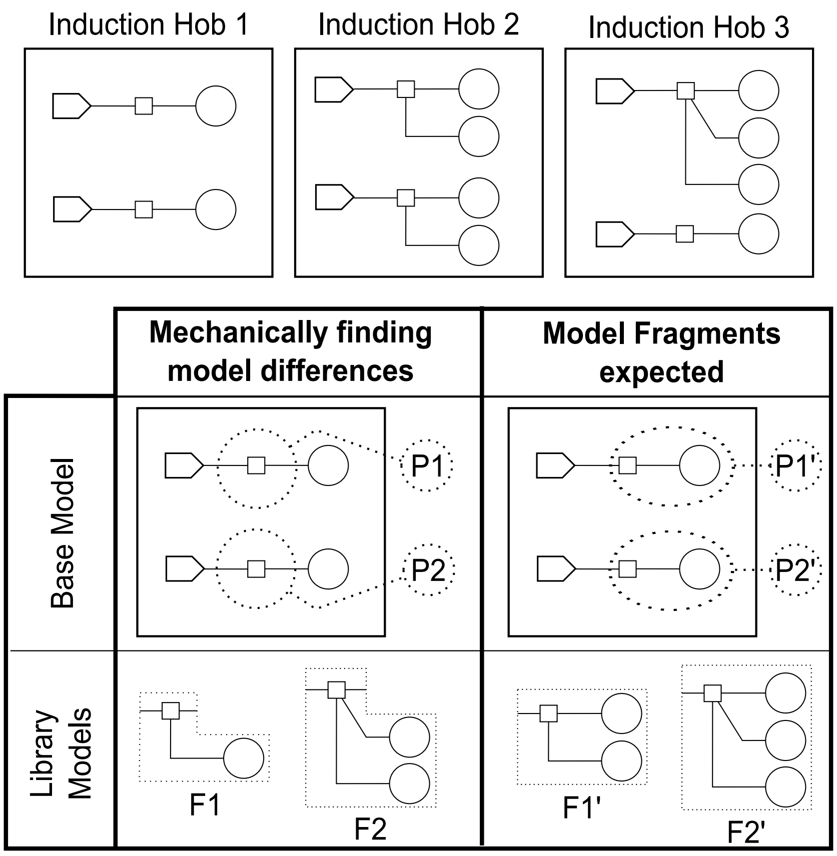


Figure 3.5: Motivation of the proposed approach to address FLiM challenge

our own reverse engineering model differencing approach [10] (see 11.1. The inverter, power manager, and main inductor are identified as common parts to the three IHs and are therefore placed into the base model. Then a placement to hold the rest of the inductors (when they exist) is created. The first fragment holds the slave inductor that is present in the hotplate of the IH2. The second fragment holds the two auxiliary inductors that are present in the hotplate of the IH3.

The IH1 can be obtained without any substitution. The IH2 can be obtained by substituting the placement by the first fragment (IH2 = P1 $\rightarrow$ F1, P2 $\rightarrow$ F1). The IH3 can be obtained by substituting the placement by the second fragment (IH3 = P1 $\rightarrow$ F2). This division of the IH product models is valid, and the three input IHs can be derived from them. However, the results differ from the expected results; the groups of inductors have been divided, resulting in fragments that do not hold model units that are recognizable by our industrial partner's engineers.

The bottom right part of Figure 3.5 shows the expected result when dividing the IHs models into fragments. The base model is similar, but the placements are different, holding the power manager and the inductors to avoid the division of the groups of inductors. As previously, the three IHs can be derived from the model fragments (IH2 = P1' $\rightarrow$ F1', P2' $\rightarrow$ F1', and IH3 = P1' $\rightarrow$ F2'). Although the main inductor is the same for the three IHs, our industrial partner expects to have fragment models that hold whole conceptual patterns. Then, a new placement could be created inside the group of inductors to hold the main inductor. This is just an example, but the problem enlarges when we take into account real models (for example, power generating elements mixed with inductors in the same fragment).

This results in a lack of resemblance between the model fragments produced and the reusable units handled by our industrial partner. Then, when those model fragments are used to build product models, engineers have problems when determining the model fragments to be used in each case (because they do not recognize them). To address this issue we propose a human-in-the-loop approach where the domain experts can take part in the feature location process by contributing their knowledge to tailor the process. As a result the model fragments will match the reusable units that they have in mind.

3.3 Evolution of Model Fragments

This section includes works from literature that are related to the second challenge addressed in this dissertation, the evolution of model fragments, with the focus on the co-evolution of model fragments and language. Works from literature are classified into three categories: (1) Model & Metamodel Co-evolution; (2) Software Product Line Evolution; (3) Traditional Software Evolution. Fig 3.6 shows an overview of the scope.

3.3.1 Model & Metamodel Co-evolution

The main problem when evolving a metamodel is that the conformance between instance models and the metamodel may be broken (depending on the type of changes performed in the metamodel). Our located model fragments relies on a metamodel to define the domain, but the domain needs to be evolved over time (for different reasons such as improvements in the domain or bug fixes). Therefore, we must assure that existing model fragments conform to the evolved metamodel.

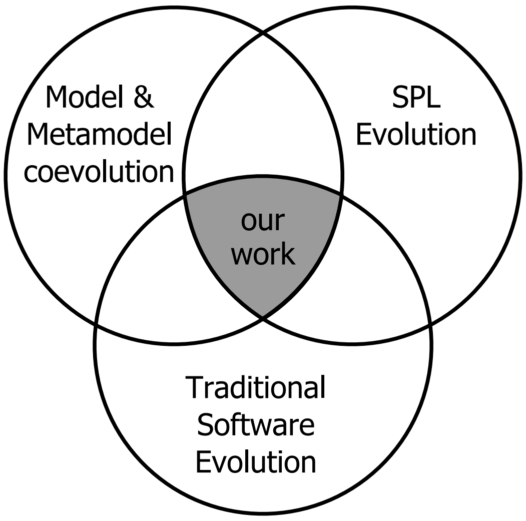


Figure 3.6: Overview of the scope of the evolution of model fragments challenge

There are several approaches in the literature to achieve the co-evolution of model and metamodel while maintaining consistency among them. Those approaches focus in the migration of the models to conform to the evolved metamodel. In particular, [118] organizes existing approaches in three different categories:

Manual specification: a transformation is manually encoded. This is the most obvious solution, requiring a lot of work by the developer. There are no specific research efforts towards this topic.

Operator-based co-evolution: a library of co-evolution operators is defined. These co-operators evolve both, the metamodel and the model (actually, the operator contributes to a M2M transformation that is executed when all the changes over the metamodel are done) [119, 120, 121, 122].

Metamodel-matching: a migration strategy is inferred by analyzing the evolved metamodel and the metamodel history. It can be applied in two different ways:

Differencing approach: both the original and the evolved metamodel are compared (with a diff tool) to generate a difference model that holds the changes between the original and the evolved metamodel, then it is used to create a migration strategy [123, 124, 125, 126, 127].

Change recording approach changes performed to the metamodel are monitored and “recorded” to be used later to generate a migration strategy [128, 128, 129].

3.3.2 Traditional Software Evolution

Software evolves, and there are several works towards characterizing mechanisms of change. SPLs are composed of several artifacts, such as the code assets which are, essentially, pieces of traditional software. Therefore, the research efforts performed towards software evolution can be useful in model-based SPL evolution.

More than thirty years ago, Lientz and Swanson [130] proposed a mutually exclusive and exhaustive software maintenance typology that distinguishes between perfective, adaptive and corrective maintenance activities. This typology was extended by [131] into a classification of 12 different types of software evolution. Then, in [132] Mens et al. proposed a more wide taxonomy that extended the previous taxonomies (based solely on the purpose of the change, the “why”). Therefore, their work presents a taxonomy focused on the technical aspects of change, the so-called “when”, “where”, “what” and “how” of software changes.

There are different works towards the adaptation of traditional versioning systems to address SPL evolution. For instance, Thao [133] propose a version control system, based on product versioning model, to support the evolution, product derivation and change propagation from core assets to products and vice versa. In [134], authors create a specific versioning system adapted to SPL in order to provide more flexibility and reliability when indicating versions of components.

3.3.3 Software Product Line Evolution

There are research efforts in categorization and analysis of changes that can trigger the need to evolve an SPL. These works combine empirical studies and analysis in order to obtain a better understanding of the changes that occurs in the software life cycle. For instance [135] provides a taxonomy of requirements-driven SPL evolution, [136] presents a case study of the changes of two different SPL during five years of evolution and [137] presents an exhaustive report on how evolutionary changes affect the different types of assets.

Lotufo et al. [138] provide empirical evidence of how a large real-world variability model evolves. They present their study using 21 versions of the Linux kernel over five years. Their entire development process is feature driven.

They analyze how a number of characteristics, such as number of features, height of the tree and depth of the leaves, using the feature models of those versions. Based on this research, they identify six categories of reasons for changes in the Linux variability model (New functionality, Retiring obsolete features, Clean-up/maintainability, Adherence to changes in C code, Build fix and Change variability).

Passos et al. [139] developed a vision of software evolution that is based on a feature-oriented perspective. They provided a feature-oriented project management and system development platform that supports traceability and analyses. There is also some work towards the monitoring, management and planning of the evolution of an SPL, for instance [140], presents a tool to plan and manage the evolution of an SPL focusing on goals and requirements.

There are also concerns about maintaining the consistency of the SPL when changes are performed. For instance in [141] a set of templates that preserve the integrity of the SPL are presented and in [142, 143] the focus is on maintaining the consistency among models and the feature mapping. The authors present the conceptual basis of a system capable of maintaining the consistency between the variability model (a feature model) and the model-based artifacts used as target of the feature mapping. In addition they present a set of operators to reestablish the correct binding between the feature model and the model elements.

Creff et al. [144] propose an incremental evolution by extension of the product line. They aim to benefit from the investments made during the product derivation and *reinvest* them into the SPL models. Specifically, they introduce an assisted feedback algorithm to extend the SPL to emerging product derivation requirements.

Dhungana et al. [145] present an approach that is based on model fragments that are applied at the model level. The tool support for the automated detection of changes facilitates metamodel evolution and the propagation of changes in the domain to pre existing variability models.

In [43], Batory et al. present the AHEAD model, which is based on the step-wise refinement paradigm and enables the synthesis of multiple complex programs from a simple program. In AHEAD, the software is expressed as nested sets of equations that describe feature refinements. The composition function (which is specific for each kind of asset) is used to stack the refinements applied to the base program to produce the different variants.

Deng et al. [146] argue that adding new requirements to a model-based Product Line Architecture (PLA) often causes invasive modifications to the PLA's component frameworks and DSLs. To address these modifications, they show how structural-based model transformations help maintain the stability of domain evolution by automatically transforming domain models.

3.3.4 Motivation of our Model and Language Co-Evolution Approach

Figure 3.7 shows an example of the co-evolution of models and metamodels problem. The left part shows a metamodel (metamodel1) and a model (model1) that conforms to that metamodel. That is, the model is expressed following the elements and rules among elements described in the metamodel. Then, the metamodel is evolved into metamodel2 (usually evolutions are done to address existing issues or to extend the expressiveness of the language). Depending on the changes performed to evolve the metamodel, models that conform to the previous version of the metamodel will not conform to the new version of the metamodel. In those cases, the common practice is to perform a migration of the models, needing to transform them to conform to the new version of the metamodel.

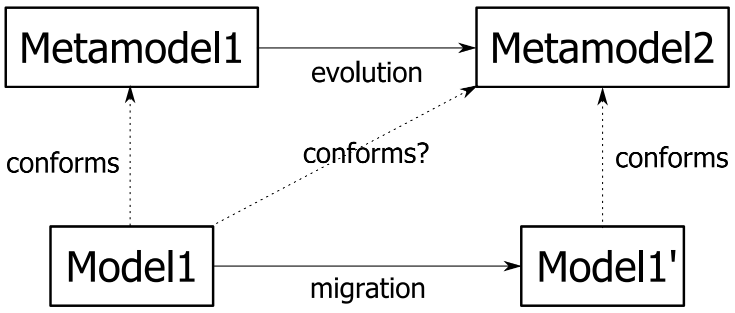


Figure 3.7: Model and Metamodel Co-evolution problem

Figure 3.8 presents the evolution of a model fragment following a migration strategy. Each column shows the same fragment (Inductor 15) for each of the metamodel generations (mm_1,mm_2 and mm_3). Although its functionality remains the same, the model is augmented to conform to each generation metamodel. In **generation 1**, the replacement of an inductor of size 15 is represented by 2 metamodel classes (Inductor and Power Table) and can be connected to a channel and controlled by a button. In **generation 2**, the model fragment is migrated to conform to mm_2. Hotplate 1 now aggregates the inductor and is the one controlled by the button. In this generation, we need 3 classes (we add the Hotplate) to model the same functionality. In **generation 3**, we need to include a cooking zone (enabling groups inside the same hotplate), so the model is now composed of four model elements. The three versions of the model fragment represent the same functionality: a heating element of size 15 that is connected with a channel and controlled from a button. However, there is an increase in model complexity.

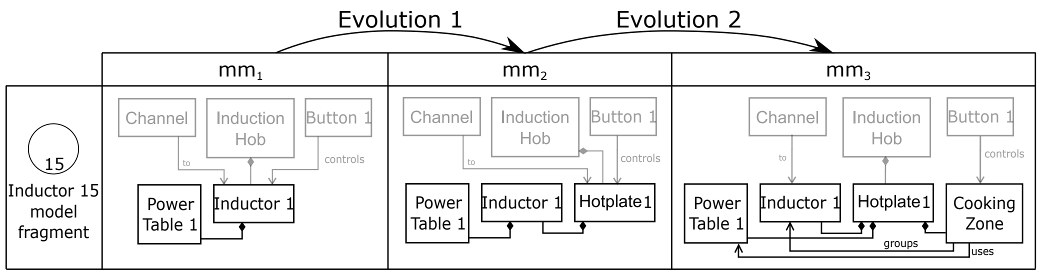


Figure 3.8: Evolution of Model Fragment through Migrations

Specifically, the migration of models from our industrial partner involves three related issues:

Overhead There is an increase in the number of elements used to model the same element of the induction hob (as in this example).

Automation Since the migration of the models cannot be performed automatically, an engineer needs to generate the M2M transformation and take decisions when applying it.

Trust leak The modification of the model fragments (through the migrations) decreases the trust gained by those models during that generation. That is, the models have acquired some reliability or trust among the engineers that have used them several times. This trust can be lost when the fragments need to be modified to be adapted to the new metamodel (not to improve its functionality), and the modification is regarded as unnecessary and error prone.

The induction hobs domain is constantly evolving, but the original elements are still present in new IHs. New types of heating elements or strategies may appear, but the simplest inductors (e.g., the inductor of size 15) are still an important part of modern IHs. Therefore, the issues related to the migration increase with each generation, as old elements are still used.

The approach presented in this dissertation to address the co-evolution challenge applies variability modeling ideas at the metamodel level. By doing so, the particularities of each metamodel and their corresponding models can be formalized into a model fragment and used to avoid the need for migration and its related issues.

Part II

FEATURE LOCATION IN MODELS

4

FEATURE LOCATION IN MODELS BY AN EVOLUTIONARY ALGORITHM (FLIMEA)

Contents

4.1 Overview of the Chapter

This chapter presents an overview of FLiMEA, our process for Feature Location in Models by an Evolutionary Algorithm. The chapter briefly describe the process and presents how domain experts can provide their domain knowledge in order to tailor the process.

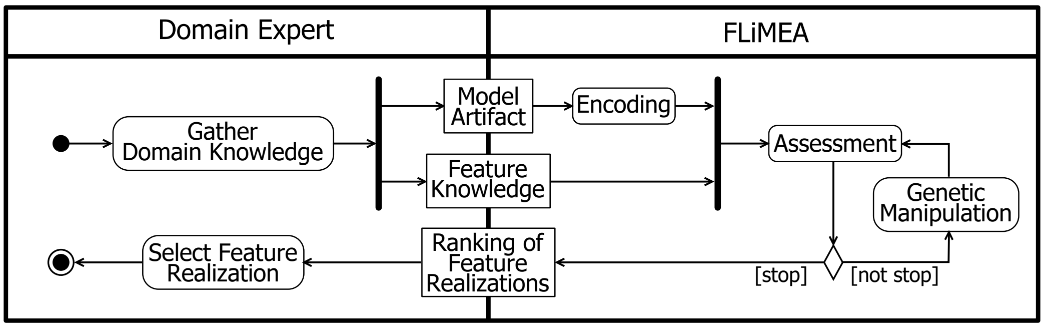


Figure 4.1: Activity diagram for the Feature Location in Models trough an Evolutionary Algorithm (FLiMEA)

Figure 4.1 shows an activity diagram for our FLiMEA process. The left side shows the activities performed by the domain expert while the right side shows the activities automatically performed by the approach. The middle side shows the objects generated in the process.

First, the domain expert gathers domain knowledge relevant for the feature that is going to be located. This knowledge is formalized as the model artifact where the feature is going to be located and the feature knowledge (that holds all the knowledge that the domain experts can gather and produce about the feature that is going to be located). Both, the artifact and the knowledge are provided to our FLiMEA approach and this will result in a ranking of feature realizations. The ranking will be presented to the domain experts and they will use their domain knowledge again to decide (with the information provided by the approach) which of the realization better fits their needs.

4.2 Model Artifact

The model artifact is the artifact where the feature will be located. That is, the realization of the feature should be part of this model artifact and the aim of the feature location process is to find it. In this dissertation we haven been working mainly with two different model artifacts: single models and families of models.

Single model (or product model): This artifact is a model that represents a single product (therefore it is also called product model) from our industrial partner. The model conforms to a particular modeling language (our industrial partners works with Domain Specific Languages built using the Meta Object Facility, see Section 2.4.1) and will consist of a set of model elements containing some properties and references among those model elements. When the feature to be located is known to be realized by a single product model, this artifact will be fed to the process.

Family of (product) models: It is also possible to use a family of models as the artifact where the feature will be located. That is, a set of similar but different models that have some commonalities will be used as the artifact. In a family of models there will exist some variability among the different product models but it is not properly formalized and the aim of the feature location is to shed some light towards this formalization. An example of a family of models has been presented at the top of Figure 3.5.

The domain experts have to identify and provide the model artifact that realizes the feature being located and thus tailor the process when doing so.

4.3 Feature Knowledge

The feature knowledge is the information that will be provided by the domain expert about the feature that is going to be located. This information will be used to narrow and guide the search, tailoring the process based on the domain experts' knowledge. Some of this information will vary depending on the type of model artifacts where the feature will be located and the information available.

Depending on the **information available**, the experts will select different sources of information. In our experience with our industrial partners the information can come from several sources such as:

Tacit knowledge: that is shared among the experts but has not been properly formalized. This kind of knowledge is usually something taken for granted among the experts and can be useful in tailoring the process.

Bug reports: usually contain information about different elements of the model and include domain terms that can be used to create a textual description of the feature being located.

Annotations in software sometimes the models or other software artifacts are annotated or commented. This type of information usually includes domain knowledge and can be used to create the feature knowledge.

Depending on the **artifacts** where the feature should be located the search can be narrowed using different means.

Seed Fragment: A seed fragment of the target feature is an element or set of elements that the domain expert believes could be part of the feature being located. In other words, the domain expert applies his knowledge of the domain and the product models to point to some elements that will be used as the starting point of the process.

Family subset: When a family of models that has been created following clone-and-own approaches [147] is used, one of the existing models serves as the base for the new one. However, the model used as base in each case may vary. This practice could lead to a situation where there are different groups of models (which are not explicitly defined) that have a closer relation. If the humans are aware of the existence of these kinds of groups among the set of models, they can take advantage of it, scoping the input to just a subset of the family of models.

Metamodel constraints: As the approach is using model artifacts, the experts can indicate which meta-elements must be included in or excluded from the resulting feature realization in order to narrow the search. The process can be tailored by defining constraints at the metamodel level.

The resulting domain knowledge is fed to the approach and will be used to guide and narrow the search.

4.4 Evolutionary Algorithm

The approach for feature location in models presented is in this dissertation (FLiMEA in Figure 4.1) comes in the form of an Evolutionary Algorithm. In Evolutionary Algorithms, a population of individuals (candidate solutions for the problem) is evolved and assessed through several iterations in the search for the best possible individual. When applied to model artifacts, the population of individuals will be in the form of model fragments. These individuals need to be properly encoded (see **Encoding** in Figure 4.1), enabling the evolutionary algorithm to work efficiently with them. Next, each candidate solution from the population is evaluated using a fitness function (a formalization of the overall quality goal) to determine how well it performs as a solution to the problem (see **Assessment**). As a result, the population of solutions is ranked depending on their fitness value and, based on the ranked population, some genetic manipulations are performed over the individuals (see **Genetic Manipulation**). This cycle of genetic manipulations and assessment will be repeated until some stop criteria is met.

4.4.1 Encoding

In Evolutionary Algorithms, each possible solution to the problem (called individual) needs to be encoded so the genetic operations can be applied to them. Traditionally, individuals are encoded as a fixed-size string of binary values, but other encodings can be used such as tree encodings. In fact, it is suggested [148] to use an encoding natural for the problem and then devise genetic operations capable of working for that specific encoding. The individuals of our problem are model fragments (extracted from some product model); therefore, the encoding must be able to represent a model fragment extracted from a product model.

As part of this dissertation we have explored two different encodings for the individuals depending on the type of artifact used as input (see Section 4.2): (1) a binary encoding designed to work with single models; (2) a CVL-based encoding designed to work with families of product models.

4.4.2 Assessment

The fitness function is used as an heuristic to guide the search performed by the evolutionary algorithm. To do so, the function assigns a fitness value to each of the feature candidates based on their quality as feature realization. This information can be used in two ways: to determine that the algorithm should terminate as a desirable level of fitness has been reached and to determine the best candidates to be used as parents for the next generation.

In this dissertation we propose and evaluate three different fitness functions: (1) the first one is based on Latent Semantic Analysis [51] and Information Retrieval techniques, to compare textual descriptions with the model fragments; (2) the second one is based on Formal Concept Analysis [149] and Latent Semantic Analysis, model fragments are grouped intro conceptual groups and then the textual comparison is performed at conceptual group level; (3) the third one is based on Conceptual Model Patterns [11], based on pattern repetitions of a model fragment among the products;

4.4.3 Genetic Manipulation

Different operations are performed to manipulate the individuals, with the hope that manipulated individuals (offspring) will perform better after manipulation. Then, to perform the genetic manipulations some parents are selected based on previous fitness assessment, giving priority to the solutions with higher fitness values. Then two types of genetic operations can be performed: crossover, that combines two parents into a new individual; mutation, the individual is evolved and some of its characteristics are modified (added or removed).

As part of this dissertation we have proposed and evaluated four different operations: (1) mask crossover, a crossover operator designed to combine individuals (in the form of model fragments) obtained from the same parent model; (2) parent and model crossover, a crossover operator designed to combine model fragments obtained from different parent models; (3) sequential mutation, a mutation operation designed to perform evolutions of model individuals following a prescribed order; (4) random mutation, a mutation operations designed to perform random modifications over model individuals while keeping the consistency of the model.

4.5 Ranking of feature realizations

Our FLiMEA provides a ranking (see bottom part of Figure 4.1) of different possible realizations for the feature located. The ranking is ordered based on different search criteria depending on the particular fitness being used to locate the features. The domain experts will be in charge of selecting the model fragment from the ranking, using their knowledge of the domain. At the end, the experts will be the ones working and manipulating these feature realizations as part of their everyday work; therefore they should understand and recognize them well to be able to use them with confidence.

As part of this dissertation we have explored two distinct forms of feature realization: (1) realization of features as model fragments; (2) realization of features as variation points. Each of them have an impact on the approach, the next chapters will present the particularities of the presented approach when the features are located as model fragments (Chapter 5) and when the features are located as variation points (Chapter 6).

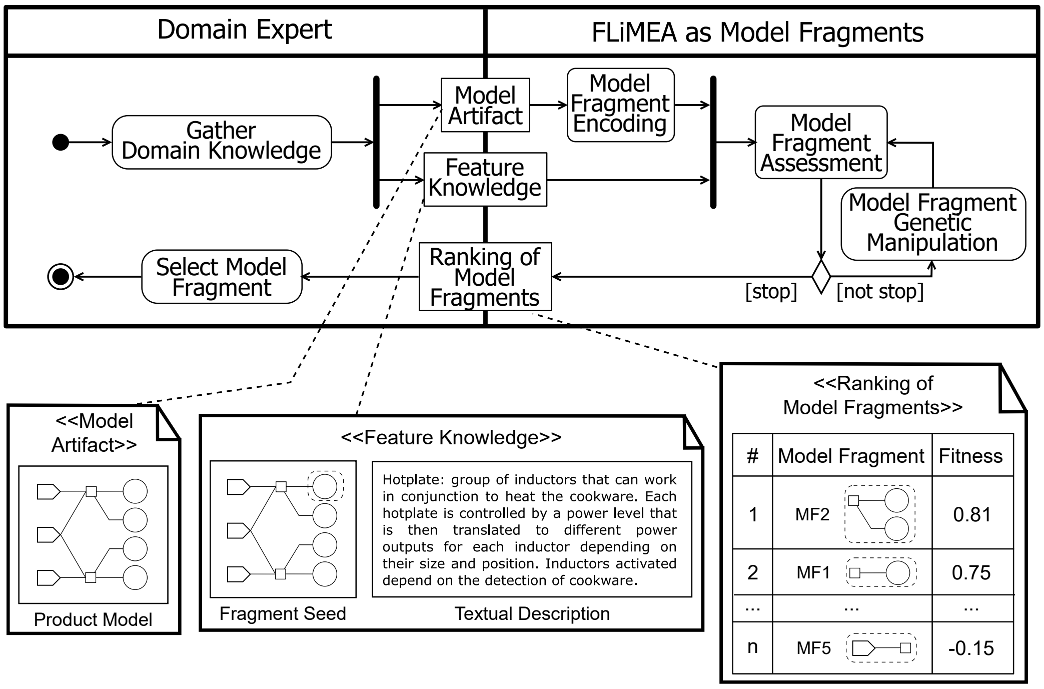
5

FLiMEA as Model Fragments

Contents

5.1 Overview of the Chapter

This chapter presents the particularities of FLiMEA when the feature is realized in the form of a Model Fragment. To accomplish that, we have tailored the approach to work with model fragments as individuals, we have created a fitness function to assess those individuals based on the feature knowledge provided and we have created genetic operations to perform genetic manipulations over those individuals.



Hotplate: group of inductors that can work in conjunction to heat the cookware. Each hotplate is controlled by a power level that is then translated to different power outputs for each inductor depending on their size and position. Inductors activated depend on the detection of cookware.

#	Model Fragment	Fitness
1	MF2	0.81
2	MF1	0.75
...	...	...
n	MF5	-0.15

Figure 5.1: Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Model Fragments

Figure 5.1 shows the instantiation of the generic process for FLiMEA (see Figure 4.1) designed to locate the features as Model Fragments. The bottom-left corner of Figure 5.1 shows the model artifact where the feature realization must be located, a single product model. The bottom-middle part of Figure 5.1 shows the feature knowledge provided by domain experts, in this case a textual description obtained from any of the sources available and a seed in the form of a model fragment obtained from the model artifact provided (the seed is depicted by a dashed line that enclose some elements of the product model).

Then, individuals are encoded as model fragments and a fitness function designed to assess model fragments is used to perform the assessment of the individuals. Similarly, the genetic manipulations of the individuals are performed applying genetic operators capable of manipulating model fragments.

As a result, our FLiMEA approach provides a ranking of alternate feature realizations in the form of Model Fragments, including the fitness score obtained by each realization. The bottom-right corner of Figure 5.1 shows an example of a feature realization ranking when the features are model fragments. Using the information on the ranking and their domain knowledge, the experts select the feature that best realizes the feature that was being located. The next sections provide the details of the encoding, fitness functions and genetic operators that we have designed to work with model fragments.

5.2 Encoding: Binary

Traditionally, in evolutionary algorithms each individual is encoded as a fixed-size string of binary values. Each position of the binary string corresponds to a particular element that may be or not part of the solution and has two possible values 0 or 1. To encode a model fragment as a string of binary values we need to decide what information will be encoded in the binary string. For instance, each position of the string can represent one model element of the parent model; then, each individual will have that bit at 0 to indicate that the element is not part of the model fragment or at 1 to indicate that it is part of the model fragment. By doing so, we can indicate the subset of elements from the parent model that are part of the model fragment.

Figure 5.2 shows two examples of the binary encoding for model fragments. The left part shows the encoding for “individual 1”, a model fragment obtained from “Parent model 1” while the right part shows the encoding for “individual 2”, a model fragment obtained from “parent model 2”. Each element of the parent models has been tagged with a letter, to establish the link between the model element and the position in the string of binary values. For instance, in “Parent Model 1” the letter A_1 corresponds to the upper inverter, and F_1 correspond to the lower inverter. The bottom part shows the individuals encoded, the string of binary values holds a 0 or 1 value for each of the positions, that represent each of the elements on the parent model. For each position, if the corresponding model element is part of the model fragment, the value will be 1; if the corresponding model element is not part of the model fragment, the value will be 0.

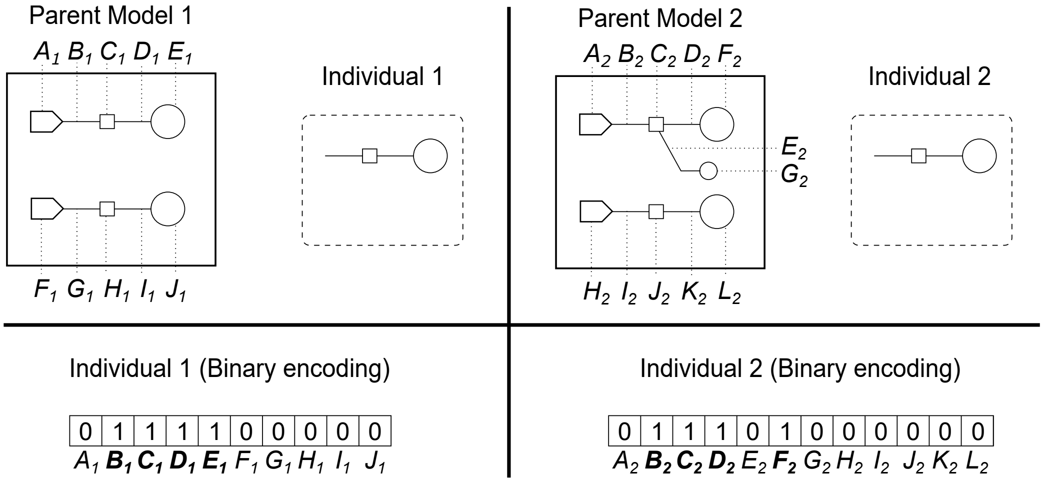


Figure 5.2: Binary-based encoding

It is important to note that both individual 1 and individual 2 are the same model fragment. However, each individual is a model fragment obtained from a different parent model and, given that the encoding depends on the parent model, the resulting encoded individual is different. For instance, E_1 corresponds to an inductor of “parent model 1” while in “parent model 2” the corresponding inductor is tagged as F_2 and corresponds to a different position in the binary string.

5.3 Fitness: Text-based similarity

As part of this dissertation we have explored two different fitness functions that can be used when locating features as model fragments using textual descriptions as feature knowledge: (1) Latent Semantic Analysis; and (2) Formal Concept Analysis & Latent Semantic Analysis.

Both approaches rely on Latent Semantic Analysis (LSA) to measure textual similarities between the individuals and the feature description provided as part of the feature knowledge. The first fitness function applies LSA to the model fragments and assesses them based on the similarity with the textual description. The second fitness function first groups model fragments into formal concepts and then applies LSA to those formal concepts, spreading the fitness assigned to the formal concept to the model fragments belonging to it. The next subsections present the details of both fitness functions.

5.3.1 Latent Semantic Analysis

To assess the relevance of each individual of the population in relation to the feature description provided by the user, we apply methods based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Analysis (LSA) [51] to analyze the relationships between the description of the feature provided by the domain expert and the model fragments. Recent studies observed that there is not a statistically significant difference among different IR techniques [150, 151] when applied to software artifacts [152]. Hence, we choose LSA because it produces similar results to other IR techniques for software documents.

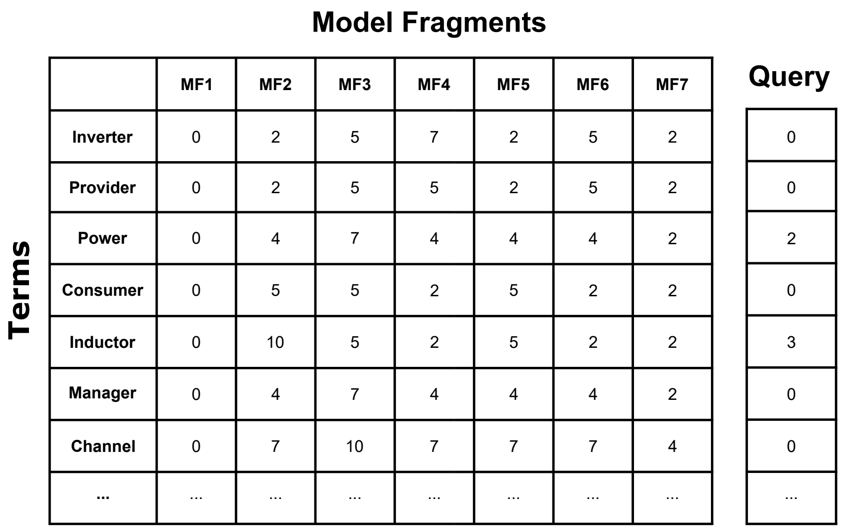
LSA constructs vector representations of a query and a corpus of text documents by encoding them as a *term-by document co-occurrence matrix* (i.e., a matrix where each row corresponds to terms, each column corresponds to documents, and the last column correspond to the query). We use the *term-frequency (tf)* as the term weighting schema to construct the matrix. That is, each cell holds the number of occurrences of a term (row) inside either a document or the query (column).

In our work, all documents are model fragments, i.e., a document of text is generated from each of the model fragments. The text of the document corresponds to the names and values of the properties and methods of each model fragment. The query is constructed from the terms that appear in the feature description. The text from the documents (model fragments) and the text from the query (feature description) are homogenized by applying well-known Natural Language Processing techniques:

Tokenize: First, the textual description is tokenized (divided into words). Usually, a white space tokenizer can be applied (which splits the strings whenever it finds a white space), but for some sources of description, more complex tokenizers need to be applied. For instance, when the description comes from documents that are close to the implementation of the product, some words could be using CamelCase naming.

Parts-of-Speech: Second, we apply the Parts-of-Speech (POS) tagging technique. POS tagging analyzes the words grammatically and infers the role of each word into the text provided. Recent studies in software engineering have proved the usefulness of POS-tagging techniques to remove textual noise in software documents [153]. In addition, the use of word-selection strategies [154, 155] can improve the results in feature location [156]. After applying this technique, each word is tagged, which allows the removal of some categories that do not provide relevant information. For instance, conjunctions (e.g., *or*), articles (e.g., *a*) or prepositions (e.g., *at*) are words that are commonly used and do not contribute relevant information that describes the feature, so they are removed.

Stemming: Third, stemming techniques are applied to unify the language that is used in the text. This technique consists of reducing each word to its roots, which allows different words that refer to similar concepts to be grouped together. For instance, plurals are turned into singulars (*inductors* to *inductor*) or verb tenses are unified (*using* and *used* are turned into *use*).



Model Fragments

Terms	MF1	MF2	MF3	MF4	MF5	MF6	MF7	Query
Inverter	0	2	5	7	2	5	2	0
Provider	0	2	5	5	2	5	2	0
Power	0	4	7	4	4	4	2	2
Consumer	0	5	5	2	5	2	2	0
Inductor	0	10	5	2	5	2	2	3
Manager	0	4	7	4	4	4	2	0
Channel	0	7	10	7	7	7	4	0
...	...	...	...	...	...	...	...	...

Figure 5.3: Term-by-document co-occurrence matrix for Model Fragments

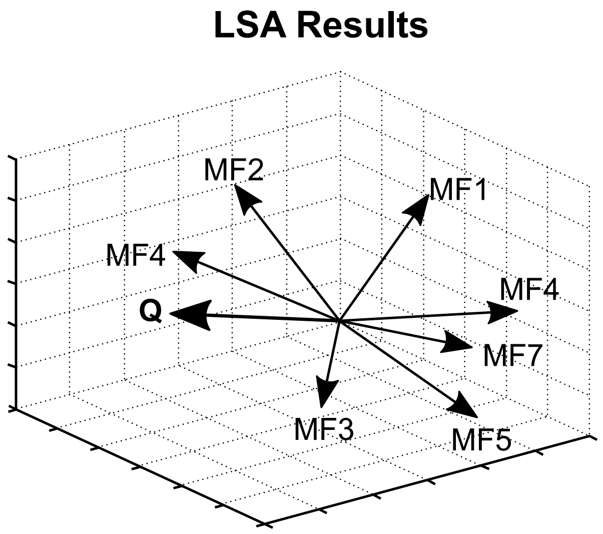
Figure 5.3 shows an example of the co-occurrence matrix. Each column corresponds to one of the individuals from the population. The last column is the query obtained from the textual description provided as part of the feature knowledge. Each row is one of the terms extracted from the corpora of text composed by all of the model fragments and the query itself (to improve readability we show the terms before the stemming process). Each cell shows the number of occurrences of each term (row) in each document obtained form the individuals (column). The union of all the keywords extracted from the documents (model fragments) and from the query (feature description) are the terms (rows) used by our LSA fitness operation.

Once the matrix is built, it is normalized and decomposed into a set of vectors using a matrix factorization technique called Singular Value Decomposition

(SVD) [51]. SVD project the original term-by-document co-occurrence matrix in a lower dimensional space k. We use the value of k suggested by Kuhn et al. [157], which provides good results [158]. One vector that represents the latent semantic of the document is obtained for each model fragment and the query. Finally, the similarities between the query and each model fragment are calculated as the cosine between the two vectors. The fitness value that is given to each model fragment is obtained as the cosine similarity between the two vectors, obtaining values between -1 and 1.

$$fitness(p_1) = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \tag{5.1}$$

Let p_1 be an individual of the population; let A be the vector representing the latent semantic of p_1; let B be the vector representing the latent semantic of the query where the angle formed by the vectors A and B is θ. The fitness function can be defined as:



Fitness
MF4 : 0.9
MF2 : 0.5
MF3 : 0.3
MF1 : -0.2
MF5 : -0.4
MF7 : -0.7
MF4 : -0.8

Figure 5.4: LSA Fitness Results

Figure 5.4 (left) shows a three-dimensional graph of the LSA results. The graph shows the representation of each one of the vectors, which are labeled with letters that represent the names of the model fragments. Finally, after the cosines are calculated, we obtain a value for each of the model fragments, indicating its similarity with the query. As a result of the fitness assessment, each individual of the population is assigned with a fitness value, that measures the similarity of the individual with the textual description (see right part of Figure 5.4).

5.3.2 Formal Concept Analysis

When locating features realized through model fragments, it is important to notice that a feature can be realized by the combination of more than one model fragment. To address this situation, model fragments can be combined into groups, hoping to create groups that match with the realization for the feature being located. Then, a fitness value will be calculated for each of the groups (using LSA) instead of applying it to each individual separately. If the stop criteria is met, the groups of individuals can be transformed into regular model fragments and included as feature realizations into the resulting ranking. If the stop criteria is not met and the process keeps evolving the population, the fitness value of the group can be spread to the model fragments that belong to that group.

When using grouping strategies, the fitness function can be divided into three steps: (1) the population of individuals is combined into groups; (2) each of the groups receive a fitness value obtained using LSA; (3) either the groups are turned into feature realizations or the fitness value assigned to the group is spread to the individuals (depending on the fulfilment of the stop criteria).

Grouping of Fragments into Feature Candidates

To perform the grouping of model fragments into feature candidates we rely on Formal Concept Analysis (FCA) [149], a branch of mathematical lattice theory that provides means to identify meaningful groups of objects that share common attributes. Groupings are identified by analysing a binary relationship between the set of all objects and all attributes. FCA takes as input a formal context (an incidence table indicating which attributes are possessed by each object) and returns a set of concepts where every concept is a maximal collection of objects that share some common attributes. Each concept will be considered as a feature candidate.

Therefore, in order to apply FCA we need to define a set of objects (model fragments), a set of common attributes (the metamodel elements used to build those model fragments) and a binary relationship between them (the presence or absence of a particular metamodel element in the model fragment). Then, a formal context that represents the relationship between the objects and the attributes can be produced.

Figure 5.5 shows an example of a formal context relating model fragments and the metamodel elements used to build them. Columns show each of the at-

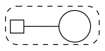
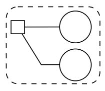
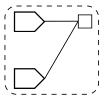
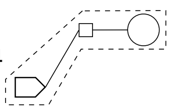
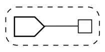
	Inverter	Provider Channel	Power Manager	Consumer Channel	Inductor
MF1 			✓	✓	✓
MF2 			✓	✓	✓
MF3 	✓	✓	✓		
MF4 	✓	✓	✓	✓	✓
MF5 	✓	✓	✓		
...	...	...	...	...	...

Figure 5.5: Formal Context between model fragments and metamodel elements

tributes present in the context, in this case the different metamodel elements used to build the model fragments. Rows show each of the objects of the context, in this case the different candidate model fragments present in the population. Each cell indicates if a particular metamodel element has been used to build each of the model fragments. For instance, MF1 and MF2 (first and second rows) are built using three different metamodel elements (power manager, consumer channel and inductor), while MF4 (fourth row) is built using all the elements from the metamodel (Inductor, Inverter, Provider Channel, Consumer Channel and Power Manager).

Using the formal context as input, FCA generates a lattice: a set of interrelated concepts where each one is a maximal collection of model fragments that share common metamodel elements. Figure 5.6 shows the lattice obtained applying FCA to the formal context presented before. Each of the circles represents one concept (there are seven in total). The concepts are labeled with the metamodel elements (grey background labels) and the model fragments (white background labels) grouped by that concept. The concepts are organized hierarchically, indicating containment relationships between the sets of model fragments and metamodel elements of the concepts.

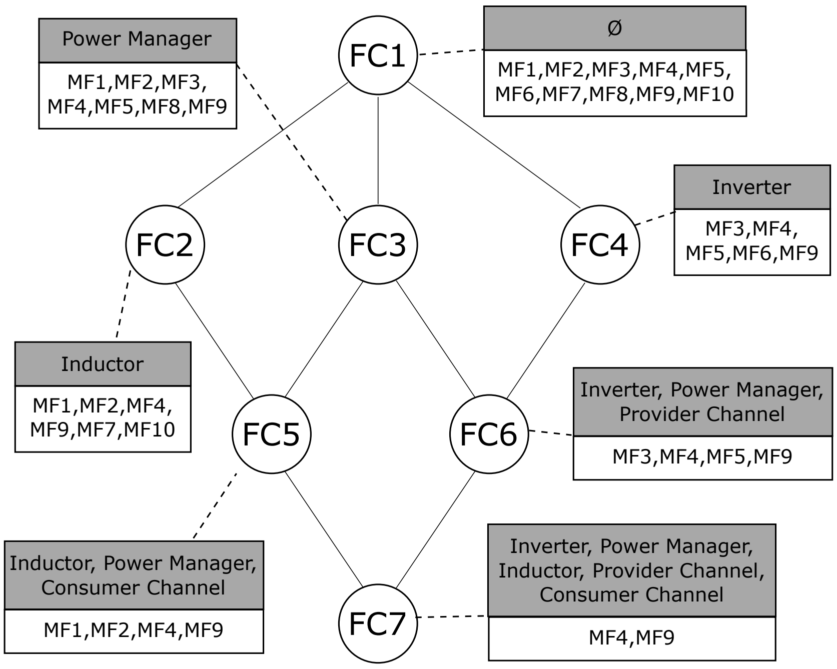


Figure 5.6: Lattice obtained from the Formal Context

That is, the set of model fragments of a concept is contained by all the connected concepts above it and contains all the model fragments from connected concepts below it. For instance, the model fragments in FC6 (MF3,MF4,MF5,MF9) will be also part of all concepts above FC6 (FC4 and FC1). Likewise, the metamodel elements in FC3 (Power Manager) will be also part of all concepts below it (FC5, FC6 and FC7).

As a result of the application of the FCA, a set of Feature Candidates (FC1, FC2, FC3, FC4, FC5, FC6 and FC7) that clusters some of the model fragments based on their use of the elements of the metamodel is provided.

Feature Candidates assessment through LSA

To assess the relevance of each feature candidate with relation to the query extracted from the textual description provided by the user, we apply Latent Semantic Analysis (LSA) to analyse the relationships between the description of the feature provided by the user and the candidate features previously obtained.

We apply LSA to the feature candidates generated by FCA and the query. A document of text is generated from each of the feature candidates using the model fragments present in the feature candidate. That is, the names and values of properties and methods are processed to extract the terms by applying Natural Language Processing techniques (as explained before, see Section 5.3.1). As a result we obtain a list of relevant terms present in the documents and the query. Finally, after the matrix is turned into vectors and the cosines are calculated, we obtain a value for each of the feature candidates indicating its similarity with the query.

Spread the fitness values across the groups

The next step is to spread the similarity values obtained by each feature candidate to the model fragments present in that feature candidate. However, each model fragment can be part of more than one feature candidate. Therefore, in order to obtain the similarity of a model fragment with the query we need to combine the similarity values obtained by each of the feature candidates where the model fragment is present. As a result each model fragment is assigned with a value (fitness value).

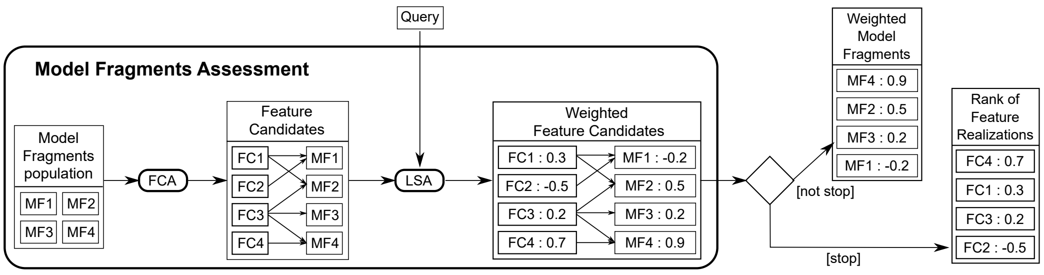


Figure 5.7: Fitness assessment for Feature Candidates and spread to Individuals

Figure 5.7 shows an example of the assessment process. First, the set of model fragments from the population is used to build a set of feature candidates through FCA. Then, the set of feature candidates is compared with the query through the use of LSI, resulting in a set of weighted feature candidates. At this point, if the stop condition is met, the process will stop returning the rank of feature candidates. If the stop condition is not yet met, the evolutionary algorithm will continue its execution for another generation more.

The next time that the genetic operators are applied, it will be necessary to select the best candidates as parents for the new generation. This will be done based on the score obtained by each model fragment. As a result, model fragments with higher similarities will have more chances to be selected as parents of the new generation. Notice that being part of more feature candidates does not guarantee a higher score for the model fragment, as the similarity between a feature candidate and the query can be negative.

5.4 Genetic Operations for Model Fragments

This section presents the genetic operations designed to work over model fragments: (1) a selection operator that chooses the best model parents based on the fitness value previously calculated; (2) a crossover operation that will combine two model fragments into a new model fragment; (3) a mutation operation that introduces random variations of the model fragment while keeping consistency with the parent model.

5.4.1 Parent Selection: Model Fragment selection

The first step when performing genetic manipulations to individuals of the population is the selection of parents. That is, a number (the number depends on the operations that will be performed but typically is two) of individuals are selected from the population based on their fitness value. These individuals will be manipulated through genetic operations in order to create new individuals with the hope that they will be fitter than their predecessors.

However, if the individuals with the best fitness are always the only ones selected the genetic algorithm could suffer from premature convergence, neglecting areas of the search space that could provide better results throughout. There are some strategies to cope with this issue [159], and one of the most frequent is to use a selection mechanism that ensures a proper distribution of the selection.

Therefore, the process will use a selection mechanism where fitter individuals are selected more times than others but that also mitigates the premature convergence issue. The most usual option is to follow the wheel selection mechanism [160]. That is, each model fragment from the population has a probability of being selected proportional to its fitness score. Therefore, candidates with high fitness values will have higher probabilities of being chosen as parents for the next generation.

Figure 5.8 shows an example of application of the selection operator. The operator determines which model fragments (from the population of model fragments) will be used as parents. In this case, Model Fragment 1 and Model Fragment 3 are selected as parents of new individuals.

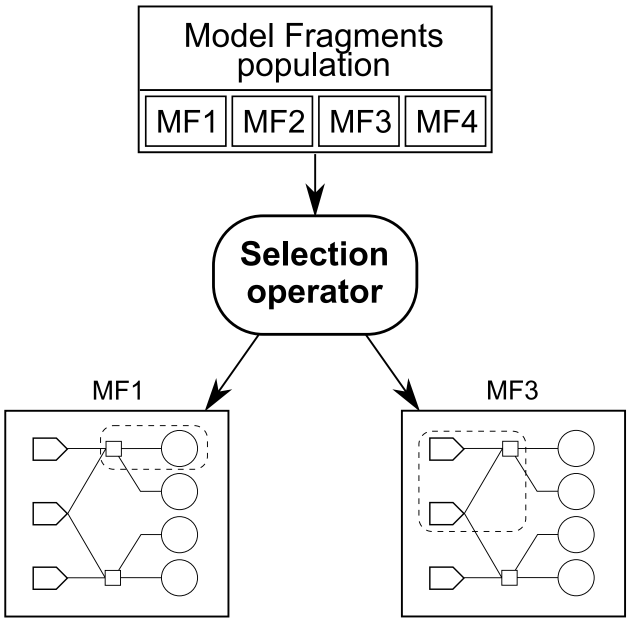


Figure 5.8: Selection Operator for Model Fragments

5.4.2 Crossover: Mask-based

The crossover operation is used to imitate the sexual reproduction followed by some living beings in nature to breed new individuals. That is, two individuals mix their genomic information to give birth to a new individual that holds some genetic information from one parent and some from the other one. This could make him adapt better (or worse) to his living environment depending on the genetic information inherited from his parents.

Following this idea, our crossover operation applied to model fragments takes as input two model fragments and a randomly generated mask (as usual in genetic algorithms) to combine them into two new individuals. The mask determines how the combination is done, indicating for each element of the model fragments if the offspring should inherit from one parent or the other (including the element or not if the element is present in the parent or not).

A model fragment is a subset of the elements present in a product model. As both model fragments have been extracted from the same product model the combination (applying the mask) of them will always return a model fragment that is part of the product model. As a result, two individuals will be generated, one by directly applying the mask and another one by applying the inverse of the mask as usually done in genetic algorithms [148].

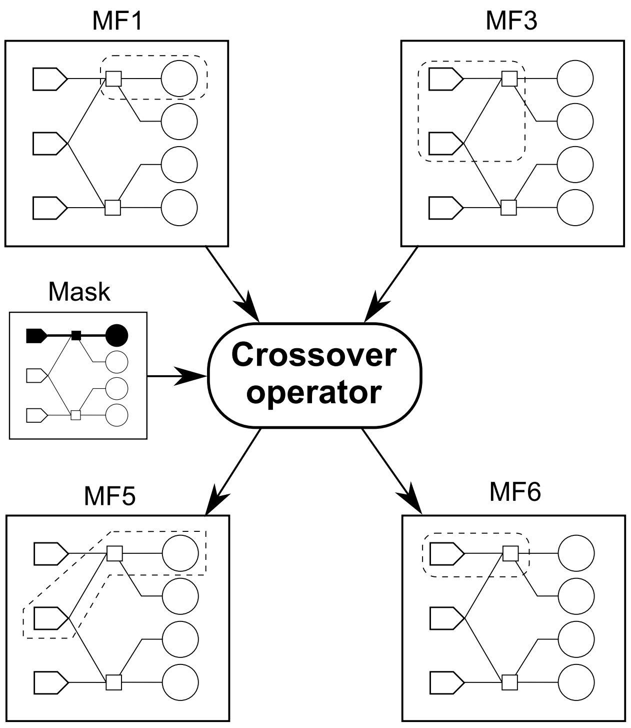


Figure 5.9: Mask-based Crossover for model Fragments

Figure 5.9 shows an example of application of the crossover operation. The input of the operation is the first parent (MF1), a mask indicating two sets of elements (one regular and one marked in black) and the second parent (MF3). To create the first of the new individuals the approach interprets the mask selecting the blacked out elements from the first parent (MF1) and the regular elements from the second parent (MF3). That is, the elements on the top part of the product model that are in black in this mask are selected depending on whether they are part of MF1 or not, while the rest of the elements that are not blacked out in the mask are selected depending on whether they are part of MF3 or not. As a result, the new MF5 contains some elements from the first parent (power group connected to the inductor) and some others from the second parent (the inverter that connects with the power group).

In addition, the mask is also interpreted in the opposite way, selecting the blacked out elements form the second parent and the regular elements from the first parent. This produces MF6 (see right part of Figure 5.9), where an inverter connected to a power manager has been inherited from the second parent (MF3) and nothing has been inherited from parent 1 (MF1) as all the elements not blacked out in the mask are not part of MF1.

For the crossover operation to work, it is not necessary to have elements shared by both parents. It is possible to perform crossovers that return fragments where not all the elements are connected. Indeed, the feature being located could be realized by several model elements that are not directly connected in the model. Therefore, it is necessary to create this kind of fragments as they could be the ones realizing the feature being located.

5.4.3 Mutation: Random

The mutation operator is used to imitate the mutations that randomly occur in nature when new individuals are born. That is, a new individual holds a small difference in regards to its parents that could make him adapt better (or worse) to their living environment.

Following this idea, the mutation operator applied to model fragments takes as input a model fragment and mutates it into a new one produced as output. As the approach is looking for fragments of the product model that realize a particular feature, the new modified fragment must remain being part of the product model. Therefore, the modifications that can be done to the model fragment are driven by the product model. In particular, the mutation operator can perform two kinds of modifications, addition of elements to the fragment, or removal of elements from the model fragment.

Removal of elements: This kind of mutation randomly removes one element from the model fragment. As the resulting model fragment is a subset of the original model fragment, it will always be a model fragment present in the parent model. The right part of Figure 5.10 shows an example of a mutation that removes elements from the model fragment. MF6 is mutated and results in MF8; the mutation operator has removed the power manager.

Addition of elements: This kind of mutation randomly adds some elements to the model fragment. The only constraint is that the new model fragment is in effect a model fragment of the parent model (that is, a subset of the elements present in the parent model). This is ensured by the binary encoding used, as only model elements present in the parent can be part of the individuals following this encoding. The left part of Figure 5.10 shows an example of a mutation that adds elements to the model fragment. MF5 is mutated

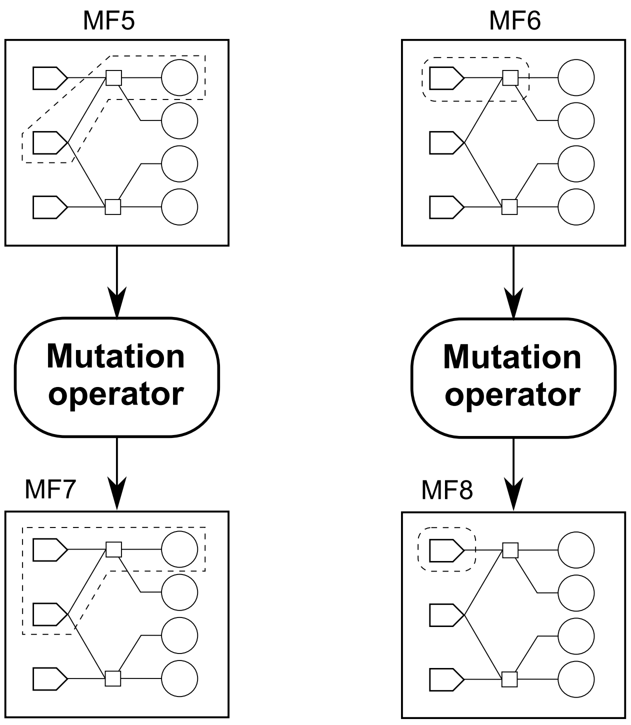


Figure 5.10: Random Mutation for model Fragments

and results in M7; the mutation operation has added some elements (a new inverter connected to the power manager).

5.5 Variability in FLiMEA as Model Fragments

Figure 5.11 shows a recapitulation of the different options presented for the Feature Location in Models as Model Fragments process. The model artifact used by this process is the single product model. The feature knowledge provided to the process consists in a textual description and optionally model fragment seeds. The encoding presented was the binary encoding, where individuals are represented as binary strings. For the individual assessment, Latent Semantic Indexing is used and optionally Formal Concept Analysis can be used. The individual genetic manipulation presented included a parent selection operation (model fragment), a crossover operation (mask based) and a mutation operation (Random). Finally, the resulting ranking of feature realizations will be in the form of model fragments.

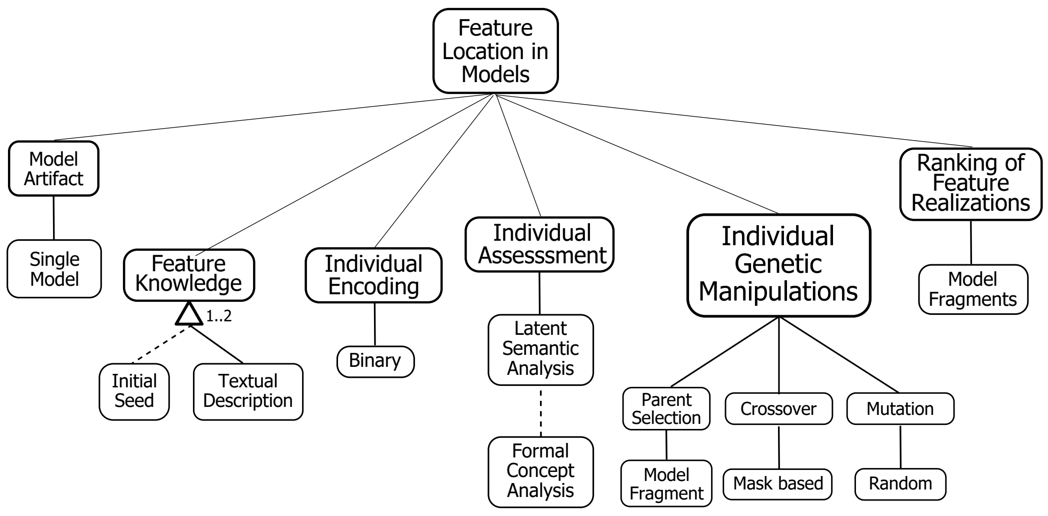


Figure 5.11: Variability of the Feature Location in Models as Model Fragments process

6 FLiMEA as Variation Points

Contents

6.1 Overview of the chapter

This chapter presents the particularities of our FLiMEA approach when the feature realization is in the form of Variation Points. Figure 6.1 shows an example of a variation point compared to a model fragment. The previous chapter showed the particularities of FLiMEA in locating the features as a single model fragments (see left part). This chapter shows the particularities of our approach to locate the features as a variation point, a set of different alternatives found across a set of product models (see right part).

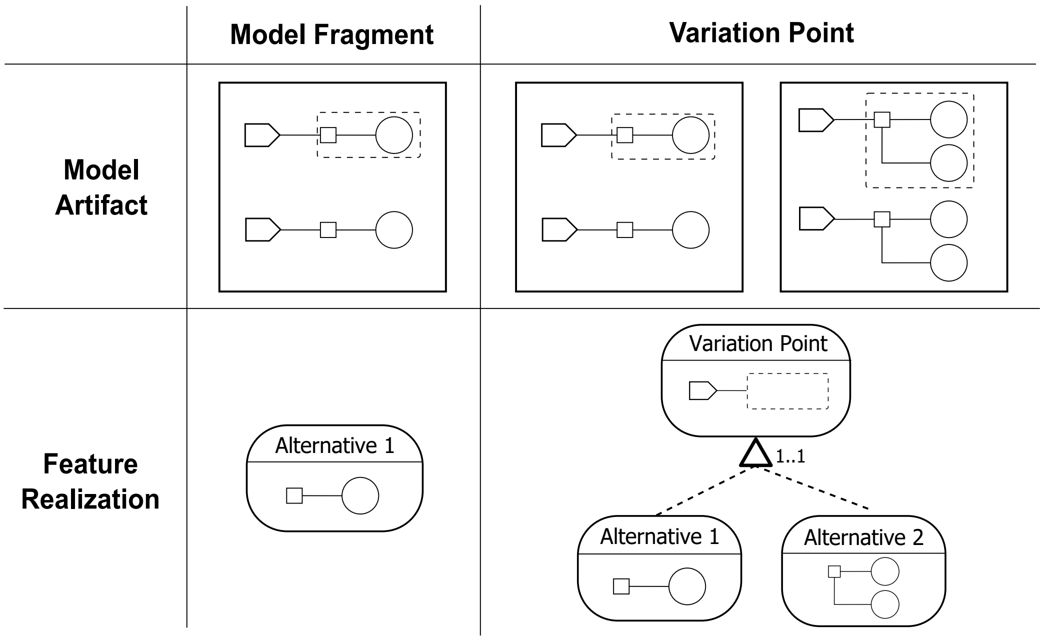


Figure 6.1: Model Fragment and Variation Point

To accomplish this, we augment FLiMEA with an encoding capable of holding variation points, a fitness function designed to assess the individuals as variation points and genetic operations capable of evolving individuals in the form of variation points.

Figure 6.2 shows the instantiation of the generic process for FLiMEA (see 4.1) designed to locate the features as variation points. The top-left corner of Figure 6.2 shows the model artifact where the feature realization must be located, a family of product models. This family of product models will be selected by the experts based on their domain knowledge and taking into account the relationships between existing product models (see 4.3). The top-right corner of Figure 6.2

shows the feature knowledge provided by domain experts, in this case: a product model (selected from the family of product models provided as model artifact); a model fragment seed (extracted from that product model); and a set of metamodel constraints that will guide the genetic operations.

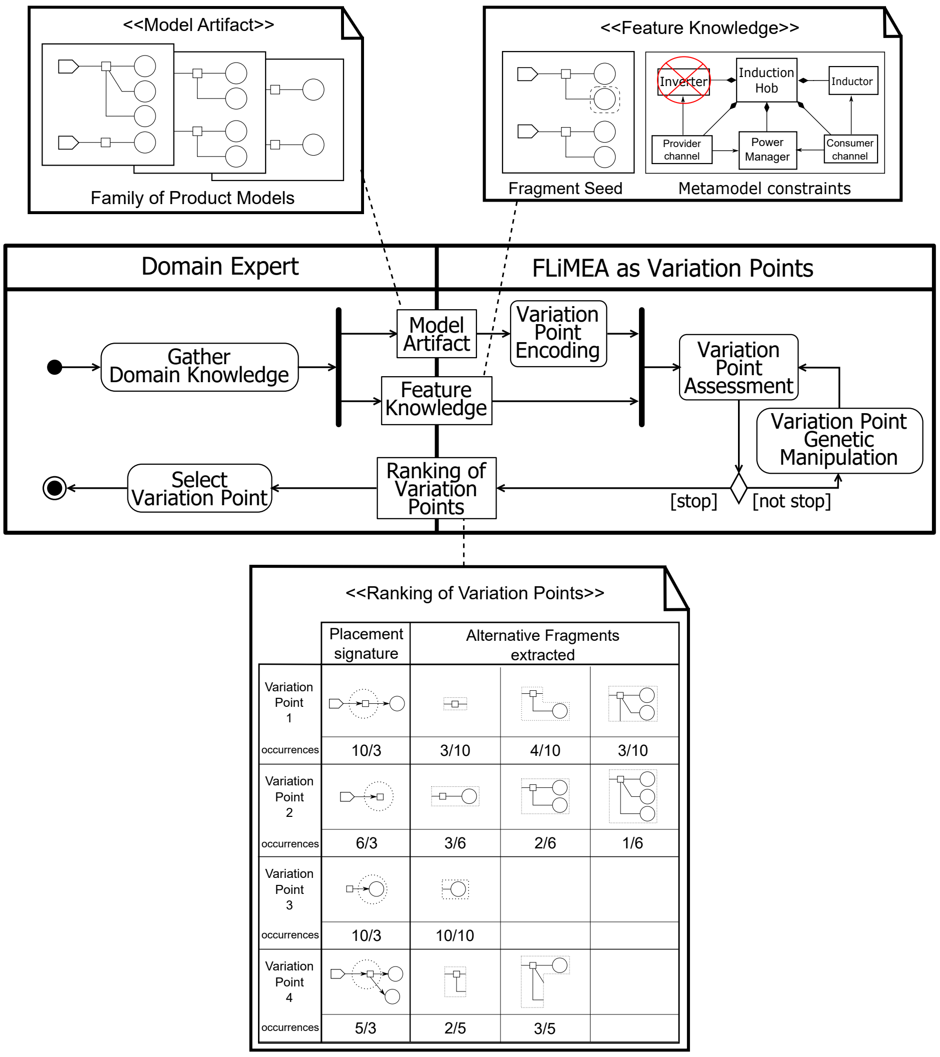


Figure 6.2: Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Variation Points

Then, individuals will be encoded as variation points and assessed by the fitness function designed to work over variation points. Similarly, genetic operators capable of manipulating the variations points will be applied to evolve the population.

As a result, the FLiMEA approach will provide a ranking of alternate feature realizations in the form of variation points, including the fitness score obtained by each variation point. The bottom of Figure 6.2 shows an example of a feature realization ranking in the form of variation points. Using the information of the ranking and their domain knowledge, the experts will select the feature realization that best fits their understanding of the domain. The next section provides the details of the encoding, fitness and genetic operators designed for this scenario.

6.2 Encoding: Boundary-based

In the previous chapter the encoding used was Binary-based as, the artifact where the feature needed to be located, was a single product model. Therefore, the individual could be encoded as a fixed size binary string where each bit represented the presence or absence of each model element (with relation to the product model).

However, when locating the features across a family of product models, not all individuals will be based on the same product model and thus it is not possible to use a fixed size binary string to encode them. In addition, the fitness function will be based on model comparisons (to discover different alternatives for a placement) and binary strings where each index represents different elements are not te best format to perform this kind of comparisons.

Therefore, to locate features as variation points, FLiMEA needs individuals encoded in a format that allows comparisons between model fragments extracted from different parents and capable of representing variation points. To achieve this, we created a new encoding based on the Common Variability Language (CVL) [47, 48, 49].

CVL defines variants of a base model (conforming to MOF) by replacing variable parts of the base model (placement) by alternative model fragments (replacements). The variability specification through CVL is divided across two different layers: the feature specification layer (where variability can be specified following a feature model syntax) and the product realization layer (where variability specified in terms of features is linked to the actual models in terms of model fragments).

Figure 6.3 shows an example of encoding of an individual using the Boundary-based encoding. CVL can be used to specify model fragments over any MOF-based DSL, through the use of pointers to the relevant elements. The middle part, shows the specification of the model fragment using CVL

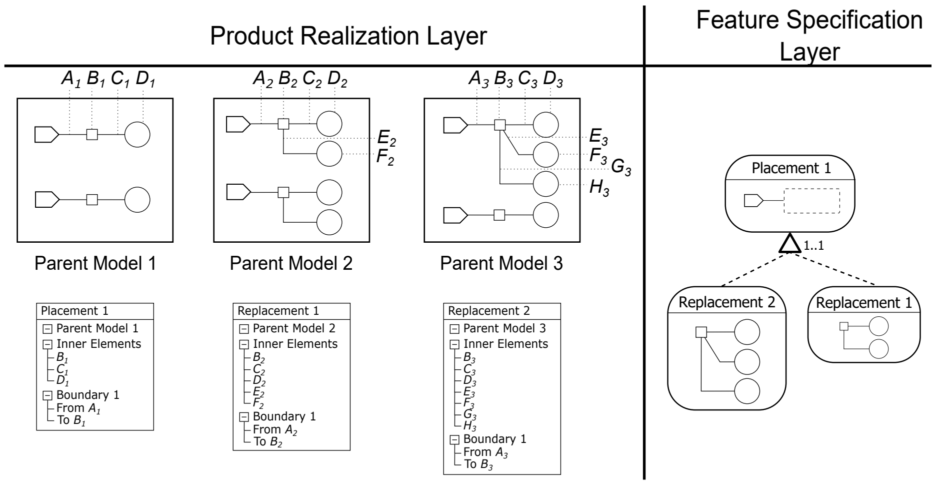


Figure 6.3: Example of a variation point using CVL-based encoding

Parent Model: a reference towards the parent model, the model where the fragment is extracted from. The model fragment is defined over the parent model, as a subset of it. Any model fragment defined using CVL will be a subset of a parent model.

Inner Elements: a reference to each of the model elements that are part of the model fragment. In this example, four elements are part of the model fragment: provider channel 1, Power Manager 1, Consumer Channel 1 and Inductor 1.

Boundary 1: the set of boundaries between the model fragment and the rest of the parent model. Each boundary is composed of two elements, a "from" and a "to", each element is a reference to a model element (one of them present in the model fragment and another one that is not part of the model fragment). In this example there is only one boundary (Boundary 1):

From: the source of the relationship, in this example is the inverter 1. The reference points to one of the elements of the parent model that is not part of the model fragment.

To: The target of the relationship, in this example is the provider channel 1. The reference points to one of the inner elements of the model fragment, as that element is part of the model fragment.

When using the Boundary-based encoding, each individual will be a model fragment defined over one of the product models through the expressiveness of

CVL (as shown in the example above). However, in order to work with this specific encoding we need genetic operations that can be applied directly to those CVL model fragments. There are no genetic operations that can be applied to that encoding in the literature, therefore we need to create them.

6.3 Fitness: Conceptual Model Patterns

We want FLiMEA to locate features in the form of Variation Points, but individuals of the population are in the form of Model Fragments extracted from a parent model. However, the use of Boundary-based encoding enables the manipulation of the model fragment and eases the transition from a Model Fragment to a Variation Point. The Conceptual Model Pattern (CMP) fitness function is based on the identification and matching of conceptual patterns across the family of product models provided as model artifact.

The CMP fitness consists of three steps that are applied to all individuals of the population: (1), the process abstracts from each individual to a placement signature in their parent model; (2), each resultant placement signature is matched against all the product models from the family of product models used as model artifact to obtain alternatives that match the placement signature; (3), a fitness value is computed for each placement signature and then spread to individuals of the population.

6.3.1 Placement Signature Abstraction

The first step is the Placement Signature Abstraction, that analyses an individual and extracts a placement signature that can be matched against other product models. The placement signature formalizes the set of elements that must be present in a product model in order to connect to the given model fragment. This is done using the boundary elements present in the individual. The process looks for those boundary elements that link an element from the model fragment with the rest of the product model and extracts them as a placement signature.

Fig 6.4 shows an example of the Placement Signature Abstraction. The left part shows an individual (a model fragment extracted from a product model). Then, the model fragment is interpreted as a CVL placement, and its boundaries are extracted into a placement signature. In this example, the model fragment only has one boundary, a provider channel (outside the model fragment) connects to a power manager (inside the model fragment). This information will be formalized

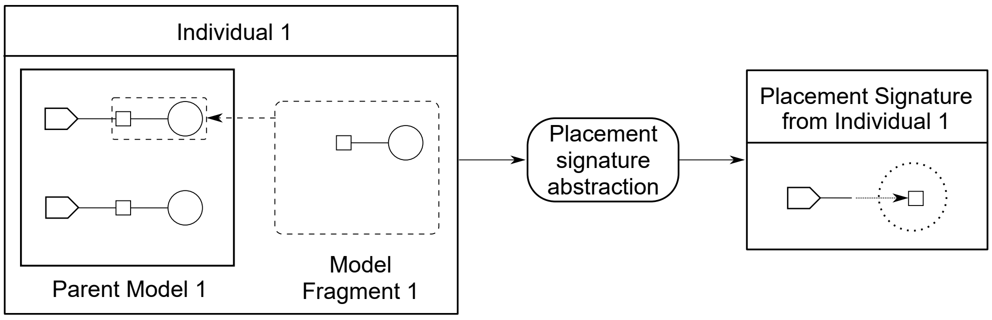

Individual 1
Parent Model 1
Model Fragment 1
Placement signature abstraction
Placement Signature from Individual 1

Figure 6.4: Placement Signature Abstraction

into the placement signature. The right part of Figure 6.4 shows the placement signature extracted. Dotted lines represent the the boundaries of the model fragment. As a result, a placement signature that can be matched against other models is obtained.

A placement signature is obtained for each of the individuals of the population. Then, placement signatures are compared and duplicates are grouped together. To do so, the process compares pairwise the placement signatures. If two placement signatures have the same elements in the boundaries, they are considered to be equal. Then, both placement signatures are grouped together. As a result, this step produces a set of unique placement signatures and each individual is associated to a single placement signature (a placement signature can represent several individuals).

6.3.2 Placement Signature Matching

Once the placement Signatures have been extracted, the next step is to match them against the product models of the family. That is, each placement signature is matched against the product models from the family used as model artifact. The product models are traversed in search for the elements present in the placement signature, if there exists a set of elements as described by the placement signature, it is considered a match. A match means that the placement could be used in that product model.

Figure 6.5) shows an example of the placement signature matching. The left part shows the placement being currently matched while top-right part shows the family of product models. The bottom-right part shows the different matches identified for each of the product models. For each match, the corresponding model fragment that matches is identified, so it can be extracted to build a variation

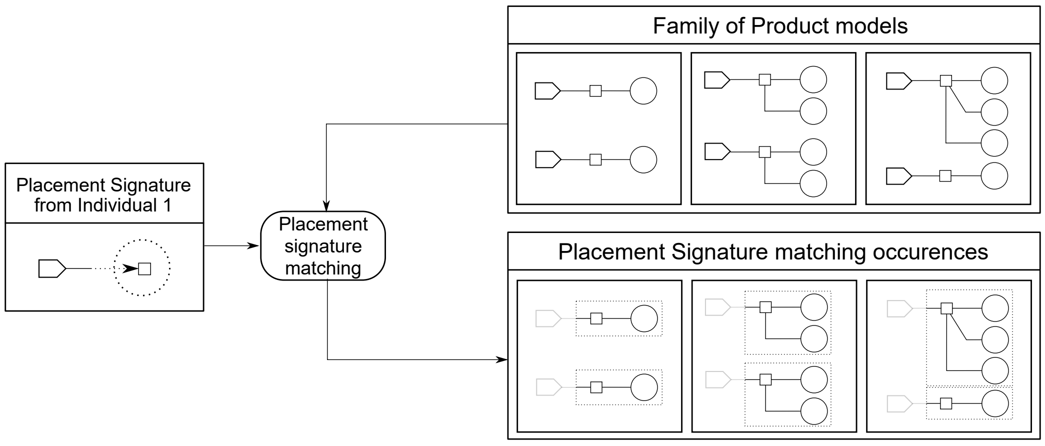

Family of Product models
Placement Signature from Individual 1
Placement signature matching
Placement Signature matching occurences

Figure 6.5: Placement Signature Matching

point. That is, the placement signature could be applied in several places and those places would be identified so a variation point could be built from them in case the genetic algorithm met the stop condition during that generation. As a result, the Placement Signature Matching provides a set of spots (across all the product models) where the given placement signature matches.

6.3.3 Fitness computation

Once the Placement Signatures matching occurrences have been computed for each placement signature, the next step is to build the variation point and assess each individual. To do so, the placement signature will be used as placement of the variation point and the matches across the product models will be the different alternatives for that particular placement. Then, a fitness value can be assigned to the elements of the variation point, taking into account the number of occurrences of that element all through the element. Each alternative score is the number of occurrences for that particular alternative, divided by the number of matches of the placement signature. For the placement signature, the score is the number of matches across all the product models divided by the number of product models.

Figure 6.6 shows an example of the fitness assessment. The left part shows the placement signature being assessed while the top-right part shows the placement signature matches identified in the last step. The bottom-part shows the resulting variation point and its fitness score. First, the placement signature is used as the placement and its score computed (in this case, there are 6 matches of that placement signature across 3 product models). Then, each of the alternatives are

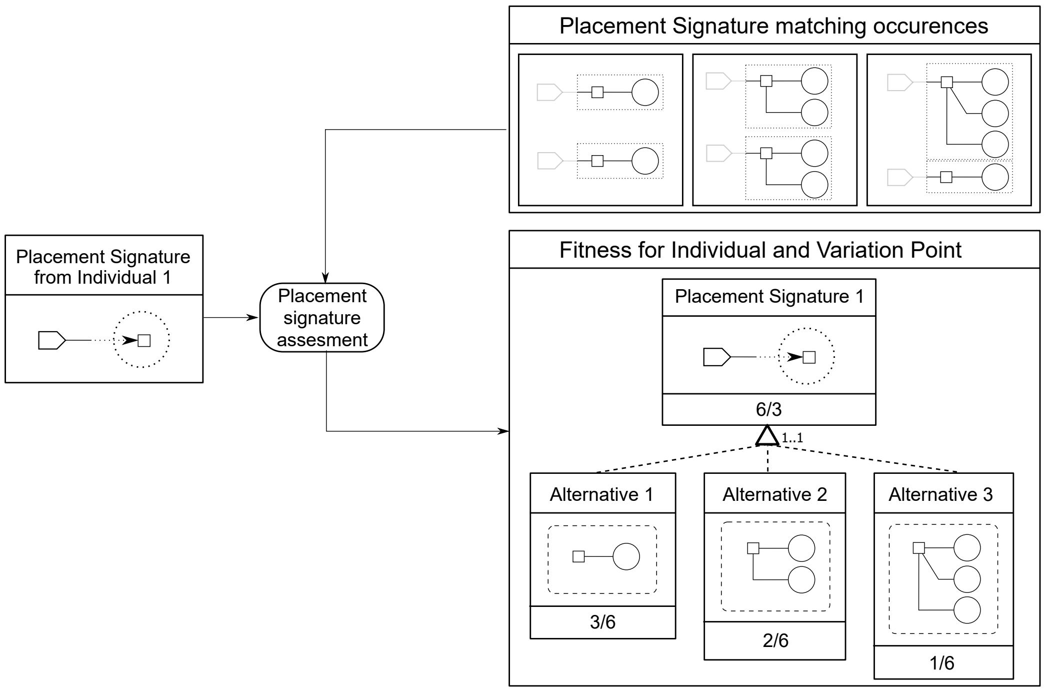

Placement Signature matching occurences
Placement Signature from Individual 1
Placement signature assesment
Fitness for Individual and Variation Point
Placement Signature 1
6/3
1..1
Alternative 1
3/6
Alternative 2
2/6
Alternative 3
1/6

Figure 6.6: Fitness Assesment and Variation Point construction

extracted as a model fragment and their scores computed. Alternative 1 has three occurrences, divided by the total number of 6 matches of the placement signature. Alternative 2 receives a score of 2 divided by 6 and Alternative 3 a score of 1 divided by 6.

In case the approach meets the stop condition at this point, the variation point will be presented to the user (along with other variations points obtained from the rest of individuals of the population). The fitness score for each of the elements of the variation point will be useful for the domain expert to ponder which alternate variation point should be used as feature realization. If the approach keeps iterating, the fitness score of the placement signature will be assigned to the original individual of the population that generated it and the process will continue.

6.4 Genetic Operations for Variation Points

This section presents the genetic operations designed to work over variation points: (1) a selection operator that chooses the best model parents based on the fitness previously calculated; (2) a crossover operation that will combine two individuals into a new individual; (3) a mutation operator that introduces random variations of the model fragment while keeping consistency with the parent model.

FLiMEA can locate features in the form of Variation Points, and to do so performs a search over a family of product models (to locate features as Model Fragments FLiMEA performs the search over a single product model). Therefore, the genetic operations must be designed having in mind that individuals are model fragments extracted from different product models; therefore, the crossover operation will be designed to combine two individuals with different parents resulting in a new individual that combines the model fragment from one individual and the parent model from the other individual.

6.4.1 Parent Selection: Different Parents

The parent selection used by FLiMEA to locate features as Variation Points will follow a strategy similar to the one used when locating features as Model Fragments. However, this time the operation needs to ensure that the two individuals do not share the parent model where the model fragments were extracted from. By doing so, the crossover operation will have the chance of swapping the parent of the first individual for the parent of the second individual (it is necessary to fulfil more conditions than having a different parent, but this condition is necessary and it is ensured at this step of the process).

Therefore, the first parent will be selected freely using the same strategy as before (roulette wheel selection). Then, to select the second parent an extra constraint will be imposed, the model from where the individual was extracted must be different from the first selected parent. To do so, after selecting the first parent, individuals extracted from the same parent are removed from the list of selectable elements, ensuring that a different one is selected next time.

6.4.2 Crossover: Parent change

In genetic algorithms, crossover enables the creation of a new individual generated by combining the genetic material of both parents. In our encoding there are two elements that can be mapped across the different individuals: the model fragment and the parent product model where the model fragment was extracted from. Therefore, our crossover operation will take the model fragment from the first parent and the product model from the second parent, generating a new individual that contains elements from both parents and thus preserving the basic mechanics of the crossover operation.

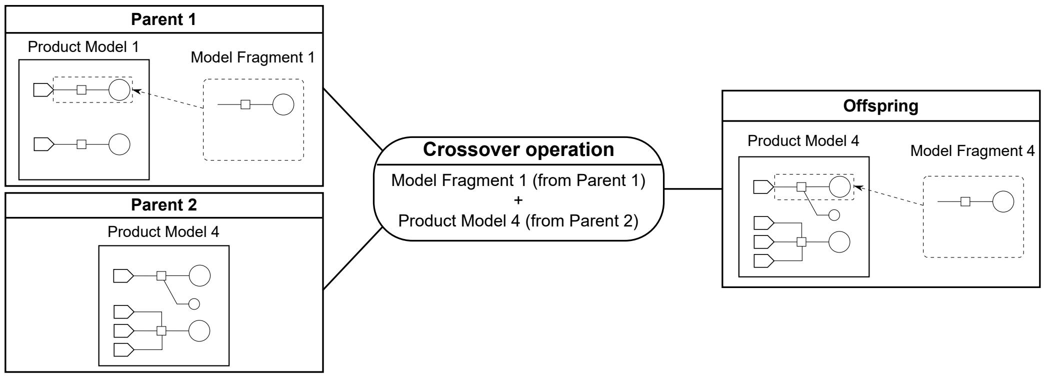


Figure 6.7: Crossover Operation

To achieve the latter, our crossover operation is based on model comparisons. Fig 6.7 shows an example of the application of the crossover operation over model fragments. First we select the model fragment from the first parent. Then we select the product model from the second parent. Then the model fragment (from first parent) is compared with the product model (from the second parent). If the comparison finds the model fragment in the product model, the process creates a new individual with the model fragment taken from the first parent but referencing the product model from the second parent. In the case that the comparison does not find a similar element, the crossover will return the first parent unchanged.

This operation enables to broaden the search space to a different product model. That is, both model fragments (the one from the first parent and the other from the new individual) will be the same. However, as each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations. The feature realization that the approach is looking for will be in the form of a variation point (combining elements from different product models). Therefore, the same variation point can be reached from different individuals (during the fitness step, the approach matches and combines different product models into a single variation point), but some individuals can yield to the solution faster than others.

6.4.3 Mutation: Sequential mutation

The mutation used by FLiMEA to locate features as Variation Points is similar to the random mutation presented to locate features as Model Fragments. However, this time the encoding is not binary, but Boundary-based and the operation is much heavier in terms of computation. Therefore, in order to mitigate the amount of computations done, the mutations are done sequentially instead of randomly. In addition, some constraints can be provided as part of the feature knowledge in order to further reduce the number of computations performed, avoiding model fragments that include model elements that the domain experts know should not be part of the resulting variation point.

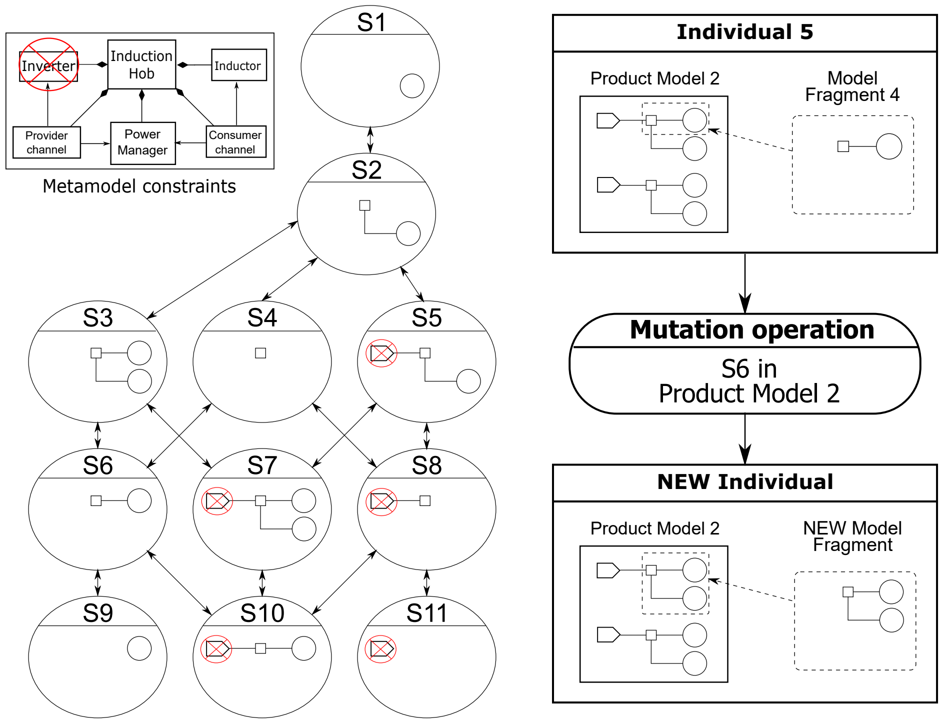


Figure 6.8: Sequential Mutation with constraints

The mutations are performed based on the model fragment and the product model. Taking the model fragment as starting point, some model elements are added to or removed from the fragment. However, the elements added during mutations are obtained from the product model, ensuring that the generated model fragment is part of the product model. In other words, apart from the individual model fragment, the process proposes other variations of that fragment that are also part of the product model.

The first step of the process is to build a state machine for the product model, that includes the different model fragments that can be extracted from that product model and the transitions from one to another in terms of mutations. This state machine will be used to drive the mutations in a sequential way, performing transitions from one state to another and avoiding previously visited states. In addition, states that do not fulfil the constraints provided as part of the feature knowledge, will be removed from the state machine.

The left part of Figure 6.8 shows an example of the state machine built for product model 2. It includes the different model fragments that can be extracted from that particular product model as different states of the state machine. In addition, there are transitions from one state to another indicating the valid mutations. Finally, some states (S5, S7, S8, S10, S11) have been discarded as they include elements that have been identified as non relevant for the feature being located (see top-left part of Figure 6.8 that shows the metamodel constraints provided as part of the feature knowledge).

The second step of the process is to match the model fragment of the individual being mutated with one of the states of the corresponding state machine (the state machine of the product model where the fragment was extracted from). Once it has been matched, the next state will be randomly chosen using the transition available at the current state.

The right part of Figure 6.8 shows an example of the mutation. Individual 5 is going to mutate and the current model fragment (Model Fragment 4) is compared with the states of the state machine (it corresponds with state 6). Then, the mutation continues and the transition to follow is randomly selected, in this case available transitions go to states S3, S4 and S9 (S10 is not valid as it includes an inverter). In this case the transition followed is to state S3. A new individual is created using the same product model of the original individual (product model 2) and the model fragment from the chosen state (S3 becomes the new model fragment).

6.5 Variability in FliMEA as Variation Points

Figure 6.9 shows a recapitulation of the different options presented for the Feature Location in Models as Variation Points process. The model artifact used by this process is a family of product models. The feature knowledge provided to the process consists of an initial model fragment seed, and optionally a set of constraints and a subset of the family of product models can be provided. The encoding presented was the Boundary-based encoding, where individuals are represented as placements, replacements and model fragments. For the individual assessment, Conceptual Model Patterns is used, based on the comparison of individuals. The

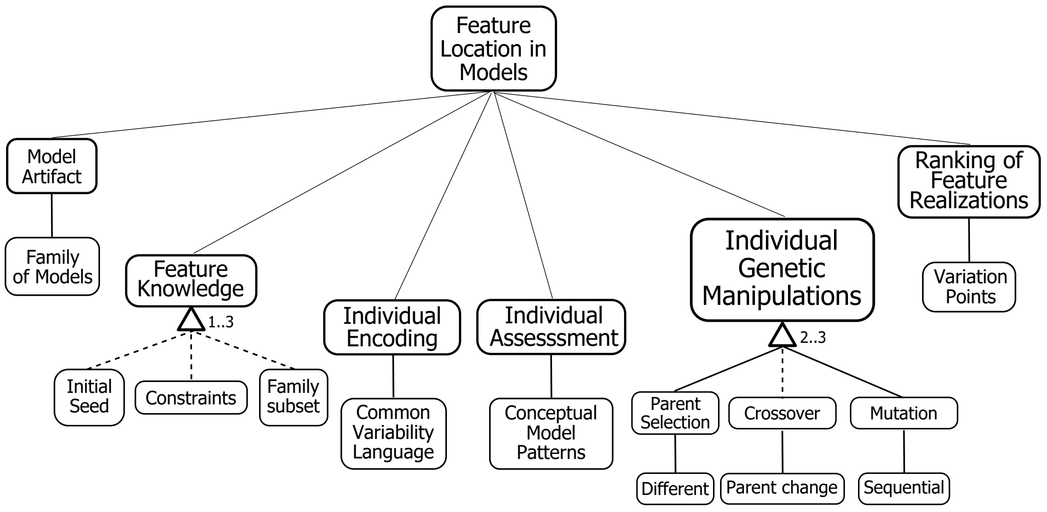


Figure 6.9: Variability of the Feature Location in Models as Variation Points process

individual genetic manipulation presented included a parent selection operation (different parents), an optional crossover operation (parent change) and a mutation operation (sequential). Finally, the resulting ranking of feature realizations will be in the form of Variation Points.

7

Evaluation of FLiMEA

Contents

7.1 Overview of the Chapter

This chapter presents the evaluations performed to test out FLiMEA, the approach proposed in this dissertation to address the problem of Feature Location in Models. We give an overview of the evaluation framework, describing each of the elements. First, we describe the two industrial case studies and the oracles extracted from them. Next, we explain how results are measured. Finally we present the results of the tree evaluations performed.

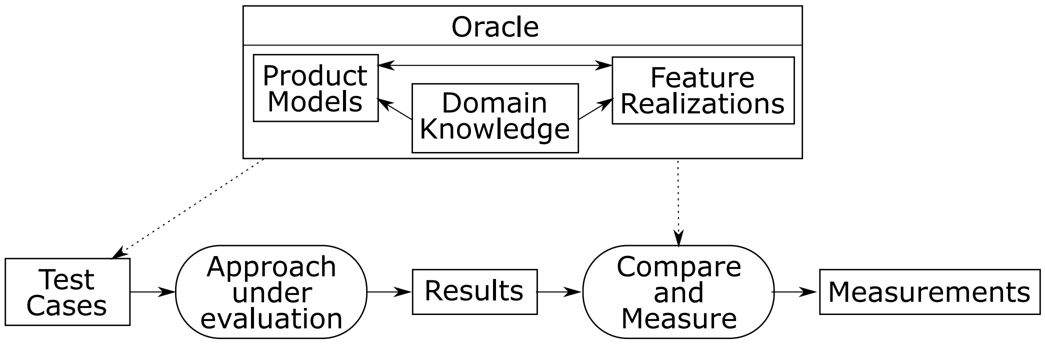


Figure 7.1: Setup of the evaluation

Figure 7.1 shows an overview of the generic setup of the evaluation, all the evaluations performed as part of this dissertation follow this scheme. The setup is composed of: (1) an oracle obtained from our industrial partner; (2) a set of test cases extracted from the oracle; (3) an approach (or approaches) that is being evaluated; (4) the set of results obtained when applying the approach to the test cases; (5) the measure (or measures) that we want to evaluate; and (6) the measurements obtained based on the results yielded by the approach and the information available from the oracle.

7.2 Oracle

The oracle is the mechanism that we will use to evaluate the results provided by our approach. The oracle will be considered the ground truth and the results provided by the approach will be compared (when needed) with the oracle in terms of the measures that we want to obtain. In addition the oracle will be used to obtain the test cases used for the evaluation.

The oracle will be mainly composed by a set of product models and a set of features located over those product models. That is, a set of features whose realizations are model fragments and a set of product models built using those model fragments. Therefore, we have the traceability information between the features, the model fragments realizing those features, and the features being used by each product model. In addition, the oracle includes domain knowledge in different forms, such as descriptions and technical documentations for each product model, descriptions about the features etc.

The oracle is extracted directly from the family of models of our industrial partner, that is being used to manage the products that are under production. Therefore, we consider it to be the best version available. However, when extracting the oracle we also have access to the domain experts from the company, who will validate the correctness of the oracle.

We have used two different oracles built from two different industrial domains: BSH the leading manufacturer of home appliances in Europe; and CAF, a worldwide provider of railway solutions. The next subsections present both case studies, providing details about the dimensions and nature of the features.

7.2.1 Induction Hob Domain

The first case study where we applied our approach is BSH (already presented in section 2.4.1 as the running example). Their induction division has been producing Induction Hobs under the brands of Bosch and Siemens for the last 15 years.

The oracle extracted from BSH is composed of 46 induction hob models where, on average, each product model is composed of more than 500 elements. The oracle includes 96 different features that can be part of a specific product model. Those features correspond to products that are currently being sold or will be released to the market in the near future.

7.2.2 Train Control and Management Domain

The second case study where we applied our approach was CAF, a worldwide provider of railway solutions. Their trains can be seen all over the world and in different forms (regular trains, subway, light rail, monorail, etc.). A train unit is furnished with multiple pieces of equipment through its vehicles and cabins. These pieces of equipment are often designed and manufactured by different providers,

Case Study	**Number of elements in DSL**	**Number of Features**	**Number of Product Models**	**Size of Product Models**
BSH	∼300 elements	96	46 IHs	∼500 elements
CAF	∼1000 elements	121	23 trains	∼1200 elements

Table 7.1: Overview of the oracles extracted from BSH and CAF

and their aim is to carry out specific tasks for the train. Some examples of these devices are: the traction equipment, the compressors that feed the brakes, the pantograph that receives power from the overhead wires, or the circuit breaker that isolates or connects the electrical circuits of the train. The control software of the train unit is in charge of making all the equipment cooperate in providing the train with functionality while guaranteeing compliance with the specific regulations of each country.

The DSL of our industrial partner has the required expressiveness to describe the interaction between the main pieces of equipment installed in a train unit. Moreover, this DSL also has the required expressiveness to specify non-functional aspects related to regulation, such as the quality of signals from the equipment or the different levels of installed redundancy. This results in a DSL that is composed of around 1000 different elements.

As an example, the high voltage connection sequence can be described using the DSL. This high voltage connection sequence is initiated when the train driver requests its start by using interface devices fitted inside the cabin. The control software is in charge of raising the pantograph to receive power from the overhead wire and of closing the circuit breaker so the energy can get to converters that adapt the voltage to charge batteries which, in turn, power the traction equipment.

Again, we extracted an oracle that is composed of 23 trains where, on average, each product model is composed of around 1200 elements. The product models are built using 121 different features that can be part of a specific product model.

Table 7.1 shows a summary of both oracles, providing details about the product models that are part of the oracle, the Domain Specific Languages used to build the product models and the features present in those product models.

7.3 Test Cases

A set of test cases is extracted from the oracle, so the approach under evaluation can be applied to them. Each test case must fulfill the input requirements of the approach, so they will vary mainly depending on the fitness function being used by the approach. However, each test case will be composed by a Feature Description and a set of model fragments where the feature should be located.

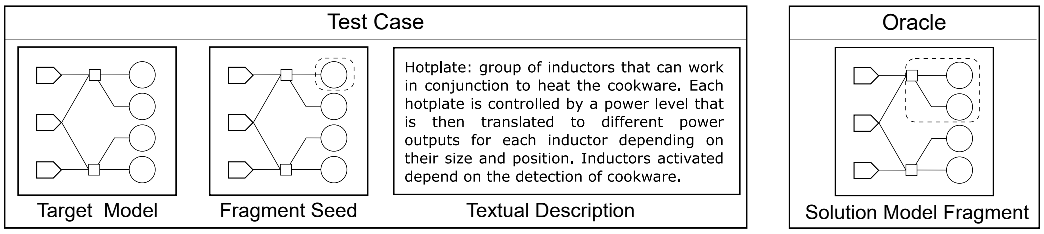

Test Case
Oracle
Hotplate: group of inductors that can work in conjunction to heat the cookware. Each hotplate is controlled by a power level that is then translated to different power outputs for each inductor depending on their size and position. Inductors activated depend on the detection of cookware.
Target Model
Fragment Seed
Textual Description
Solution Model Fragment

Figure 7.2: Test Case example

Figure 7.2 shows an example of a test case. It includes a feature description in the format required by the approach (in this case, a seed fragment and a textual description of the feature) and the target product model (where the feature will be located). In addition, the test case has been extracted from the oracle and there is a corresponding model fragment for that feature description (that will be used to compare with the output provided by the approach).

7.4 Approach under Evaluation

The presented FLiMEA can be configured depending on the needs of the user and the nature of the product models that will be used to locate the features. Figure 7.3 shows a feature model that describes the different elements that can be configured such as genetic operations, the fitness function or the input provided. This feature model corresponds to the different options explained in previous chapters.

For instance, FLiMEA can be configured to locate features over a single model (scope), to return the feature located as a single model fragment (output), requiring a single model and a textual description (input), and using random mutations, mask crossovers and LSA as fitness.

7.5 Comparison and Measure

Once the results from applying the approach to the test cases are obtained, we proceed to compare them with the oracle and measure them in terms of some software quality properties. Figure 7.4 shows an example of a model fragment from the oracle (left part), and two model fragment candidates obtained from the

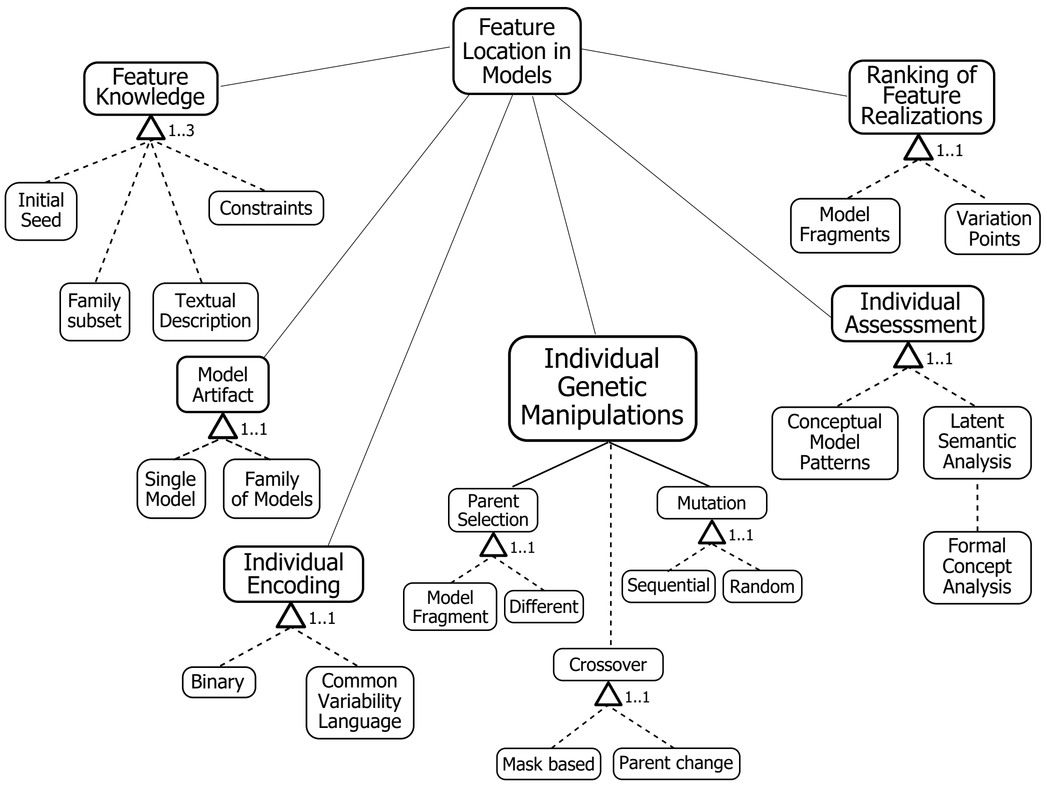
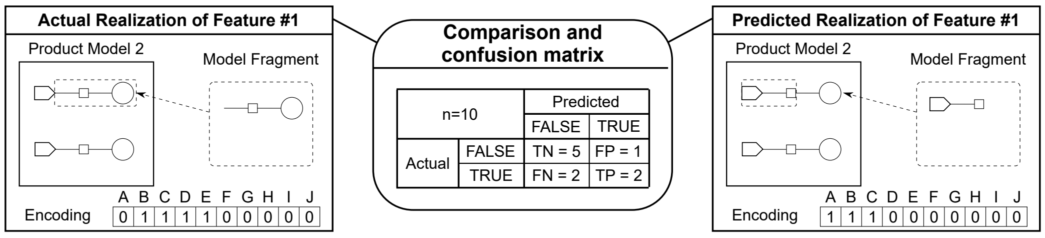


Figure 7.3: Feature Model of FLiMEA approach

application of the approach (middle part and right part). To compare them we are going to use an error matrix [161], also known as confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case our approach under evaluation) on a set of test data (the resulting model fragments) for which the true values are known (from the oracle). In this case, each feature realization returned by the approach is a model fragment composed of a subset of the model elements that are part of the product model (where the feature is being located). Since the granularity will be at the level of model elements, each model element presence or absence will be considered as a classification. Therefore, our confusion matrices will distinguish between two values (TRUE or presence and FALSE or absence).

Figure 7.4 shows an example of the comparison process performed to compare a result from one of the evaluated approaches with the ground truth from the oracle and the resulting confusion matrix. The left part shows the actual realization of the feature #1 (obtained from the oracle and considered the ground truth) while the right part shows the predicted realization of the feature #1 outputted by the


Actual Realization of Feature #1
Product Model 2
Model Fragment
A B C D E F G H I J
Encoding 0 1 1 1 1 0 0 0 0 0
Comparison and confusion matrix
Predicted Realization of Feature #1
Product Model 2
Model Fragment
A B C D E F G H I J
Encoding 1 1 1 0 0 0 0 0 0 0

n=10		Predicted	
		FALSE	TRUE
Actual	FALSE	TN = 5	FP = 1
	TRUE	FN = 2	TP = 2

Figure 7.4: Example of confusion matrix for two candidate model fragments

FLiMEA approach. The confusion matrix arranges the results of the comparison into four categories:

True positive (TP): A model element present in the predicted realization that is also present in the actual realization (e.g.: model element B is a TP).

True Negative (TN): A model element not present in the predicted realization that is not present in the actual realization (e.g.: model element H is a TN)

False Positive (FP): A model element present in the predicted realization that is not present in the actual realization (e.g.: model element A is a FP)

False Negative (FN): A model element not present in the predicted realization that is present in the actual realization (e.g.: model element D is a FN)

The confusion matrix holds the results of the comparison between the predicted results and the actual results; it is just a specific table layout to help the visualization of the performance of a classifier. However, in order to evaluate the performance of the approach it is necessary to derive some measurements from the values of the confusion matrix. The next subsection presents the four measurements that we use to evaluate the performance of our approach.

7.6 Measurements

In this subsection we present the three measurements (derived from the confusion matrices) used to evaluate the performance of the presented approach (FLiMEA). The three measurements are Precision, Recall and F-Measure.

Precision: measures the number of elements from the prediction (result of the approach) that are correct according to the ground truth (the oracle).

$$Precision = \frac{TP}{TP + FP}$$

Recall: measures the number of elements of the ground truth (the oracle) that are correctly retrieved by the prediction (result of the approach).

$$Recall = \frac{TP}{TP + FN}$$

F-measure: combines both recall and precision as the harmonic mean of precision and recall. The Recall, Precision and F-Measure are calculated as follows:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Recall values can range between 0% (which means that no single model element from the realization of the feature obtained from the oracle is present in any of the model fragments of the feature candidate) to 100% (which means that all the model elements from the oracle are present in the feature candidate).

Precision values can range between 0% (which means that no single model fragment from the feature candidate is present in the realization of the feature obtained from the oracle) to 100% (which means that all the model fragments from the feature candidate are present in the feature realization from the oracle). A value of 100% precision and 100% recall implies that both feature realizations are the same.

Following up with the example of confusion matrix in Figure 7.4, we can calculate the precision, recall and F-measure for the model fragment (see Table 7.2). The model fragment has a measurement of 66.7% precision (two out of the three elements included in the candidate model are present in the model fragment from the oracle) and 50% recall (2 out of the 4 elements that are present in the oracle are also present in the model fragment). This results in a combined F-measure of 57%.

	Precision	Recall	F-Measure
Values	$\frac{2}{3} = 66.7\%$	$\frac{2}{4} = 50\%$	$\frac{4}{7} = 57\%$

Table 7.2: Performance Measurements

7.7 Results

This section presents the results from the different evaluations performed as part of this dissertation. For each evaluation, it describes the particularities of the FLiMEA being evaluated and the research questions addressed.

7.7.1 Evaluation 1 (SPLC'15)

The first evaluation is part of the work SPLC'15 [11] (see 11.2). Figure 7.5 shows the configuration of the FLiM-EA for this evaluation. As input the approach receives a family of product models, an initial seed and a set of constraints. This approach relies on a sequential mutation as genetic operation, no crossover operation is used. The fitness function used is the Conceptual Model Pattern (CMP). As output, the approach will provide the features realizations in the form of a ranking of variation points.

The evaluation was designed to address two research questions:

SPLC-RQ1: Is feasible to locate features in industrial domains using the evolutionary algorithm presented so far?

SPLC-RQ2: What is the rationale followed by domain experts to perform the selection from the ranking of variation points outputted by the approach?

The approach was used to locate patterns with the collaboration of our industrial partner's engineers. We were able to obtain some patterns that satisfied the engineers, and therefore we conclude positively the feasibility of the approach to locate features in industrial domains.

In addition, we conducted a usability test, including a focus group around the rationale followed when selecting from the ranking. In overall, we obtained three reasons to select a model pattern different from the one proposed by the approach (the first on the ranking):

Odd elements: some patterns were automatically discarded when particular elements were found. This situation can be tailored and reduced though the use of the constraints.

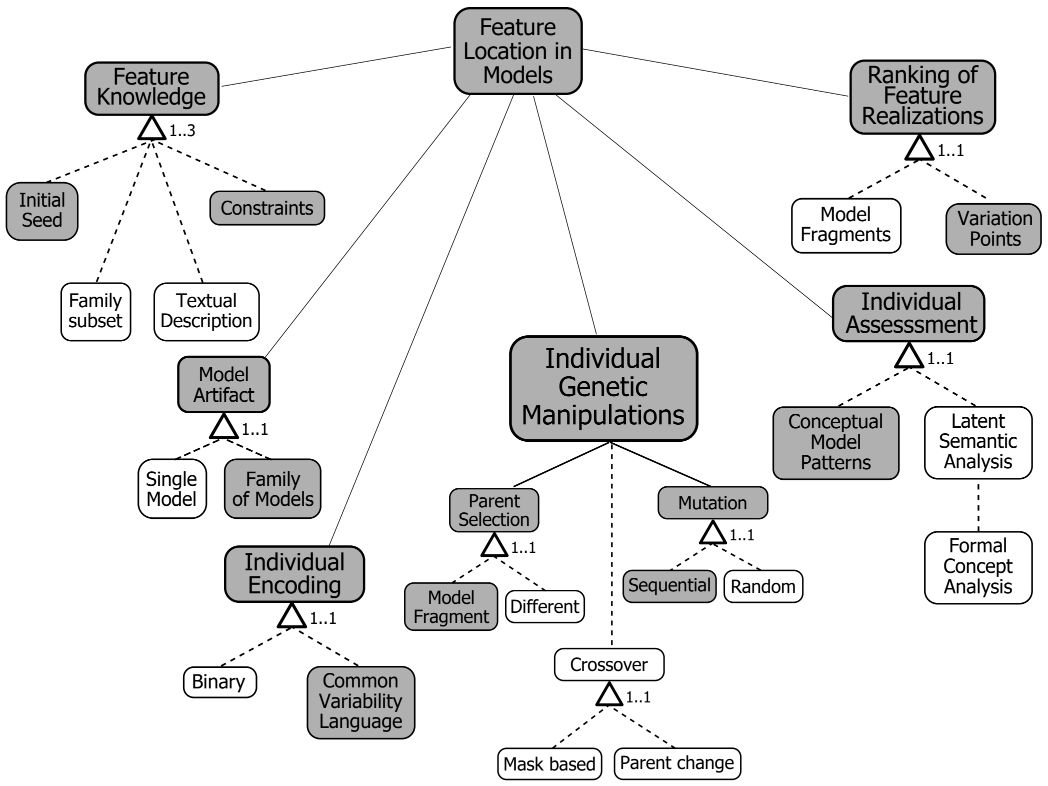


Figure 7.5: Configuration 1

Deprecated elements: Knowing that some elements would be deprecated in a near future was another reason for not selecting the option proposed by the approach.

Future developments: again, the humans selecting from the ranking have more information than the approach and use the knowledge about future developments when deciding the best option from the ranking.

7.7.2 Evaluation 2 (ICSR'16)

The second approach evaluated is part of the work ICSR'16 [12] (see 11.3). Figure 7.6 shows the configuration of the approach for this evaluation. As input the approach receives a family of product models, and an initial seed. This approach relies on a random mutation and a product model and fragment crossover as genetic operations. The fitness function used is the Conceptual Model Pattern (CMP). As output, the approach will provide the features location in the form of a ranking of variation points.

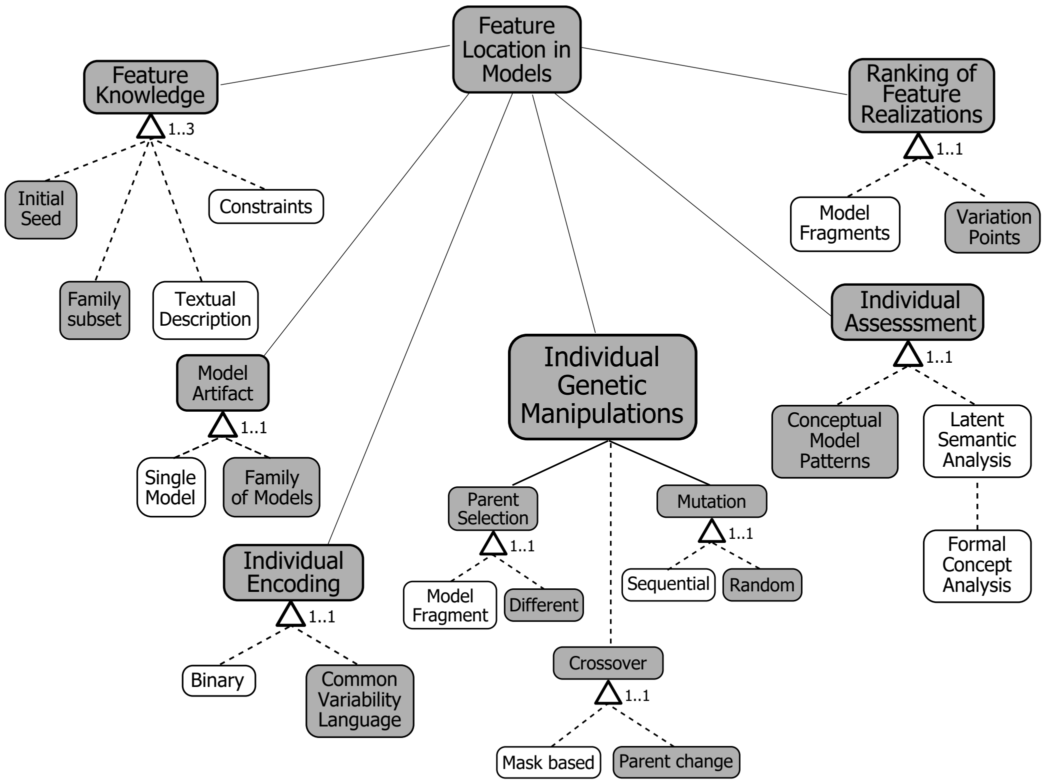


Figure 7.6: Configuration 2

The evaluation was designed to address one research question:

ICSR-RQ1: What is the impact of the accuracy when providing the input for the approach?

To address the **ICSR-RQ1**, we created a set of test cases including three different scenarios regarding the accuracy of the input provided:

High accuracy: more than 75% of the product models provided as input contain the feature being located.

Medium accuracy: between 25% and 75% of the product models provided as input contain the feature being located.

Low accuracy: less than 25% of the product models provided as input contain the feature being located.

Then, the test cases are fed as input to the approach and also to the previous approach 7.5 and results are compared. It turns out that the inclusion of the crossover operation helps mitigate the bad results obtained with low accuracy inputs. Table 7.7.2 shows a summary of the results from both approaches. Each cell indicates the percentage of times where the correct (from the oracle) feature formalization was included into the ranking provided as output.

	ICSR'16	SPLC'15
high accuracy	79%	86%
medium accuracy	73%	48%
low accuracy	63%	16%

Table 7.3: Comparison between ICSR'16 and SPLC'15 based on input accuracy

7.7.3 Evaluation 3 (MODELS'16)

The third approach evaluated is part of the work MoDELS'16 [13] (see 11.4). Figure 7.7 shows the configuration of the approach for this evaluation. As input the approach receives a single product model, a textual description and an initial seed. This approach relies on a random mutation and a mask crossover as genetic operations. The fitness function used is the Latent Semantic Analysis (LSA) combined with Formal Concept Analysis (FCA). As output, the approach will provide the feature realization in the form of a ranking of model fragments.

The evaluation was designed to address three research questions:

MODELS-RQ1: Does the LSA+FCA fitness help when guiding the process or tampers it?

MODELS-RQ2: Does the selection of the seed have an impact on the results?

MODELS-RQ3: Does the selection of the textual description have an impact on the results?

To address the **MoDELS-RQ1**, a set of test cases was obtained from the oracle and then fed to the approach. In addition, a baseline fitness function was created (based on random) and the test cases fed to an approach using this fitness. Then, results from both fitness functions where compared and analysed. The LSA+FCA fitness prove to be able to guide the search, and results where not due to mere

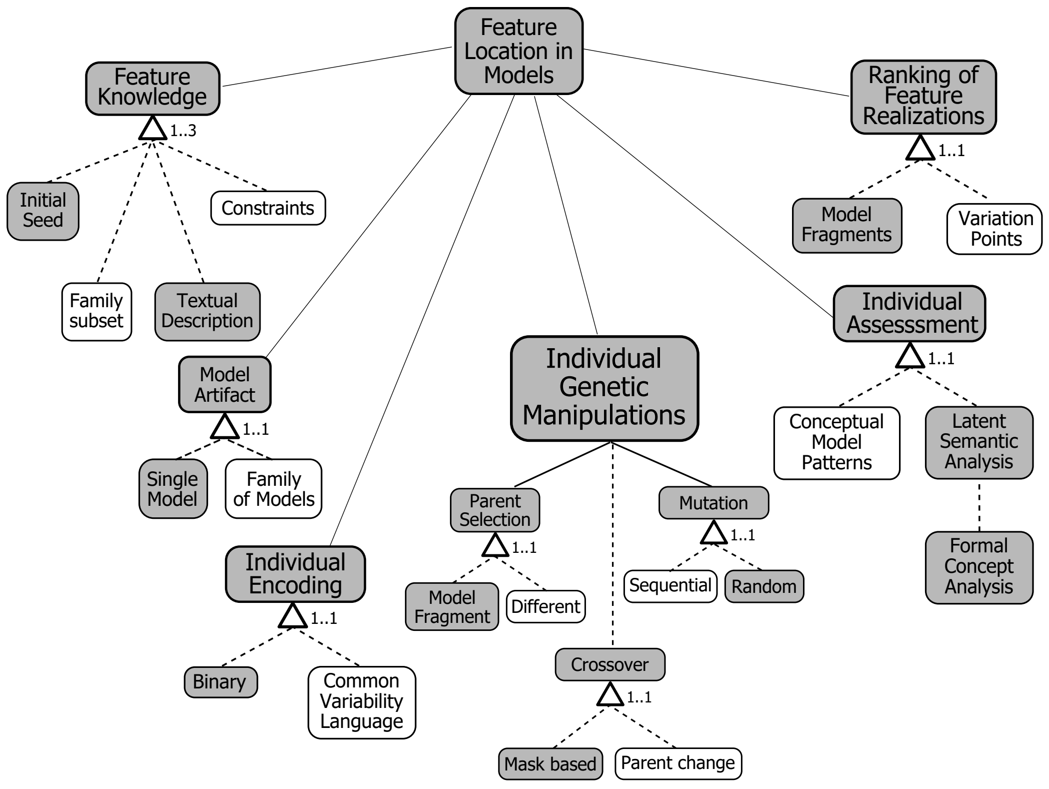


Figure 7.7: Configuration 3

chance. The approach was able to provide mean values of Precision, Recall and F-measure between 80% and 90%.

Figure 7.8 shows the mean precision and recall values measured for the 96 features located by both executions (FLiMEA and the Baseline). The top chart shows the results for the execution of FLiMEA while the bottom part shows the results for the Baseline. The values for the recall measure are in blue, the values for the precision measure are in red and the values for the F-measure are in black (in both charts). Each measure includes the standard deviation (shaded in the same color). The x axis of the charts indicates the number of generations of the evolutionary algorithm while the y axis measures the % value of the recall, precision and F-measures.

Each of the lines corresponds to the mean values for the location of the 96 test cases obtained from the oracle. First, we have calculated the values for each of the test cases (including all the feature candidates from their rank). Then, mean values and standard deviations for the 96 test cases have been calculated.

The recall values for the presented approach (top chart blue line) start in a

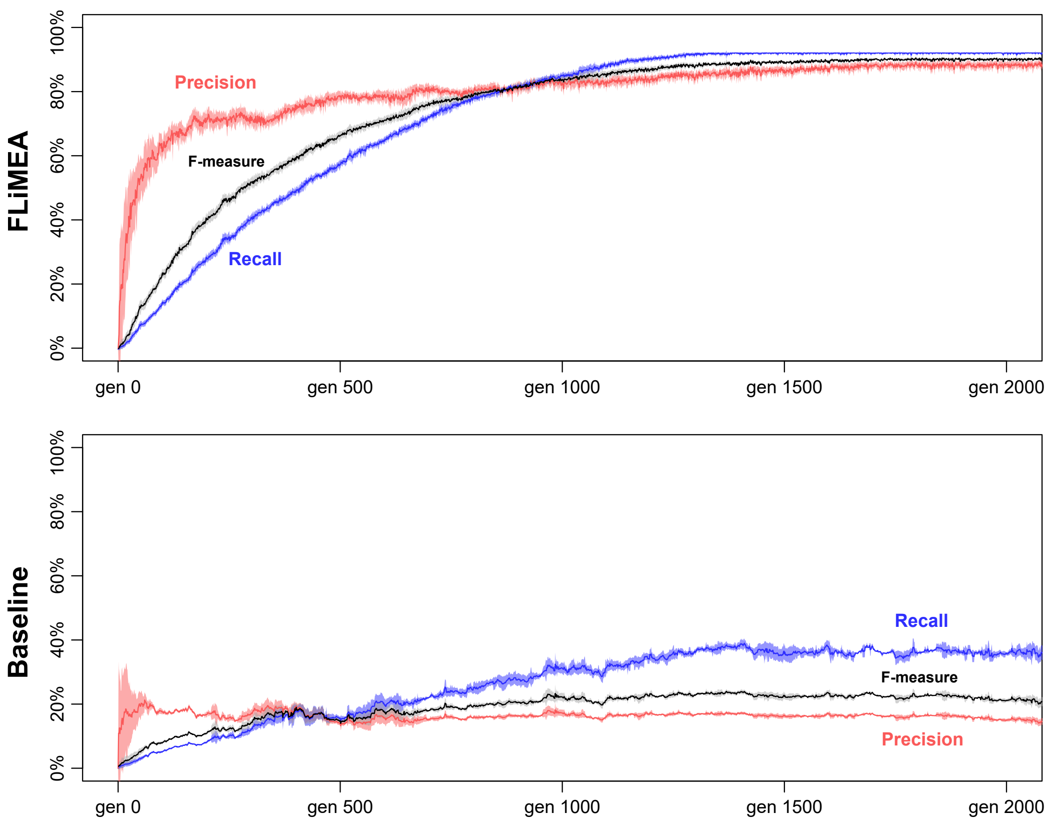


Figure 7.8: Mean Precision, Recall and F-measure for FLiMEA and the Baseline

range between 0% and 20% for the first hundreds of generations but then start raising up to the 90% (around generation 1.400). Beyond generation 1.400, the recall values keep close to 100%. The precision values for the presented approach (top chart red line) start in a range between 0% and 60% for the first hundreds of generations. Then, the precision values raise up to the range between 80% and 90% (around generation 1.500), beyond that generation there are no further changes in the tendency.

The recall values for the Baseline (bottom chart blue line) start in a range between 0% and 20% for the first hundreds of generations. Then, the recall values reach the range between 30% and 40% (around generation 1.400) and oscillate in that range for the rest of the generations. The precision values for the Baseline (bottom chart red line) raise sharply to 20% and then drop slightly to a value around 15%, remaining steady for the rest of generations.

Overall, results show that the use of IR techniques as the fitness function of the GA (our approach) guides it to locate the feature better than if a random guide is provided (Baseline). The comparison with the oracle enables to obtain the recall and precision values for both approaches and the IR provides higher mean values of precision and recall for any number of generations.

To address the **MoDELS-RQ2**, a set of test cases including different seeds were created. The seed was varied in terms of accuracy (the elements selected are part of the feature being located) and size (number of elements in the seed). Then, test cases were executed and results analysed. When the seed provided contained about 50% of the elements that are part of the feature being located, the time needed by the approach was reduced up to 15%.

To address the **MoDELS-RQ3**, a set of test cases including different textual descriptions were created. The textual description was varied in terms of size and the type of terms being used to build it. Then, test cases were executed and results analysed. When the textual description includes terms that are also names of meta-classes, the results include several elements that are not relevant. When avoiding the use of those terms, in favour of more specific terms (such as terms used as values for the properties of the elements), precision values raised up to 20%.

Part III

Evolution of Model Fragments

8

VARIABLE METAMODEL (VMM)

Contents

8.1 Overview of the Chapter

This chapter focuses on the maintenance and evolution of a variability specification based on model fragments and is based on published work [15, 16]. In the previous part we have explored how to locate the features among a set of product models in the form of model fragments and variation points. However, that variability formalization represents the variability of a family of products in a given point in time. Software evolves and that is also the case of the model fragments used to realize the features. Therefore, this chapter focuses on the model and language co-evolution problem.

In Section 3.3.4 we analyzed the existing state-of-the-art solution (model migration) and identified three issues related to its application. In this chapter we present a retrospective case study of the variability evolution among the induction hobs from our industrial partner. Then, we propose an alternative (based on variability modeling applied at meta-model level) to the model and language co-evolution that does not involve those issues.

8.2 Retrospective Case Study

This section presents the retrospective case study that was extracted from the evolution of our industrial partner's (BSH, see 7.2.1) models and metamodels over the last 13 years. Although the evolution data provided involves all the elements present in the initial DSL, for simplicity and due to intellectual property rights, we are going to focus on the evolution related to the inductor concept.

Let MM be the set of all models that conform to the MOF language (i.e., the set of all metamodels) and let M be the set of all models. Let m_i (the index i will be explained shortly) be in M and let mm_i be in MM. Then, we say that a model (m_i) conforms to a metamodel (mm_i) if it is expressed by the terms that are encoded in the metamodel; this conformance is denoted as $C(m_i, mm_i)$.

Let $CVLSPL$ be the set of all CVL-based product lines. One such product line, $cvlspl_i$, is denoted as follows:

$$\begin{aligned} CVLSPL =& \ MM \times M \times M \\ cvlspl_i =& < mm_i \ , \ b_i \ , \ l_i > \end{aligned} \tag{8.1}$$

where mm_i is the metamodel of the DSL (conforming to MOF), b_i is the base model (over which placements for the variable parts are defined), l_i is the li-

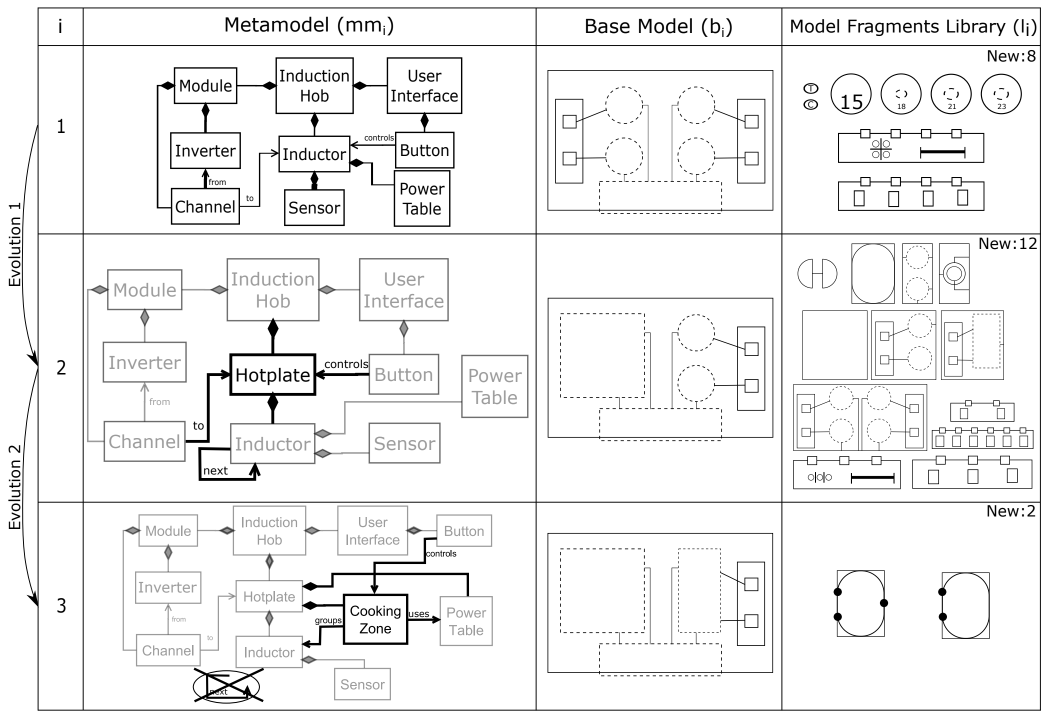


Figure 8.1: Model Generations of the CVLSPL

brary of replacements for those placements, and the conformance between models $C(b_i, mm_i)$ and $C(l_i, mm_i)$ is fulfilled. In addition, let i be a consecutive index that is assigned based on when models and metamodels are created, i.e., we will refer to the *generation* i of the base model, the *generation* i of the metamodel, the *generation* i of the library, and the *generation* i of the CVLSPL. The use of the index i is only as an annotation to identify the generation. For each generation there may be several base models and libraries adhering to the same metamodel.

We perform a $CVLSPL$ evolution (a shift from one $cvlspl_i$ generation to the next generation, $cvlspl_{i+1}$) whenever there is a *breaking and unresolvable change* (from now on *breaking change*) [124] in the metamodel. Breaking changes break the conformance of models and the metamodel in a way that cannot be resolved by automatic means [124] (e.g., the addition of a mandatory meta-class or a restriction in the multiplicities). There are other metamodel changes that do not break the conformance of models and the metamodel (e.g., the addition of an optional class) or metamodel changes that can be resolved automatically by existing approaches [121, 119, 124, 162, 118] (e.g., eliminating a property). However, in this dissertation we focus on the evolution triggered by breaking changes.

Figure 8.1 shows a summary of the CVLSPL generations and the evolutions performed. Specifically, we present three CVLSPL generations: the first row shows $cvlspl_1$, which includes the concept of inductor; the second row shows $cvlspl_2$, which includes the concept of Hotplate; and the third row shows $cvlspl_3$, which includes the concept of cooking zone. This figure shows the breaking changes that were overcome by our industrial partner, such as the addition or removal of meta-elements. The first column shows the metamodel for each generation, the second column shows the base model, and the third column shows the replacements library. The full variability specification is not shown, but the shape of each placement in the base model and the shape of each replacement in the model fragments library are indicators of which placements can be substituted by which replacements.

Evolution 1 (from $cvlspl_1$ to $cvlspl_2$) is triggered by a new concept called Hotplate (see the first and second rows of Figure 8.1)

MM level : A Hotplate consists of a group of inductors that can work together. There is a hierarchy (*next* relationship) among the inductors; some must be turned on before their subordinates are turned on. Since we need to control the whole Hotplate (two inductors) with just one user interface controller, the controller will now act over hotplates instead of inductors. This is reflected in the metamodel mm_2 (see the second row, first column).

Model level : There are also modifications at the model level. A new placement is created over the base model b_2 to enable substitutions of the new hotplate replacements. In addition, new replacements (l_2) that instantiate the hotplate concept are created; for example, the split hotplate (formed by two inductors, one main and one auxiliary) or the double hotplate (formed by two inductors, requiring twice the space and power as the rest of hotplates).

Evolution 2 (from $cvlspl_2$ to $cvlspl_3$) is triggered by a new concept called cooking zone (see the second and third rows of Figure 8.1).

MM level Cooking zones improve the hotplate by introducing the ability to heat two different pieces of cookware at the same time and with different power levels. Now each hotplate will have cooking zones, which will be controlled by the user interface controller. Since the number of combinations of inductors that are working at the same time increases, the power table is now aggregated by the hotplate, and the cooking zones use it. By means of this modification, several hotplates will share the same power tables (when the inductor configurations are equivalent). Furthermore, the hierarchy that is present among inductors is now controlled by the cooking zone (one cooking zone having the main inductor and another cooking zone having both inductors); therefore, the relationship *next* is removed from the metamodel (mm_3).

Model level A new placement to include hotplates on both sides is created over the base model b_3. Similarly, new replacements that exercise the new concept of cooking zone are created (l_3). For instance, the pool hotplate has four inductors that are divided into two different cooking zones, which are controlled by two different buttons.

8.3 The Variable MetaModel (VMM)

As a result of applying the migration strategy three issues arise (Overhead, Automation and Trust Leak see Section 3.3.4). In this chapter we present our proposal for addressing the co-evolution in model-based SPLs whose variability is realized through model fragments, the Variable MetaModel (VMM). In order to eliminate the need for migration when a new generation (metamodel revision) is created by the engineers, a new metamodel that supports both generations can be automatically built: the VMM. For instance, models that conform to generation 1 and models that conform to generation 2 will also conform to this VMM. A model that contains replacements from both generations will conform to the VMM. Since the VMM will be enhanced each time a new generation is created, a single VMM that includes all generations of the CVLSPL will exist.

The VMM is the result of applying variability modeling ideas at metamodel level. In this scenario the mechanism used to specify the variability at metamodel level will be CVL; we have a base model in a given DSL (in this case, MOF) with placements defined over it and a library of replacements. VMM is defined as follows:

$$\begin{gathered} VMM = MM \times MM \\ vmm_i = < mmb_i \, , mml_i > \end{gathered} \tag{8.2}$$

where mmb_i is the base model at the metamodel level and mml_i is the library of

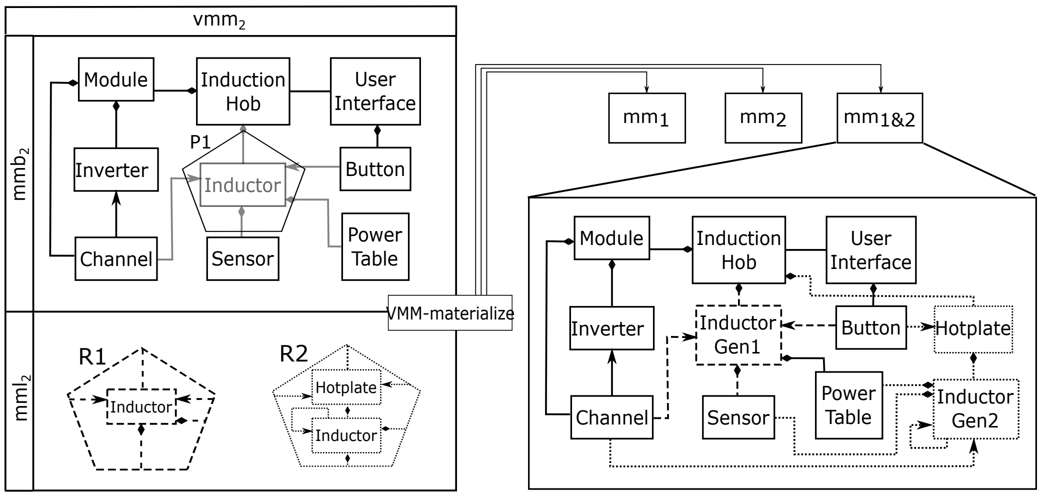


Figure 8.2: VMM and VMM-materialize

replacements at the metamodel level.

The vmm_i hold all metamodel variations from starting generation (generation 1) to generation i. Similarly to CVL at the model level, we can materialize models that conform to the given DSL (in this case, MOF). Let G be the set of all generations and let $\mathcal{P}(G)$ be its power set. We define the $VMMmat$ (VMM Materialization) operation as follows:

$$\begin{aligned} VMMmat\colon \quad & VMM \quad \times\mathcal{P}(G)\longrightarrow MM \\ VMMmat(< mmb_i, mml_i > , \quad & g) \quad = mm_g \\ & where g \neq \emptyset \end{aligned} \tag{8.3}$$

That is, given a vmm_i where i generation is included in G and selecting a non-empty generation set g, $VMMmat$ retrieves the mm_g for the $cvlspl_g$ of the given generation set g.

Figure 8.2 (left) shows an example of VMM, the vmm_2 for generation 2. The top-left corner shows the base model (mmb_2). It is the metamodel from $cvlspl_1$, with a placement (P1) defined over the inductor. The bottom-left corner of Figure 8.2 shows the replacement library (mml_2), which contains two different replacements: R1 (in dashed lines) defined over the $cvlspl_1$ metamodel and R2 (in dotted lines) defined over the $cvlspl_2$ metamodel.

Figure 8.2 (right) shows the models produced with the vmm_2 presented. The materialization of CVL produces models that conform to the same language that the base model and replacements conform to; therefore, in this case the models produced will conform to MOF. With the library that is available (two replacements), we can produce three different models: 1) mm_1 (the metamodel of $cvlspl_1$) with a substitution of P1 by R1; 2) mm_2 (the metamodel of $cvlspl_2$) with a substitution of P1 by R2; and 3) $mm_{1\&2}$ (a new metamodel with the concepts from the mm_1 and the mm_2 metamodels) with the substitution of P1 by R1 and P1 by R2.

The cardinality property of placements in CVL enables the creation of $mm_{1\&2}$. In other words, one placement can be substituted more than once (the number of times can be specified). The first time that a placement is substituted, the existing references of the placement are replaced. The second time that the same placement is substituted, the same references are used (the multiplicity must be *many* to allow this). In Figure 8.2, the aggregation of Inductors in vmm_2 is duplicated into an aggregation of Inductor Gen1 (in dashed lines), and an aggregation of Hotplate (in dotted lines) in the $mm_{1\&2}$. We have limited the substitution of the same replacement several times as the result will be the same as replacing it only once ($mm_{1\&1}$ produces the same metamodel as mm_1).

The $mm_{1\&2}$ metamodel contains concepts from both $cvlspl_1$ and $cvlspl_2$ at the same time. To achieve this, VMM renames the elements that conflict (e.g., Inductor from mm_1 and from mm_2). The advantages of this $mm_{1\&2}$ is that any model that conforms to mm_1 also conforms to $mm_{1\&2}$ and any model that conforms to mm_2 also conforms to $mm_{1\&2}$. In other words, $mm_{1\&2}$ is used when materializing IH models that contain replacements from both libraries (l_1 and l_2), and the resulting model conforms to $mm_{1\&2}$. When combining replacements from different generations into the same product, unexpected interactions between them might arise. However, dealing with feature interactions is not straightforward and there are several works focusing on this topic (such as [163]); thus, feature interactions will be left out of the scope of this paper.

The vmm_2 enables the materialization of mm_1 and mm_2 that are used directly by the engineers to create new replacements. By doing so, the replacements created will conform to a specific generation and will not include unnecessary indirection. If the functionality required for a particular replacement can be achieved with the expressiveness of a previous generation, that metamodel will be used.

Furthermore, if the engineers try to create new replacements using the $mm_{1\&2}$ directly, they could end up creating models that do not conform to either mm_1 or to mm_2. Therefore, we need to keep the original metamodels (mm_1 and mm_2) in order to restrict the creation of new replacements.

8.4 VMM operations

There are two main operations in relation to the VMM: the initialization of the VMM and the addition of new metamodel revisions. Both operations are capable of spotting the commonalities and variabilities among metamodels and formalizing them in terms of CVL. The initialization is executed only one time, to generate the initial VMM. Then, the addition of new metamodel revisions is performed each time a new revision is created. The following subsections present both operations in detail.

8.4.1 InitVMM operation

Figure 8.3 shows an example of the initVMM operation. InitVMM receives two metamodel revisions (e.g., Metamodel A and Metamodel B) as input and produces a VMM that includes both generations as output. Either of the two metamodels can be used as the base model and will lead to valid CVL specification of the metamodels provided. Different base models result in different model fragments, which are used to specify the variability. This can be highly relevant when there are users that interact directly with the model fragments [11], but it is not important for the VMM approach since those model fragments will be managed automatically. Therefore, one revision is randomly selected as the base model (in this example, Metamodel A); we will refer to the other metamodel revision as the new revision (in this example, Metamodel B).

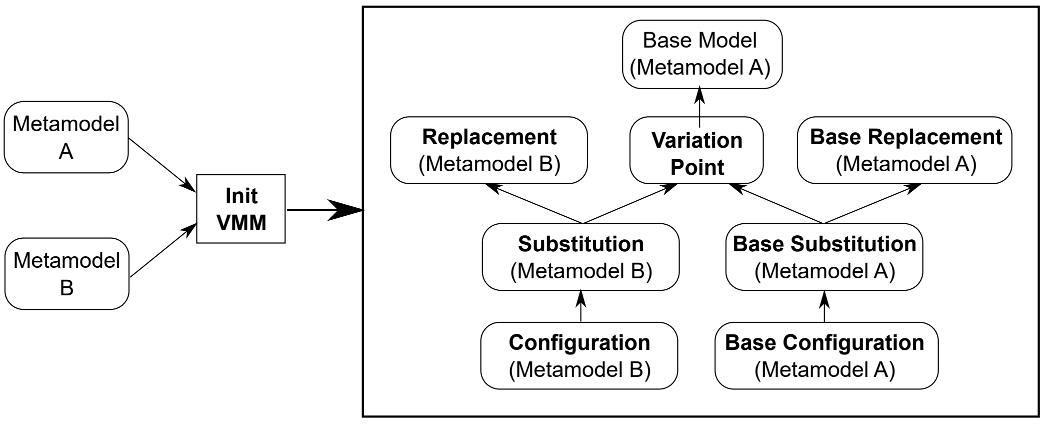

Metamodel A
Metamodel B
Init VMM
Base Model (Metamodel A)
Replacement (Metamodel B)
Variation Point
Base Replacement (Metamodel A)
Substitution (Metamodel B)
Base Substitution (Metamodel A)
Configuration (Metamodel B)
Base Configuration (Metamodel A)

Figure 8.3: InitVMM operation

Then, the operation follows a five-step process to generate the VMM. The aim of this process is to formalize the commonalities and particularities of each metamodel revision in terms of CVL (placements, replacements, substitutions, and configurations). To do this, the operation will perform comparisons between the Base Model (Metamodel A) and the new revision provided as input (Metamodel B):

1. **Compare**: Metamodel B is automatically compared with the base model. The result is a list of differences between the two revisions. Each difference is composed of two elements (the differing element from the base model and the differing element from the new revision (Metamodel B)). Then, each of the differences is processed and formalized as CVL elements as follows:
   (a) **Placement**: A placement is created over the base model (if that placement does not previously exist). Using the information from the comparison, the boundaries of the placement are generated accordingly.
   (b) **Replacement**: A replacement that formalizes the differences between the base model and the new revision must be created. A replacement holding the particularities of Metamodel B is created; this replacement will turn the base model into the Metamodel B. As with the placement, we use the information from the comparison to determine the boundaries of the replacement.
   (c) **Substitution**: Once a placement and a replacement have been created the process generates a substitution. That is, the boundaries of the placement and the replacement are mapped accordingly, so the placement can be substituted by the replacement.
2. **Configuration**: The process is repeated for all of the differences obtained in Step 1. Finally, a configuration is defined, specifying what substitutions need to be executed to turn the base model (Metamodel A) into the new metamodel (Metamodel B).

As a result of this process, the commonalities and variabilities among the new revision (Metamodel B) and the base model (Metamodel A) are formalized in terms of CVL and thus, there are replacements holding the particularities of the new revision. However, the VMM also needs to capture the particularities of the metamodel revision that is used as the base model in separate fragments. Therefore, each time a new placement is generated over the base model, Steps 1.(b), 1.(c) and 2. will be replicated to generate the CVL specification for the metamodel revision that is used as the base model:

   (b) **Base replacement**: The process needs a replacement that formalizes the particularities of the base model. Therefore, all the elements included in the placement will be included in a new replacement. This replacement holds the particularities of Metamodel A and will be necessary to generate combined metamodels (joining two revisions).

   (c) **Base substitution**: As previously, we need to map the placement and the replacement boundaries so that the substitution can be properly executed. The execution of this substitution might seem unnecessary since the result would be the same base model (in this example, Metamodel A); however, the replacement and substitutions generated will be necessary when generating metamodel revisions that make use of different generations.

2. **Base configuration**: Finally, a new configuration describing the substitutions needed to generate that revision from the base model is generated. Again, this may seem redundant, but it is done this way to keep the consistency and explicitly formalize which replacements and substitutions belong to that particular revision (regardless of whether the revision is being used as the base model or not).

In summary, the initVMM operation formalizes a metamodel revision in terms of CVL, generating placements, replacements, substitutions, and configurations as needed. The first time it is executed, it also formalizes the base model in terms of CVL, so all of the metamodel revisions are formalized in terms of CVL independently of the revision used as base model.

Figure 8.2 (left) shows an example of the result of initVMM applied to the induction hobs. Two different revisions, mm_1 and mm_2, were used as input. Then, mm_1 was selected as base model (mmb_2), a new placement was created (P1), and then two replacements (mml_2) were generated to formalize the particularities of each revision (R1 to formalize mm_1 and R2 to formalize mm_2). In addition, the cardinality of the placement was updated accordingly as there were two substitutions using that placement (the configurations do not have a graphical representation in the figure).

8.4.2 AddGen operation

Once the VMM has been created, following the initVMM operation, it is necessary to have an operation to include new metamodel revisions in this VMM. This is accomplished by the addGen operation. The operation receives a VMM and a new metamodel revision as input and returns an extended VMM that includes the new revision.

The operation proceeds similarly to the initVMM operation; however, this time there is only one metamodel that will be compared with the base model (it is not necessary to capture the base model as separate fragments). Furthermore, the addition of new metamodel revisions can result in the reutilization of already existing placements. In other words, when creating a new placement as part of Step 1.(b) Variation Point, the placement may already exist and there is no need to create a new one. The same placement will be used and its multiplicity will be increased. By doing so, we will enable the materialization of models that combine several generations.

As a result of this operation, new placements, replacements, substitutions, and configurations are automatically created to formalize the new metamodel revision. The resulting VMM will now include the new metamodel and it will be possible to materialize it as a single generation metamodel or as part of a metamodel that combines several generations.

Both operations (InitVMM and AddGen) are automatic processes and there is no need for human assistance to run them. The first time that a new metamodel revision is generated, initVMM will be executed and the following times, addGen will be executed. The comparisons performed by the operations have been implemented based on the EMF Compare Framework [164]. This framework provides functionality to compare EMF (the implementation of MOF within the Eclipse environment) models and can be customized to perform the comparisons based on different criteria. In our case, we compared models at the finest level of granularity capturing any change from one revision to the next (e.g., the addition of a property or even a change in the name of a class).

9

EVALUATION OF VMM

Contents

9.1 Overview of the Chapter

This chapter presents the evaluation performed to test out the Variable Meta-Model, the approach presented to address the problem of model and language co-evolution. First we analyze the three issues entailed by the migration and how the VMM behaves in relation them. Then we present a set of Lesson Learned from the application of the presented approach to address the co-evolution of our industrial partners' models and languages.

9.2 Migration Issues in VMM

We have applied both strategies (migration and the VMM) to the retrospective of 13 years of our industrial partner's SPL models. In this dissertation, we only show a simplification of the evolution related to the inductor concept even though we have applied it to all of the concepts. This involves about 32 different IH models composed of approximately 72 different model replacements (each of which is composed of multiple model elements). The average number of model elements of a model fragment replacement is 43, while the average number of elements of an IH model is about 470. Figure 9.1 shows a summary of the comparison obtained from the collaboration with our industrial partner of both the migration strategy and the VMM strategy in terms of three dimensions: (a) overhead, (b) automation, and (c) trust leak.

With this evaluation we aim to address the following research questions:

GPCE-RQ1: What is the level of overhead introduced when applying VMM compared to traditional migration?

GPCE-RQ2: What is the degree of involvement of the user required by the VMM when compared to traditional migration?

GPCE-RQ3: Are there differences in the trust of the users towards their model fragments when applying VMM compared to traditional migration?

9.2.1 Overhead

Overhead refers to an increase in model elements in order to conform to an evolved metamodel while keeping the same functionality, for instance, the inductor that migrates into a hotplate and then into a cooking zone (see Section 3.3.4).

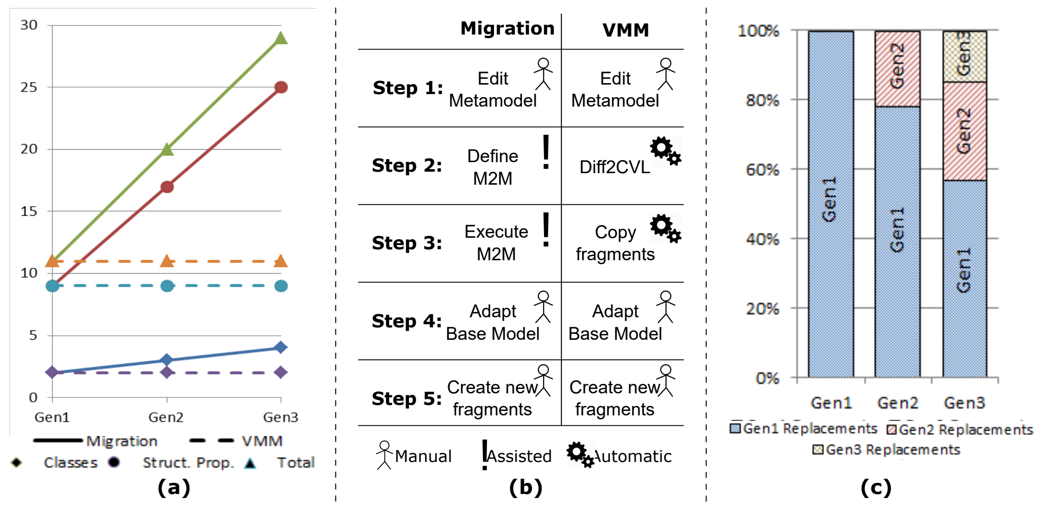


Figure 9.1: Comparison between Migration and VMM Strategy

Figure 9.1 (a) shows the comparison of both strategies in terms of the overhead that is present in the replacements. The graph shows the number of model elements (classes and structural properties) used in each generation to represent an inductor. In the migration strategy (solid lines), the inductor grows from a total of 11 elements in Gen 1 to a total of 29 elements in Gen 3. This growth trend is common for all of the concepts studied in this work. Although it is out of the scope of this paper, there are transformations based on the metamodel to transform IHDSL models into code, and this overhead requires modifications and produces an increase in the complexity of the transformations and the code generated. In contrast, the VMM strategy (dashed lines) avoids the migration of replacements, and the number of elements needed to represent the inductor concept (11) remains the same over all of the generations.

9.2.2 Automation

Depending on the degree of involvement of the user, the execution of the steps of both strategies can be either manual, assisted, or automatic. A step is automatic when it is done without user intervention; it is assisted when the user must help in the process; and it is manual when the whole process is performed by the user.

Figure 9.1 (b) shows the comparison of the two strategies in terms of automation for each of the steps of the strategies. Step 1 (Edit Metamodel) is the same for both strategies and must be performed manually. Step 2 is different; the migration strategy requires the definition of a M2M transformation. With the options that are available (manual [118], operator-based [121, 119], or metamodel matching [124, 162]), the process is, at best, assisted [123, 124]. In contrast, in the VMM Strategy Step 2 (InitVMM and addGen) is fully automatizable, (CVL applied to the model and the metamodel level enabled us to resolve all kinds of changes presented by [124] in an automatic way). Step 3 in the migration strategy is the execution of the M2M transformation. Breaking changes (e.g., the addition of obligatory properties) are not automatically resolvable ([123, 124]), so the step needs to be assisted. In contrast, in the VMM strategy replacements are used "as is" (i.e., no migration is required and only an automatic copy is performed). Finally Steps 4 (Adapt base model) and 5 (Create new replacements) are performed manually in both strategies.

9.2.3 Trust Leak

Model fragments are used to produce code; once they have been used repeatedly on many IHs, the familiarity of our industrial partner's engineers with the models increase and the engineers develop trust towards the model fragments. However, when the replacements are modified, there is a loss of this trust on the part of the engineers, which has been reported as *trust leak*.

Figure 9.1 (c) shows the evolution of the replacements being used in each generation, regarding the generation when they were created. That is, the graph shows the weight of the replacements originated in each generation in relation to the total number of products created with the SPL (i.e., the average percentage of replacements originating from each generation present in the induction hobs taking into account all of the IHs derived from the SPL for that generation). This is highly relevant for the *trust leak* phenomena, as it is related to the number of migrations that the replacements overcome.

In generation 1, all the fragments used to build the products were originated in that generation. However, only 22% of the replacements used by products in generation 2 are originated in that generation. The rest 78% of replacements were created in generation 1 and if not using the VMM strategy need to be migrated to conform to generation 2 metamodel (resulting in a decrease in the trust, as the model elements are modified). In generation 3 the effect is increased, as only a 17% of the replacements are created in that generation. The rest of the fragments have been created in previous generation but are still being used by products of generation 3. Therefore, if we apply a migration strategy 83% of the fragments needed by products of that generation will need to be migrated from previous generations (58% of them twice, from Gen1 to Gen2 and then to Gen3).

It turns out that the replacements from generation 1 are the ones that are most frequently used to build IHs (in all generations), and they are also the ones that require more migrations when following the migration strategy. Therefore, those are the replacements that have the highest level of trust leak as the trust is reduced each time that the replacement needs to be modified.

9.3 Lessons Learned

This section presents three lessons learned from the adoption of the presented VMM approach as part of the SPL of our industrial partner. After a period of using the approach by our industrial partner, we reviewed the VMM created to determine whether it was working properly. As part of this review process, we learned some lessons that enabled us to improve the approach. The first lesson is related to the creation of false revisions, the second lesson is related to the folding of revisions, and the third lesson is related to isolated revisions.

9.3.1 False Revisions

We designed the presented VMM to automatically include new metamodel revisions. In other words, each time a new metamodel was created, the addGen operation was triggered and the new revision was formalized in terms of CVL (when needed due to a breaking and unresolvable change). However, when reviewing the VMM generated by our industrial partner after the period of use, we realized that some false revisions were being created in the VMM.

Figure 9.2 shows an example of false revisions. The horizontal arrows represent the *VMM before* and after addressing the false revision issue. The *VMM before* shows 7 different revisions (circles). The number of model fragments generated for each revision is represented by the size of the circle. In addition, there are some products that were built based on the model fragments from the revisions. For instance, Product Set 1 is composed of model fragments from three different revisions (R1, R2, and R5). However, there are some revisions that were not used to build any of the products (R3, R4, and R6). We discussed this situation with our industrial partner. It turned out that those revisions where tests that were discarded and not used to build real products.

Therefore, we decided to remove those revisions from the VMM (as in the *VMM after*) and thereby reduce the complexity of the VMM. It turns out that

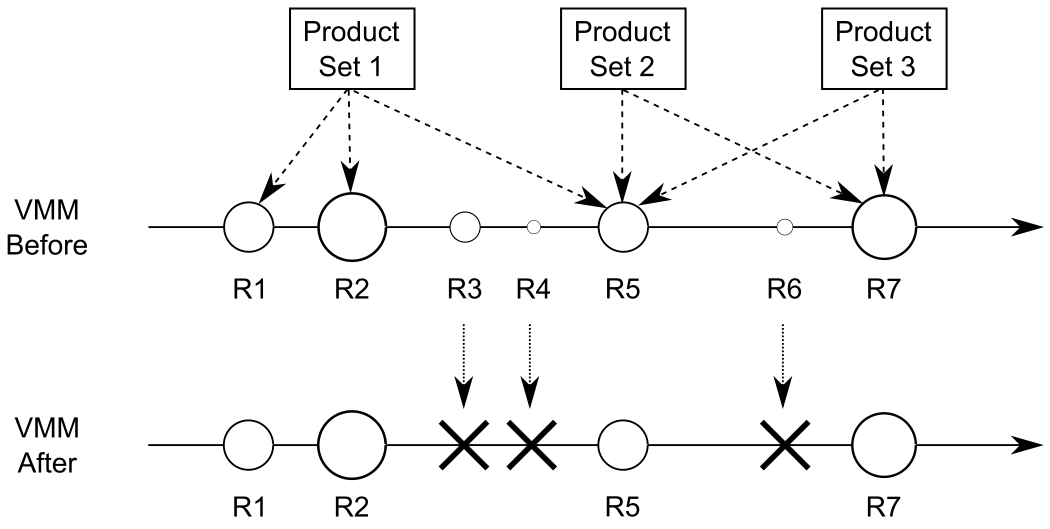


Figure 9.2: False Revisions

what defines a new generation is not just the creation of a new metamodel revision or the creation of new model fragments for that revision. Those tasks (the creation of a revision and the addition of model fragments) are common for testing purposes. What defines the creation of a new generation is the use of model fragments (that belong to the new revisions) to build new products. Therefore, we decided to postpone the addition of new metamodel revisions until they are used for the creation of new products.

However, the false revisions (R1, R2, and R5) are not deleted as they might be used to create products in the future. Therefore, we store them into an auxiliary VMM, a copy of the 'main' VMM that is used only for storing purposes (not to build new products). Then, the user can create new replacements using those metamodel revisions in the auxiliary VMM. When the user includes a replacement created with one of those revisions to build a product, the revision (that is stored into an auxiliary VMM) is added to the 'main' VMM and is no longer considered a false revision.

9.3.2 Revision Folding

Some situations also suggested the need for removing a particular revision from the VMM even though they are not false revisions (i.e., being used by some products). In other words, a new metamodel revision that includes a concept is created, model fragments for that revision are developed, and products using those model

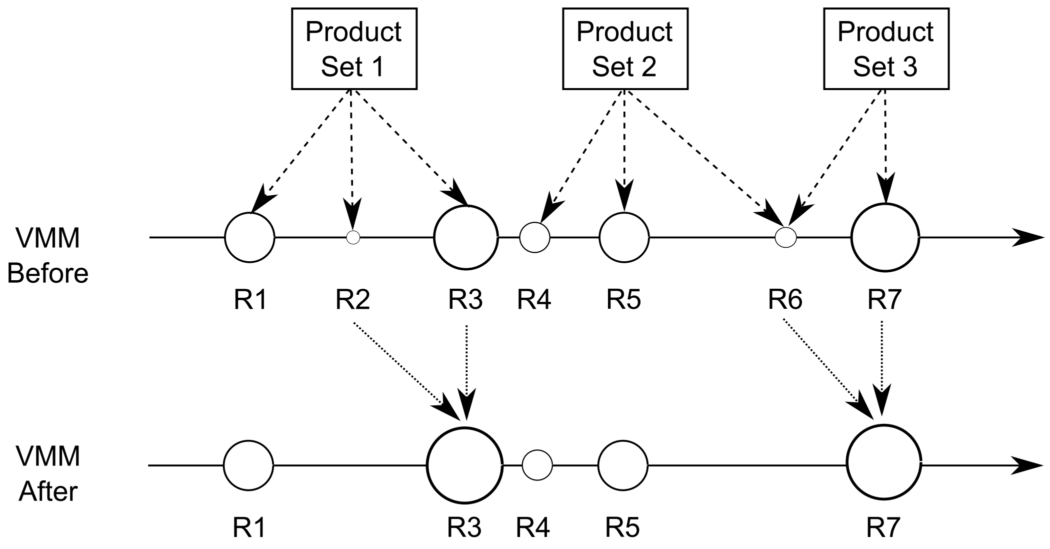


Figure 9.3: Revision Folding

fragments are created. Then, an issue with the revision is found and a new revision (fix revision) that properly represents the concept and addresses the issue discovered needs to be created. After the fix revision is created, the old revision is not used anymore, but it is not possible to remove it directly (as there are products using it). To manage situations of this kind, we introduced revision folding.

Figure 9.3 shows an example of revision folding. In the VMM *before*, an issue is discovered in R2 after some products from Product Set 1 have already been created. Then, our industrial partner created revision R3 to address the issues discovered in R2 and started using it. R2 is no longer needed, but some of the model fragments (which were not affected by the issue discovered) are still in use. To address this kind of situation, we propose migrating the model fragments from R2 to R3 and folding both revisions into a single one. The VMM after, shows how the R2 and R3 revisions have been folded (into R3). The same situation occurs with R6 and R7.

As a result, the products previously using model fragments from R2 now are using the migrated fragments from R3. This migration usually only affects a small set of fragments, and the lifespan of those fragments is short. Therefore, the disadvantages of migration are outweighed by having a clearer and smaller set of revisions under the VMM. In other words, when the engineer considers that two metamodel revisions are mutually exclusive and the later revision is a direct fix of the previous one, the engineer can fold both revisions, migrating the fragments

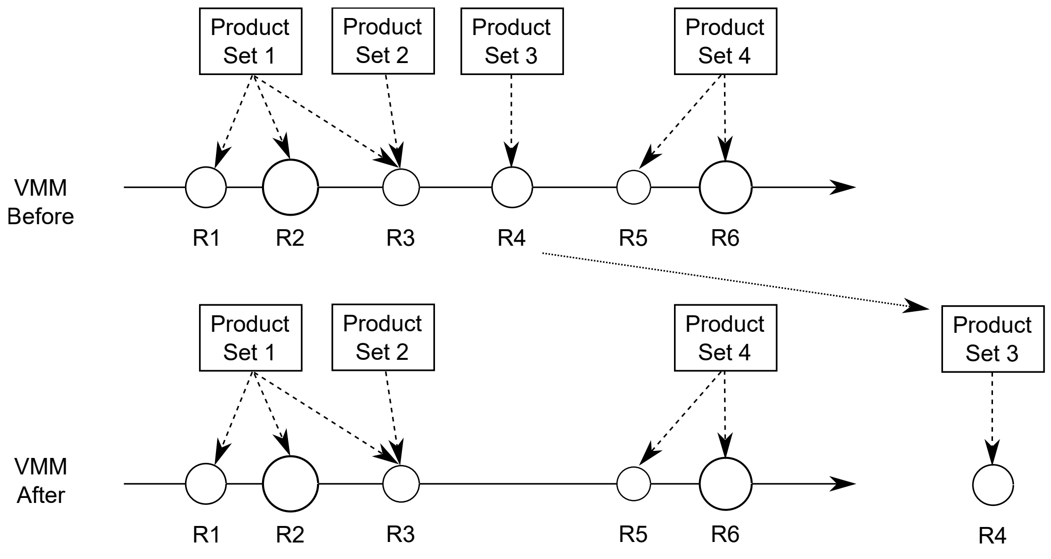


Figure 9.4: Isolated Revisions

that belong to the unused metamodel revision.

When the engineer decides to fold two revisions, the traditional migration strategy is followed. That is, fragments are migrated from the fault revision to the fix revision. The engineer is guided through the process that can be fully automated if there are not breaking changes between the two revisions.

9.3.3 Isolated Revisions

When reviewing the VMM generated by our industrial partner, we also discovered some isolated revisions (i.e., some revisions are only used to build products that do not include other revisions). Therefore, the products conform to that particular metamodel revision and it is not necessary to combine it with other metamodel revisions. As a result, that revision can be extracted from the VMM, decreasing the number of revisions managed and its complexity.

Figure 9.4 shows an example of an isolated revision. The VMM *before* shows four product sets built with model fragments from six different revisions. However, Product Set 3 is built only with model fragments from R4. In addition, R4 model fragments are not used to build any other product. As a result, R4 can be extracted from the VMM since it is not used in combination with any other revision. Product Set 2 is also built only with model fragments from a single revision (R3). However, R3 model fragments are also used to build Product Set 1, where R3 is combined with R1 and R2. Therefore it is not possible to extract R3 from the

VMM. Only revisions that are not combined with other revisions can be extracted from the VMM.

When isolated revisions are extracted from the 'main' VMM, they are stored into an auxiliary VMM. It is important to consider that, although at that moment the revision is isolated, it could stop being isolated if the engineer creates a product that combines replacements from the isolated revision and other revisions. Therefore, in that event, the isolated revision that is stored into the auxiliary VMM is moved to the 'main' VMM.

The VMM strategy eliminates the need for migration and properly manages different metamodel revisions. However, the inclusion of the VMM strategy also entails the need to properly manage the generations. As indicated by these lessons, in order to reduce the complexity of the VMM, the creation of false revisions must be avoided, the means for folding revisions must be provided, and isolated revisions must be properly handled.

Part IV

CONCLUSION

10

CONCLUSION

Contents

10.1 Overview of the Chapter

This chapter recapitulates the results presented so far and concludes the dissertation. First we connect the particular research questions presented for each of the evaluations performed with the three research questions proposed in the dissertation. Then, the ongoing research is described. Finally, we conclude the dissertation.

10.2 Research Questions

The three research questions presented as part of the dissertation have been addressed through the evaluations performed in each of the evaluations presented so far. Next we present in which way they are connected:

Research Question 1: How to identify and formalize the variability present among a set of product models in terms of features realized by model fragments?

SPLC-RQ1: Is feasible to locate features in industrial domains using the evolutionary algorithm presented so far?

MODELS-RQ1: Does the LSA+FCA fitness help when guiding the process or tampers it?

Answer to RQ1: To address RQ1, we search for the model fragment that best represents the feature being located. To do so, we rely on an evolutionary algorithm (FLiMEA) that iterates a set of candidate solutions until the model fragment that best represents the feature being located is found. Results shows that FLiMEA can be used to identify and formalize the variability across product models from industrial domains such as the ones from our partners. In addition, we have explored different operations and fitness functions for the approach. As a result, FLiMEA can be tailored to work under different environments, depending on the models and the type of domain knowledge present in the industrial domain.

Research Question 2: How to capitalize on expert domain knowledge to boost the process of feature location?

SPLC-RQ2: What is the rationale followed by domain experts to perform the selection from the ranking of variation points outputted by the approach?

ICSR-RQ1: What is the impact of the accuracy when providing the input for the approach?

MODELS-RQ2: Does the selection of the seed have an impact on the results?

MODELS-RQ3: Does the selection of the textual description have an impact on the results?

Answer to RQ2: To address RQ2, FLiMEA has been designed so the expert domain knowledge can be contributed in different ways; Specifically, domain experts will provide the feature description (based on the information available) and will choose the best option among the ranking of feature realizations proposed by FLiMEA. We have researched the impact of the different inputs provided into the process. Results show a boost on the feature location process when domain knowledge is embedded following the different options available in FLiMEA.

Research Question 3: How to co-evolve the model fragments that capture the features and the language used to create them?

GPCE-RQ1: What is the level of overhead introduced when applying VMM compared to traditional migration?

GPCE-RQ2: What is the degree of involvement of the user required by the VMM when compared to traditional migration?

GPCE-RQ3: Are there differences on the trust of the users towards their model fragments when applying VMM compared to traditional migration?

Answer to RQ3: To address RQ3, we analyzed the state-of-the-art solution to co-evolve models and metamodels and identified three issues when used to co-evolve model fragments and language. We have presented VMM, a co-evolution strategy based on variability management at metamodel level that avoids the issues entailed by migration strategies. We have applied it to address the co-evolution of a model-based SPL from an industrial domain. Results shows that VMM can be applied to address the co-evolution of model fragments and language, as such is being done by our industrial partner. We also include a set of lesson learned from the application in that domain.

Software Product Line Engineering has proven to be a mature approach to manage families of products. When combined with Model Driven Engineering, a model-based SPL can be built to manage a family of product models. However, in order to shift from a clone-and-own approach to a model-based SPL approach a great upfront investment is needed. With extractive techniques as those presented in this dissertation, we contribute to ease the transition to a model-based SPL.

In particular, the techniques for Feature Location in Models presented in Part II will enable the re-engineering of an existing family of product models into the features present in the products and realizes them in the form of model fragments or variation points. The approach helps to embed the domain knowledge into the resulting variability formalization, producing model fragments that properly capture the concepts used by the company.

Then, the techniques for co-evolution presented in Part III enable the co-evolution over time of the model fragments and the language used by the models. By applying this technique, the burden imposed by the migration can be avoided and thus the model fragments do not need to be changed when the language evolves. Therefore, the trust gathered by the model fragments is not lost and the engineers can keep working with the feature realizations that they already understand and trust.

10.3 Ongoing Research

The contributions presented in this dissertation are the results of an ongoing work that is currently being developed further. Specifically, the FLiM-EA approach is being currently adapted to work under different conditions and applied to serve different purposes. This section presents some open research questions and the ongoing work that is being done to address them.

10.3.1 Parameter values of the Evolutionary algorithm

Evolutionary Algorithms (as the one presented as contribution of this dissertation) have several different parameters that can be modified and that can determine whether the search is successful or fails [165].

The problem of setting the parameters of an evolutionary algorithm is usually divided into two cases, parameter tuning [165] and parameter control [166]. Parameter control implies that the parameter values are changing during the execution of the evolutionary algorithm while in parameter tuning, the values are determined before running the algorithm and do not change during the execution of the algorithm.

Literature distinguishes between two types of parameters: (1) qualitative parameters, those that has a finite domain and no sensible distance or ordering among the values (e.g. crossover, mutation and parent selection operations); (2) quantitative parameters, those that have an infinite domain with structure and order among the values (e.g. size of the population being evolved, number or percentage of replacements preformed in each generation).

As part of this dissertation we have already presented different qualitative parameters (see Chapters 5 and 6). However, which quantitative parameters provide better results for each domain remains as an open question and we are currently doing research to address it.

10.3.2 Multi-Objective Evolutionary Algorithms

In this dissertation we have presented three types of fitness functions (one based on Latent Semantic Analysis, another using Formal Concept Analysis and the last one based on model comparisons). Each fitness function is used to guide the search towards a single objective and this is the common case for Single-Objective Evolutionary Algorithms (SOEA). Multi-Objective Evolutionary Algorithms (MOEA) combine different fitness functions and thus guide the search towards multiple objectives.

However, determining how the combination of different fitness functions is performed is not trivial and must be carefully analyzed in order to obtain good results. The most common method to combine several fitness functions into the same search is to use the NSGA-II evolutionary algorithm [167]. This algorithm relies on nondominated sorting to find the best individuals according to several fitness dimensions. An individual is considered nondominated when there is no better individual (in the population) than that one in a specific fitness dimension without worsening other fitness dimensions.

In addition, selecting which fitness functions are used to guide the search and what are the implications of its combination also remains as an open research question. We are currently researching [168] towards the combination of the fitness functions already covered in this dissertation with new ones such as complexity measurements of the model fragments, longevity of the model fragments and its elements, and size of the model fragments.

10.3.3 Bug Location

In this dissertation, the presented approach for Feature Location in Models (FLiMEA) is applied to locate features based on domain knowledge available about the feature that needs to be located. The main purpose of performing such an operation is to identify the variability existing among the products and formalizing it in the form of features. However, the approach can also be tailored to be applied with other purposes, such as the location of bugs.

In particular, we are currently working on a modified version of the FLiMEA for Bug Location in Models (BLiMEA) [169]. To do so, we are currently modifying the approach to consume domain knowledge provided in different forms, with a focus on bug reports obtained from issue tracking systems. We are also working on new fitness functions that help towards the identification of the source of bugs, taking into account different parameters such as the nature of the modifications performed to the model fragments (type of modification, time of commit, developer that commits the changes).

10.3.4 Machine Learning Fitness

Machine Learning is known as the branch of artificial intelligence that gathers statistical, probabilistic, and optimization algorithms which learn empirically. Machine Learning has a wide range of applications, including search engines, text recognition, marketing adv sales diagnosis, etc. and has provided good results when applied to software engineering tasks that target source code artifacts.

We believe that Machine Learning can also be applied to software engineering tasks that target model artifacts and we are currently working on adapting FLiMEA towards this direction. Particularly, to apply Machine Learning techniques in models, the first challenge consists in identifying the set of elements from a model that are truly relevant for the problem and encoding them into vectors.

We believe that the approach presented in this dissertation can be adapted to obtain the model fragment that is truly relevant for the particular problem being addressed and we are currently working towards this [170].

10.3.5 Trust Leak

As part of this dissertation we have identified a major concern among the industrial scenarios that consider whether to migrate to a model-based Software Product

Line Engineering approach or not; the trust leak. Owners of the models want to be in control all through the entire process, from the identification of features and its extraction as model fragments to its further evolution and maintenance over time.

Therefore, we are working on new industrial scenarios in order to evaluate the impact of the proposed approaches (FLIMEA and VMM) on the trust deposited in the models over the years. In particular, we plan to conduct empirical evaluations focused on the trust issue in CAF (a worldwide provider of railway solutions). We want to provide insights on the reasoning followed to choose whether to migrate the artifacts or to let different generations coexists with the application of the VMM.

10.4 Concluding Remark

As a concluding remark, although there are some open research questions, the work presented in this dissertation has provided a step forward in terms of addressing the issue of re-engineering a family of product models into a model-based SPL and its further evolution in time. In particular, the work presented in this dissertation:

Conferences: Our work has been presented at scientific venues (such as the *ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems* and the *19th International Conference on Software Product Lines*).

Journals: Our work has been published in international journals (specifically in *Computer Languages, Systems and Structures* [16] and the *IEEE Transactions on Evolutionary Computation* [14]).

Research Projects: has been contributed to national and international research projects such as *VARIAMOS* (Spanish national research project) and REVaMP[2] (an international ITEA 3 Call 2 project).

Industrial Scenarios: has been evaluated in industrial scenarios such as BSH (the leading manufacturer of home appliances in Europe) and CAF (a worldwide provider of railway solutions).

BIBLIOGRAPHY

[1] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[2] Paul Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, August 2001.

[3] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[4] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[5] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. A survey of variability modeling in industrial practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 7:1–7:8, New York, NY, USA, 2013. ACM.

[6] Charles W. Krueger. Easing the transition to software mass customization. In *Revised Papers from the 4th International Workshop on Software Product-Family Engineering*, PFE '01, pages 282–293, London, UK, UK, 2002. Springer-Verlag.

[7] Julia Rubin and Marsha Chechik. A survey of feature location techniques. In Iris Reinhartz-Berger, Arnon Sturm, Tony Clark, Sholom Cohen, and Jorn Bettin, editors, *Domain Engineering*, pages 29–58. Springer Berlin Heidelberg, 2013.

[8] Bogdan Dit, Meghan Revelle, Malcom Gethers, and Denys Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[9] Mathieu Acher, Anthony Cleve, Philippe Collet, Philippe Merle, Laurence Duchien, and Philippe Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Softw. Syst. Model.*, 13(4):1367–1394, October 2014.

[10] Jaime Font, Manuel Ballarín, Øystein Haugen, and Carlos Cetina. Automating the variability formalization of a model family by means of common variability language. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 411–418, New York, NY, USA, 2015. ACM.

[11] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Building software product lines from conceptualized model patterns. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 46–55, New York, NY, USA, 2015. ACM.

[12] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Feature location in model-based software product lines through a genetic algorithm. In *Proceedings of the 15th International Conference on Software Reuse: Bridging with Social-Awareness - Volume 9679*, ICSR 2016, pages 39–54, New York, NY, USA, 2016. Springer-Verlag New York, Inc.

[13] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Feature location in models through a genetic algorithm driven by information retrieval techniques. In *Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems*, MODELS '16, pages 272–282, New York, NY, USA, 2016. ACM.

[14] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Achieving feature location in families of models through the use of search-based software engineering. *IEEE Transactions on Evolutionary Computation*, PP(99):1–1, 2017.

[15] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Addressing metamodel revisions in model-based software product lines. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2015, Pittsburgh, PA, USA, October 26-27, 2015*, pages 161–170, 2015.

[16] Jaime Font, Lorena Arcega, Øystein Haugen, and Carlos Cetina. Leveraging variability modeling to address metamodel revisions in model-based software product lines. *Computer Languages, Systems Structures*, 48:20–38, 2017. Special Issue on the 14th International Conference on Generative Programming: Concepts Experiences (GPCE'15).

[17] Salvatore T. March and Gerald F. Smith. Design and natural science research on information technology. *Decis. Support Syst.*, 15(4):251–266, December 1995.

[18] V. Vaishnavi and W. Kuechler. Design research in information systems. http://www.isworld.org/Researchdesign/drisISworld.htm, January 2004.

[19] Joaquin Miller and Jishnu Mukerji. Mda guide version 1.0.1, 2003.

[20] Stuart Kent. Model driven engineering. In *Integrated Formal Methods, Third International Conference, IFM 2002, Turku, Finland, May 15-18, 2002, Proceedings*, pages 286–298, 2002.

[21] Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi, and Gabor Karsai. Generative programming via graph transformations in the model-driven architecture. In *OOPSLA 2002 Workshop in Generative Techniques in the context of Model Driven Architecture*, 2002.

[22] Stephen J. Mellor, Anthony N. Clark, and Takao Futagami. Guest editors' introduction: Model-driven development. *IEEE Softw.*, 20(5):14–18, September 2003.

[23] Amílcar Sernadas, Cristina Sernadas, and Hans-Dieter Ehrich. Object-oriented specification of databases: An algebraic approach. In *Proceedings of the 13th International Conference on Very Large Data Bases*, VLDB '87, pages 107–116, San Francisco, CA, USA, 1987. Morgan Kaufmann Publishers Inc.

[24] Ralf Jungclaus, Gunter Saake, Thorsten Hartmann, and Cristina Sernadas. TROLL - A language for object-oriented specification of information systems. *ACM Trans. Inf. Syst.*, 14(2):175–211, 1996.

[25] Michael Rohs and Jürgen Bohn. Entry points into a smart campus environment " overview of the ethoc system. In *Proceedings of the 23rd Interna-*

tional Conference on Distributed Computing Systems, ICDCSW '03, pages 260–, Washington, DC, USA, 2003. IEEE Computer Society.

[26] Object Management Group. Unified modeling language: Superstructure version 2.1.1. OMG Specification, feb 2007.

[27] Philippe Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2004.

[28] Ivan Ivanov. *Adaptability of Model Transformations*. PhD thesis, 5 2005.

[29] Jean Bézivin, Nicolas Farcet, Jean-Marc Jézéquel, Benoît Langlois, and Damien Pollet. Reflective model driven engineering. In *Proceedings of UML 2003*, San Francisco, United States, October 2003.

[30] Jean Bézivin. In search of a Basic Principle for Model-Driven Engineering. *Novatica – Special Issue on UML (Unified Modeling Language)*, 5(2):21–24, 2004.

[31] Douglas C. Schmidt. Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25–31, February 2006.

[32] Robert Balzer. A 15 year perspective on automatic programming. *IEEE Trans. Softw. Eng.*, 11(11):1257–1268, November 1985.

[33] Krzysztof Czarnecki. *Generative Programming. Principles and Techniques of Software Engineering Based on Automated Conguration and Fragment-Based Component Models*. PhD thesis, Technical University of Ilmenau, October 1998.

[34] Ulrich W. Eisenecker. Generative programming (GP) with C++. In *Modular Programming Languages, Joint Modular Languages Conference, JMLC '97, Linz, Austria, March 19-21, 1997, Proceedings*, pages 351–365, 1997.

[35] Arie van Deursen, Paul Klint, and Joost Visser. Domain-specific languages: An annotated bibliography. volume 35, pages 26–36, New York, NY, USA, June 2000. ACM.

[36] Juha-Pekka Tolvanen and Steven Kelly. Modelling languages for product families: A method engineering approach. In *Proceedings of OOPSLA workshop on Domain-Specific Visual Languages*, 2001.

[37] Mark A. Simos. *Organization Domain Modeling (ODM): Formalizing the Core Domain Modeling Life Cycle.* SSR '95. ACM, New York, NY, USA, 1995.

[38] Robert Esser and Jörn W. Janneck. A framework for defining domain-specific visual languages. In *OOPSLA 2001 Workshop on Domain Specific Visual Languages*, 2001.

[39] M. Douglas McIlroy. Mass-produced software components. In J. M. Buxton, Peter Naur, and Brian Randell, editors, *Software Engineering Concepts and Techniques (1968 NATO Conference of Software Engineering)*, pages 88–98. NATO Science Committee, October 1968.

[40] D. L. Parnas. On the design and development of program families. *IEEE Trans. Softw. Eng.*, 2(1):1–9, January 1976.

[41] Frank van der Linden, editor. *Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3-5, 2001, Revised Papers*, volume 2290 of *Lecture Notes in Computer Science*. Springer, 2002.

[42] P. Donohoe. *Proceedings of the 1st International Software Product Lines Conference (SPLC 2000).* Number ISBN 0-7923-7940-3. Denver, Colorado, USA, August 28-31.

[43] Don Batory, Jacob Neal Sarvela, and Axel Rauschmayer. Scaling stepwise refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.

[44] Linda M. Northrop. Sei's software product line tenets. *IEEE Softw.*, 19(4):32–40, July 2002.

[45] Gary Chastek and John McGregor. Guidelines for developing a product line production plan. Technical Report CMU/SEI-2002-TR-006, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2002.

[46] John D. McGregor, Linda M. Northrop, Salah Jarrad, and Klaus Pohl. Guest editors' introduction: Initiating software product lines. *IEEE Softw.*, 19(4):24–27, July 2002.

[47] OMG. Common variability language (CVL), OMG revised submission 2012. OMG document: ad/2012-08-05, 2012.

[48] Øystein Haugen, Birger Møller-Pedersen, Jon Oldevik, Gøran K. Olsen, and Andreas Svendsen. Adding standardized variability to domain specific languages. pages 139–148, 2008.

[49] Andreas Svendsen, Xiaorui Zhang, Roy Lind-Tviberg, Franck Fleurey, Øystein Haugen, Birger Møller-Pedersen, and Gøran K. Olsen. Developing a software product line for train control: A case study of cvl. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 106–120, Berlin, Heidelberg, 2010. Springer-Verlag.

[50] Wesley K. Assunção Jabier Martinez and Tewfik Ziadi. Espla: A catalog of extractive spl adoption case studies. In *Proceedings of the 2011 21st International Systems and Software Product Line Conference. (SPLC 2017), Sevilla, Spain 25-29 September*, SPLC '17, 2017.

[51] Thomas K Landauer, Peter W Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25(2-3):259–284, 1998.

[52] G. H. Golub and C. Reinsch. Singular value decomposition and least squares solutions. *Numer. Math.*, 14(5):403–420, April 1970.

[53] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, November 1975.

[54] Andrian Marcus, Andrey Sergeyev, Vaclav Rajlich, and Jonathan I. Maletic. An information retrieval approach to concept location in source code. In *Proceedings of the 11th Working Conference on Reverse Engineering*, WCRE '04, pages 214–223, Washington, DC, USA, 2004. IEEE Computer Society.

[55] Wei Zhao, Lu Zhang, Yin Liu, Jiasu Sun, and Fuqing Yang. Sniafl: Towards a static noninteractive approach to feature location. *ACM Trans. Softw. Eng. Methodol.*, 15(2):195–226, April 2006.

[56] Meghan Revelle, Bogdan Dit, and Denys Poshyvanyk. Using data fusion and web mining to support feature location in software. In *Proceedings of the 2010 IEEE 18th International Conference on Program Comprehension*,

ICPC '10, pages 14–23, Washington, DC, USA, 2010. IEEE Computer Society.

[57] Dapeng Liu, Andrian Marcus, Denys Poshyvanyk, and Vaclav Rajlich. Feature location via information retrieval based filtering of a single scenario execution trace. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 234–243, New York, NY, USA, 2007. ACM.

[58] Denys Poshyvanyk, Yann-Gael Gueheneuc, Andrian Marcus, Giuliano Antoniol, and Vaclav Rajlich. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. *IEEE Trans. Softw. Eng.*, 33(6):420–432, June 2007.

[59] Fatemeh Asadi, Massimiliano Di Penta, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. A heuristic-based approach to identify concepts in execution traces. In *Proceedings of the 2010 14th European Conference on Software Maintenance and Reengineering*, CSMR '10, pages 31–40, Washington, DC, USA, 2010. IEEE Computer Society.

[60] Yguaratã Cerqueira Cavalcanti, Ivan do Carmo Machado, Paulo A. da Mota S. Neto, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. Combining rule-based and information retrieval techniques to assign software change requests. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 325–330, New York, NY, USA, 2014. ACM.

[61] Markus Kimmig, Martin Monperrus, and Mira Mezini. Querying source code with natural language. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, ASE '11, pages 376–379, Washington, DC, USA, 2011. IEEE Computer Society.

[62] Shaowei Wang, David Lo, and Lingxiao Jiang. Active code search: Incorporating user feedback to improve code search relevance. In *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ASE '14, pages 677–682, New York, NY, USA, 2014. ACM.

[63] Emily Hill, Lori Pollock, and K. Vijay-Shanker. Automatically capturing source code context of nl-queries for software maintenance and reuse. In *Proceedings of the 31st International Conference on Software Engineering*,

ICSE '09, pages 232–242, Washington, DC, USA, 2009. IEEE Computer Society.

[64] Yanzhen Zou, Ting Ye, Yangyang Lu, John Mylopoulos, and Lu Zhang. Learning to rank for question-oriented software text retrieval (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 1–11, Washington, DC, USA, 2015. IEEE Computer Society.

[65] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 181–190, New York, NY, USA, 2011. ACM.

[66] Yuan Tian, David Lo, and Julia L. Lawall. Automated construction of a software-specific word similarity database. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, pages 44–53, 2014.

[67] Timothy Dietrich, Jane Cleland-Huang, and Yonghee Shin. Learning effective query transformations for enhanced requirements trace retrieval. In *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, ASE'13, pages 586–591, Piscataway, NJ, USA, 2013. IEEE Press.

[68] Fei Lv, Hongyu Zhang, Jian-guang Lou, Shaowei Wang, Dongmei Zhang, and Jianjun Zhao. Codehow: Effective code search based on api understanding and extended boolean model (e). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 260–270, Washington, DC, USA, 2015. IEEE Computer Society.

[69] Norman Wilde and Michael C. Scully. Software reconnaissance: Mapping program features to code. *Journal of Software Maintenance*, 7(1):49–62, January 1995.

[70] W. Eric Wong, Joseph R. Horgan, Swapna S. Gokhale, and Kishor S. Trivedi. Locating program features using execution slices. In *Proceedings of the 1999 IEEE Symposium on Application - Specific Systems and Software Engineering and Technology*, ASSET '99, pages 194–, Washington, DC, USA, 1999. IEEE Computer Society.

[71] Andrew David Eisenberg and Kris De Volder. Dynamic feature traces: Finding features in unfamiliar code. In *Proceedings of the 21st IEEE International Conference on Software Maintenance*, ICSM '05, pages 337–346, Washington, DC, USA, 2005. IEEE Computer Society.

[72] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Locating features in source code. *IEEE Transactions on Software Engineering*, 29(3):210–224, March 2003.

[73] Rainer Koschke and Jochen Quante. On dynamic feature location. In *Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering*, ASE '05, pages 86–95, New York, NY, USA, 2005. ACM.

[74] Marc Eaddy, Alfred V. Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *Proceedings of the 2008 The 16th IEEE International Conference on Program Comprehension*, ICPC '08, pages 53–62, Washington, DC, USA, 2008. IEEE Computer Society.

[75] Kunrong Chen and Václav Rajlich. Case study of feature location using dependence graph. In *Proceedings of the 8th International Workshop on Program Comprehension*, IWPC '00, pages 241–, Washington, DC, USA, 2000. IEEE Computer Society.

[76] Neil Walkinshaw, Marc Roper, and Murray Wood. Feature location and extraction using landmarks and barriers. In *23rd IEEE International Conference on Software Maintenance (ICSM 2007), October 2-5, 2007, Paris, France*, pages 54–63, 2007.

[77] David Shepherd, Zachary P. Fry, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th International*

Conference on Aspect-oriented Software Development, AOSD '07, pages 212–224, New York, NY, USA, 2007. ACM.

[78] Martin P. Robillard. Automatic generation of suggestions for program investigation. pages 11–20, 2005.

[79] Emily Hill, Lori Pollock, and K. Vijay-Shanker. Exploring the neighborhood with dora to expedite software maintenance. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 14–23, New York, NY, USA, 2007. ACM.

[80] Martin P. Robillard. Topology analysis of software dependencies. *ACM Trans. Softw. Eng. Methodol.*, 17(4):18:1–18:36, August 2008.

[81] Christian Kästner, Alexander Dreiling, and Klaus Ostermann. Variability mining: Consistent semi-automatic detection of product-line features. *IEEE Trans. Software Eng.*, 40(1):67–82, 2014.

[82] Christian Kästner, Paolo G. Giarrusso, Tillmann Rendel, Sebastian Erdweg, Klaus Ostermann, and Thorsten Berger. Variability-aware parsing in the presence of lexical macros and conditional compilation. In *Proceedings of the 2011 ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '11, pages 805–824, New York, NY, USA, 2011. ACM.

[83] Christian Kästner, Klaus Ostermann, and Sebastian Erdweg. A variability-aware module system. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, OOPSLA '12, pages 773–792, New York, NY, USA, 2012. ACM.

[84] Slawomir Duszynski, Jens Knodel, and Martin Becker. Analyzing the source code of multiple software variants for reuse potential. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 303–307, Washington, DC, USA, 2011. IEEE Computer Society.

[85] Sarah Nadi, Thorsten Berger, Christian Kästner, and Krzysztof Czarnecki. Where do configuration constraints stem from? an extraction approach and an empirical study. *IEEE Trans. Software Eng.*, 41(8):820–841, 2015.

[86] Steven She, Rafael Lotufo, Thorsten Berger, Andrzej Wąsowski, and Krzysztof Czarnecki. Reverse engineering feature models. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 461–470, New York, NY, USA, 2011. ACM.

[87] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *12th International Software Product Line Conference*, pages 67–76, September 2008.

[88] Krzysztof Czarnecki and Andrzej Wasowski. Feature diagrams and logics: There and back again. In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 23–34, Washington, DC, USA, 2007. IEEE Computer Society.

[89] Mark Harman, Yue Jia, Jens Krinke, William B. Langdon, Justyna Petke, and Yuanyuan Zhang. Search based software engineering for software product line engineering: A survey and directions for future work. In *Proceedings of the 18th International Software Product Line Conference - Volume 1*, SPLC '14, pages 5–18, New York, NY, USA, 2014. ACM.

[90] Roberto E. Lopez-Herrejon, Javier Ferrer, Francisco Chicano, Lukas Linsbauer, Alexander Egyed, and Enrique Alba. A hitchhiker's guide to search-based software engineering for software product lines. *CoRR*, abs/1406.2823, 2014.

[91] Jules White, Brian Doughtery, and Douglas C. Schmidt. Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints. In *Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8-12, 2008, Proceedings. Second Volume (Workshops)*, pages 209–216, 2008.

[92] 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. *J. Syst. Softw.*, 84(12):2208–2221, December 2011.

[93] Jian Li, Xijuan Liu, Yinglin Wang, and Jianmei Guo. *Formalizing Feature Selection Problem in Software Product Lines Using 0-1 Programming*, pages 459–465. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[94] Abdel Salam Sayyad, Tim Menzies, and Hany Ammar. On the value of user preferences in search-based software engineering: A case study in software product lines. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 492–501, Piscataway, NJ, USA, 2013. IEEE Press.

[95] Abdel Salam Sayyad, Joseph Ingram, Tim Menzies, and Hany Ammar. Optimum feature selection in software product lines: Let your model and values guide your search. In *Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering*, CMSBSE '13, pages 22–27, Piscataway, NJ, USA, 2013. IEEE Press.

[96] Ying-lin Wang and Jin-wei Pang. Ant colony optimization for feature selection in software product lines. *Journal of Shanghai Jiaotong University (Science)*, 19(1):50–58, February 2014.

[97] Kit Yan Chan, Che Kit Kwong, and T. C. Wong. Modelling customer satisfaction for product development using genetic programming. *Journal of Engineering Design*, 22(1):55–68, 2011.

[98] Evelyn Nicole Haslinger, Roberto E. Lopez-Herrejon, and Alexander Egyed. Reverse engineering feature models from programs' feature sets. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, WCRE '11, pages 308–312, Washington, DC, USA, 2011. IEEE Computer Society.

[99] Roberto Erick Lopez-Herrejon, José A. Galindo, David Benavides, Sergio Segura, and Alexander Egyed. Reverse engineering feature models with evolutionary algorithms: An exploratory study. In *Proceedings of the 4th International Conference on Search Based Software Engineering*, SSBSE'12, pages 168–182, Berlin, Heidelberg, 2012. Springer-Verlag.

[100] Lukas Linsbauer, Roberto Erick Lopez-Herrejon, and Alexander Egyed. Feature model synthesis with genetic programming. In *Search-Based Software Engineering - 6th International Symposium, SSBSE 2014, Fortaleza, Brazil, August 26-29, 2014. Proceedings*, pages 153–167, 2014.

[101] Roberto E. Lopez-Herrejon, Lukas Linsbauer, José A. Galindo, José A. Parejo, David Benavides, Sergio Segura, and Alexander Egyed. An assess-

ment of search-based techniques for reverse engineering feature models. *J. Syst. Softw.*, 103(C):353–369, May 2015.

[102] Sergio Segura, José A. Parejo, Robert M. Hierons, David Benavides, and Antonio Ruiz-Cortés. Automated generation of computationally hard feature models using evolutionary algorithms. *Expert Syst. Appl.*, 41(8):3975–3992, June 2014.

[103] Tewfik Ziadi, Luz Frias, Marcos Aurélio Almeida da Silva, and Mikal Ziane. Feature identification from the source code of product variants. In *16th European Conference on Software Maintenance and Reengineering, CSMR 2012, Szeged, Hungary, March 27-30, 2012*, pages 417–422, 2012.

[104] Tewfik Ziadi, Christopher Henard, Mike Papadakis, Mikal Ziane, and Yves Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1064–1071, New York, NY, USA, 2014. ACM.

[105] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve. Reverse engineering web configurators. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 264–273, Feb 2014.

[106] Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, pages 90–99, Washington, DC, USA, 2011. IEEE Computer Society.

[107] Xiaorui Zhang, Øystein Haugen, and Birger Møller-Pedersen. Augmenting product lines. In *19th Asia-Pacific Software Engineering Conference, APSEC 2012, Hong Kong, China, December 4-7, 2012*, pages 766–771, 2012.

[108] David Wille, Sönke Holthusen, Sandro Schulze, and Ina Schaefer. Interface variability in family model mining. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, SPLC '13 Workshops, pages 44–51, New York, NY, USA, 2013. ACM.

[109] Sönke Holthusen, David Wille, Christoph Legat, Simon Beddig, Ina Schaefer, and Birgit Vogel-Heuser. Family model mining for function block diagrams in automation software. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools - Volume 2*, SPLC '14, pages 36–43, New York, NY, USA, 2014. ACM.

[110] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. Bottom-up adoption of software product lines: A generic and extensible approach. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 101–110, New York, NY, USA, 2015. ACM.

[111] Jabier Martinez, Tewfik Ziadi, Tegawende F. Bissyande, Jacques Klein, and Yves le Traon. Automating the extraction of model-based software product lines from model variants (t). In *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '15, pages 396–406, Washington, DC, USA, 2015. IEEE Computer Society.

[112] Kangtae Kim, Hyungrok Kim, and Woomok Kim. Building software product line from the legacy systems "experience in the digital audio and video domain". In *Proceedings of the 11th International Software Product Line Conference*, SPLC '07, pages 171–180, Washington, DC, USA, 2007. IEEE Computer Society.

[113] Ronny Kolb, Dirk Muthig, Thomas Patzke, and Kazuyuki Yamauchi. Refactoring a legacy component for reuse in a software product line: A case study: Practice articles. *J. Softw. Maint. Evol.*, 18(2):109–132, March 2006.

[114] Hyesun Lee, Hyunsik Choi, Kyo C. Kang, Dohyung Kim, and Zino Lee. Experience report on using a domain model-based extractive approach to software product line asset development. In *Proceedings of the 11th International Conference on Software Reuse: Formal Foundations of Reuse and Domain Engineering*, ICSR '09, pages 137–149, Berlin, Heidelberg, 2009. Springer-Verlag.

[115] Julia Rubin and Marsha Chechik. Combining related products into product lines. In *Proceedings of the 15th International Conference on Fundamental*

Approaches to Software Engineering, FASE'12, pages 285–300. Springer-Verlag, Berlin, Heidelberg, 2012.

[116] Julia Rubin. *Cloned product variants: From ad-hoc to well-managed software reuse*. PhD thesis, University of Toronto, 2014.

[117] Julia Rubin and Marsha Chechik. N-way model merging. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 301–311, New York, NY, USA, 2013. ACM.

[118] Louis M Rose, Richard F Paige, Dimitrios S Kolovos, and Fiona A Polack. An analysis of approaches to model migration. In *Proceedings of the Joint MoDSE-MCCM Workshop*, pages 6–15, 2009.

[119] Guido Wachsmuth. Metamodel adaptation and model co-adaptation. In *Proceedings of the 21st European Conference on Object-Oriented Programming*, ECOOP'07, pages 600–624, Berlin, Heidelberg, 2007. Springer-Verlag.

[120] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. Automatability of coupled evolution of metamodels and models in practice. In *Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems*, MoDELS '08, pages 645–659, Berlin, Heidelberg, 2008. Springer-Verlag.

[121] Markus Herrmannsdoerfer, Sebastian Benz, and Elmar Juergens. Cope - automating coupled evolution of metamodels and models. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming*, Genoa, pages 52–76, Berlin, Heidelberg, 2009. Springer-Verlag.

[122] Markus Herrmannsdoerfer, Daniel Ratiu, and Guido Wachsmuth. Language evolution in practice: The history of gmf. In *Proceedings of the Second International Conference on Software Language Engineering*, SLE'09, pages 3–22, Berlin, Heidelberg, 2010. Springer-Verlag.

[123] Boris Gruschko, Dimitrios Kolovos, and Richard Paige. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*, 2007.

[124] Antonio Cicchetti, Davide Di Ruscio, Romina Eramo, and Alfonso Pierantonio. Automating co-evolution in model-driven engineering. In *Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, EDOC '08, pages 222–231, Washington, DC, USA, 2008. IEEE Computer Society.

[125] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. *Metamodel Matching for Automatic Model Transformation Generation*, pages 326–340. MoDELS '08. Springer-Verlag, Berlin, Heidelberg, 2008.

[126] Jonathan Mark Sprinkle. *Metamodel Driven Model Migration*. PhD thesis, Nashville, TN, USA, 2003. AAI3100940.

[127] Jonathan Sprinkle and Gabor Karsai. A domain-specific visual language for domain model evolution. *J. Vis. Lang. Comput.*, 15(3-4):291–307, 2004.

[128] Moritz Eysholdt. Emf ecore based meta model evolution and model co-evolution. Master's thesis, Carl von Ossietzky Universität Oldenburg, 2008.

[129] Moritz Eysholdt, Sören Frey, and Wilhelm Hasselbring. Emf ecore based meta model evolution and model co-evolution. *Softwaretechnik-Trends*, 29(2):20–21, 2009.

[130] Bennet P. Lientz and Burton E. Swanson. *Software Maintenance Management: A Study of the Maintenance of Computer Application Software in 487 Data Processing Organizations*. Addison-Wesley, 1980.

[131] Ned Chapin, Joanne E. Hale, Khaled Md. Kham, Juan F. Ramil, and Wui-Gee Tan. Types of software evolution and software maintenance. *Journal of Software Maintenance*, 13(1):3–30, January 2001.

[132] Tom Mens, Jim Buckley, Matthias Zenger, and Awais Rashid. Towards a taxonomy of software evolution. In *Proceedings of the International Workshop on Unanticipated Software Evolution*, number LAMP-CONF-2003-005, pages 1–18, 2003.

[133] Cheng Thao. Managing evolution of software product line. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1619–1621, Piscataway, NJ, USA, 2012. IEEE Press.

[134] Jacky Estublier, Idrissa A. Dieng, and Thomas Leveque. Software product line evolution: The selecta system. In *Proceedings of the 2010 ICSE Workshop on Product Line Approaches in Software Engineering*, PLEASE '10, pages 32–39, New York, NY, USA, 2010. ACM.

[135] Klaus Schmid and Holger Eichelberger. A requirements-based taxonomy of software product line evolution. *ECEASST*, 8, 2007.

[136] Mikael Svahnberg and Jan Bosch. Evolution in software product lines: Two cases. *Journal of Software Maintenance*, 11(6):391–422, November 1999.

[137] John McGregor. The evolution of product line assets. Technical Report CMU/SEI-2003-TR-005, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

[138] Rafael Lotufo, Steven She, Thorsten Berger, Krzysztof Czarnecki, and Andrzej Wąsowski. Evolution of the linux kernel variability model. In *Proceedings of the 14th International Conference on Software Product Lines: Going Beyond*, SPLC'10, pages 136–150, Berlin, Heidelberg, 2010. Springer-Verlag.

[139] Leonardo Passos, Krzysztof Czarnecki, Sven Apel, Andrzej Wąsowski, Christian Kästner, and Jianmei Guo. Feature-oriented software evolution. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 17:1–17:8, New York, NY, USA, 2013. ACM.

[140] Mathias Schubanz, Andreas Pleuss, Ligaj Pradhan, Goetz Botterweck, and Anil Kumar Thurimella. Model-driven planning and monitoring of long-term software product line evolution. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, VaMoS '13, pages 18:1–18:5, New York, NY, USA, 2013. ACM.

[141] Laís Neves, Leopoldo Teixeira, Demóstenes Sena, Vander Alves, Uirá Kulesza, and Paulo Borba. Investigating the safe evolution of software product lines. In *Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering*, GPCE '11, pages 33–42, New York, NY, USA, 2011. ACM.

[142] Christoph Seidl. *Evolution in feature-oriented model-based software product line engineering*. PhD thesis, Technische Universität Dresden, 2011.

[143] Christoph Seidl, Florian Heidenreich, and Uwe A. Co-evolution of models and feature mapping in software product lines. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC '12, pages 76–85, New York, NY, USA, 2012. ACM.

[144] Stephen Creff, Joël Champeau, Jean-Marc Jézéquel, and Arnaud Monégier. Model-based product line evolution: An incremental growing by extension. In *Proceedings of the 16th International Software Product Line Conference - Volume 2*, SPLC '12, pages 107–114, New York, NY, USA, 2012. ACM.

[145] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *J. Syst. Softw.*, 83(7):1108–1122, July 2010.

[146] Gan Deng, Douglas C Schmidt, Aniruddha Gokhale, Jeff Gray, Yuehua Lin, and Gunther Lenz. Evolution in model-driven software product-line architectures. *Designing Software-Intensive Systems: Methods and Principles*, 2008.

[147] Nam H. Pham, Hoan Anh Nguyen, Tung Thanh Nguyen, Jafar M. Al-Kofahi, and Tien N. Nguyen. Complete and accurate clone detection in graph-based models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 276–286, Washington, DC, USA, 2009. IEEE Computer Society.

[148] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York, 1991.

[149] Bernhard Ganter and Rudolf Wille. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1st edition, 1997.

[150] Malcom Gethers, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. On integrating orthogonal information retrieval methods to improve traceability recovery. In *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*, pages 133–142, 2011.

[151] Andrea Lucia, Massimiliano Penta, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Labeling source code with information retrieval

methods: An empirical study. *Empirical Softw. Engg.*, 19(5):1383–1420, October 2014.

[152] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar Devanbu. On the naturalness of software. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 837–847, Piscataway, NJ, USA, 2012. IEEE Press.

[153] Samir Gupta, Sana Malik, Lori L. Pollock, and K. Vijay-Shanker. Part-of-speech tagging of program identifiers for improved text-based software engineering tools. In *IEEE 21st International Conference on Program Comprehension, ICPC 2013, San Francisco, CA, USA, 20-21 May, 2013*, pages 3–12, may 2013.

[154] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. On the role of the nouns in ir-based traceability recovery. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 148–157, May 2009.

[155] Giovanni Capobianco, Andrea De Lucia, Rocco Oliveto, Annibale Panichella, and Sebastiano Panichella. Improving ir-based traceability recovery via noun-based indexing of software artifacts. *Journal of Software: Evolution and Process*, 25(7):743–762, 2013.

[156] Sima Zamani, Sai Peck Lee, Ramin Shokripour, and John Anvik. A noun-based approach to feature location using time-aware term-weighting. *Information & Software Technology*, 56(8):991–1011, 2014.

[157] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. Semantic clustering: Identifying topics in source code. *Inf. Softw. Technol.*, 49(3):230–243, March 2007.

[158] Pieter van der Spek, Steven Klusener, and Piërre van de Laar. *Complementing Software Documentation*, pages 37–51. Springer Netherlands, Dordrecht, 2011.

[159] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs (3rd Ed.).* Springer-Verlag, London, UK, UK, 1996.

[160] Michael Affenzeller, Stephan Winkler, Stefan Wagner, and Andreas Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*. Chapman & Hall/CRC, 1$^{\text{th}}$ edition, 2009.

[161] Stephen V. Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote Sensing of Environment*, 62(1):77–89, 1997.

[162] Kelly Garcés, Frédéric Jouault, Pierre Cointe, and Jean Bézivin. Managing model adaptation by precise detection of metamodel changes. In *Proceedings of the 5th European Conference on Model Driven Architecture - Foundations and Applications*, ECMDA-FA '09, pages 34–49, Berlin, Heidelberg, 2009. Springer-Verlag.

[163] Sven Apel, Sergiy Kolesnikov, Norbert Siegmund, Christian Kästner, and Brady Garvin. Exploring feature interactions in the wild: The new feature-interaction challenge. In *Proceedings of the 5th International Workshop on Feature-Oriented Software Development*, FOSD '13, pages 1–8, New York, NY, USA, 2013. ACM.

[164] Eclipse Foundation. Eclipse modeling framework compare (emfcompare) website. http://wiki.eclipse.org/index.php/EMF_Compare, 2008.

[165] A. E. Eiben and S. K. Smit. *Evolutionary Algorithm Parameters and Methods to Tune Them*, pages 15–36. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[166] A. E. Eiben, Robert Hinterding, and Zbigniew Michalewicz. Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 3(2):124–141, July 1999.

[167] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[168] Raúl Lapeña, Jaime Font, Carlos Cetina, and Oscar Pastor. Model fragment reuse driven by requirements. In *Proceedings of the Forum and Doctoral Consortium Papers Presented at the 29th International Conference on Advanced Information Systems Engineering, CAiSE 2017, Essen, Germany, June 12-16, 2017*, pages 73–80, 2017.

[169] Lorena Arcega, Jaime Font, Øystein Haugen, and Carlos Cetina. On the influence of modification timespan weightings in the location of bugs in models. In *Proceedings of the 26th International Conference on Information Systems Development, ISD 2017*, 2017.

[170] Ana Cristina Marcén, Francisca Pérez, and Carlos Cetina. Ontological evolutionary encoding to bridge machine learning and conceptual models: Approach and industrial evaluation. In *Conceptual Modeling - 36th International Conference, ER 2017, Valencia, Spain, November 6-9, 2017, Proceedings*, 2017.