Location of Features as Model Fragments and their Co-Evolution

Jaime Font Burdeus

September 18, 2017

Thesis submitted for the degree of Philosophiæ Doctor

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

Abstract iii

Acknowledgements v

Contents xi

List of Figures xv

Part I Introduction 1

1 Introduction 3

1.1 Motivation of the Dissertation 4
1.2 Problem Statement 5
1.3 Contribution 6
1.4 Overview of the Work 7
1.5 Research Methodology 8
1.6 Quick Reference 10
1.7 Structure of the dissertation 11

2 Background 13

2.1 Overview of the Chapter 14
2.2 Model Driven Development 14
2.2.1 Definition 15
2.2.2 Model Driven Software Development Initiatives 15
2.2.3 Domain Specific Languages 17
2.3 Software Product Lines 18
2.3.1 Definition 19
2.3.2 Software Product Line Processes 20
2.4 Running Example 21
2.4.1 The Induction Hobs Domain 21
2.4.2 The Common Variability Language applied to Induction Hobs 23

3 **State of the Art** **25**
3.1 Overview of the Chapter 26
3.2 Feature Location in Models 26
3.2.1 Feature Location 26
3.2.2 Search Based Software Engineering 30
3.2.3 Model Driven Engineering 32
3.2.4 Motivation of our Feature Location in Models Approach 35
3.3 Evolution of Model Fragments 37
3.3.1 Model & Metamodel Co-evolution 37
3.3.2 Traditional Software Evolution 39
3.3.3 Software Product Line Evolution 39
3.3.4 Motivation of our Model and Language Co-Evolution Approach 41

Part II Feature Location in Models **45**

4 **Feature Location in Models by an Evolutionary Algorithm (FLiMEA)** **47**
4.1 Overview of the Chapter 48
4.2 Model Artifact 48
4.3 Feature Knowledge 49
4.4 Evolutionary Algorithm 50
4.4.1 Encoding 51
4.4.2 Assessment 51
4.4.3 Genetic Manipulation 52
4.5 Ranking of feature realizations 52

5 **FLiMEA as Model Fragments** **55**
5.1 Overview of the Chapter 56
5.2 Encoding: Binary 57
5.3 Fitness: Text-based similarity 58
5.3.1 Latent Semantic Analysis 59
5.3.2 Formal Concept Analysis 62
5.4 Genetic Operations for Model Fragments 66
5.4.1 Parent Selection: Model Fragment selection 66
5.4.2 Crossover: Mask-based 67

5.4.3 Mutation: Random 69
5.5 Variability in FLiMEA as Model Fragments 70

6 FLiMEA as Variation Points 73
6.1 Overview of the chapter 74
6.2 Encoding: Boundary-based 76
6.3 Fitness: Conceptual Model Patterns 78
6.3.1 Placement Signature Abstraction 78
6.3.2 Placement Signature Matching 79
6.3.3 Fitness computation 80
6.4 Genetic Operations for Variation Points 81
6.4.1 Parent Selection: Different Parents 82
6.4.2 Crossover: Parent change 82
6.4.3 Mutation: Sequential mutation 83
6.5 Variability in FliMEA as Variation Points 85

7 Evaluation of FLiMEA 87
7.1 Overview of the Chapter 88
7.2 Oracle 88
7.2.1 Induction Hob Domain 89
7.2.2 Train Control and Management Domain 89
7.3 Test Cases 90
7.4 Approach under Evaluation 91
7.5 Comparison and Measure 91
7.6 Measurements 93
7.7 Results 95
7.7.1 Evaluation 1 (SPLC'15) 95
7.7.2 Evaluation 2 (ICSR'16) 96
7.7.3 Evaluation 3 (MODELS'16) 98

Part III Evolution of Model Fragments 103

8 Variable MetaModel (VMM) 105
8.1 Overview of the Chapter 106
8.2 Retrospective Case Study 106
8.3 The Variable MetaModel (VMM) 109
8.4 VMM operations 112
8.4.1 InitVMM operation 112
8.4.2 AddGen operation 114

9 Evaluation of VMM 117
9.1 Overview of the Chapter 118
9.2 Migration Issues in VMM 118
9.2.1 Overhead 118
9.2.2 Automation 119
9.2.3 Trust Leak 120
9.3 Lessons Learned 121
9.3.1 False Revisions 121
9.3.2 Revision Folding 122
9.3.3 Isolated Revisions 124

Part IV Conclusion 127

10 Conclusion 129
10.1 Overview of the Chapter 130
10.2 Research Questions 130
10.3 Ongoing Research 132
10.3.1 Parameter values of the Evolutionary algorithm 132
10.3.2 Multi-Objective Evolutionary Algorithms 133
10.3.3 Bug Location 134
10.3.4 Machine Learning Fitness 134
10.3.5 Trust Leak 134
10.4 Concluding Remark 135

Bibliography 136

Part V Publications 159

11 Feature Location in Models 161
11.1 REVE'15 Paper 162
11.2 SPLC'15 Paper 171
11.3 ICSR'16 Paper 182
11.4 MODELS'16 Paper 199
11.5 TEVC'17 Paper 211

12 Evolution of Model Fragments 227
12.1 GPCE'15 Paper 228
12.2 COMLAN'17 Paper 239

LIST OF FIGURES

1 Introduction **3**

1.1 Motivation of the Dissertation 5
1.2 Overview of the work performed as part of the dissertation . . . 7
1.3 Research methodology followed in this Dissertation. 9
1.4 Cheat Sheet 10

2 Background **13**

2.1 CVL applied to IHDSL 23

3 State of the Art **25**

3.1 Overview of the scope of Feature Location in Models challenge 26
3.2 Overview of the scope of Feature Location in relation to FLiM challenge 27
3.3 Overview of the scope of the Search Based Software Engineering in relation to FLiM challenge 30
3.4 Overview of the scope of the Model Driven Engineering in relation to FLiM challenge 32
3.5 Motivation of the proposed approach to address FLiM challenge 36
3.6 Overview of the scope of the evolution of model fragments challenge 38
3.7 Model and Metamodel Co-evolution problem 41
3.8 Evolution of Model Fragment through Migrations 42

4 Feature Location in Models by an Evolutionary Algorithm (FLiMEA) **47**

4.1 Activity diagram for the Feature Location in Models trough an Evolutionary Algorithm (FLiMEA) 48

5 FLiMEA as Model Fragments **55**
5.1 Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Model Fragments 56
5.2 Binary-based encoding 58
5.3 Term-by-document co-occurrence matrix for Model Fragments . 60
5.4 LSA Fitness Results 61
5.5 Formal Context between model fragments and metamodel elements 63
5.6 Lattice obtained from the Formal Context 64
5.7 Fitness assessment for Feature Candidates and spread to Individuals 65
5.8 Selection Operator for Model Fragments 67
5.9 Mask-based Crossover for model Fragments 68
5.10 Random Mutation for model Fragments 70
5.11 Variability of the Feature Location in Models as Model Fragments process 71

6 FLiMEA as Variation Points **73**
6.1 Model Fragment and Variation Point 74
6.2 Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Variation Points 75
6.3 Example of a variation point using CVL-based encoding 77
6.4 Placement Signature Abstraction 79
6.5 Placement Signature Matching 80
6.6 Fitness Assesment and Variation Point construction 81
6.7 Crossover Operation 83
6.8 Sequential Mutation with constraints 84
6.9 Variability of the Feature Location in Models as Variation Points process 86

7 Evaluation of FLiMEA **87**
7.1 Setup of the evaluation 88
7.2 Test Case example 91
7.3 Feature Model of FLiMEA approach 92
7.4 Example of confusion matrix for two candidate model fragments 93
7.5 Configuration 1 96
7.6 Configuration 2 97

7.7 Configuration 3 99
7.8 Mean Precision, Recall and F-measure for FLiMEA and the Baseline 100

8 Variable MetaModel (VMM) 105
8.1 Model Generations of the CVLSPL 107
8.2 VMM and VMM-materialize 110
8.3 InitVMM operation 112

9 Evaluation of VMM 117
9.1 Comparison between Migration and VMM Strategy 119
9.2 False Revisions 122
9.3 Revision Folding 123
9.4 Isolated Revisions 124

Part I

INTRODUCTION

1

INTRODUCTION

Contents

1.1 Motivation of the Dissertation 4
1.2 Problem Statement 5
1.3 Contribution 6
1.4 Overview of the Work 7
1.5 Research Methodology 8
1.6 Quick Reference 10
1.7 Structure of the dissertation 11

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.

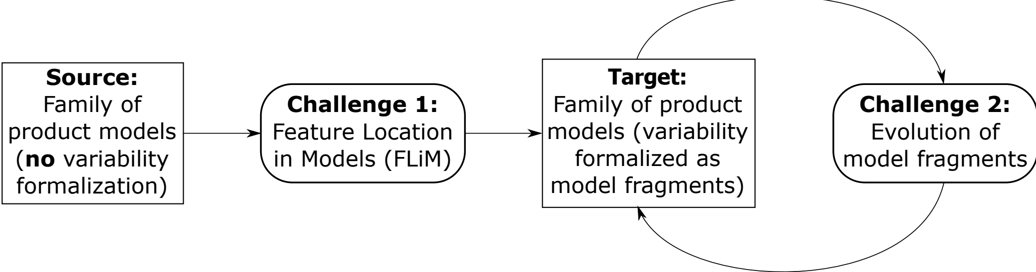


Figure 1.1: Motivation of the Dissertation

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		
	REVaMP²: 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.

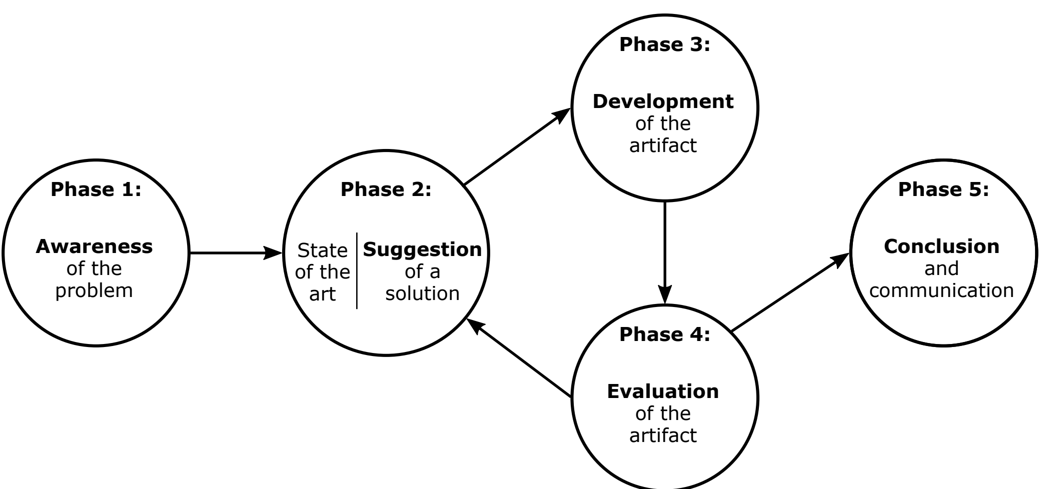


Figure 1.3: Research methodology followed in this Dissertation.

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.

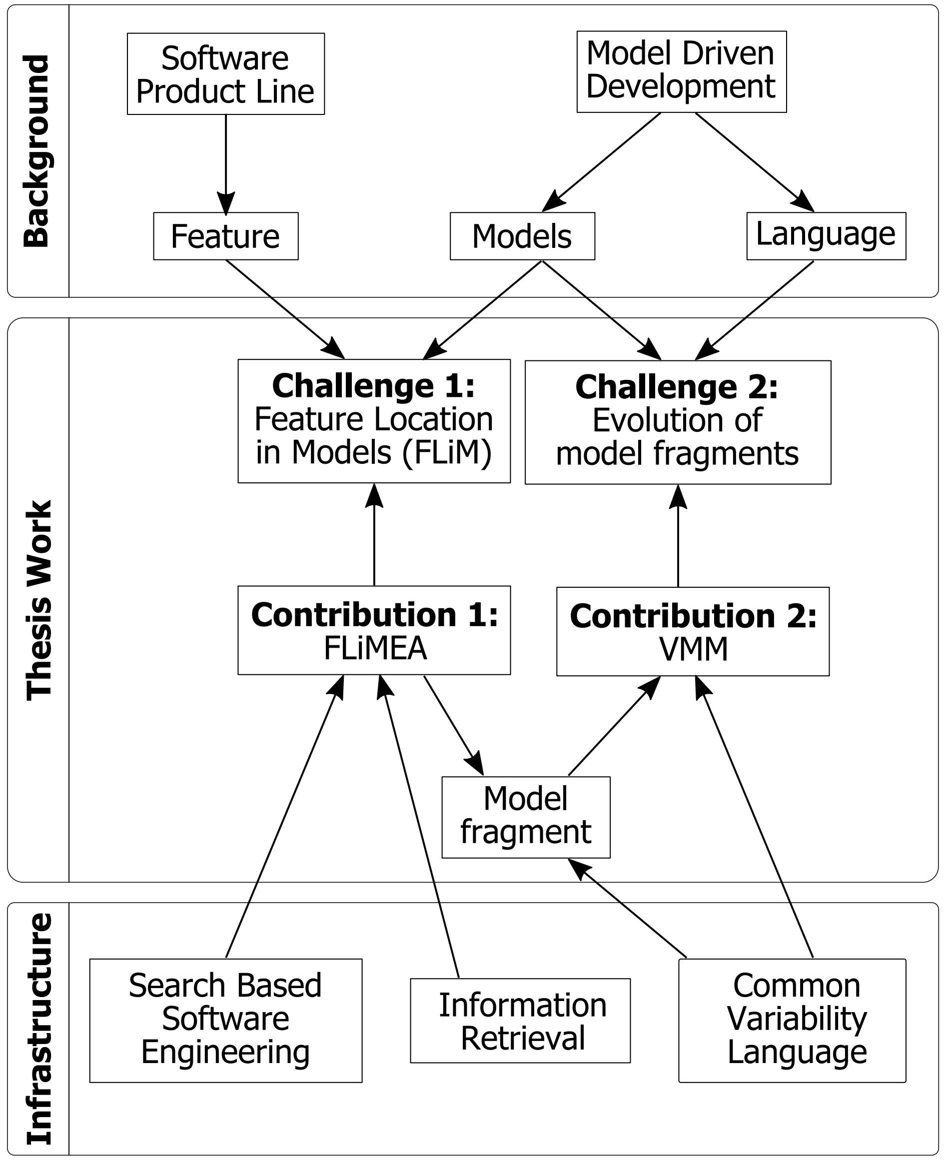


Figure 1.4: Cheat Sheet

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 Coevolution 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 14

2.2 Model Driven Development 14

2.2.1 Definition 15

2.2.2 Model Driven Software Development Initiatives . . 15

2.2.3 Domain Specific Languages 17

2.3 Software Product Lines 18

2.3.1 Definition 19

2.3.2 Software Product Line Processes 20

2.4 Running Example 21

2.4.1 The Induction Hobs Domain 21

2.4.2 The Common Variability Language applied to Induction Hobs 23

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.

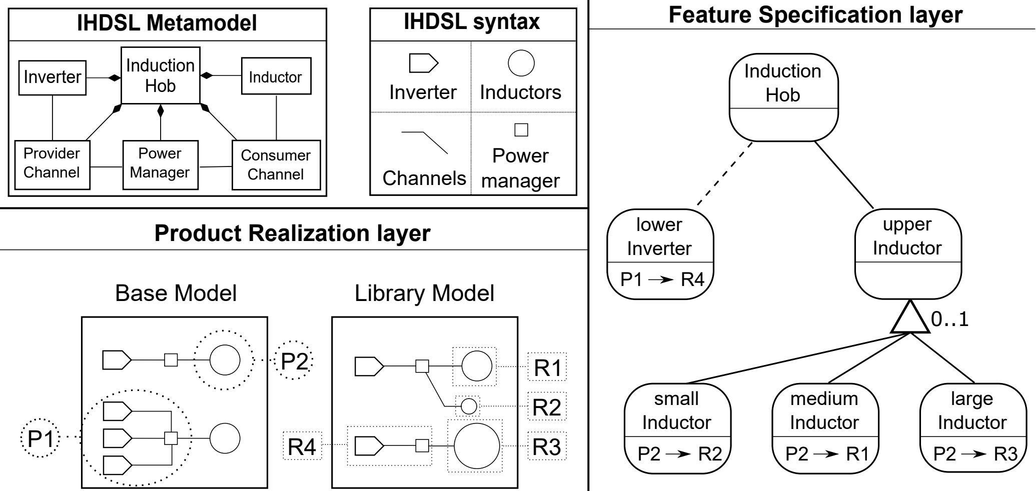


Figure 2.1: CVL applied to IHDSL

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 26

3.2 Feature Location in Models 26

3.2.1 Feature Location 26

3.2.2 Search Based Software Engineering 30

3.2.3 Model Driven Engineering 32

3.2.4 Motivation of our Feature Location in Models Approach 35

3.3 Evolution of Model Fragments 37

3.3.1 Model & Metamodel Co-evolution 37

3.3.2 Traditional Software Evolution 39

3.3.3 Software Product Line Evolution 39

3.3.4 Motivation of our Model and Language Co-Evolution Approach 41

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.

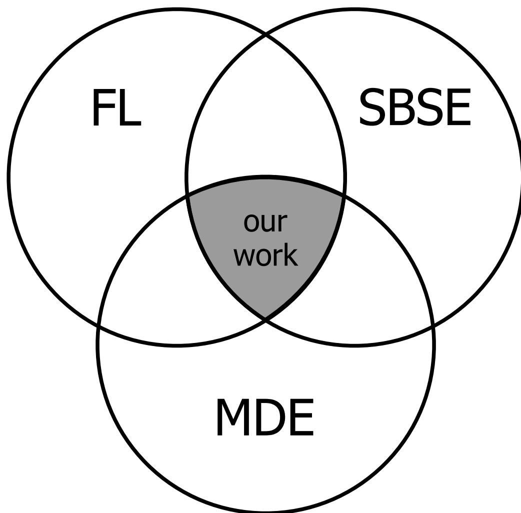


Figure 3.1: Overview of the scope of Feature Location in Models challenge

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 Fea-

ture 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.

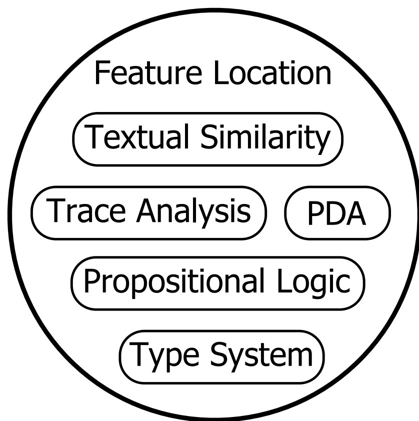


Figure 3.2: Overview of the scope of Feature Location in relation to FLiM challenge

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).

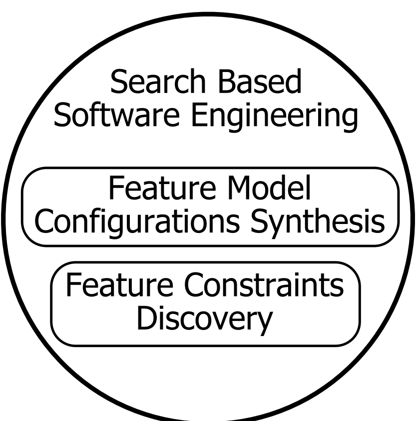


Figure 3.3: Overview of the scope of the Search Based Software Engineering in relation to FLiM challenge

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.

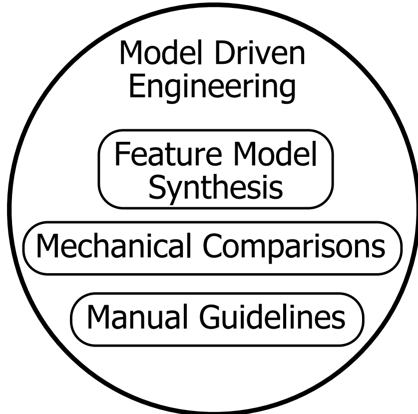


Figure 3.4: Overview of the scope of the Model Driven Engineering in relation to FLiM challenge

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

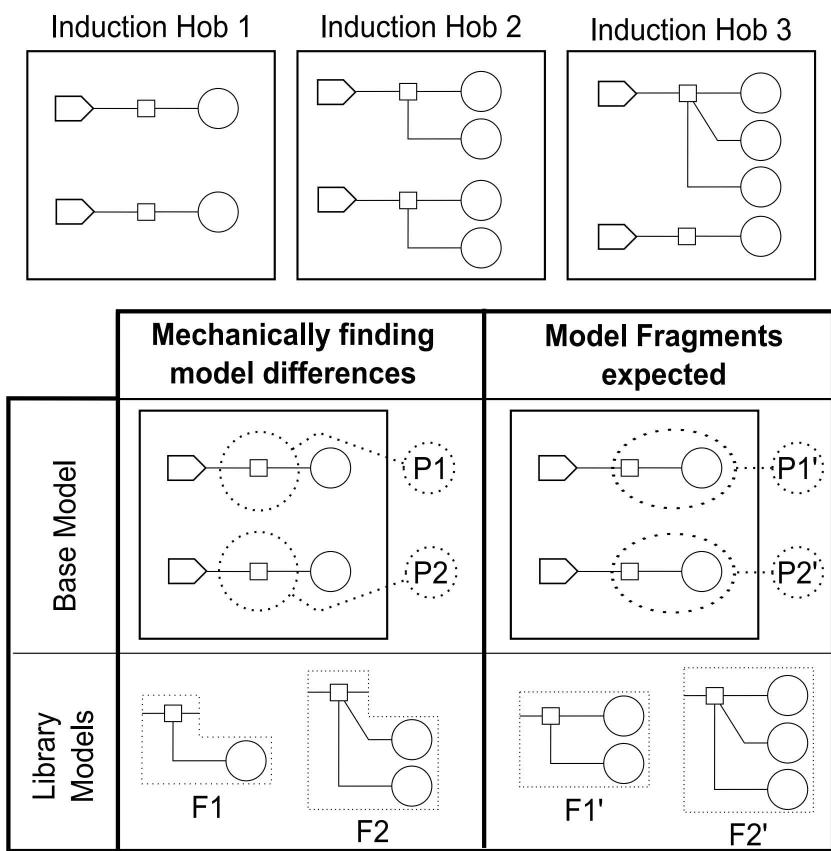


Figure 3.5: Motivation of the proposed approach to address FLiM challenge

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.

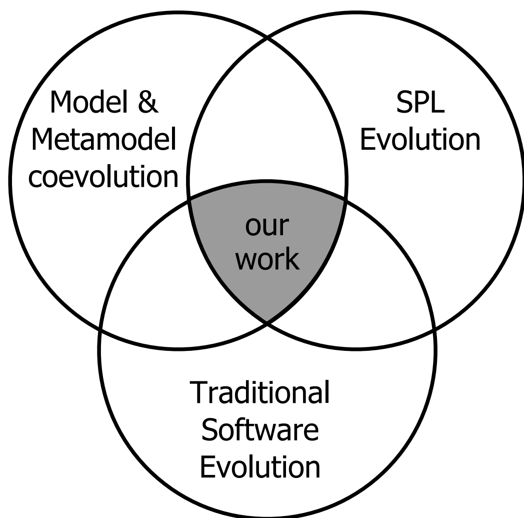


Figure 3.6: Overview of the scope of the evolution of model fragments challenge

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.

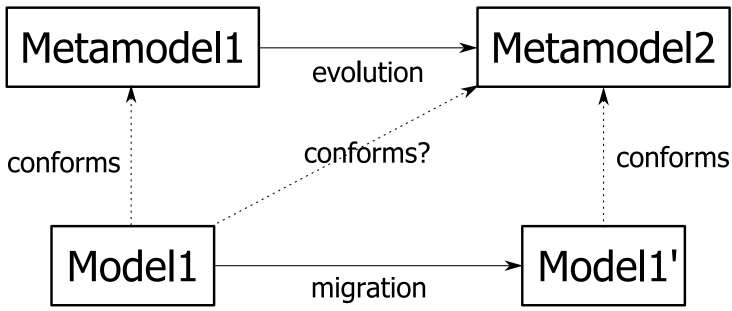


Figure 3.7: Model and Metamodel Co-evolution problem

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.

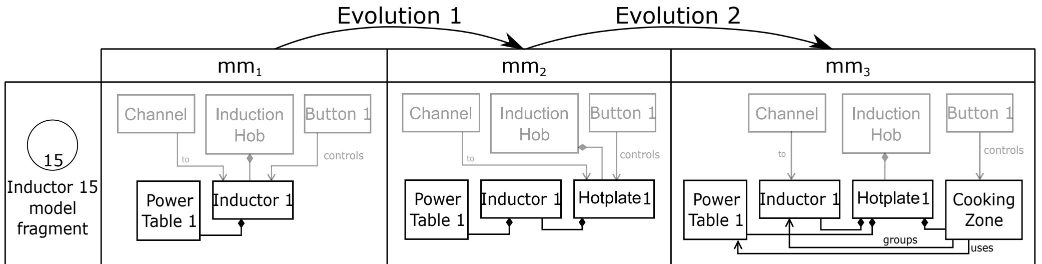


Figure 3.8: Evolution of Model Fragment through Migrations

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 **48**

4.2 Model Artifact **48**

4.3 Feature Knowledge **49**

4.4 Evolutionary Algorithm **50**

4.4.1 Encoding 51

4.4.2 Assessment 51

4.4.3 Genetic Manipulation 52

4.5 Ranking of feature realizations **52**

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.

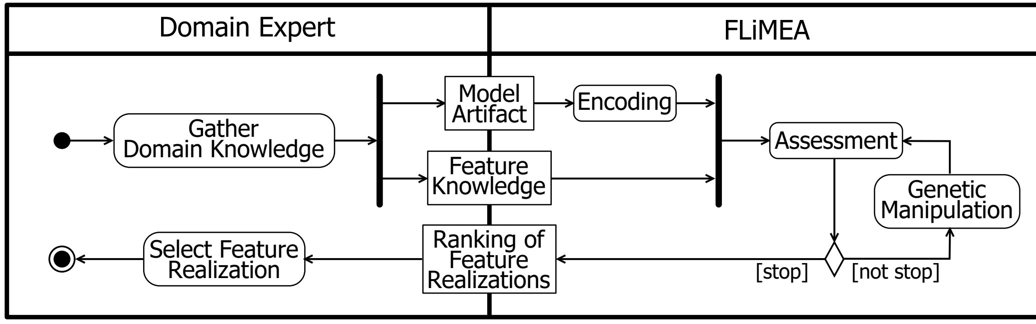


Figure 4.1: Activity diagram for the Feature Location in Models trough an Evolutionary Algorithm (FLiMEA)

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 Re-

trieval 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** **56**
5.2 **Encoding: Binary** **57**
5.3 **Fitness: Text-based similarity** **58**
5.3.1 Latent Semantic Analysis 59
5.3.2 Formal Concept Analysis 62
5.4 **Genetic Operations for Model Fragments** **66**
5.4.1 Parent Selection: Model Fragment selection 66
5.4.2 Crossover: Mask-based 67
5.4.3 Mutation: Random 69
5.5 **Variability in FLiMEA as Model Fragments** **70**

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.

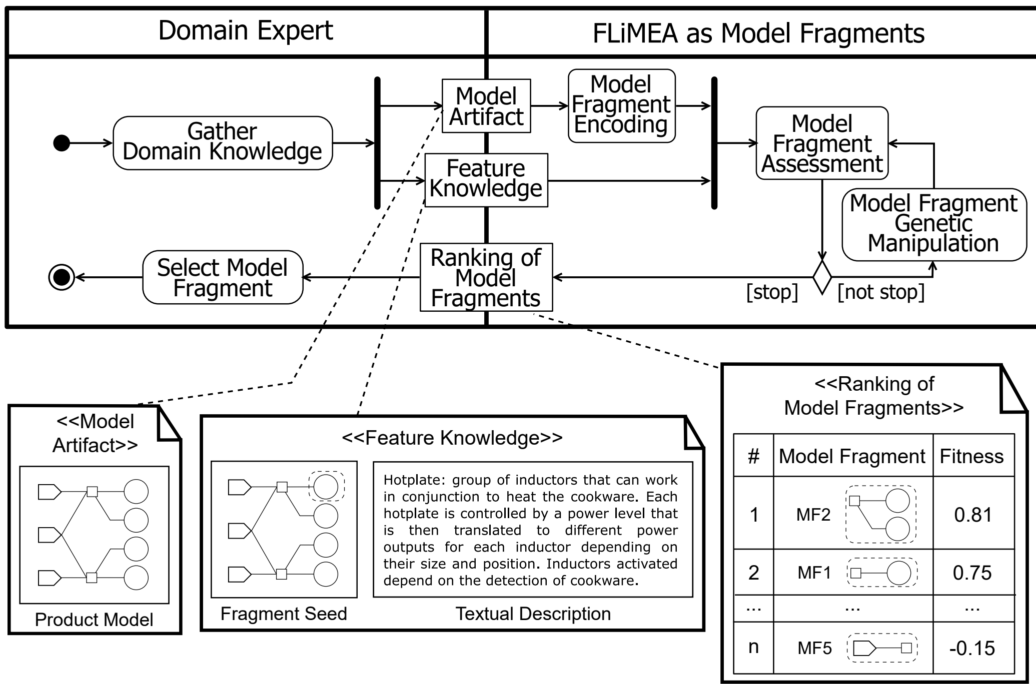


Figure 5.1: Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Model Fragments

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.

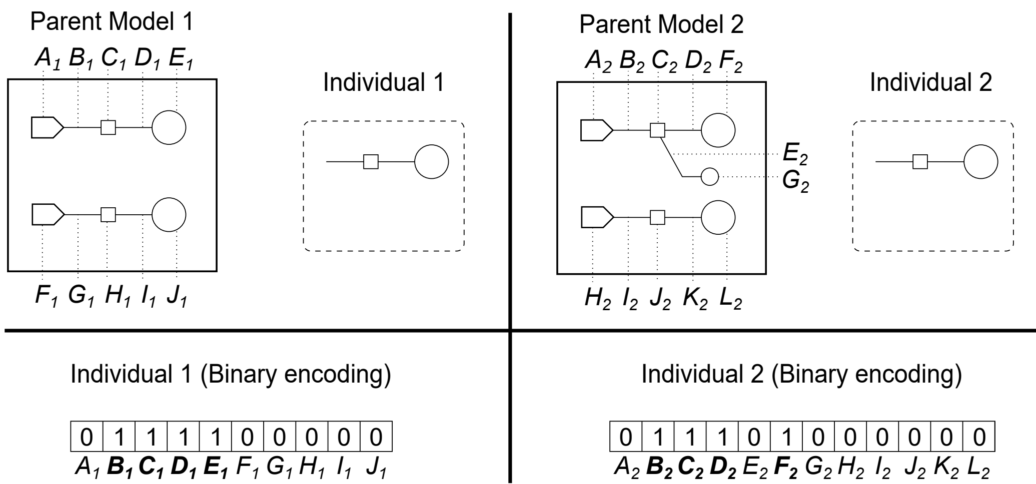


Figure 5.2: Binary-based encoding

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

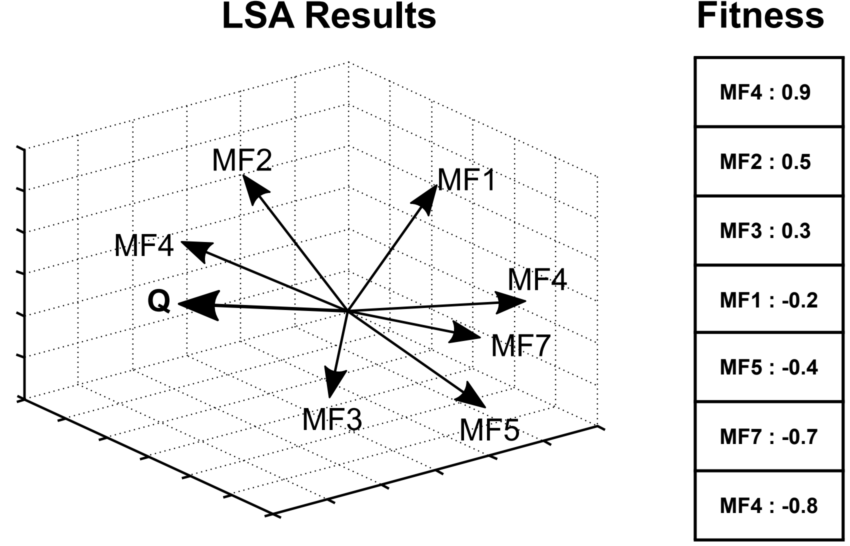
Figure 5.3 shows an example of the co-occurrence matrix. Each column corresponds to one of the individuals from the population. The last column is the query obtained from the textual description provided as part of the feature knowledge. Each row is one of the terms extracted from the corpora of text 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.

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

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.

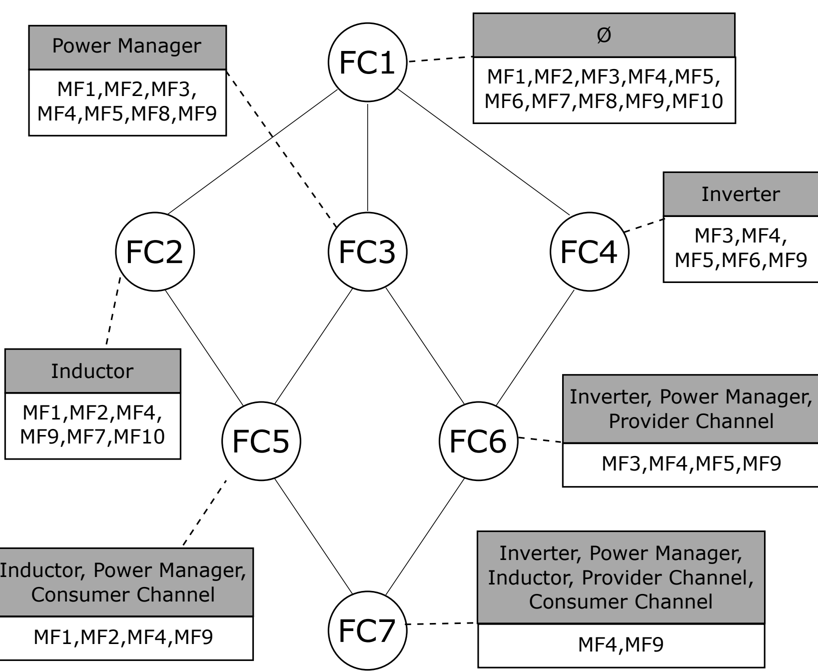


Figure 5.6: Lattice obtained from the Formal Context

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).

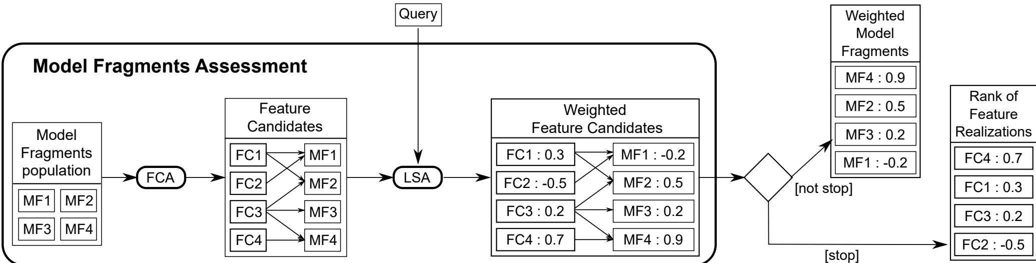


Figure 5.7: Fitness assessment for Feature Candidates and spread to Individuals

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.

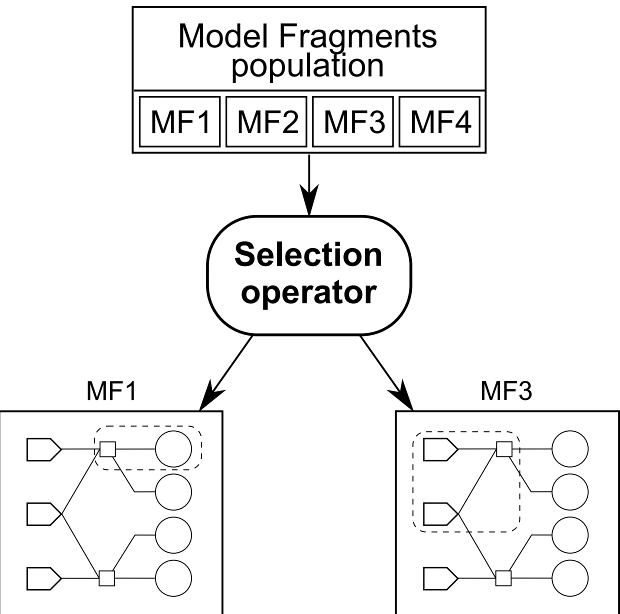


Figure 5.8: Selection Operator for Model Fragments

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].

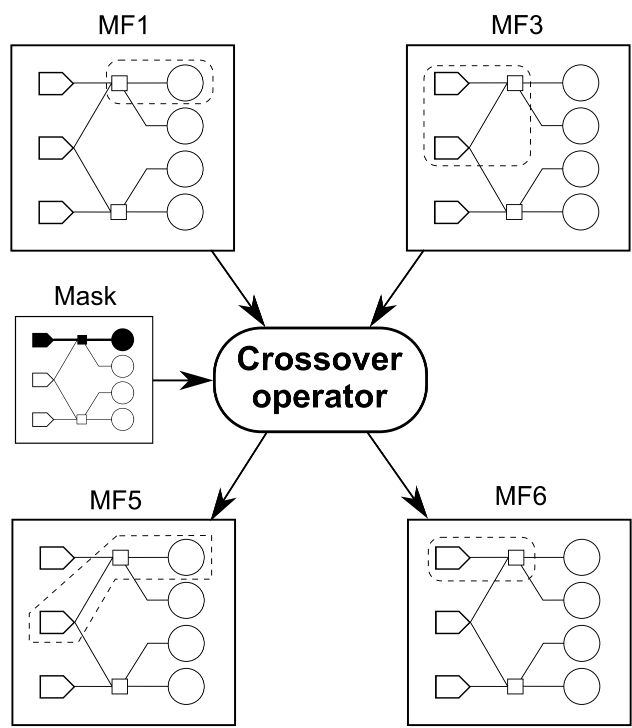


Figure 5.9: Mask-based Crossover for model Fragments

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

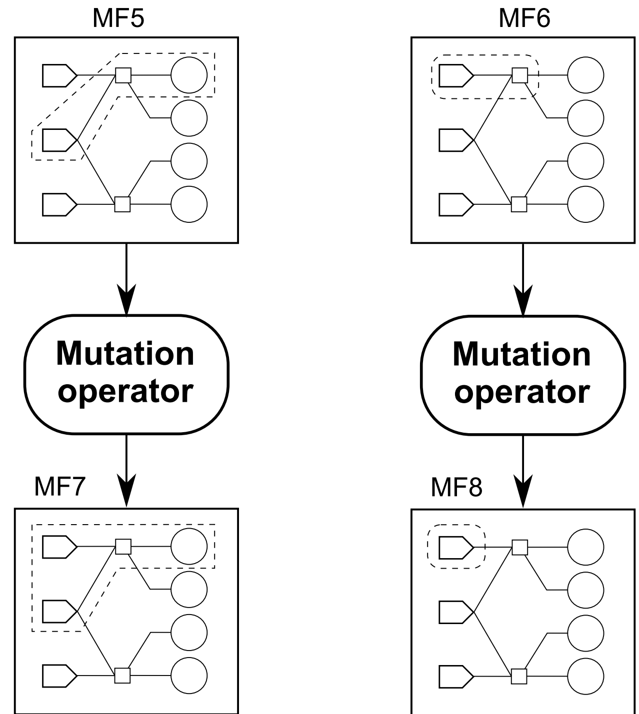


Figure 5.10: Random Mutation for model Fragments

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.

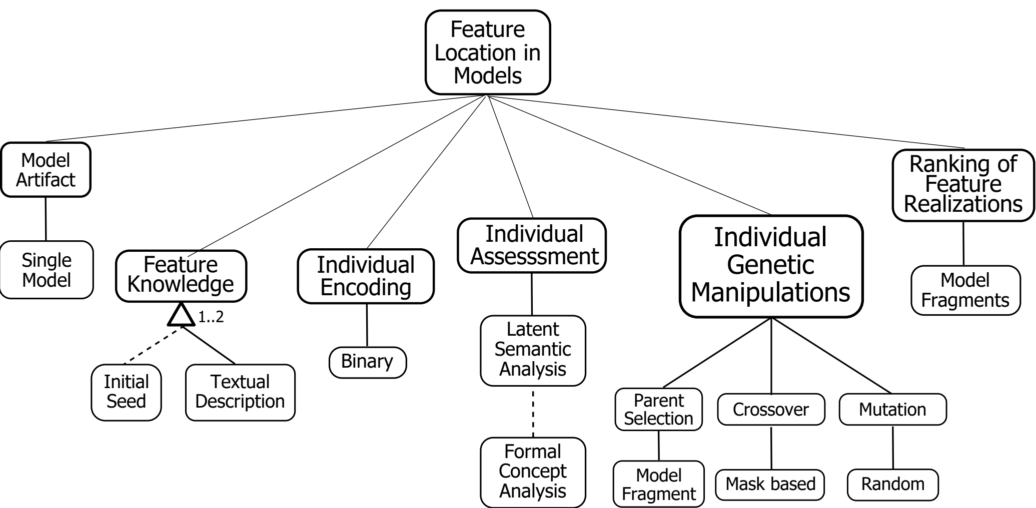


Figure 5.11: Variability of the Feature Location in Models as Model Fragments process

6

FLiMEA as Variation Points

Contents

6.1 Overview of the chapter 74

6.2 Encoding: Boundary-based 76

6.3 Fitness: Conceptual Model Patterns 78

6.3.1 Placement Signature Abstraction 78

6.3.2 Placement Signature Matching 79

6.3.3 Fitness computation 80

6.4 Genetic Operations for Variation Points 81

6.4.1 Parent Selection: Different Parents 82

6.4.2 Crossover: Parent change 82

6.4.3 Mutation: Sequential mutation 83

6.5 Variability in FliMEA as Variation Points 85

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).

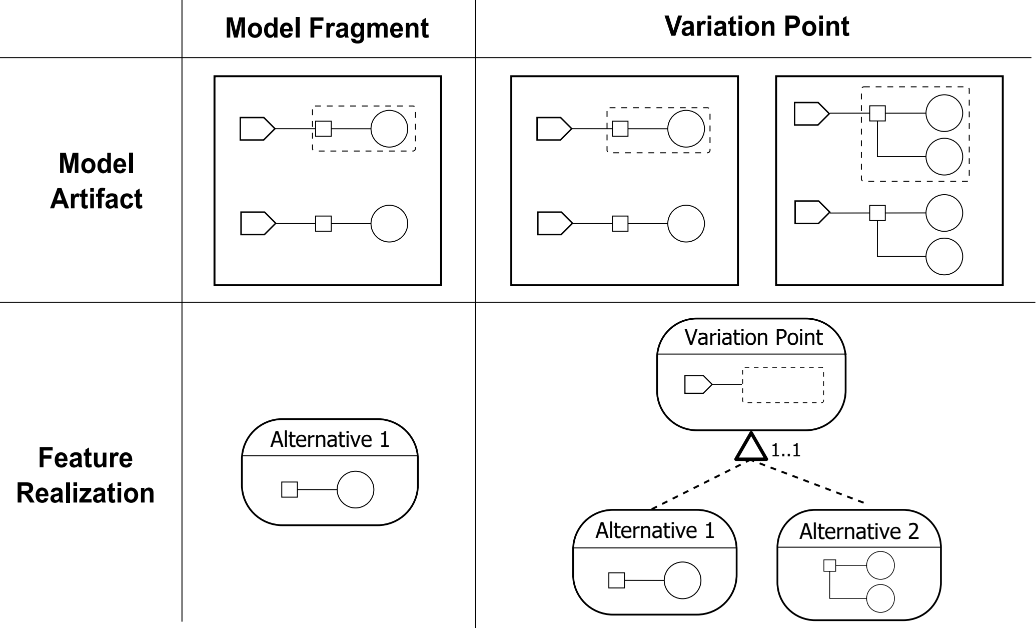


Figure 6.1: Model Fragment and Variation Point

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.

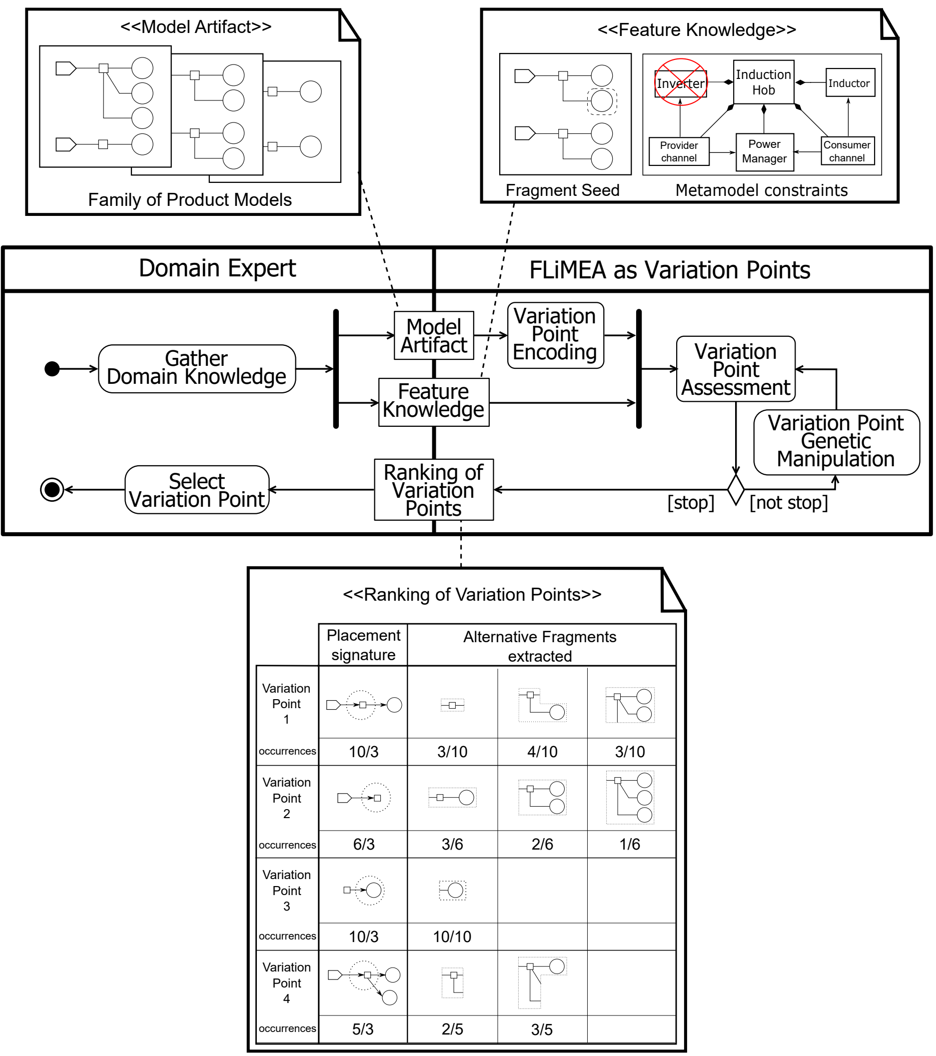


Figure 6.2: Activity diagram for the Feature Location in Models by an Evolutionary Algorithm as Variation Points

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

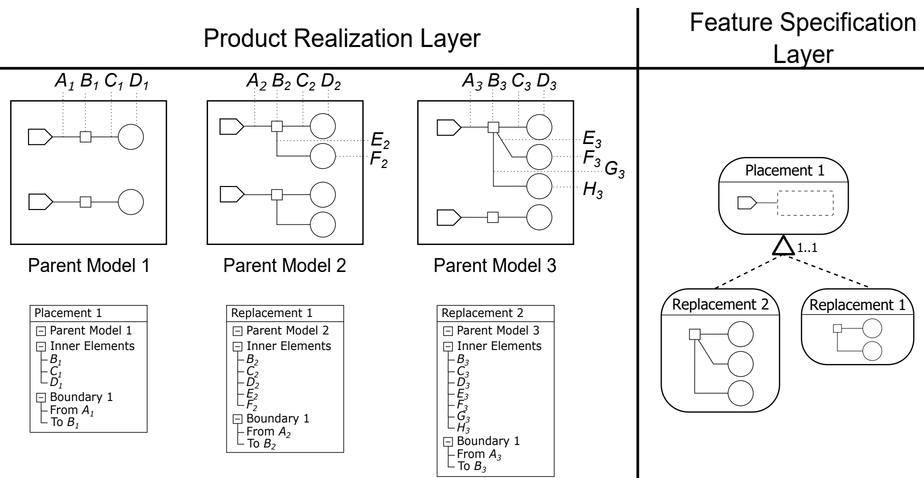


Figure 6.3: Example of a variation point using CVL-based encoding

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

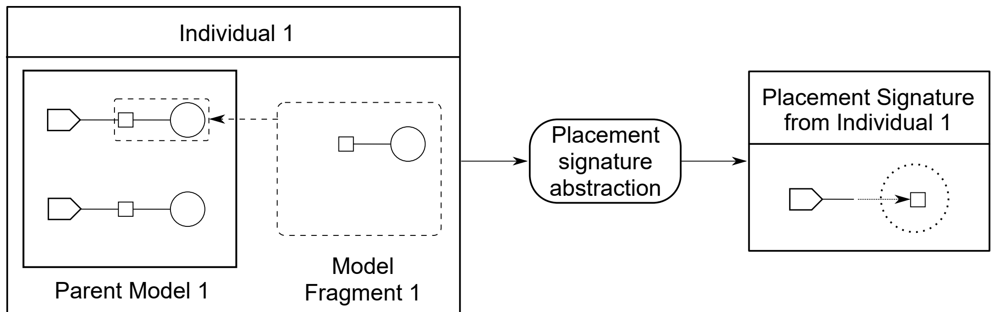


Figure 6.4: Placement Signature Abstraction

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

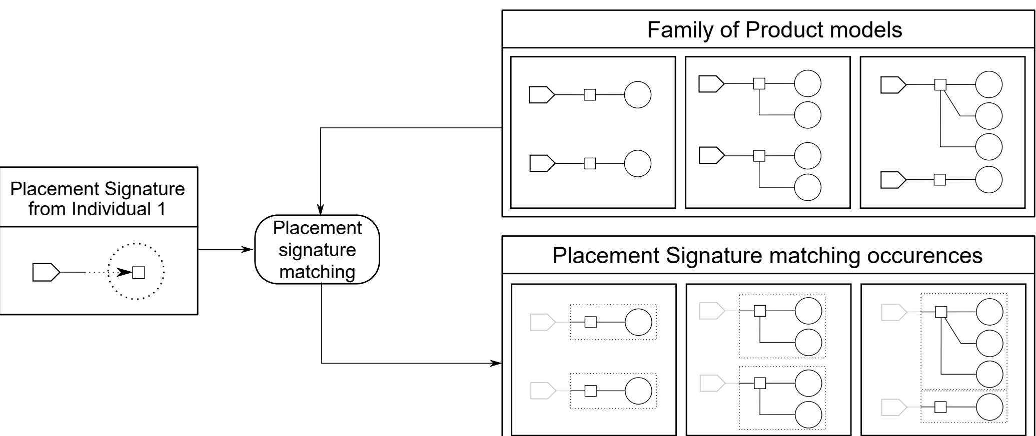


Figure 6.5: Placement Signature Matching

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

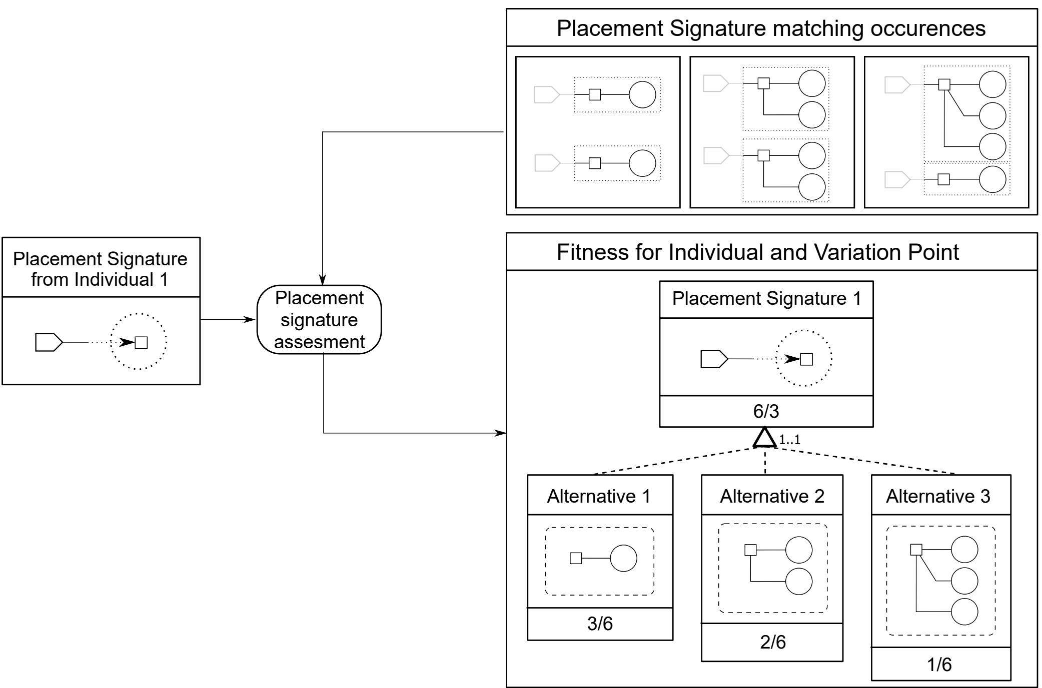


Figure 6.6: Fitness Assesment and Variation Point construction

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.

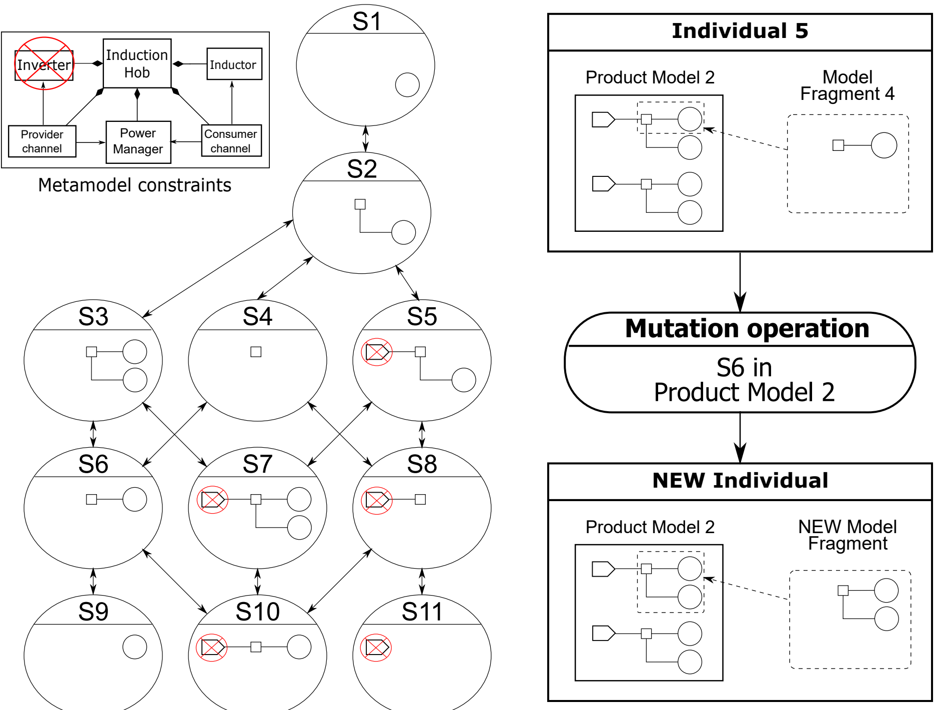


Figure 6.8: Sequential Mutation with constraints

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

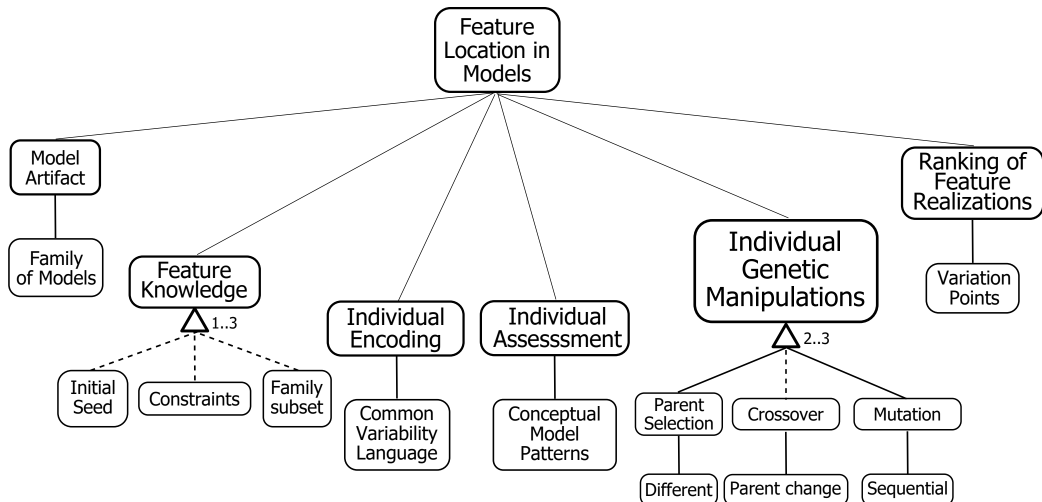


Figure 6.9: Variability of the Feature Location in Models as Variation Points process

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 **88**

7.2 Oracle **88**

7.2.1 Induction Hob Domain 89

7.2.2 Train Control and Management Domain 89

7.3 Test Cases **90**

7.4 Approach under Evaluation **91**

7.5 Comparison and Measure **91**

7.6 Measurements **93**

7.7 Results **95**

7.7.1 Evaluation 1 (SPLC'15) 95

7.7.2 Evaluation 2 (ICSR'16) 96

7.7.3 Evaluation 3 (MODELS'16) 98

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.

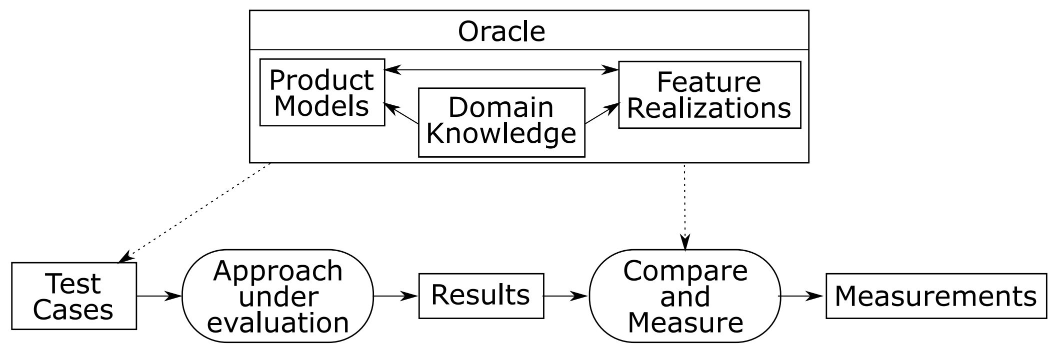


Figure 7.1: Setup of the evaluation

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.

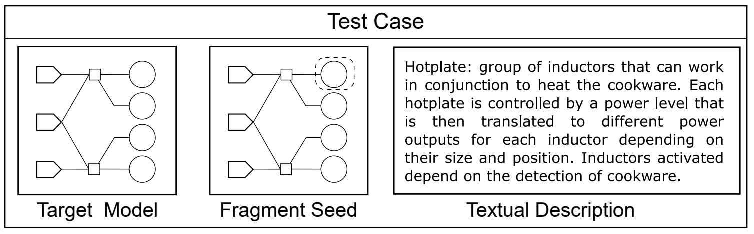
Again, we extracted an oracle that is composed of 23 trains where, on average, each product model is composed of around 1200 elements. The product models are built using 121 different features that can be part of a specific product model.

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.



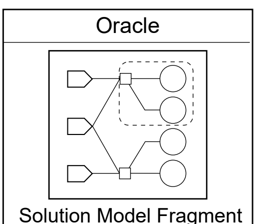


Figure 7.2: Test Case example

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

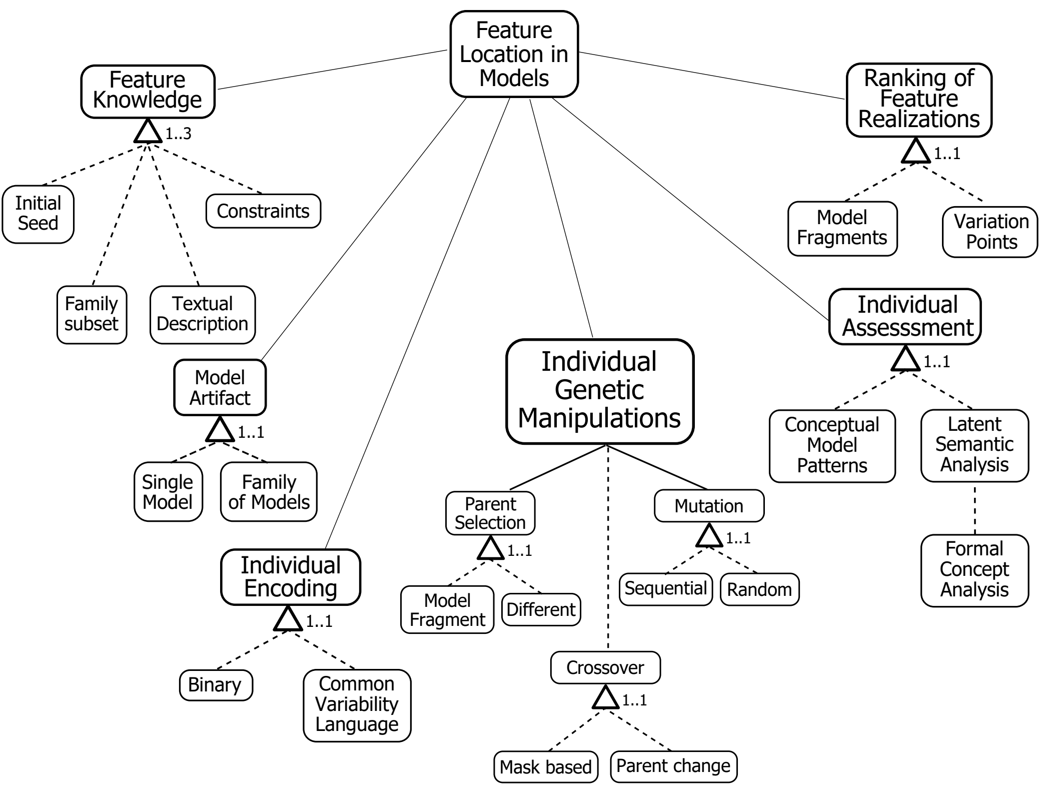


Figure 7.3: Feature Model of FLiMEA approach

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

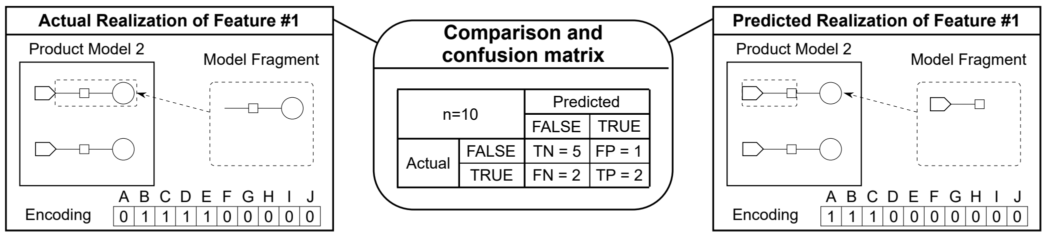


Figure 7.4: Example of confusion matrix for two candidate model fragments

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.

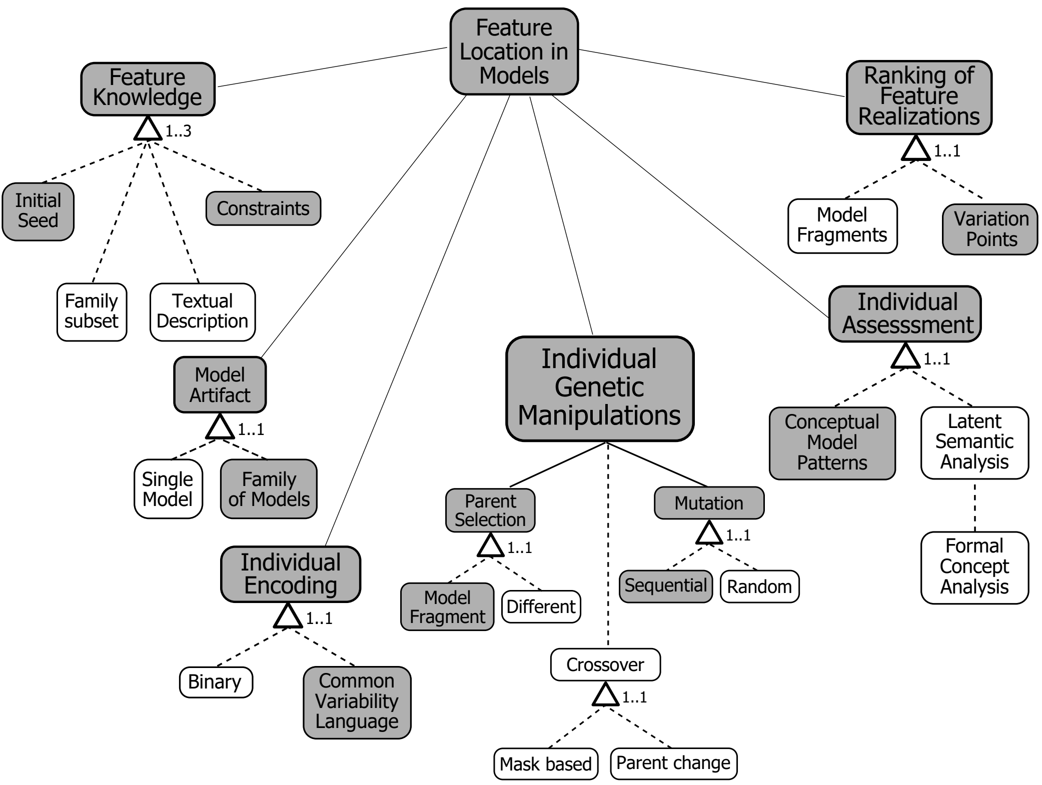


Figure 7.5: Configuration 1

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

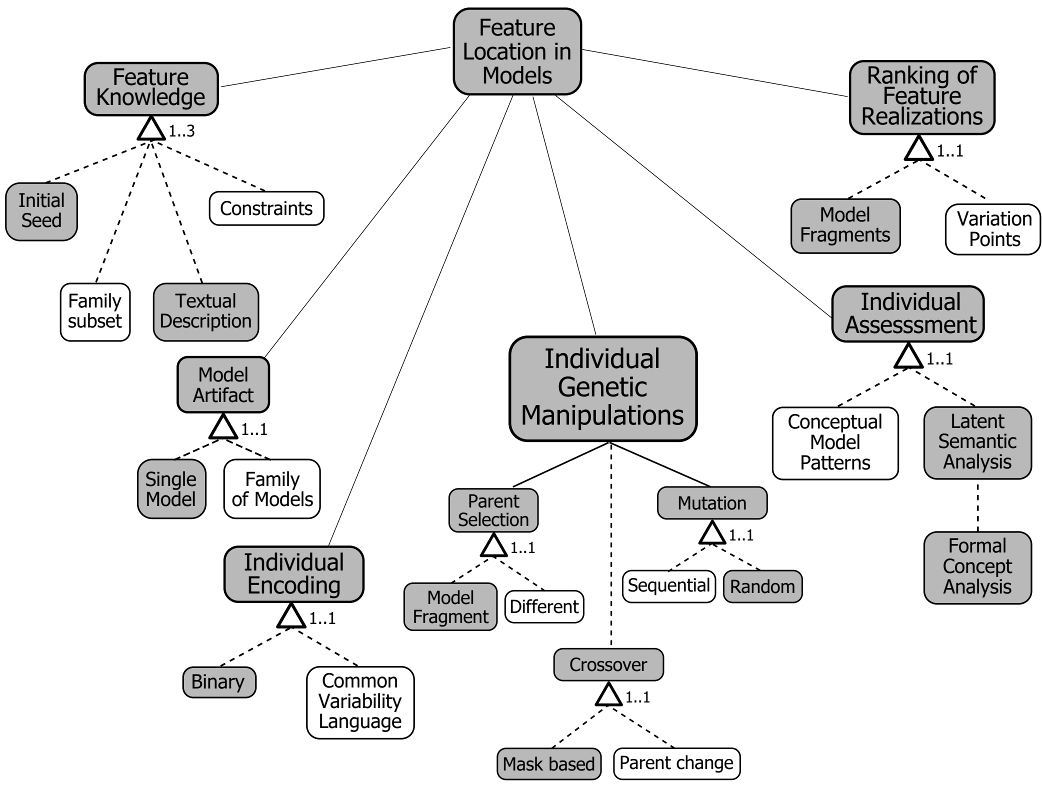


Figure 7.7: Configuration 3

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

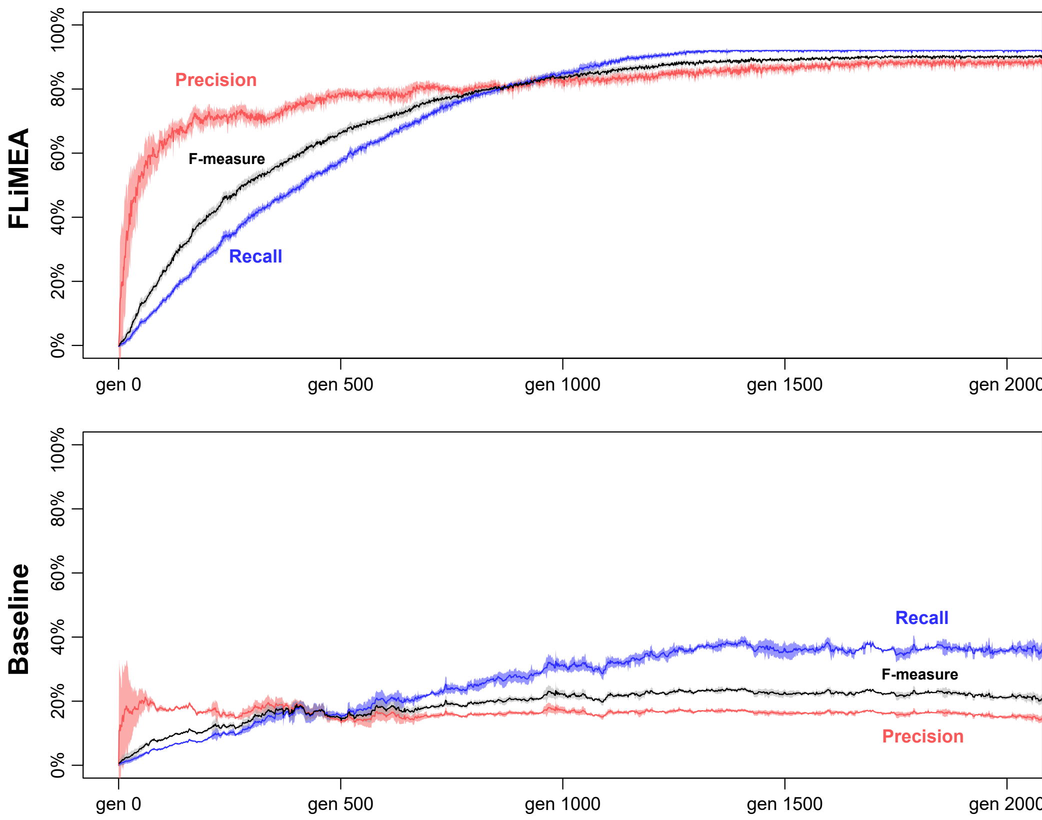


Figure 7.8: Mean Precision, Recall and F-measure for FLiMEA and the Baseline

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 106

8.2 Retrospective Case Study 106

8.3 The Variable MetaModel (VMM) 109

8.4 VMM operations 112

8.4.1 InitVMM operation 112

8.4.2 AddGen operation 114

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-

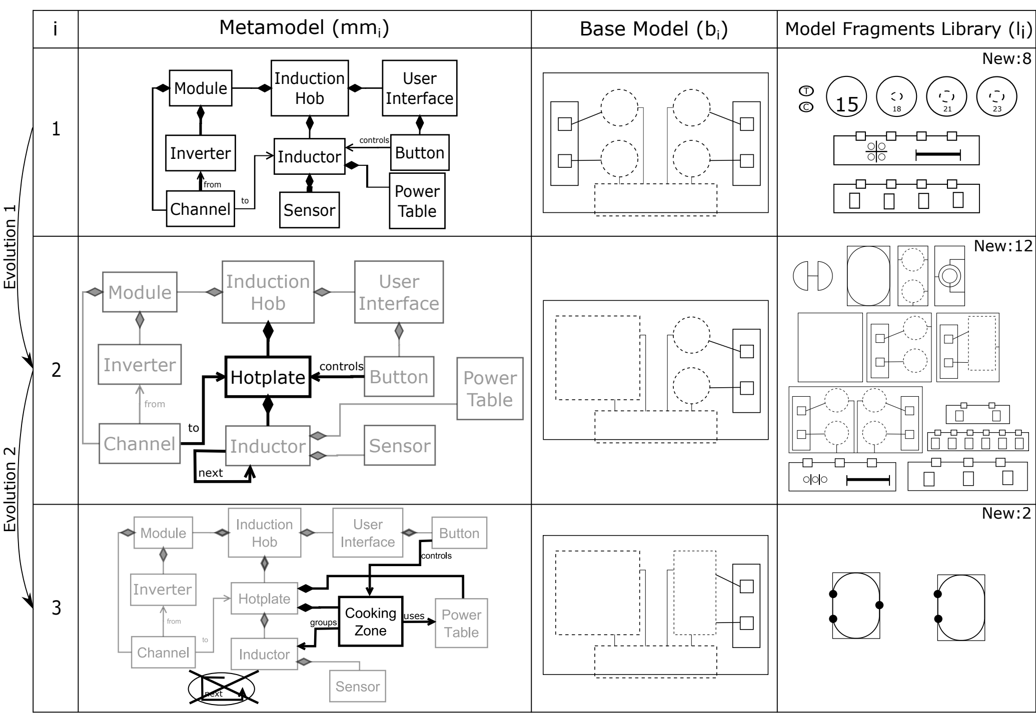


Figure 8.1: Model Generations of the CVLSPL

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:

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

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

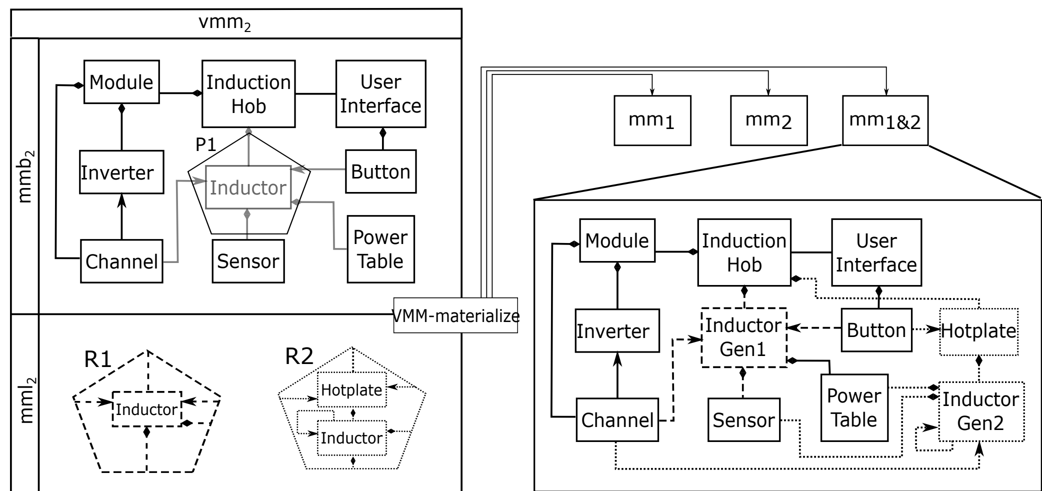


Figure 8.2: VMM and VMM-materialize

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:

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

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 mod-

els 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).

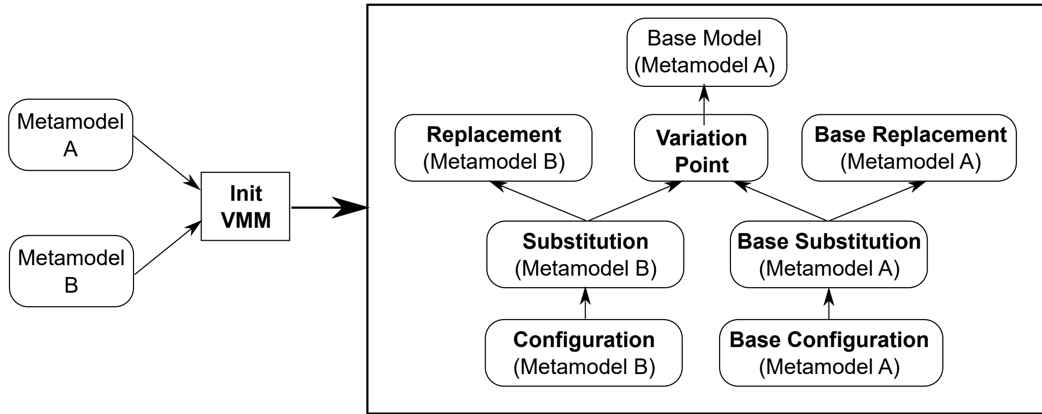


Figure 8.3: InitVMM operation

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 **118**

9.2 Migration Issues in VMM **118**

9.2.1 Overhead 118

9.2.2 Automation 119

9.2.3 Trust Leak 120

9.3 Lessons Learned **121**

9.3.1 False Revisions 121

9.3.2 Revision Folding 122

9.3.3 Isolated Revisions 124

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).

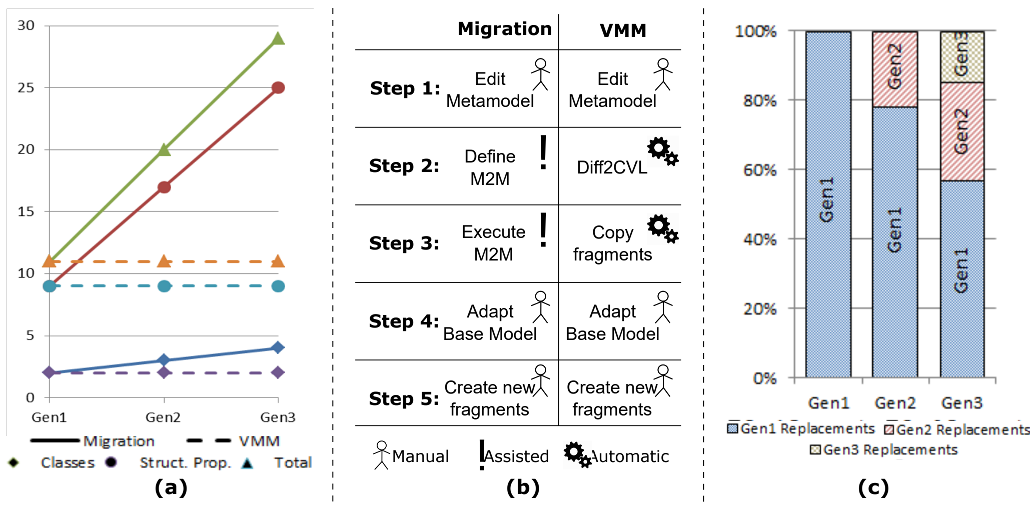


Figure 9.1: Comparison between Migration and VMM Strategy

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

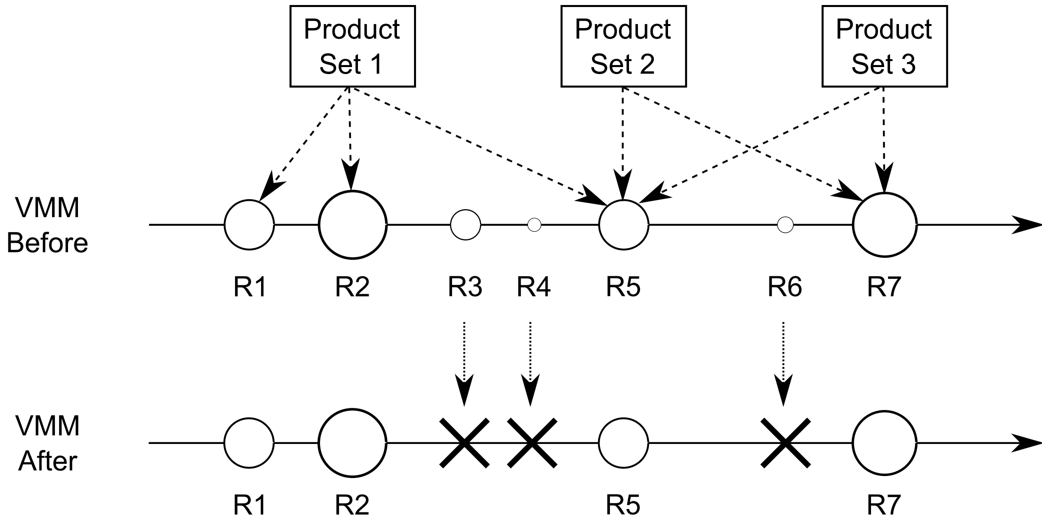


Figure 9.2: False Revisions

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

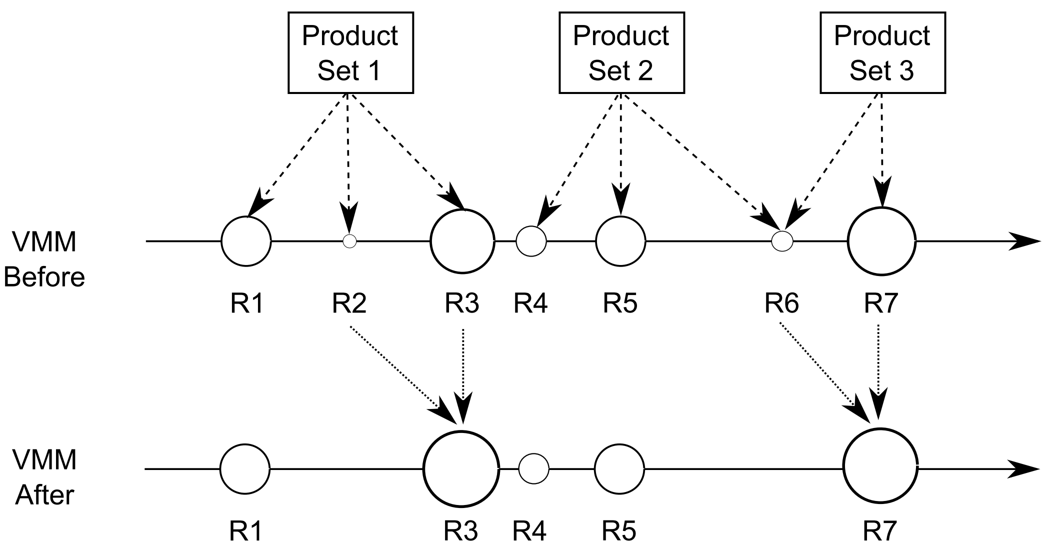


Figure 9.3: Revision Folding

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

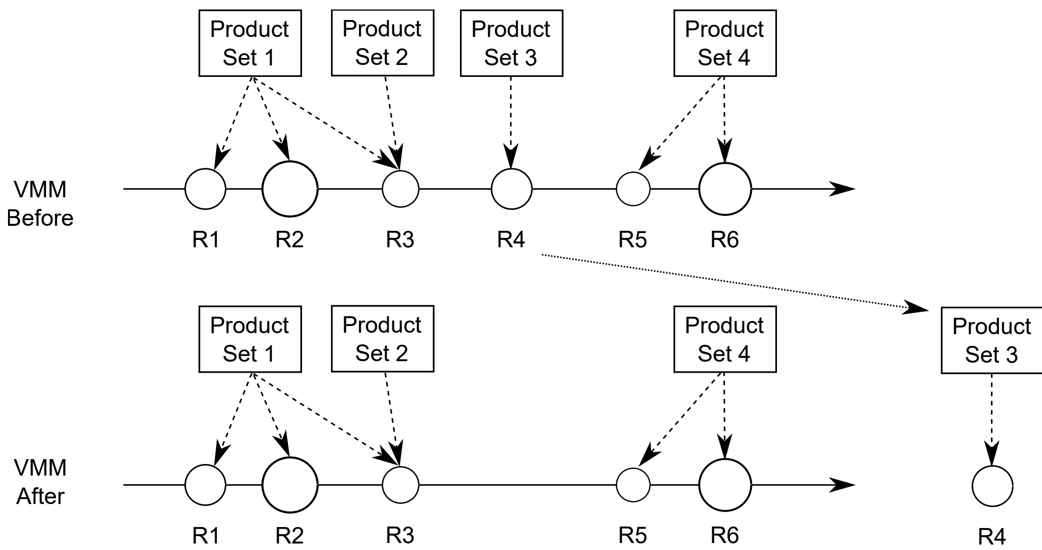


Figure 9.4: Isolated Revisions

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 **130**

10.2 Research Questions **130**

10.3 Ongoing Research **132**

10.3.1 Parameter values of the Evolutionary algorithm . . . 132

10.3.2 Multi-Objective Evolutionary Algorithms 133

10.3.3 Bug Location 134

10.3.4 Machine Learning Fitness 134

10.3.5 Trust Leak 134

10.4 Concluding Remark **135**

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 (FLiM-EA) 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, 1th 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.

Part V

PUBLICATIONS

11

Feature Location in Models

Contents

11.1 REVE'15 Paper 162
11.2 SPLC'15 Paper 171
11.3 ICSR'16 Paper 182
11.4 MODELS'16 Paper 199
11.5 TEVC'17 Paper 211

11.1 REVE'15 Paper

Title:	Automating the Variability Formalization of a Model Family By Means of Common Variability Language.
Authors:	Jaime Font, Manuel Ballarín, Øystein Haugen, Carlos Cetina.
Proceedings:	Proceedings of the 19th International Conference on Software Product Line (SPLC '15).
Location:	Nashville, Tennessee - July 20 - 24, 2015
Publisher:	ACM, New York, NY, USA
Pages:	411-418
DOI:	`http://doi.acm.org/10.1145/2791060.2793678`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work. He was also responsible for the oral presentation of the work which took place during the conference.

Automating the Variability Formalization of a Model Family By Means of Common Variability Language

Jaime Font[1,2] Manuel Ballarín[1] Øystein Haugen[2,3] Carlos Cetina[1]

[1]San Jorge University
SVIT Research Group
Autovía A-23 Km. 299
50830 Zaragoza, Spain
{jfont,mballarin,ccetina}@usj.es

[2]University of Oslo
Department of Informatics
Postboks 1080 Blindern
0316 Oslo, Norway
oysteinh@ifi.uio.no

[3]Østfold University College
Department of Information Technology
Postboks 700
1757 Halden, Norway
oystein.haugen@hiof.no

ABSTRACT

The aim of domain engineering process is to define and realise the commonality and variability of a Software Product Line. In the context of a family of models, spotting the commonalities and differences may become cumbersome and error prone as the number of models and its complexity increases. This work presents an approach to automate the formalization of variability in a given family of models. As output, the variability is made explicit in terms of Common Variability Language. The model commonalities and differences are specified as placements over a base model and replacements in a model library. The resulting Software Product Line (SPL) enables the derivation of new product models by reusing the extracted model fragments. Furthermore, the SPL can be evolved by the creation of new models, which are in turn automatically decomposed as model fragments of the SPL. The approach has been validated with our industrial partner (BSH), an induction hobs company. Finally, we present five different evolution scenarios encountered during the validation.

CCS Concepts

•Software and its engineering → Software product lines;

Keywords

Reverse Engineering, Model-based Software Product Lines, Variability Identification, Common Variability Language

1. INTRODUCTION

A Software Product Line (SPL) enables a planned reuse of software components into products within the same scope. The software product line engineering paradigm separates two processes; domain engineering (where the variability of the SPL is defined and realized) and application engineering (where specific software products are derived by reusing the variability of the SPL) [9].

The proactive strategy for the adoption of an SPL is traditionally regarded as the typical approach. Following this strategy, the assets of the SPL are developed prior to the derivation of any product [6]. However, a recent survey reveals that only a minority of industrial SPLs are planned proactively, being the extractive approach more used (where existing products are re-engineered into an SPL) [3].

In particular, in model-based SPLs, the members of the SPL are specified in the form of models. However, in the context of a family of models, manually spotting the commonalities and variability among the models may become cumbersome and error prone, particularly as the number of models and its complexity increases.

There are several research efforts towards automating the formalization of the variability existing among products [1, 2, 11, 14]. However, those works are mainly based on Feature Models extraction and do not properly support variability formalization by means of the Common Variability Language (CVL). In addition, existing works [10,12] are not designed with the evolution of the SPL on mind. The evolution of SPLs should be considered as the normal case, not as an anomaly [4].

This work presents Model Family to SPL, an approach to automate the variability formalization of a given family of models into an SPL. As output, the variability is made explicit in terms of CVL [5]. The model commonalities are formalized as a base model and variabilities are specified as placements over the base model and replacements in a model library. In addition, the resulting SPL can be further evolved to include new products. In particular, our approach enables the automatic decomposition of new product models into model fragments that are incorporated to the model library.

We have validated the approach with our industrial partner (BSH), the largest manufacturer of home appliances in Europe. Their induction division has been producing induction hobs (under the brands of Bosch and Siemens among others) over the last 15 years. We have applied the presented approach to a set of their induction hobs models to build an SPL to generate the firmware for their products. In addition, we present the five evolution scenarios faced by our industrial partner when evolving the SPL to incorporate new product models.

The rest of the paper is structured as follows: next section introduces some background about the CVL and our indus-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA

© 2015 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ISBN 978-1-4503-3613-0/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2791060.2793678

trial partner's domain. Section 3 presents our approach for extracting variability from a set of product models. In section 4 we present our experience applying the approach to our industrial partner's domain, focusing on the evolution scenarios encountered. Section 5 discusses related work. Finally we conclude the paper.

2. BACKGROUND

This section presents the main concepts of the Domain Specific Language (DSL) used to specify Induction Hobs (hereinafter referred as IHs) and the CVL. Both, the Induction Hob Domain Specific Language (IHDSL) and CVL are the techniques which we use to describe the model-based SPL of our industrial partner.

2.1 Induction Hob Domain Specific Language (IHDSL)

The IHDSL metamodel used by our industrial partner is composed of 46 metaclasses, 74 references among them and more than 180 metaclass properties. However, in order to gain legibility and due to intellectual property rights concerns, in this paper we use a meaningful simplification of it (see top-left corner of Figure 1).

Induction Hobs use electromagnetic induction phenomenon to cause the generation of heat on the cookware that is then transferred to the food. Induction hobs are composed of several elements, being the most important the inverter (where the energy is modulated) and the inductor (where the electromagnetic field is generated).

Top-right corner of Figure 1 shows the graphical representation of the IHDSL. The big rectangle represents the IH itself. It is composed of two power modules (vertical rectangles at both sides of the IH) and each of them holds two inverters (squares). Inverters are connected to the inductors (circles). The number inside each inductor represents the diameter of the inductor. The line that connects inverters and inductors represent the channel, which transfers energy from the inverter to the inductor. The user interface of an IH has buttons to configure the power level of each inductor. In top-right corner of Figure 1, the horizontal rectangle at the bottom of the IH represents the user interface. It has ports to connect each inductor with his button.

In order to gain legibility through the rest of the paper we will focus on the variability regarding the inductor. However, there are several parts of the induction hobs that are subject to variation such as the inverters and how are connected with the inductors. Our work with our industrial partner has covered the variability of the whole induction hob although only a subset is presented.

2.2 Common Variability Language (CVL)

CVL is a DSL for modeling variability in any model of any DSL based on Meta-Object Facility (MOF), an OMG's specification to define a universal metamodel for describing modeling languages. CVL defines variants of the base model by replacing parts of the base model by Model replacements found in a library. Figure 1 presents an overview of CVL.

The **base model** is a model described by a given DSL (here: IHDSL) that serves as the base for different variants defined over it. In CVL the elements of the base model subject to variations are the **placement fragments** (hereinafter placements). A placement can be any element or set of elements that is subject to variation.

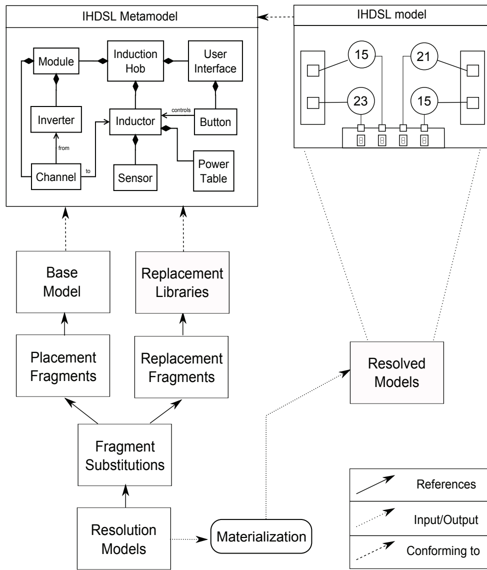


Figure 1: CVL Overview

To define alternatives for a placement we use a **replacement library**, a model described in the same DSL as the base model that will serve as a base 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** (hereinafter substitution). Each substitution references to a placement and a replacement and includes the information necessary to substitute the placement by the replacement. That is, 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 executed, the base model (with a placement substituted by a replacement) continues to conform to the same metamodel.

Each **resolution model** represents one variant of the base model. The resolution model references a set of substitutions that needs to be executed in order to create the variant. When a resolution model is materialized, produces a resolved model, which is a variant of the base model where the substitutions defined by the resolution model have been executed. For further details about the inner workings of CVL see [5].

3. MODEL FAMILY TO SPL

This section presents Model Family to SPL, our software process capable of turning the implicit variability existing

among a given set of similar models into explicit variability. In particular, Model Family to SPL takes a product family modeled in any DSL (conforming to MOF) as input and generates a CVL based SPL where commonalities and variabilities among the model family are explicitly defined. That is, each of the models of the given model family are expressed in terms of CVL, resulting in an SPL capable of generating all the products from the given model family.

Figure 2 shows an example of execution of *Model Family to SPL*. Top part shows the input of the process, the model family. Bottom part shows the output of the process, an SPL formalized by CVL models. Middle part shows how the execution of the processes is performed. Product Family to SPL is composed of two sub-processes, **Select Base Model** and **Product Model to SPL**. *Select Base Model* analyses the given family of models and determines which one of them is more suitable to be the base model. Once the base model is selected, *Product Model to SPL* compares a product model from the model family with the base model and updates CVL models to include the product into the variability definition. *Product Model to SPL* formalizes the variability of each given product model and incorporates it into the SPL

3.1 Select Base Model

The selection of the base model phase designates the base model that is used through the rest of the process. In this example we use the number of differences between the base model and the rest of the model family to determine the base model. Using the number of differences among the models produces simpler CVL models in terms of the number of substitutions needed to formalize each model. However, other values can be used to select the base model, the rest of the process can be executed no matter which base model is selected.

The first process executed as part of the Model Family to SPL is *Select Base Model*. Figure 2 shows an example of the execution of the process (top part). In addition, Figure 3 shows the state machine associated to the *Select Base Model* process (top part). Given a Product Model Family, the process *Select Base Model* takes the model family as input and proceeds as follows:

1.1 Compare. All the models from the model family (IH1, IH2 and IH3) are input into the compare operation. The models are paired two by two in all the possible combinations (the order doesn't matter) and then each pair is compared. The comparison is performed at element level, matching one element from first model with another from the second model. The process continues comparing pairs until there are no more pairs. The result is a set of differences between each pair of models processed. Figure 2 shows the result of the operation. For instance, the comparison between IH1 and IH2 produces a set of 4 diffs, because the four inductors of IH1 are different from the inductors of IH2.

1.2 Aggregate. The number of differences among each model and the rest of the models from the model family is added together. This is done for each of the models of the model family (IH1, IH2 and IH3). This aggregate value indicates the total number of differences among a given model and the rest of the product models. That is, the value indicates the number of differences that would need to be addressed if that particular model were the base model. For instance, if IH1 were the base model a total number of 7 differences would need to be addressed (4 differences with IH2 and 3 differences with IH3).

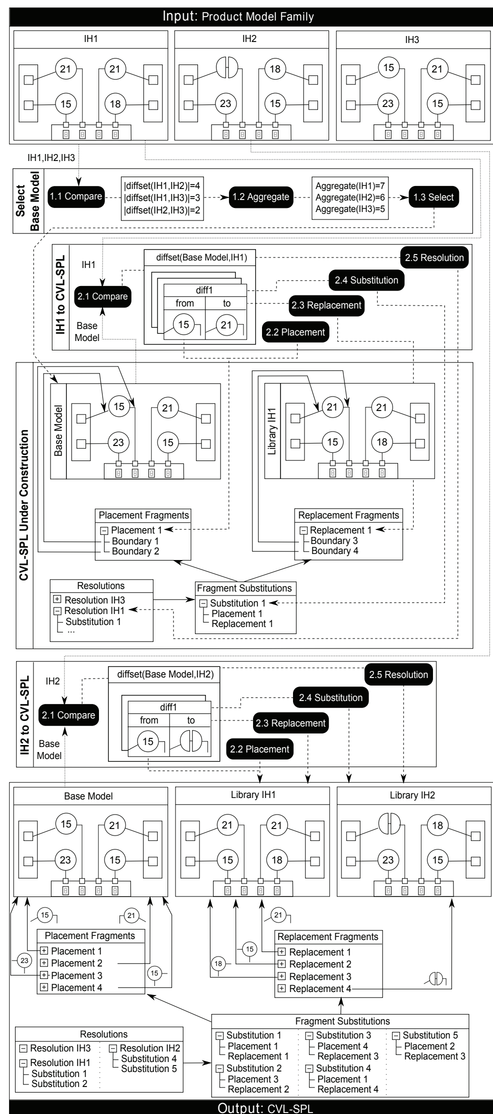


Figure 2: Model Family to SPL execution

1.3 Select. When the aggregated values have been cal-

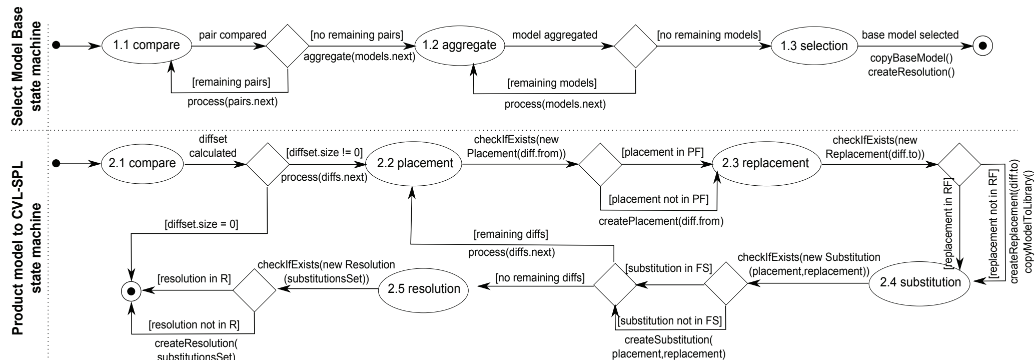


Figure 3: Select Base Model process and Product Model to SPL process state machines

culated for all the models, the model with the lowest value is designated as the base model. Therefore, it is included into the SPL, as it will be the base for all the products generated with the SPL. The model designated as base model (IH3 in this case) must be derivable from the SPL, therefore, a resolution model capable of generating the base model is created (Resolution IH3). As IH3 is the base model itself, there is no need of substitutions and Resolution IH3 is empty.

3.2 Product Model to SPL

After executing the first process (the base model has been designated), the second phase of the process starts, the population of the SPL. The population consists of executing the process *Product Model to SPL* for each of the product models of the input (except for the base model, that has been already included into the SPL).

The *Product Model to SPL* process, performs compare operations between each of the models from the given model family and the base model, using the same compare operation as in the previous process. However, this time we shall create and update the CVL models that define each of the differences among the model family as sets of placements, replacements, substitutions and resolutions.

Figure 2 shows an example of the execution of the *Product Model to SPL* process for IH1 (middle part, below *Select Base Model* process) and a snapshot of the SPL that is being constructed. In addition, Figure 3 shows the state machine for *Product Model to SPL* process (bottom part). Given a product model and a Base Model (designated by previous process), *Product Model to SPL* proceeds as follows:

2.1 Compare. The first model from the input model family (IH1) is compared with the base model. The result is a list of differences between the two models, *diffset(Base Model,IH1)*. Each difference has two elements, the *from* element references elements from the base model and the *to* element references elements from the other compared model (IH1 in this case). They reference the elements spotted as different by the compare operation. For instance, *diff1.from* element references the inductor of size 15 of the base model while *diff1.to* element references the inductor of size 21 of the IH1 model. It is important to notice that the difference not only holds the element that is different (inductor), but also the references involving that element (in this case references from the button and from the inverter).

2.2 Placement. The process checks if a placement holding exactly the same elements of *diff1.from* exists in the Placement Fragments model. As it does not exist, the process defines a placement over the base model (Placement 1). The references involving the differing element (the inductor) are defined as the boundaries of the placement (Boundary 1 and Boundary 2). If the placement is already defined in the Placement Fragments model it is not created again (see bottom of Figure 3).

2.3 Replacement. Once the placement is retrieved (created a new one or retrieving the existing one from the Placement Fragments model), the process continues with the replacement. Similarly as in previous step, the process checks if a replacement holding the information from *diff1.to* exists in the Replacement Fragments model. It does not exists, therefore it needs to be created, but this time will be defined over a model of the Replacements Library. To accomplish that, the model being processed (IH1) is copied into the fragments library and then a replacement is defined over it (Replacement 1). As with placement fragments, the references involving the differing element (the inductor) are defined as the boundaries of the replacement (Boundary 3 and Boundary 4). If the replacement is already defined in the Replacement Fragments model, there is no need to create a new one as indicated by the state machine (see bottom right part of Figure 3).

2.4 Substitution. Once the placement (Placement 1 from step 2.2) and the replacement (Replacement 1 from step 2.3) had been retrieved (creating them if necessary), the process is ready to create the substitution of the placement by the replacement. Similarly to previous steps, the process first checks if the substitution already exists in the Fragment Substitutions model. As the substitution does not exist, the process needs

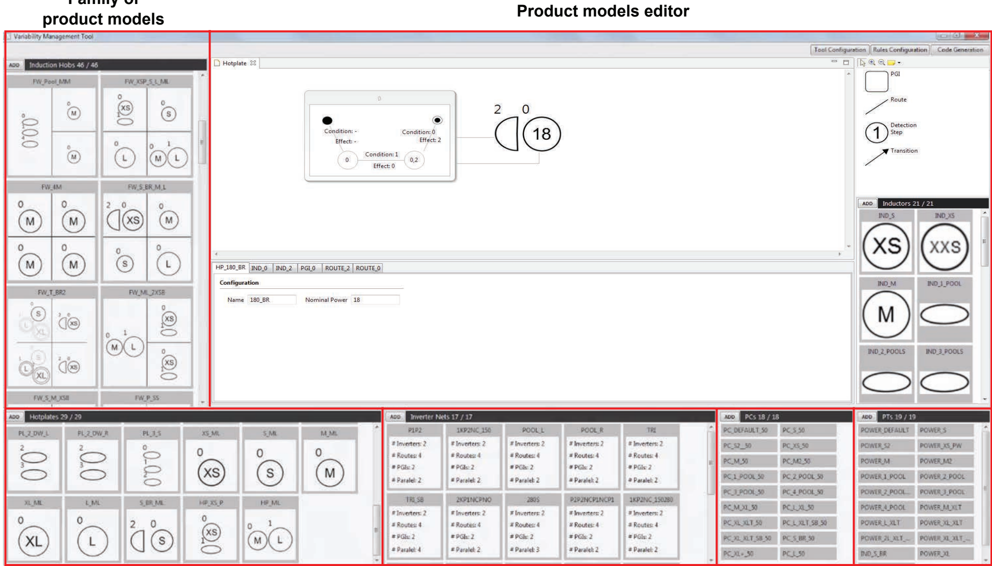


Figure 4: Resulting SPL for Induction Hobs domain

to create it. The substitution indicates that the Placement 1 can be substituted by the Replacement 1. As part of the definition of the substitution, links between the boundaries from the placement and the replacement are established. Therefore, when the fragment substitution is executed the elements can be updated properly and the model continues to conform to the metamodel. Similarly to previous step, if the substitution already exists there is no need to create it (see bottom of Figure 3).

At this point, the first difference (*diff1*) from the *diffset(Base Model,IH1)* has been processed. Now, the steps 2.2, 2.3 and 2.4 are performed for the rest of differences of the diffset. For each difference, a placement, a replacement and the proper substitution of the placement by the replacement are obtained (created or retrieved if already exists). The iterations for *diff2* and *diff3* are not shown in Figure 2.

2.5 Resolution. When all the differences from the diffset had been processed, the process is ready to create a resolution for the processed model (IH1). First, the process checks if the resolution already exists in the resolutions model. As the resolution does not exist, the process creates a new one (Resolution IH1). In this case, the process indicates that the resolution of IH1, involves the Substitution 1 (substitution of Placement 1 by Replacement 1) corresponding to the first diff processed. Similarly, substitutions for the rest of the differences of the diffset are included in this resolution.

This five-step process is repeated for all the models from the input (except for the base model). After executing *IH1 to SPL*, comes the execution of *IH2 to SPL*. The result is an SPL populated with all the models from the input family. Bottom part of Figure 2 shows the output of the *Model Family to SPL* operation. There is a base model and two library models conforming to the IHDSL. In addition, there are placements defined over the base model, and replacements defined over the library models. Moreover, substitutions are defined referencing placements and replacements; resolutions that generate each of the models received as input have been created based on those substitutions.

With the above process we obtain a CVL-based SPL capable of generating exactly the same models provided as input of the process. However, the commonalities and variabilities among the products are now explicitly formalized in terms of CVL. In addition, the *Product Model to SPL* process presented can be used to further evolve the variability of the SPL, decomposing new products and expressing them in terms of CVL. Next section presents the application of the approach to our industrial partner and the set of evolution scenarios encountered.

4. CASE STUDY: INDUCTION HOBS

This section presents our experience building a Product Line from an existing set of products from our industrial partner (BSH group). This company is the largest manufacturer of home appliances in Europe and one of the leading companies in the sector worldwide. Their induction division has been producing induction hobs (the brand portfolio is composed by Bosch and Siemens among others) over the last 15 years.

In order to implement the approach, several technologies

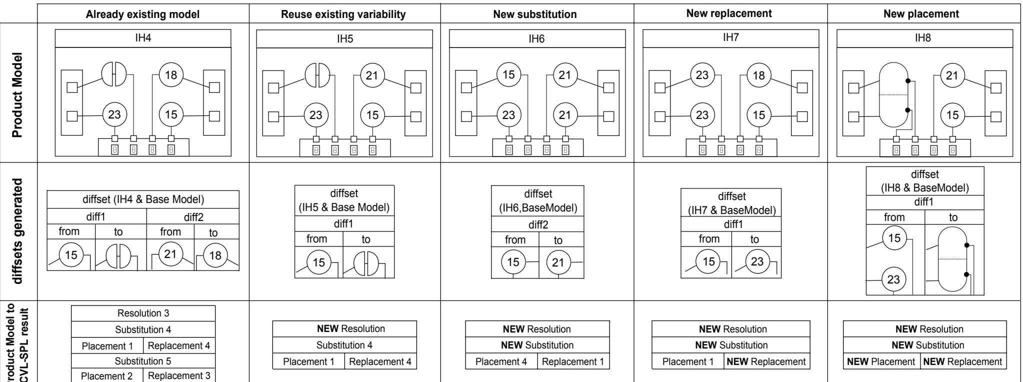


Figure 5: Five Scenarios of SPL evolution

are involved. Specifically, CVL can be applied to MOF based models, so the approach is developed within the Eclipse environment using the Ecore implementation and the Eclipse Modeling Framework (EMF)[1]. The comparisons among models are implemented based on EMF-Compare[2], which is an Eclipse framework to compare instances of EMF models. To build the frontend of the SPL we have used the Graphical Modeling Project (GMP)[3], a framework that provides a set of generative components and runtime infrastructures for developing graphical editors based on EMF. Finally, to add variability management capabilities to the graphical editor we have integrated the CVL tool from Sintef [5], a CVL prototype implementation that can be integrated into editor.

The initial input of the approach is a set of 46 induction hob models, corresponding to products that are currently being sold or that will be launched to the market in the immediate future. The set of models were developed following a clone and own [8] approach, where each IH has been modeled modifying a copy of the most similar IH present in the collection. For instance, a modification includes taking some elements from other induction hobs and customize them (if necessary, sometimes the elements do not require further customization). Therefore, the variability present among the models has not been explicitly defined, resulting in a set of models with implicit variability among its members. With regard to the products complexity, each of the IH models is composed of more than 500 elements, including around 100 class elements on average.

Figure 4 presents the resulting SPL tool that makes use of the variability information obtained applying the *Product Model to SPL* process. Top left part presents the Induction Hobs that have already been derived from the SPL. The set of products is the same as the one used as input; however, those induction hobs have been expressed in terms of the reusable model fragments extracted through the *Product Model to SPL* process. Bottom part presents the libraries of model fragments, holding the 102 replacement model fragments obtained by the approach. When deriving new products, the model fragments presented by the libraries can be reused. Finally, top right part presents the editor area, where product models can be derived and customized.

The tool contains the variability information extracted from the set of product models used as input. However, this information is extended when new product models are derived reusing existing model fragments (as the variability model needs to include the new product) and when new reusable model fragments are needed (the fragments need to be added to the model fragment libraries). For instance, one of our industrial partner engineer's creates a new empty model and populates it reusing elements from the library. Then, the engineer customizes some elements of the induction hob model using the editor and saves it. The *Product Model to SPL* process is automatically executed to include the new induction hob into the SPL, which can lead to an increment in the variability that is defined in the SPL or in the reusable model assets available to derivate further products.

Figure 5 presents five different examples that illustrates five different situations encountered when adding new models to the SPL. Each column presents one of the five examples. First row present the product model that is going to be added to the SPL. Second row shows the diffset generated when each model is compared with the Base Model (see bottom of Figure 2). Third row presents a summary of the changes that the application of *Product Model to SPL* produces over the CVL models. Next subsections present the five different scenarios.

4.1 Already existing model

First column is an example of the addition of a model that already exists in the SPL. The comparison between IH4 and the Base model produces a set of two differences (second row). When performing steps 2.2, 2.3 and 2.4, the placement, replacement and substitutions necessary to model diff1 already exists in the CVL models. Diff1 corresponds to already existing Substitution 4 (substitute Placement 1 by Replacement 4). Therefore, no placement, replacement or substitution is created for diff1. The same happens with diff2, that corresponds to Substitution 5 (sub-

[1] http://www.eclipse.org/modeling/emf/
[2] https://www.eclipse.org/emf/compare/index.html
[3] http://eclipse.org/modeling/gmp/

stitute Placement 2 by Replacement 3). During the creation of the resolution model (step 2.5), the process detects that the resolution already exists in the SPL (Resolution 3 composed of Substitution 4 and Substitution 5). Therefore, no resolution is created as part of step 2.5.

When the step 2.5 does not involve the creation of a new resolution model (as in this scenario), denotes that the model being processed is already part of the SPL. The process *Product Model to SPL* automatically skips the inclusion of this model in order to avoid duplicates. By means of this scenario, we avoid the inclusion of redundancy into the SPL.

4.2 Model reusing existing variability

Second column is an example of the addition of a model that reuses the variability already defined in the SPL to generate a new product model. The comparison between IH5 and the Base model produces a set of one difference (second row). Execution of steps 2.2, 2.3 and 2.4 detects that diff1 corresponds to Substitution 4 (substitute Placement 1 by Replacement 4). During step 2.5 the resolution model does not exist in the SPL, therefore, a new resolution model that includes Substitution 4 is created.

When *Product Model to SPL* does not create any substitution means that already existing variability is being used to create a new product model. However, if the resolution model does not exist in the SPL, a new resolution including the substitutions identified for each diff is created. By means of this scenario, we have created a new product reusing existing variability.

4.3 Model requiring a new substitution

Third column shows the addition of a model that needs the creation of a new substitution in order to be included into the SPL. The comparison between IH6 and the Base model produces a set of only one difference (second row). Then, during step 2.2 a placement for diff1.from is identified (Placement 1). Similarly, during step 2.3 a replacement for diff1.to is identified (Replacement 4). However, during step 2.4 no existing substitution is identified, therefore a new one is created. Then, during step 2.5 a new resolution is created, holding the new substitution created in previous step.

Sometimes the placement, replacement and substitution for a given diff already exists in the SPL (as in previous scenario) while other times only the placement and replacement exists and a new substitution is created (as in this scenario). However, in both cases we are reusing already existing model fragments to create new product models. By means of this scenario we show how the existing variability is reused in the creation of new product models.

4.4 Model requiring a new replacement

Fourth column is an example of the addition of a model that requires the creation of a new replacement in order to be formalized and included into the SPL. The comparison between IH7 and the base model produces a set of one difference (second row). Step 2.2 determines that diff1.from correspond to the already existing Placement 1. By contrast, Step 2.3, determines that there is no replacement corresponding to diff1.to in the SPL models, therefore it is created. As a new replacement has been created in step 2.3, step 2.4 creates a new substitution of the Placement 1 by the just created replacement. Finally step 2.5 creates a new resolution model including the new substitution.

When the step 2.3 involves the creation of a new replacement, the next step 2.4 will always require the creation of a new substitution (as the substitution involves a new created placement, it cannot exist in the SPL). By means of this scenario, the variability defined in the SPL has been increased, including a new replacement that now is available for the construction of other models.

4.5 Model requiring a new placement

Fifth column is an example of the addition of a model that requires the creation of a new placement. The comparison between IH8 model and the base model returns a set of one difference (second row). Then, step 2.2 detects that there is no placement corresponding to diff1.from; therefore a new placement is defined over the base model. Then, in step 2.3 a new replacement defined by diff1.to is created in the SPL. As part of step 2.4, a new substitution (that substitutes the new placement by the new replacement) is created. Finally, the resolution model including the just created substitution is created as part of step 2.5.

If the step 2.2 involves the creation of a new placement, then, a new replacement (step 2.3) and a new substitution (step 2.4) will be also created. It is important to notice that during the inclusion of IH8 model into the SPL a new replacement has been created. This replacement overlaps with other existing replacements (Replacement 1 and Replacement 3), as it is defined over the same model elements as other existing placements. However, this situation does not poses a threat to the stability of the SPL models. Substitution in CVL can be restricted, to avoid situations where two overlapping placements try to be replaced. Therefore, it is safe to define overlapping placements as long as the restrictions among them are correctly defined.

5. RELATED WORK

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, [2] 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 [1], the authors presents 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.

Other research efforts rely on the source code of the products in order to extract the variability model. In [11] the authors present a tool-supported approach for reverse engineering FMs from different sources, such as Makefiles, preprocessor declarations, and documentation. They focus on identifying parents and combine logic formulas and descriptions as complementary sources of information. In addition, [14] 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 produce feature candidates that are manually pruned (to remove non-relevant candidates). However, these approaches rely on the source code level as

input for the process, focusing in the generation of Feature Models. By contrast, our approach deals with the particularities of the CVL and is applied at model level.

In [10], the authors propose a generic framework for mining legacy product lines and automating their refactoring to contemporary feature-oriented SPLE approaches. In [7] the authors present MoVaC, an approach to identify and analyse commonalities and variability among a set of models, with the focus on the visualization of the results. In [12] the authors propose an approach to synthesize an SPL from the comparison of a set of models. The variability is extracted from the set of models and then a CVL model for the SPL is proposed. The approach is further refined in [13] to enable the inclusion of new models to the SPL. As output, a CVL model for the SPL is proposed to be manually enhanced. We further extend those works, automatically selecting a base model among the input models based on the metric desired. In addition, we have validated the approach building an SPL for an industrial environment, extracting the variability of a set of real induction hob models. Furthermore, we present the five different evolution scenarios encountered during the validation and how the approach handles them in order to evolve the variability of the SPL.

6. CONCLUSIONS

We have presented the *Model Family to SPL process*, capable of automating the formalization of the variability among a given set of similar product models. In addition, the generated SPL can be further extended in order to increase the variability specification. The presented approach has been tooled within the Eclipse environment using already existing technologies such as EMF Runtime, EMF Compare and GMP. Then, the approach has been validated with our industrial partner. Finally, we have presented the five different evolution scenarios encountered when evolving the variability specification by our industrial partner.

However, our current implementation has some limitations. For instance, the concrete syntax used to represent each of the elements from the library is not automatically produced. Therefore, some customization regarding the concrete syntax has been performed in order to present the resulting SPL tool. We plan to provide means for automating the generation of a graphical syntax following a generative approach (similar to the Graphical Modeling Project).

CVL materialization generates product models from resolution models. However, the graphical editor shows diagrams that need to be automatically generated for each resolved model. Therefore, the position of each graphical element needs to be calculated by custom layouts that automatically position each element in the correct place. Nevertheless, these limitations constitute our future work.

7. REFERENCES

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

[2] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Software & Systems Modeling*, pages 1–28, 2013.

[3] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. 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.

[4] D. Dhungana, T. Neumayer, P. Grunbacher, and R. Rabiser. Supporting evolution in model-based product line engineering. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 319–328, 2008.

[5] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang. A generic language and tool for variability modeling. *Technical Report SINTEF A13505*, 2009.

[6] C. Krueger. Easing the transition to software mass customization. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2002.

[7] J. Martinez, T. Ziadi, J. Klein, and Y. le Traon. Identifying and visualising commonality and variability in model variants. In J. Cabot and J. Rubin, editors, *Modelling Foundations and Applications*, volume 8569 of *Lecture Notes in Computer Science*, pages 117–131. Springer International Publishing, 2014.

[8] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen. Complete and accurate clone detection in graph-based models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 276–286, May 2009.

[9] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[10] J. Rubin and M. Chechik. Combining related products into product lines. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 285–300. Springer Berlin Heidelberg, 2012.

[11] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. 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.

[12] X. Zhang, O. Haugen, and B. Moller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference (SPLC), 2011 15th International*, pages 90–99, Aug 2011.

[13] X. Zhang, O. Haugen, and B. Moller-Pedersen. Augmenting product lines. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 766–771, Dec 2012.

[14] T. Ziadi, L. Frias, M. da Silva, and M. Ziane. Feature identification from the source code of product variants. In *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*, pages 417–422, March 2012.

11.2 SPLC'15 Paper

Title:	Building Software Product Lines from Conceptualized Model Patterns.
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Proceedings:	Proceedings of the 19th International Conference on Software Product Line (SPLC '15).
Location:	Nashville, Tennessee - July 20 - 24, 2015
Publisher:	ACM, New York, NY, USA
Pages:	46-55
DOI:	`http://doi.acm.org/10.1145/2791060.2791085`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work. He was also responsible for the oral presentation of the work which took place during the conference.

Building Software Product Lines from Conceptualized Model Patterns

Jaime Font[1,2] Lorena Arcega[1,2] Øystein Haugen[2,3] Carlos Cetina[1]

[1]San Jorge University
SVIT Research Group
Autovía A-23 Km. 299
50830 Zaragoza, Spain
{jfont,larcega,ccetina}@usj.es

[2]University of Oslo
Department of Informatics
Postboks 1080 Blindern
0316 Oslo, Norway
oysteinh@ifi.uio.no

[3]Østfold University College
Department of Information Technology
Postboks 700
1757 Halden, Norway
oystein.haugen@hiof.no

ABSTRACT

Software Product Lines (SPLs) can be established from a set of similar models. Establishing the Product Line by mechanically finding model differences may not be the best approach. The identified model fragments may not be seen as recognizable units by the application engineers. We propose to identify model patterns by human-in-the-loop and conceptualize them as reusable model fragments. The approach provides the means to identify and extract those model patterns and further apply them to existing product models. Model fragments obtained by applying our approach seem to perform better than mechanically found ones. It turns out that the repetition of a fragment does not guarantee its relevance as reusable asset for the SPL engineers and vice versa, a fragment that has not been repeated yet, may be relevant as a reusable asset. We have validated these ideas with our industrial partner BSH, an induction hobs manufacturer that generates the firmware of their products from a model-driven SPL.

CCS Concepts

•**Software and its engineering → Software product lines;**

Keywords

Reverse Engineering, Model-based Software Product Lines, Variability Identification, Human-in-the-loop

1. INTRODUCTION

Software Product Lines (SPLs) aim at reducing development cost and time to market while improving quality of software systems by exploiting commonalities and variabilities across a set of software applications [10]. The SPL engineering paradigm separates two processes; domain engineering (where the variability of the SPL is defined and realized) and application engineering (where specific software products are derived by reusing the variability of the SPL) [2].

The extractive approach to SPLs capitalizes on existing systems to initiate a product line [7], formalizing variability among a set of similar products into a variability model. However, manually spotting the commonalities and variability among the set of product models may become cumbersome and error prone [1], especially as the number of models and its complexity increases.

Some reverse engineering model differencing approaches [15, 11] aim to automatically extract and formalize the variability among a set of similar product models. By performing several comparisons among the product models, commonalities and variabilities are identified and formalized as variability models. As a result, a variability model is automatically built from a set of similar products.

We have applied a model differencing approach to build a SPL around a set of product models from our industrial partner. However, the results are not the reusable model assets expected by our industrial partner, due to the lack of domain engineering knowledge embedded in the process; model assets that are automatically extracted may not match the expectations of domain experts and application engineers.

We propose a human-in-the-loop differencing approach to identify and extract reusable model patterns from a set of similar product models. Domain experts and application engineers become part of the decision-making process, contributing their knowledge of the domain to tailor the approach. Then, the model patterns are extracted and formalized into variability models. Finally, model patterns can be applied to the set of models, resulting in a formalization of the variability in terms that are relevant for the engineers.

We have validated the presented ideas with our industrial partner (BSH group), the induction division has been producing induction hobs (under the brands Bosch and Siemens among others) over the last 15 years. The Induction Hobs domain is facing big changes, which has triggered the creation of a Software Product Line around the Induction Hob models that already exist. The first attempt follows a model differencing approach; then we apply our human-in-the-loop approach. As a result, we have identified the main situations where domain experts prefer a model asset that is different to the one proposed by automatic approaches.

The rest of the paper is structured as follows: next section introduces some background about the Common Variability Language and our industrial partner's domain. Section 3

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPLC 2015, July 20 - 24, 2015, Nashville, TN, USA
© 2015 ACM. ISBN 978-1-4503-3613-0/15/07...$15.00
DOI: http://dx.doi.org/10.1145/2791060.2791085

motivates the approach with an example extracted from our experience. Section 4 presents our approach to identify and extract resusable model patterns from a set of similar product models. In section 5 we present our experience applying the approach to our industrial partner's domain. Section 6 discusses related work. Finally we conclude the paper.

2. 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.

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 paper we use a simplified subset of the IHDSL (see the top of Figure 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 the 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).

2.1 The Common Variability Language applied to Induction Hobs

The Common Variability Language (CVL) [3, 13] 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.

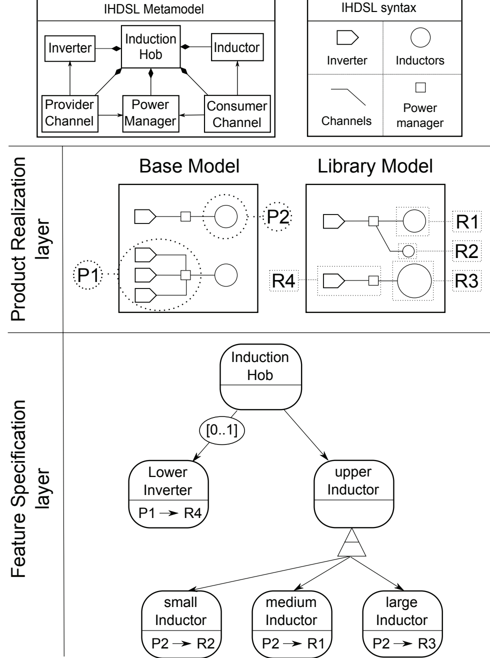


Figure 1: CVL applied to IHDSL

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.

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) that serves as the 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 that is described in the same DSL as the base model that will serve as a base 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 out-

side 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 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.

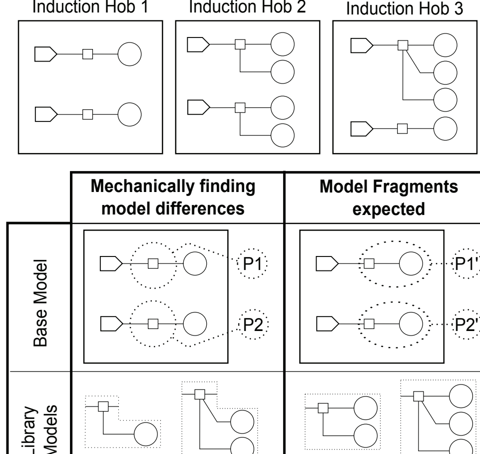


Figure 2: Motivation of the approach

3. MOTIVATION OF THE APPROACH

Reverse engineering approaches [11, 15] rely on mechanically finding model differences among the models. 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 reverse engineering approaches of this kind to extract and formalize the variability existing among the IH product models of our industrial partner. Specifically, fragments obtained by these approaches do not match the expectations of our industrial partner. Figure 2 illustrates the issue experienced.

The top part of Figure 2 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 bigger than the main inductor). It is important to note that the three IHs share a common point (i.e., 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 2 presents a representation of the results obtained after applying a reverse engineering model differencing approach. 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 → F1, P2 → F1). The IH3 can be obtained by substituting the placement by the second fragment (IH3 = P1 → 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 2 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' → F1', P2' → F1', and IH3 = P1' → 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 a running example, but the problem grows bigger when taking into account real models (for instance, elements in charge of generating power are mixed with inductors in the same fragment). This results in a lack of recognition of the model

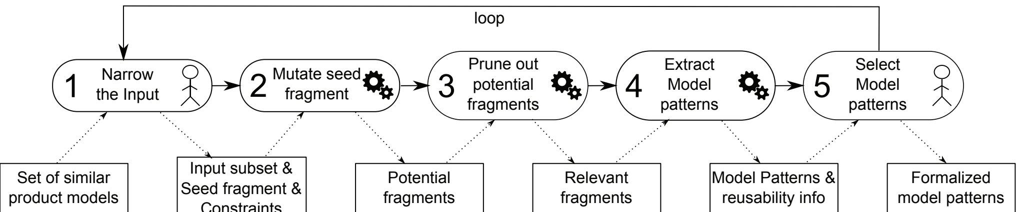


Figure 3: The model pattern identification and extraction process

fragments produced that do not match the reusable units handled by our industrial partner.

To address this issue we propose a human-in-the-loop approach where the SPL engineers can take part in the variability identification and extraction process, contributing their knowledge and tailoring the process. As a result, the approach produces model patterns that hold the variability among the set of product models in terms of CVL placements and replacements. Those model patterns can then be applied to the product models to formalize the variability.

4. THE MODEL PATTERN IDENTIFICATION AND EXTRACTION PROCESS

Figure 3 presents an overview of the human-in-the-loop process to identify and extract model patterns, which consists of 5 steps. The initial input of the process is the set of similar models around which the SPL will be built. Some steps of the process are automatic (represented by two gears) while others are performed by the humans-in-the-loop (represented by a stickman). The domain experts and application engineers will contribute their knowledge to the process. Since the human roles involved in the establishment and further operation of a Product Line may vary depending on the particular approach adopted, we will refer to the people contributing knowledge to the process as the "humans".

The approach itself can be seen as a web search engine. After providing a search query (and some advanced search options) the search engine returns a set of webs that match the query, which are automatically ordered from the most accurate match to the last. The web search engine may return several millions of results, but only the most relevant will be browsed by the user (usually the top of the list). Then, the user can select any of the results provided or perform a new search trying to refine the results.

The first step enables the humans-in-the-loop to narrow the scope of the comparisons that will take place in further steps. By doing so, the task of identifying the model patterns is modularized, resulting in a manageable task. The humans also select the initial fragment that will be used as seed to identify the model pattern (i.e., a model fragment that the humans believe conforms a recognizable unit).

The second step performs mutations of the fragment designated as seed, taking into account the actual product model where the seed comes from. In other words, this step performs mutations (following the scope parameters provided in the first step), resulting in a set of potential fragments that are variations of the seed fragment. The selected seed will be used as a starting point, but a set of fragments built around the seed will be produced. The provided seed is not always accurate and by providing alternatives we facilitate the selection of the proper model pattern.

In the third step, the set of potential fragments is pruned out to discard the model fragments that do not fulfill the constraints stated by the humans in the first step. The objective of this step is to discard non-relevant fragments (according to the humans' constraints) as early in the process as possible, decreasing the cost of further steps. As a result, we produce a set of relevant fragments that has been built taking into account the humans' knowledge.

In the fourth step, a set of model patterns involving the relevant fragments is calculated. For each potential fragment, the corresponding placement signature in the original product model is calculated. Then, the process matches each signature with the product models selected in the first step; if the match is positive, a model fragment is extracted. Each model pattern consists of a placement signature and the set of model fragments that can be used with that particular placement signature. In addition, information about the occurrences for each model pattern is calculated (for the placement and for each of the matching fragments).

In the fifth step, the set of model patterns and the reusability information gathered is presented to the humans. The model patterns can be ordered following different criteria, such as the size of the placement or the number of alternatives for that placement. In addition, one of the model patterns is selected as the default choice, taking into account the number of times it has been reused in the models (following criteria similar to the fully automatic approaches [11, 15]). Finally, the humans select the model pattern that is relevant for them and that better formalizes their understanding of the domain, guided by the reusability information provided.

As a result of these five steps, a single model pattern is extracted and formalized. The process can be iterated to identify and extract new model patterns or to extend already extracted model patterns with the information present in new product models. The following subsections present a running example of the application of the approach to our industrial partner's domain.

4.1 Step 1 - Narrow the input

The presented process includes several comparisons among the product models which can be costly in terms of time and memory. The number of resulting model patterns presented to the humans in the fifth step depends on the products used as input and can be too high. Therefore, the approach provides the means to narrow the input, so that the number of resulting model patterns can be limited.

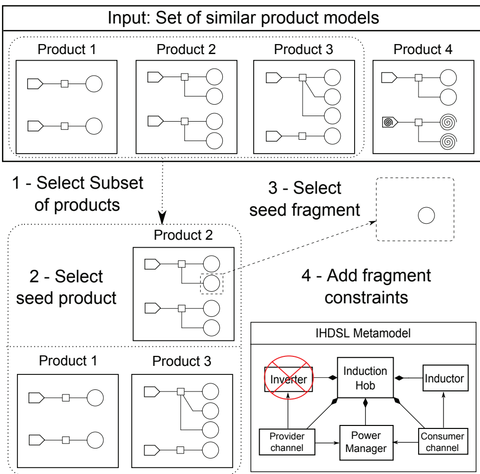


Figure 4: Step 1 - Narrow the input

As part of the first step, domain experts are in charge of four tasks: (1) selecting a subset of products from the input, (2) selecting one product as the seed product, (3) selecting one fragment from that product as seed fragment, and (4) narrowing the scope of the input to reduce the costs in further steps of the process.

The selection of the seed model and fragment is the most basic way of narrowing the input and the one that has the biggest impact on the results of the process. The resulting model patterns will include the selected fragment or slight mutations of it (i.e., a set of model patterns related to the selected fragment). Taking into account the knowledge of the domain (and the results of previous iterations) the humans select a fragment model from one of the product models used as input. The selection of the model fragment is done based on the intuition of what parts of the product model could be reused across several product models.

In addition, the knowledge of the humans about the domain and the product models enables them to have an accurate idea of the model fragments that they expect to obtain from this process. Taking into account this knowledge, the scope can be further refined in two different ways:

Input models: When a model is created following clone-and-own approaches [9], 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 iteration to just a subset of the input models.

Metamodel level: In order to narrow the set of potential fragments, the humans can indicate which meta-elements must be included in or excluded of from the resulting fragments. The process can be tailored so that each functional unit of the product is formalized as part of a different model pattern by defining constraints at the metamodel level. For instance, when applying the approach with large teams it is common to have experts of different parts of the domain, enabling each expert to work only with the subset of the model that the expert knows best, limiting the model patterns to just that subset.

Figure 4 shows a running example of the first step of the process, the scope of the input. A set of 4 products is provided as input (Product 1 - Product 4). First, the humans select a subset of the input (Product 1, Product 2 and Product 3); Product 4 is discarded since it is an odd product (it mixes induction with glass-ceramic radiant heaters). Then, the product model seed and the model fragment seed is selected from the subset of the input models. In this case, an inductor from Product 2 is selected; the humans are aware that inductors are present in all IHs and can be turned into a model pattern. Finally, a constraint is defined at the metamodel level and the resulting model patterns must comply with this constraint. In this case, the model patterns will not contain inverters to avoid the mix between power control and inductors in the same model pattern.

4.2 Step 2 - Mutate seed fragment

This step is performed automatically and takes as input the product model seed and the fragment seed selected in the previous step and the configuration of the mutations (if any). In this step, some potential fragments (which are closely related to the seed) are obtained. By doing this, the humans can evaluate whether the selected seed is a relevant fragment or there is another mutation that is more suitable.

The mutations are performed taking into account the fragment seed and the product model seed. Taking the seed fragment as starting point, some model elements are added to or removed from the seed fragment. However, the elements added during mutations are obtained from the seed product model, guaranteeing that the generated fragment is part of the seed model. In other words, apart from the selected seed fragment, the process proposes other variations of that fragment that are also part of the product model.

In order to obtain the mutations, the process performs additions, substractions, and combinations of both. In addition, the number and type of mutation operations can be configured or restricted to tailor the process towards the desired potential fragments. The number of chained mutations performed can also be adjusted to restrict the number of results.

Figure 5 shows the application of the step. A directed graph is built with all the potential fragments obtained by the mutations. Each node (circles) represents a potential fragment while each arc (arrows) represents one mutation (both directions, additive or substractive mutations). The nodes are classified by levels so that the process can be restricted to calculating fragments up to a fixed depth. The top part of Figure 5 shows the model fragment selected as seed (labeled as F1). It corresponds to Level 0 since no mutations are needed. Level 1 shows the fragments that can be obtained by one mutation; which in this case is F2, the result of adding the power manager to the original fragment. In this example, we are only considering the addition of elements that are connected in the seed model. We have restricted the generation of potential fragments up to Level 4, (i.e., fragments that can be obtained chaining up to four mutations). Note that some potential fragments (such as F5 or F8) are marked with a red crossed circle (this is part of

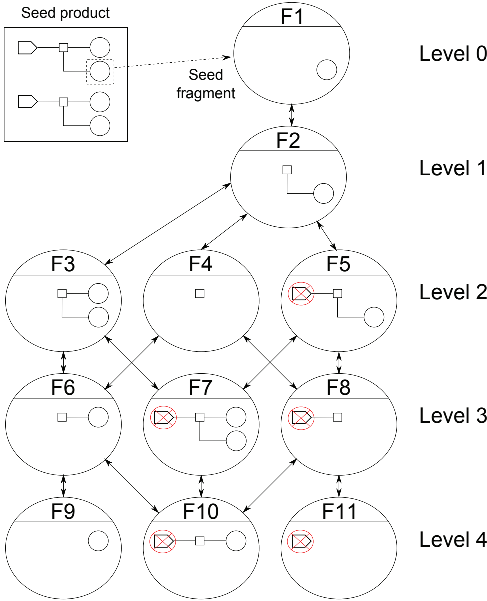


Figure 5: Step 2 - Mutate seed fragment

the prune out explained in the next step).

This step only produces fragments that are part of the original model. It does not create new elements; it just proposes variations of the seed fragment that are actual fragments of the original model. Note that all potential fragments produced (F1 to F11) are part of the seed product model. The elements added during mutations are obtained from the seed product model, so the resulting fragment is a subset of the seed product model.

As a result of the application of several chained mutations (addition or substractions), duplicated fragments may be produced. However, the process keeps track of the fragments that have already been produced to avoid duplication of work and results. Therefore, this step produces a set of potential fragments, including the seed fragment that was selected in the first step.

4.3 Step 3 - Prune out potential fragments

This step is performed automatically and prunes out the potential fragments calculated in the previous step, discarding the fragments that do not fulfill the constraints provided in the first step. In other words, for each of the potential fragments calculated, the process checks wether the constraints are satisfied or not; if not, the potential fragment is removed from the set. As a result, the set of potential fragments that will be used in further steps is reduced, avoiding the inclusion of undesired fragments in the result of the process.

In Figure 5, some of the potential fragments calculated contain inverters, but the humans decided to discard potential fragments containing inverter elements. Consequently, the potential fragments containing model instances of that metamodel element must be pruned out. In this example, since F5, F7, F8, F10, and F11 contain inverters they are discarded (see the bottom right corner of Figure 5).

By applying constraints to the resulting fragments, the humans can contribute their knowledge to the process. For instance, in this example, the humans are aware that there should be a reusable model fragment containing the inductor, but they are not totally sure about the exact elements that are part of the fragment. However, they know that the inverter belongs to another part of the system and want to prevent the inverter from being included in the model patterns that are calculated during this iteration. By means of this step, the potential fragments have been reduced from 11 to 6, simplifying further steps of the process. As a result, a single set containing all the potential fragments that meet the scope requirements provided by the humans is produced.

4.4 Step 4 - Extract model patterns

This step is performed automatically, and the input is the set of potential fragments that are relevant for the humans (from the previous step) and the subset of products provided by the humans as part of step 1. As output, this step produces a set of model patterns that is annotated with information about its reusability among the subset of products. A model pattern can be seen as a variation point (and the alternatives for that variation point) that can be reused several times.

First, the placement signature is calculated for each of the potential fragments; that is, we calculate the boundary information between the potential fragment and the rest of the seed model. Particularly, we calculate the set of incoming and outgoing relationships regarding the potential fragment and the seed model. This placement signature can be used to identify spots in the models where that potential fragment can be used.

Then, we match each of the placement signatures against each product model from the input subset. By doing so, we determine wether a particular placement can be found in each product model. If the match is positive, the fragment of that particular product that matches with the placement signature is extracted. Thus, each of the placements' signatures can be seen as a variation point (that is not bounded to a particular product) and each of the fragments extracted can be seen as an alternative for that variation point.

Finally, for each model pattern (placement signature and alternative fragments), the process computes the number of times it can be applied in the subset of products (i.e., the number of times that the match between the placement signature and the product is positive and the number of times that each of the fragments is used in the products).

Figure 6 shows the application of this step to the running example. Each row shows information for one of the relevant potential fragments obtained. The first column shows the potential fragment, and the second column shows the placement signature calculated for each potential fragment. The rest of the columns show the matching against each of the products from the subset provided in step 1.

For instance, the first row of Figure 6 shows the seed fragment selected by the humans (F1, the first column), the placement signature corresponding to that fragment (the second column) and the matching of the placement signature with the product models. In this case, another potential

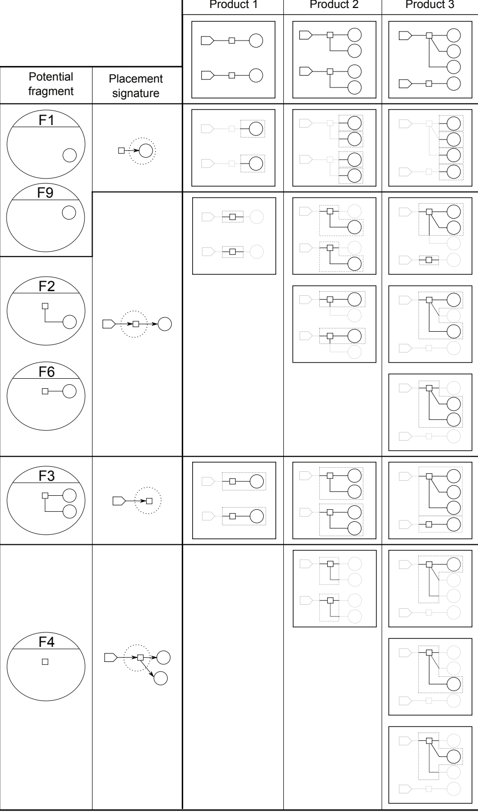


Figure 6: Step 4 - Extract model patterns and occurrences

placement (F9, the first column) produces the same placement signature as F1. Not all placement signatures match all products; for instance, the placement signature corresponding to F4 (the last row), does not match Product 1.

As a result of this step, a model pattern is generated for each of the potential fragments used as input (i.e., the calculated placement signature, the alternative fragments matching that placement extracted from the product models, and the information about occurrences for each of the fragments and the placement itself).

4.5 Step 5 - Select model patterns

In this step, the model patterns obtained are presented to the humans so they can choose the ones that are most relevant for them based on their knowledge and the information provided by the approach. Each model pattern is presented with the information about the occurrences in the product models gathered in the previous step (i.e., the number of positive matches of the placement signature among the products and the number of occurrences for each of the alternative fragments extracted).

	Placement signature	Alternative Fragments extracted		
Model Pattern 1				
occurrences	10/3	3/10	4/10	3/10
Model Pattern 2				
occurrences	6/3	3/6	2/6	1/6
Model Pattern 3				
occurrences	10/3	10/10		
Model Pattern 4				
occurrences	5/3	2/5	3/5	

Figure 7: Step 5 - Select Model Patterns

Figure 7 shows the application of this step to the running example. Each row shows information about a model pattern. The first column presents the placement signature of the model pattern and the rest of the columns show each of the extracted alternative fragments that match that particular placement signature. The number below each placement signature represents the number of possitive matches out of the total number of products used as input. The number below each alternative fragment represents the number of occurrences of that particular fragment out of the total number of matches of the placement signature.

For instance, the first row shows the model pattern selected by default. The pattern has been identified ten times in the three products analyzed and is the one that has the most fragment alternatives with three possibilities. Therefore, it has been selected as the default pattern by the process. However, even though this model pattern is the one that is most frequently repeated, the humans do not recognize all the alternative fragments presented because they are incomplete (they do not contain all the inductors), so it is discarded. The second row shows a model fragment that holds inductors; it has been identified six times and also has three different alternatives. In fact, each alternative contains the whole group of inductors connected to a power manager, as the humans were expecting. The third row shows the model pattern containing the selected seed fragment; However, even though it is matched ten times in the models provided, it does not present alternatives and would be considered as a common part of the model by traditional reverse engineering model differencing approaches.

Taking into account the reusability information, the humans select the model pattern that best fits their understanding of the domain. For instance, they can stick to the initial seed selection (model pattern 3), but some of the model patterns provided may be more accurate and relevant

for them (like model pattern 2). The model pattern presentation (along with the reusability information) enables the humans to reason about the fragments, thus facilitating the task of determining which one should be selected.

4.6 Loop

The five-step process enables the extraction of one model pattern and can be repeated until all the recognizable units that conform to the models have been extracted. In addition, already extracted model patterns may be extended to include new replacements that were not present in the original iteration.

For instance, a new iteration could be run (taking into account the model patterns already selected) to extract another model pattern holding the inverters. Similarly, an iteration to refine an existing model pattern could be run, including new models that were not part of the original iteration (discarded during the step 1) and resulting in a refinement of the former pattern.

The number of iterations needed depends on the domain where it is being applied and the amount of variability that must be formalized. In addition, some of the iterations will not produce a relevant model pattern and will need to be refined until the desired model pattern is obtained.

5. CASE STUDY: INDUCTION HOBS DOMAIN

This section presents our experience building a Product Line from an existing set of products from our industrial partner (BSH group). This company is the largest manufacturer of home appliances in Europe and one of the leading companies in the sector worldwide. Their induction division has been producing induction hobs (the brand portfolio is composed by Bosch and Siemens among others) over the last 15 years.

In order to implement the approach, several technologies are involved. Specifically, CVL can be applied to MOF based models, so the approach is developed within the Eclipse environment using the Ecore implementation and the Eclipse Modeling Framework (EMF) [1]. The mutations of the seed fragment (Step 2, Section 4.2) are implemented based on ecore-mutator [2], which is an EMF-based framework to mutate EMF models that conform to an Ecore metamodel. The comparisons among models (Step 4, Section 4.4) are implemented based on EMF-Compare [3],which is an Eclipse framework to compare instances of EMF models.

As a first attempt, we applied a reverse engineering model differencing approach [11, 15] and followed up with an evaluation of the obtained fragments that revealed the problem presented in Section 3. Then, we applied the approach presented in this paper, a human-in-the-loop approach to identify and extract conceptualized model patterns as reusable variation points. Finally, the patterns were used to formalize the variability among the set of existing products and incorporated to the Product Line supporting tool to enable the reuse of the patterns when creating new products.

The model patterns extracted following the approach include the model information necessary to create CVL variation points. Specifically, a pattern contains the placement signature of a fragment and a set of matching fragment alternatives to replace it. Therefore, the patterns can be used to formalize the variability among a set of products.

We have used the model patterns identified and extracted to improve our industrial's partner tool, enabling the formalization of variability based on those patterns. For each model pattern, we have created (1) a custom editor that enables the creation of new replacements for a particular model pattern, (2) a fragment library that holds all the replacements identified throughout the process for that particular model pattern. The resulting tool can be seen here [4].

5.1 Application of our approach

The initial input of the approach is a set of 46 induction hob models, corresponding to products that are currently being sold or that will be launched to the market in the immediate future. The set of models were developed following a clone and own [9] approach, where each IH has been modeled modifying a copy of the most similar IH present in the collection. Therefore, the variability present among the models has not been explicitly defined, resulting in a set of models with implicit variability among its members.

However, there is implicit knowledge among our industrial partner's engineers of the traceability of the clones performed. In other words, the engineers are aware of the existence of groups of models that may have more similarity among them since they share a common ancestor product used as base. Therefore, we used this information to perform the division of the input set into smaller subsets.

With regard to the products complexity, each of the IH models is composed of more than 500 elements, including around 100 class elements on average. For each IH, there are around 10^{29} different potential fragments composed of more than one class element. The magnitude gives the idea that manually processing this amount of data may be cumbersome, error prone, and could not be done in a reasonable time.

For the application of the approach, we had the collaboration of the engineers from the firmware department; they are experts in developing firmware for induction hobs and are in charge of maintaining and evolving the firmware used in the IH. Those engineers are the owners of the induction hob models and provided their knowledge to tailor the approach.

We performed several iterations looking for model patterns, but not all of them ended up in the extraction of a model pattern. Some of them were used as the base for further refinement. For instance, we performed several iterations looking for a model pattern that held groups of inductors (presented as running example in Section 4). We started with a subset containing the most basic IHs and with a very conservative set of constraints. With the information of that iteration, we refined the constraints until we obtained a pattern that satisfied our industrial partner's engineers.

Then, we performed new iterations, adding more induction hobs to the subset, in order to find new alternatives for the model pattern previously extracted. We added induction hobs that held each of the different group of inductors present in the products. Finally, we performed several extra iterations in order to confirm that the extracted model pattern was correct and contained all the alternatives desired.

[1] http://www.eclipse.org/modeling/emf/
[2] https://code.google.com/a/eclipselabs.org/p/ecore-mutator/
[3] https://www.eclipse.org/emf/compare/index.html
[4] www.carloscetina.com/variabilitytool.htm

When looking for the inverter groups, we followed a similar strategy, performing several iterations and refining the search. However, the subsets employed to search for this model pattern were totally different (as suggested by our industrial partner engineers). Even more, the domain experts that took part in the process were also different (since they had more knowledge of this specific part of the system).

As part of the application of the approach, we conducted a usability evaluation, including satisfaction tests, interviews, and focus groups, where the engineers could talk freely about the process and the fragments that were being identified. Although the usability evaluation is out of the scope of this work, we outline some of the findings below.

Specifically, we wanted to know more about the rationale behind the selection of one model pattern over the others. The approach proposes a model pattern based on the number of times it is reused across the products (as other automatic approaches do), but the proposed pattern was not always the chosen one. We identified three main reasons for not choosing the proposed model pattern:

Odd elements in the fragments: It is not clear beforehand how resulting fragments are going to be, but there are some elements that are not expected to be part of the model pattern. However, odd elements can be incorporated by automatic approaches due to the number of occurrences (as in the motivation of the approach example). In the presented approach, the domain experts can state that odd elements pertain to different parts of the system and choose to have them in separate fragments. When browsing the results of the approach, the domain experts can compare among patterns with different levels of granularity, easing the task of determining which elements should be part of the fragment and which ones should not.

Deprecated elements: Another reason for selecting a model pattern that is different from the one proposed is the inclusion of deprecated elements as part of the model pattern. Thus, the default pattern proposed has a high number of alternatives for the placement, but the humans know that those elements are going to disappear in further versions. Therefore, they prefer to not give them too much relevance as they will not be part of future products anymore.

Future developments: Sometimes, there is a model pattern that is treated as irrelevant by the approach due to its low number of occurrences or alternatives. However, this pattern was relevant for the humans because they knew that in future developments that element would be split into several elements. Therefore, they wanted to include it among the model patterns selected. For instance, in Figure 7, the Model Pattern 3 presents no alternatives, but the engineers knew that different kind of inductors would be developed in the future, so they wanted to include that model pattern.

6. RELATED WORK

Some works focus on transforming legacy products into Product Line assets. For instance, in [5], the authors present their experience in the Digital Audio & Video Domain. In [6], the authors explain their experience reengineering the Image Memory Handler from Ricoh's products into a SPL. In [8], 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.

Other works focus on the automation of the extraction process, obtaining the variability from legacy products by comparing the products with each other. In [14] 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 [4] to reduce the number of comparisons needed to mine the family model.In [11], the authors propose a generic framework for mining legacy product lines and automating their refactoring to contemporary feature-oriented SPLE approaches. They compare the elements of the input with each other, matching those whose similarity is above a certain threshold and merging them together. In [15], the authors propose a generic approach to automatically compare products and extract the variability among them in terms of a CVL variability model. These two approaches automatically turn 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 approaches. In contrast, our work enables the domain experts to decide which elements should be formalized as part of the SPL based on the results of the comparisons, instead of performing them as an automatic output of the comparisons.

Finally, some research efforts focus on the source code of the products in order to extract the variability model. For instance, the authors in [12] 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 [16] 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 [17] to introduce ExtractorPL, an automated technique that infers a full implementation of a SPL from the given code. However, these approaches focus on the source code level of the products, while our approach is applied at the model level which enables domain experts to contribute their knowledge to the identification and extraction of reusable model fragments.

7. CONCLUSION AND FUTURE WORK

As part of this work, we have identified the need for an approach to identify and extract conceptualized model patterns from a set of similar product models that match the expectations of the domain experts. We have proposed a human-in-the-loop approach that enables domain experts to contribute their knowledge to the identification and extraction process. In addition, we have implemented the presented approach within the Eclipse environment. We have validated the approach with our industrial partner, extracting the model patterns that are present in a set of real induction hob models. We have outlined the rationale followed

by our industrial partner to prefer one model pattern over the rest (even though it is reused fewer times among the models). Finally, we have applied the resulting model patterns to a modeling tool, enabling the evolution (creating new fragment alternatives) and reutilization of the model pattern.

The identification of model patterns through a human-in-the-loop approach has provided model fragments that are recognizable by the SPL engineers. The humans involved in the extraction process could contribute their knowledge to the process, tailoring it to produce the desired model fragments and thereby formalizing the variability among a set of products.

There are some open questions remaining about the presented approach, such as how many model patterns should be extracted, what model pattern granularity works best, or if nested model patterns could benefit the process. To address those open questions, we are currently applying our approach in CAF [5], an international company that builds and deploys railway solutions around the world. Similarly to our industrial partner, they need a solution to formalize the variability that exists among their products.

The use of a human-in-the-loop approach seems to be successful in identifying and extracting model patterns, as it enables the domain experts to immerse themselves in the process and contribute their knowledge. As a result, the fragments represent recognizable units that are extracted and fulfill the expectations of the engineers involved. As one of our industrial partner engineers reported: "The first fragment library (reverse engineering model differencing) was generic, but the new fragment library (our model patterns approach) was made just for us".

[5] http://www.caf.es/en

8. REFERENCES

[1] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Software & Systems Modeling*, pages 1–28, 2013.

[2] P. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. Addison-Wesley Professional, 3rd edition, Aug. 2001.

[3] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding standardized variability to domain specific languages. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 139–148, Sept 2008.

[4] S. Holthusen, D. Wille, C. Legat, S. Beddig, I. Schaefer, and B. 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.

[5] K. Kim, H. Kim, and W. Kim. Building software product line from the legacy systems "experience in the digital audio and video domain". In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 171–180, Sept 2007.

[6] R. Kolb, D. Muthig, T. Patzke, and K. 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, Mar. 2006.

[7] C. Krueger. Easing the transition to software mass customization. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 282–293. Springer Berlin Heidelberg, 2002.

[8] H. Lee, H. Choi, K. Kang, D. Kim, and Z. Lee. Experience report on using a domain model-based extractive approach to software product line asset development. In *Formal Foundations of Reuse and Domain Engineering*, volume 5791 of *Lecture Notes in Computer Science*, pages 137–149. Springer Berlin Heidelberg, 2009.

[9] N. Pham, H. Nguyen, T. Nguyen, J. Al-Kofahi, and T. Nguyen. Complete and accurate clone detection in graph-based models. In *Software Engineering, 2009. ICSE 2009. IEEE 31st International Conference on*, pages 276–286, May 2009.

[10] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[11] J. Rubin and M. Chechik. Combining related products into product lines. In J. de Lara and A. Zisman, editors, *Fundamental Approaches to Software Engineering*, volume 7212 of *Lecture Notes in Computer Science*, pages 285–300. Springer Berlin Heidelberg, 2012.

[12] S. She, R. Lotufo, T. Berger, A. Wąsowski, and K. 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.

[13] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: a case study of cvl. In *14th international conference on Software product lines*, SPLC'10, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] D. Wille, S. Holthusen, S. Schulze, and I. 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.

[15] X. Zhang, Ø. Haugen, and B. Møller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Software Product Line Conference, 2011 15th International*, pages 90–99, Aug 2011.

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

[17] T. Ziadi, C. Henard, M. Papadakis, M. Ziane, and Y. Le Traon. Towards a language-independent approach for reverse-engineering of software product lines. In *Symposium on Applied Computing*, SAC '14, 2014.

11.3 ICSR'16 Paper

Title:	Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm.
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Proceedings:	Software Reuse: Bridging with Social-Awareness: 15th International Conference (ICSR '16).
Location:	Limassol, Cyprus, June 5-7, 2016
Publisher:	Springer International Publishing
Pages:	39-54
DOI:	`https://doi.org/10.1007/978-3-319-35122-3_3`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work. He was also responsible for the oral presentation of the work which took place during the conference.

Feature Location in Model-Based Software Product Lines Through a Genetic Algorithm

Jaime Font[1,2(✉)], Lorena Arcega[1,2], Øystein Haugen[3], and Carlos Cetina[1]

[1] SVIT Research Group, San Jorge University, Zaragoza, Spain
{jfont,larcega,ccetina}@usj.es
[2] Department of Informatics, University of Oslo, Oslo, Norway
[3] Department of Information Technology, Østfold University College, Halden, Norway
oystein.haugen@hiof.no

Abstract. When following an extractive approach to build a model-based Software Product Line (SPL) from a set of existing products, features have to be located across the product models. The approaches that produce best results combine model comparisons with the knowledge from the domain experts to locate the features. However, when the domain expert fails to provide accurate information, the semi-automated approach faces challenges. To cope with this issue we propose a genetic algorithm to feature location in model-based SPLs. We have an oracle from an industrial environment that makes it possible to evaluate the results of the approaches. As a result, the proposed approach is able to provide solutions upon inaccurate information on part of the domain expert while the compared approach fails to provide a solution when the information provided by the domain expert is not accurate enough.

1 Introduction

A recent survey [2] reveals that most of the Software Product Lines (SPLs) are built following an extractive approach, where a set of existing products is reengineered into a SPL [12]. 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.

Several reverse engineering approaches can be used to identify and locate the features [4–6,14,16,18] from the existing product models and formalize them in the form of a model-based SPL (where the features are realized in the form of model fragments). In our previous work [5] we show that Conceptualized Model Patterns to Feature Location (CMP-FL) provides features more recognizable by the engineers that must use them thanks to the inclusion of information from the domain experts into the feature location process.

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO), through the Spanish National R+D+i Plan and ERDF funds under The project Model-Driven Variability Extraction for Software Product Lines Adoption (TIN2015-64397-R).

© Springer International Publishing Switzerland 2016
G.M. Kapitsaki and E. Santana de Almeida (Eds.): ICSR 2016, LNCS 9679, pp. 39–54, 2016.
DOI: 10.1007/978-3-319-35122-3_3

However, in CMP-FL the set of possible solutions is too big to be evaluated exhaustively, resulting in the need of very precise information from the domain engineers to accelerate the process. If the information provided is not accurate enough the feature location will fail, not being able to provide the expected solution. When the family of product models is built following clone-and-own techniques, the variability among the products is not always properly documented, resulting in a lack of precise information.

To cope with the above, we propose an approach based on a Genetic Algorithm to Feature Location (GA-FL) among a set of product models. Specifically, we propose new model-based genetic operations capable of working with model fragments: (1) the crossover operation, that combines information from two possible solutions into a single offspring; (2) the mutation operation, that randomly mutates one model fragment (while keeping the consistency with the product model where the fragment was extracted from); (3) a fitness function that evaluates the population of possible solutions and ranks them depending on how they solve the problem and (4) a parent selection operation to find candidates that feed the rest of genetic operations.

We have compared the CMP-FL with GA-FL through the use of an oracle extracted from our industrial partner (BSH), whose induction department produces the firmware for their induction hobs (sold under the brands of Bosch and Siemens) based on a model-based SPL. It turns out that our GA-FL is able to provide the solution expected in scenarios where the CMP-FL fails. When the information provided is accurate, the GA-FL algorithm is able to enrich the set of best solutions produced given that it explores a broader search space.

The rest of the paper is organized as follows: next section presents some background about the domain of our industrial partner and its SPL. In Sect. 3 we present our approach, the GA-FL. Section 4 compares the presented approach with the best alternative from literature. In Sect. 5 we discuss some related work. Finally, we conclude the paper.

2 Formalizing the Variability

This section presents the Domain Specific Language (DSL) used by our industrial partner to formalize their products, the IHDSL. It will be used through the rest of the paper to present a running example. Then, the Common Variability Language (CVL) is presented, CVL is the language used by our approach (GA-FL) to formalize the location of the features as reusable model fragments.

2.1 The Induction Hobs Domain Specific Language (IHDSL)

The newest Induction Hobs (IHs) feature full cooking surfaces, where dynamic heating areas are automatically generated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. In addition, 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 made possible at the cost of increasing the software complexity.

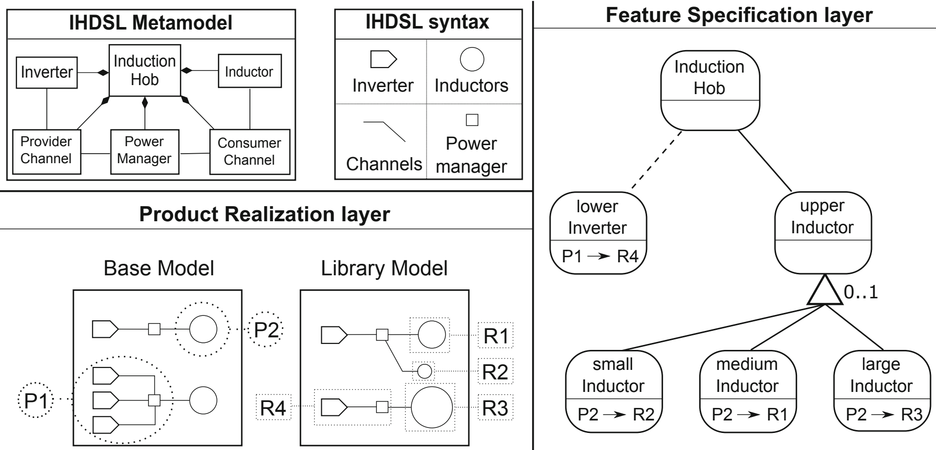


Fig. 1. CVL applied to IHDSL

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 paper we use a simplified subset of the IHDSL (see Fig. 1).

Inverters are in charge of transforming the input electric supply to match the specific requirements of the IH. Then, the energy is transferred to the inductors 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 the energy is transformed into an electromagnetic field. Inductors can be organized into groups to heat larger cookware while sharing the user interface controllers.

2.2 The Common Variability Language Applied to IHs

To formalize the variability among the products of the SPL, we need a variability model that captures which model fragments are used by each of the products that can be built from the SPL. To build it, the presented approach uses the Common Variability Language (CVL) [8], given its expressiveness to properly formalize the feature realizations in terms of model fragments. CVL defines variants of a base model conforming to MOF (Meta-Object Facility, the Object Management Group metalanguage for defining modeling languages) by replacing variable parts of the base model by alternative model replacements found in a library.

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

Figure 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. 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 (boundaries) 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.

Through the rest of the paper, we will use the term feature location in models formalized through CVL as "the process of obtaining the particular model fragments (or alternatives e.g. R1, R2 and R3) that are used in a particular placement (or variation point e.g. P1) among a set of products". Therefore, we will refer to the variation point as the feature being located and each of the alternative model fragments will be referred as different realizations for that feature (in fact, they are realizations of the alternatives of the feature).

3 Genetic Algorithm for Feature Location

This section present our approach, a Genetic Algorithm to Feature Location (GA-FL). Figure 2 shows an overview of the GA-FL process. The input of the process is a set of interrelated product models with implicit variability among them.

In the Genetic Algorithm process, the set of solutions that will be iterated need to be properly encoded (see **A - Encoding of the Population**), enabling the GA to work with them. The DE (domain expert or domain engineer) provides information about the set of product models to initialize the population of model fragments (see **B - Initialize Population**), the DE will select some product models to locate a particular feature and an initial model fragment for each of the selected product models. Next, each possible individual from the

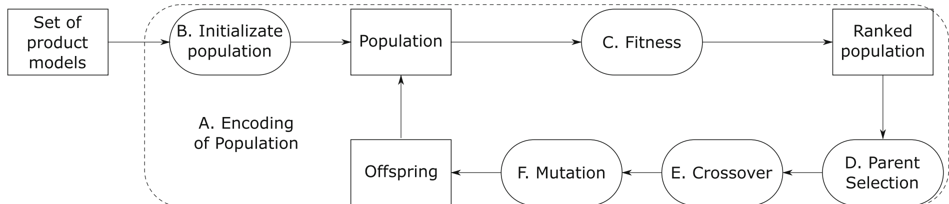


Fig. 2. Overview of the genetic algorithm to feature location

population is evaluated to determine how good is as a solution to the problem (see **C - Fitness**), as a result the population of solutions is ranked depending on their fitness value. Based on the ranked population, the parents for the new element are randomly selected (see **D - Parent Selection**), giving a higher probability to the solutions with higher fitness values. The first operation applied to the parents is the crossover, that joins two parents into a new solution (see **E - Crossover**). The resulting model fragment will be bound by a different product model and thus will evolve differently than the original one. The second operation applied to the solution resulting from the crossover is the mutation (see **F - Mutation**), the model fragment will evolve, growing or shrinking, resulting in a different model fragment that will be evaluated as possible solution in further generations. Finally, the set of solutions obtained will be presented to the DE, to select the solution that best represent their understanding of the feature being located.

3.1 Encoding of the Population

Traditionally, genetic algorithms encoded each possible solution of the problem (or chromosome) as a fixed-size string of binary values. Each position of the chromosome string (called locus) has two possible values (called alleles): 0 or 1.

However, to encode each model fragment as a string of binary values is not straightforward. As suggested by Davis [3], we decided to use an encoding natural for our problem and then devise a GA for that specific encoding. Therefore, we will encode our individuals as model fragments. To do so, we rely on MOF as the standard to define the models and CVL to specify fragments over those models and manipulate them.

Each individual of our Genetic Algorithm will be a model fragment defined over one of the product models. That is, each individual is a set of model elements and relationships among them that is present in one of the product models (see right part of Fig. 3 to see the representation of the individual). Therefore, to work with these individuals (model fragment defined over a product model), we will present genetic operations that can be applied directly to those model fragments. Through the rest of the paper we will refer to each individual as a model fragment that is always part of a product model.

3.2 Initialize Population

The first step of the process is to initialize the population of the GA. This is done by the DEs, preferably the same DE that created the products or work directly with them. The initialization is done based on DE's knowledge of the domain and the products themselves. This step is performed only one time for each feature that wants to be located.

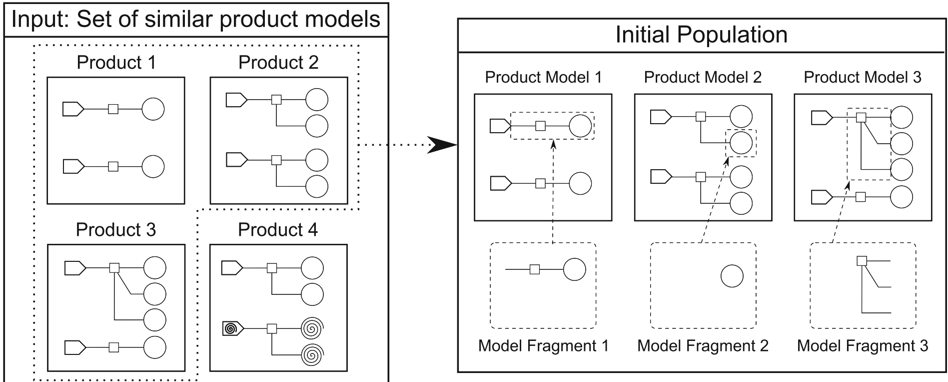


Fig. 3. Initialize population

Figure 3 shows an overview of this step. Top part shows the set of similar product models that where the feature will be located (Product Model 1 to 4). First, (1) the DE selects a subset of product models representative of the feature that will be used as input (in this example Product Model 1, 2 and 3), then, (2) for each product model from the subset the DE selects a model fragment that he believes will be part of the realization of that particular feature (Model Fragment 1, 2 and 3). As a result we get an initial population composed of pairs of model fragments and the product models where they were extracted from.

It is important to remark that we focus in the location of the features, leaving out of the scope of this work the features constraints discovery. That is, there could exists a cross-tree constraint among the feature 'upper heating spot' and the feature 'lower heating spot' (e.g. power consumption of combination of several inductors is higher than power consumption of single small inductors), but feature constraint discovery is not covered by this work.

3.3 Fitness Function

The fitness function is used as an heuristic to find the best solutions for the given problem. It is applied to each individual in the population and the function assigns a value that assesses how good is the solution. 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 as parents for the next generation.

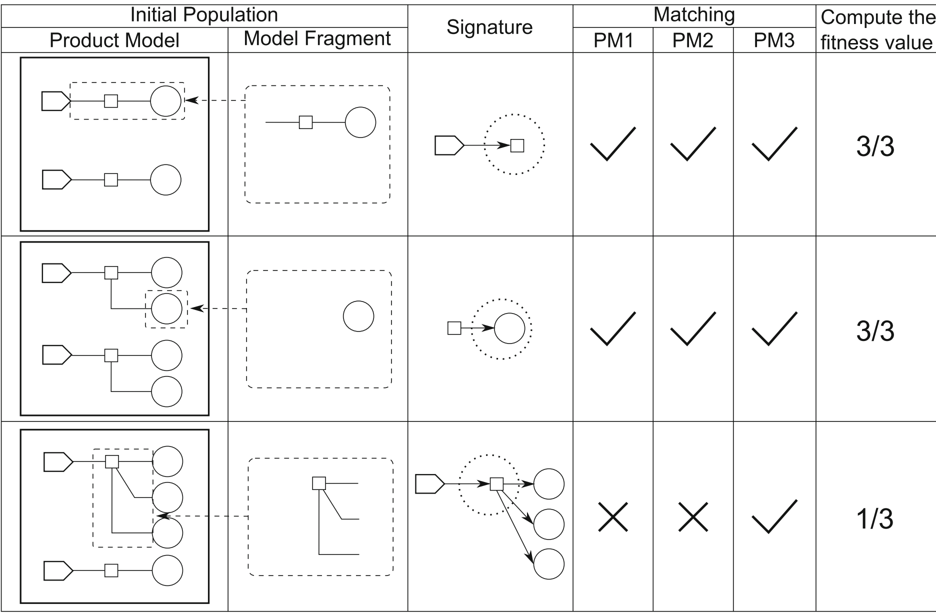


Fig. 4. Fitness function application

Our fitness function proceeds as follows: (1) the process abstracts from each model fragment to a placement signature in their referenced model fragment; (2) placement signatures are compared and grouped together if they are equal; (3) each placement signature is matched against all the product models from the initial subset of product models; (4) the fitness is computed for each placement signature and the fitness values are spread to the elements of the population.

Figure 4 depicts an overview of the model pattern extraction process [5] adapted to be used as a fitness function. The input of the process is the present population (the set of model fragments and their reference to a product model), see first and second column.

Step 1: The first step (see third column of Fig. 4) is to obtain a placement signature for each of the individuals (model fragment and the product model). The placement signature formalizes the set of elements that must be present in a model in order to connect the given model fragment. This is done comparing the model fragment with the product model from which it was originally extracted (when the initial population was created). The model fragment is present in the product model and connected to other model elements of the product model. 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. That is, the set of elements needed to connect the given model fragment. Therefore, the model fragment used as input match this placement signature. As a result, step 1 produces a placement signature for each model fragment used as input.

Step 2: The second step (not shown in Fig. 4, there are no duplicates) is to compare the placement signatures and group the ones that are equal. 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 model fragment is associated to a single placement signature.

Step 3: The third step (see fourth, fifth and sixth columns of Fig. 4) is to match each placement signature with all the product models present in the initial subset of product models. That is, the process looks for spots where a given placement signature matches in each of the given product models. When a placement signature matches a particular spot of a product model, means that the model fragments associated to that placement signature could be inserted in the given spot. As a result, step 3 provides a set of spots (across all the product models) where the given placement signature matches.

Step 4: The fourth step (see seventh column of Fig. 4) is to compute the fitness value for each of the placement fragments and spread it to the associated model fragments. The process computes the number of product models where the placement matches (no matter how many times). This value indicates the number of product models where the resulting placement could be used. As the purpose of the genetic algorithm is to locate variation points and alternative realizations across the product models, the higher the number of product models that match the better. Finally, the value of each placement signature is spread to the associated model fragments. As a result, step 4 assigns a fitness value for each model fragment present in the population.

After applying these steps, each model fragment gets a fitness value. The higher the number of products where the placement signature is present the better, as this means that it will be able to formalize the variability of a higher number of product models.

Once the population fitness has been assessed, it is time to create the next generation of individuals. This new generation will be based on present generation and the fitness value will be used to ensure that best candidates are chosen as parents for the evolution process. To do so, the process makes use of three different genetic operations that will act over the individuals of the population to generate new ones. First, a selection operation will be used to select the elements that will be used as parents of the new individual. Then, a crossover operation will be used to broad the solution space that a particular solution can reach. Finally a mutation operation will be used to introduce variations in the individual hoping that the new individual performs better than its antecessor.

3.4 Selection of Parents

The selection of parents is performed following the roulette wheel selection method [1], one of the most common methods used in GA. In this method, each individual is assigned with a share of a wheel roulette proportional to their

fitness. By doing so, fitter individuals will have higher chance to be selected and go forward with the rest of genetic operations while weaker individuals will have lower probability of being selected. Other selection strategies present in literature can be used with our model fragments, as the operation simply selects individuals, the encoding does not affect the selection.

This operation selects the individuals that will be parents of the new individual that is going to be generated. Traditionally, genetic algorithms select two elements as parents with the only restriction of avoiding the same element being 'father' and 'mother' (as this would nullify the effect of the crossover operation). However, when applying our genetic algorithm to model fragments a new restriction applies: both fragment selected must reference different product models. By doing so we ensure that the crossover operation can be properly applied.

First, we perform the selection of the first parent with no restrictions. Then, when selecting the second parent, we will only allow selections of elements referencing a product model different from the first parent. However, in order to allow the algorithm to browse into a broader search space, the product models not included into the input subset by the DE will be also eligible (with a low fitness value). That is, elements already present in the population will have the fitness value from the previous step while product models not present in the population will have a fitness value of 1.

As a result, the selection operation provides a parent model fragment (obtained from the present population) and another product model (that could not be present in the actual population) that will be used for the crossover operation.

3.5 Crossover

In genetic algorithms, crossover enables the creation of a new individual generated 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 referenced product model. 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.

To achieve the latter, our crossover operation is based on model comparisons. Figure 5 shows an example of 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 broad the search space to a different product model. That is, both model fragments (the one from the first parent and the one from

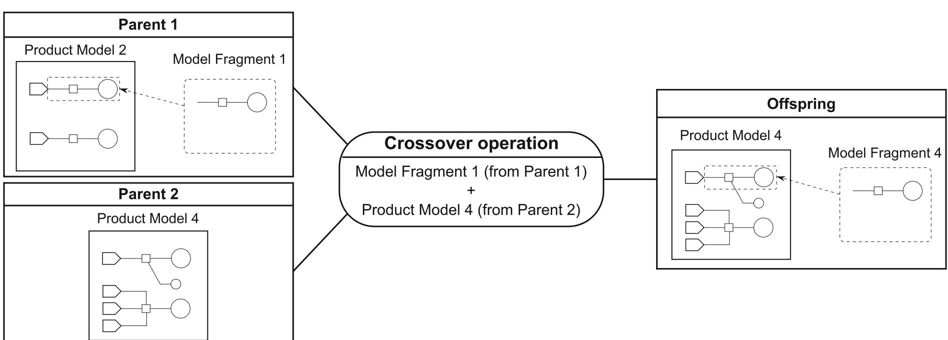


Fig. 5. Crossover operation

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. As the solution we are looking for should apply to all the models provided as input, it can be reached from any of them, but some product models can yield to the solution faster than others.

3.6 Mutation

In genetic algorithms, mutation operation introduces a random variation to the new individuals generated by the crossover operation. The mutation operation often results in a weaker individual, but occasionally the result might be a stronger individual.

Figure 6 shows an example of our mutation for model fragments. Each model fragment is associated to a product model and the model fragment mutates in the context of their associated product model. That is, the model fragment will gain or drop some elements, but the resulting model fragment will be still part of the referenced product model. The mutation possibilities of a given model fragment are driven by their associated product model.

To perform the mutation, the type of mutation that will occur (either addition or removal of elements) is decided randomly:

Removal of Elements: This kind of mutation randomly removes some elements from the model fragment. The only constraint is that elements are selected from the edges of the model fragment (they are connected with a single element), so the resulting model fragment is still connected (we can navigate from any element to any other element through the connections between the elements) and is not split in two isolated groups of elements. As the resulting model fragment is a subset of the original model fragment, and the original was present in the referenced product model, the resulting product model will be always present in the referenced product model.

Addition of Elements: This kind of mutation randomly adds some elements to the model fragment. The only constraint is that the resulting model fragment is present in the referenced product model. To achieve it, the boundaries of the

model fragment with the rest of the product model are identified and then a random element from the boundary is added to the resulting model fragment. By doing so, the mutated model fragment will be part of the referenced product model.

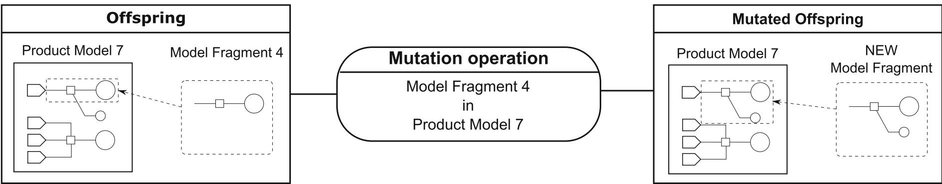


Fig. 6. Mutation operation

As a result, a new model fragment is created but it still references the same product model. That is, the individual represent other possible feature realization present in the product model for the particular feature being located. The next time the fitness is calculated, the placement signature described by this model fragment will be extracted and evaluated to assess how good it is as a solution.

4 Case Study

To evaluate the approach we are going to compare the presented GA-FL approach with CMP-FL, an approach to Feature Location in product models that makes use of the information provided by DEs. We are going to validate the results from both approaches against an oracle obtained from our industrial partner (BSH), the leading manufacturer of home appliances in Europe. Their induction division has been producing induction hobs (under the brands of Bosch and Siemens among others) for the last 15 years. The firmware of the different induction hobs is generated following a model-based SPL approach. First, a resolution for a product is created choosing from the set of features present in the variability model (each feature is formalized as model fragments). Then, a product model is generated by executing the product resolution (CVL execution capabilities produce a product model including the model fragments from the features selected). Finally, the firmware of the induction hob is obtained applying a model transformation to the resulting product model.

4.1 Case Study Setup

Figure 7 presents an overview of the process followed to evaluate the presented approach. Top part shows the oracle, a set of product models and their formalization of features. The product models from the oracle are used to construct three different scenarios regarding how good is the input fed to the approaches (left part of Fig. 7). Then each scenario is test against both approaches, (CMP-FL) and the presented approach (GA-FL). As a result each approach provides

a set of placement signatures that realize the feature being located. Each set of solutions is compared with the placement signature present in the oracle for that particular feature being located (right part of Fig. 7). We want to determine if the solution used by our industrial partner (from the oracle) is present among the solutions provided by each approach in each scenario.

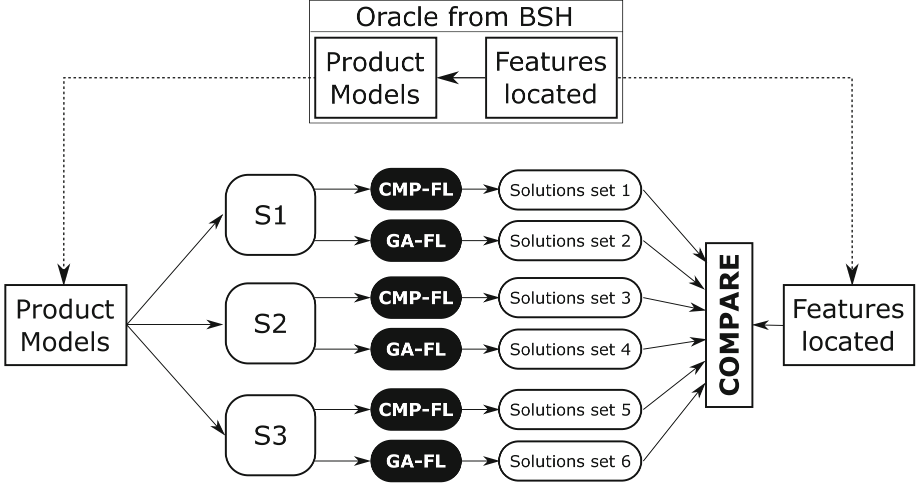


Fig. 7. Overview of the evaluation with the oracle

The oracle is composed of a set of product models and the set of features (used to define the products) properly located. That is, for each feature used by the products (around 100 features) has been previously located and validated by our industrial partner (the oracle is extracted from a set of product models that are currently under production). Therefore, we will consider the oracle as the ground truth for the evaluation process. The set of product models consist of 46 induction hob models, each of them model composed of around 100 elements (on average) that can be part or not of a model fragment. Therefore, the number of possible combinations can be calculated as the power set of the set S of elements $P(S)$, resulting in around 2^{100} ($|P(S)| = 2^n \quad where\, |S| = n$) different potential model fragments. We generate the product models attending to the oracle to distinguish three different scenarios regarding how accurate is the input fed to the approaches:

S1 High Accuracy: The first scenario corresponds to what we consider a high accuracy input from the user. More than a 75 % of the products used as input for the approaches corresponds to the subset of product models (46 available) that actually include a formalization of the feature that is being located (extracted from the oracle); and thus the placement signature will match with those product models.

S2 Medium Accuracy: The second scenario corresponds to a medium accuracy input from the user. Between 25 % and 75 % of the products used as input

for the approaches include a formalization of the feature that is being located. Therefore, a similar percentage (25 % to 75 %) of the products do not contain a formalization of the feature being located.

S3 Low Accuracy: The third scenario corresponds to a low accuracy input from the user. Only less than a 25 % of the products used as input include a formalization of the feature that is being located. This results in some deliberately bad cases (e.g. select only products that do not include the feature being located).

In the three scenarios, the size of the input is randomly selected and ranges from 1 to 5 product models. The seed fragments have been obtained randomly. For each of the features present in the oracle we generate 100 different test cases for each of the three scenarios (S1, S2 and S3). Then, each test case is tested against both approaches (CMP-FL and GA-FL). Finally, the solutions sets (placement signatures) provided by the approaches are compared against the oracle. As a result, we can determine if the feature realizations that is actually being used by our industrial partner (the expected solution) is present among the solution sets returned by the approaches. We do this comparing the placement signature from the oracle with the set of placement signatures provided as solution and determining whether it is present or not.

4.2 Results

The CMP-FL was able to provide a set of solutions that included the expected solution in 86 % of the cases from S1 (high accuracy input). Nevertheless, the presented GA-FL was able to include the expected solution in 79 % of the cases. The CMP-FL was able to include the expected solution into the solutions set in 48 % of the cases from S2 (medium accuracy input). When the information provided by the user is not accurate enough, the approach fails to include the expected (oracle) option into the resulting set. By contrast, the GA-FL was able to include it in 73 % of the cases. Finally, the CMP-FL was able to include the expected solution into the solutions set in 16 % of the cases from S3 (low accuracy input). The approach only search in the product models provided by the user and is not able to look for the solution in other product models. By contrast, the GA-FL approach was able to include the expected solution in 63 % of the cases from S3. Given the stochastic nature of the Genetic Algorithm, the approach is able to find the solution even if the input provided is not accurate.

The justification of the different results provided by both approaches resides in how the search space is traversed. That is, the different elements evaluated as possible solutions by each of the approaches. The CMP-FL approach only explores the portion of the solution space delimited by the product models used as input. In contrast, the GA-FL approach is capable of traversing the entire solution space, independently of the input.

The GA-FL approach is capable of reaching any possible solution from the search space, as it can move across the search space in any direction. The mutation enables the exploration of solutions within the same product, while the

crossover operation enables to switch to another product (an further explore it with subsequent random mutations). By contrast, the CMP-FL approach is bounded by the input of the user and only explores solutions within the product provided as input; thus, some areas of the search space cannot be reached.

As a result, the CMP-FL is not able to provide better results than the input provided; that is, upon a 75 % of accuracy will provide the expected result 75 % of the cases. In particular in all the cases where the accuracy was 0 % (from S3) the expected solution was not included. In contrast, the presented approach is able to explore solutions beyond the input provided by the user. This means that upon the scenarios where the input is not accurate enough, the crossover operation will (eventually) be able to switch to different product models that convey to the expected solution.

5 Related Work

Some works report their industrial experiences in a wide range of fields transforming legacy products into Product Line assets [10,11,13]. These approaches focus on capturing guidelines and techniques for manual transformations. In contrast, our approach introduces automation into the process while taking advantage from the knowledge of the domain experts.

Other works focus on the automation of the extraction process [6,9,14,16–18], obtaining the variability from legacy products by comparing the products with each other. In [17], the similarity between models is measured following an exchangeable metric, taking into account different attributes of the models. Then, the approach is further refined [9] to reduce the number of comparisons needed to mine the family model. Rubin et al. [16] propose a generic framework for mining legacy product lines and automating their refactoring. They compare the elements of the input with each other, matching those whose similarity is above a certain threshold and merging them together. The authors in [18], propose a generic approach to automatically compare products and extract the variability among them in terms of a CVL variability model. The authors in [14] propose an approach based on comparisons to extract the variability of any kind of asset. However, these approaches are based on mechanical comparisons, automatically turning identical elements into common parts of the SPL, similar elements as alternatives for a feature and unmatched elements into optional features. In contrast, our work enables the DE to decide which elements should be formalized as part of a feature based on the results of the comparisons.

Finally, there are some research efforts that apply genetic algorithms to the SPLs domain. For instance, the authors in [7] present GAFES, an artificial intelligence approach for optimized feature selection in SPLs. The authors in [15] present a genetic algorithm that finds optimal configurations of a Dynamic SPL at run-time. However, the solutions of those genetic algorithms are encoded as strings of binary values specifying the presence or absence of each feature. By contrast, our approach is applied directly to the product models and model fragments, resulting in a different encoding and set of genetic operations customized to work with model fragments.

6 Conclusion

In this paper we have presented a Genetic Algorithm to Feature Location (GA-FL) approach. To the best of our knowledge it is the first Genetic Algorithm applied to feature location over models. We have provided a custom encoding that enable the GA to work with model fragments and a set of genetic operations that can be applied to individuals following that encoding. We have presented a fitness function, a parent selection operation, a crossover operation (capable of bring together elements from two parents into a single offspring) and a mutation operation (that produces slight variations of the individual being mutated).

Finally we have compared the presented GA-FL with CMP-FL in terms on how both approaches traverse the search space. This comparison shows that CMP-FL does not traverse the whole space, failing to find a solution under some scenarios, while the GA-FL is capable of traversing the whole search space reaching the solutions. In addition, in scenarios where the CMP-FL approach is able to find the best solution, our GA-FL approach is also able to do so while traversing more elements from the search space, providing a more complete solution.

The ideas of the presented approach are generic and can be applied to any MOF-based models. Our next steps will involve the application of the presented GA-FL approach to CAF[1], an international company that builds and deploy railway solutions. They are currently shifting to a model-based SPL approach and there is a need of locating the features among their existing product models.

References

1. Affenzeller, M., Winkler, S., Wagner, S., Beham, A.: Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications, 1st edn. Chapman & Hall/CRC, London (2009)
2. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: 7th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS) (2013)
3. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold, New York (1991)
4. Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D.: Feature location in source code: a taxonomy and survey. J. Softw.: Evol. Process **25**(1), 53–95 (2013)
5. Font, J. , Arcega, L., Haugen, Ø., Cetina, C.: Building software product lines from conceptualized model patterns. In: Proceedings of the 19th International Conference on Software Product Line (SPLC), pp. 46–55 (2015)
6. Font, J., Ballarín, M., Haugen, Ø., Cetina, C.: 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), pp. 411–418 (2015)
7. Guo, J., White, J., Wang, G., Li, J., Wang, Y.: A genetic algorithm for optimized feature selection with resource constraints in software product lines. J. Syst. Softw. **84**(12), 2208–2221 (2011)

[1] www.caf.es/en.

8. Haugen, Ø., Moller-Pedersen, B., Oldevik, J., Olsen, G., Svendsen, A.: Adding standardized variability to domain specific languages. In: 12th International Software Product Line Conference, SPLC 2008, pp. 139–148, September 2008
9. Holthusen, S., Wille, D., Legat, C., Beddig, S., Schaefer, I., Vogel-Heuser, B.: Family model mining for function block diagrams in automationsoftware. In: Proceedings of the 18th International Software Product Line Conference, vol. 2, pp. 36–43 (2014)
10. Kim, K., Kim, H., Kim, W.: Building software product line from the legacy systems "experience in the digital audio and video domain". In: 11th International Software Product Line Conference, SPLC 2007, pp. 171–180, September 2007
11. Kolb, R., Muthig, D., Patzke, T., Yamauchi, K.: Refactoring a legacy component for reuse in a software product line: a case study: practice articles. J. Softw. Maint. Evol. **18**(2), 109–132 (2006)
12. Krueger, C.W.: Easing the transition to software mass customization. In: van der Linden, F. (ed.) Software Product-Family Engineering. LNCS, vol. 2290, pp. 282–293. Springer, Heidelberg (2002)
13. Lee, H., Choi, H., Kang, K.C., Kim, D., Lee, Z.: Experience report on using a domain model-based extractive approach to software product line asset development. In: Edwards, S.H., Kulczycki, G. (eds.) ICSR 2009. LNCS, vol. 5791, pp. 137–149. Springer, Heidelberg (2009)
14. Martinez, J., Ziadi, T., Bisyandé, T.F., Klein, J., Traon, Y.L.: Bottom-up adoption of software product lines,: a generic and extensible approach. In: Proceedings of the 19th International Conference on Software Product Line (SPLC), pp. 101–110 (2015)
15. Pascual, G.G., Pinto, M., Fuentes, L.: Self-adaptation of mobile systems driven by the common variability language. Future Gener. Comput. Syst. **47**, 127–144 (2015). Special Section: Advanced Architectures for the Future Generation of Software-Intensive Systems
16. Rubin, J., Chechik, M.: Combining related products into product lines. In: de Lara, J., Zisman, A. (eds.) FASE 2012 and ETAPS 2012. LNCS, vol. 7212, pp. 285–300. Springer, Heidelberg (2012)
17. Wille, D., Holthusen, S., Schulze, S., Schaefer, I.: Interface variability in family model mining. In: Proceedings of the 17th International Software Product Line Conference: Co-located Workshops, pp. 44–51 (2013)
18. Zhang, X., Haugen, Ø, Moller-Pedersen, B.: Model comparison to synthesize a model-driven software product line. In: Proceedings of the 15th International Software Product Line Conference (SPLC), pp. 90–99 (2011)

11.4 MODELS'16 Paper

Title:	Feature Location in Models Through a Genetic Algorithm Driven by Information Retrieval Techniques.
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Proceedings:	Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MODELS '16)
Location:	Saint-malo, France - October 02 - 07, 2016
Publisher:	ACM New York, NY, USA
Pages:	272-282
DOI:	`http://dl.acm.org/citation.cfm?id=2976789`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work. He was also responsible for the oral presentation of the work which took place during the conference.

Feature Location in Models through a Genetic Algorithm Driven by Information Retrieval Techniques

Jaime Font[1,2] Lorena Arcega[1,2] Øystein Haugen[3] Carlos Cetina[1]

[1]Universidad San Jorge
SVIT Research Group
Autovía A-23 Km. 299
50830 Zaragoza, Spain
{jfont,larcega,ccetina}@usj.es

[2]University of Oslo
Department of Informatics
Postboks 1080 Blindern
0316 Oslo, Norway
oysteinh@ifi.uio.no

[3]Østfold University College
Faculty of Computer Science
Postboks 700
1757 Halden, Norway
oystein.haugen@hiof.no

ABSTRACT

In this work we propose a feature location approach that targets models as the feature realization artifacts. The approach combines Genetic Algorithms and Information Retrieval techniques. Given a model and a feature description, model fragments extracted from the model are evolved using genetic operations. Then, Formal Concept Analysis is used to cluster the model fragments based on their common attributes into feature realization candidates. Finally, Latent Semantic Analysis is used to rank the candidates based on the similarity with the feature description. As a result, the genetic algorithm evolves the population of model fragments to find the set of most suitable feature realizations. We have evaluated the approach with an industrial case study, locating features with precision and recall values around 90% (baseline obtains less than 40%). Finally, we provide recommendations on how to provide the input to the approach to improve the location of features over the models.

1. INTRODUCTION

Feature location (FL) is known as the process of finding the set of software artifacts that realize a particular feature. The topic has gained momentum during recent years [22, 6] and several research works focus on creating and improving methods to locate the features. FL is one of the main activities performed during software evolution [6] and up to an 80% of a system's lifetime is spent on the maintenance and evolution of the system [19].

However, most of the research on FL has been directed towards the location of features into source code artifacts, neglecting other software artifacts such as the models. In addition, the approaches that perform FL over models [10, 8, 20, 26, 27, 21] are designed to generate the variability specififcation over a family of models and built a Software Product Line from them and it is not possible to apply them to isolated models. Therefore, there is a lack of proper FL techniques that can be applied to locate the model elements that belong to a feature that has to be evolved or maintanied.

In this work we propose FLM (Feature Location in Models), an FL approach that targets models as the realization artifacts and does not rely in model comparisons, but on Information Retrieval (IR) techniques. The approach is based on a Genetic Algorithm (GA) that generates alternative model fragments that can be the realizations of the feature being located. Then, we use Formal Concept Analysis (FCA) [11] to cluster the model fragments by their common attributes and to generate feature candidates. The feature candidates are assesed comparing them to a search query that describes the target feature by using Latent Semantic Analysis (LSA) [17] to measure the similarity between both. As a result, feature candidates can be ranked based on the distance with the feature description, allowing the best ones to be selected to engendrate the next generations. When the improvement of the generations is stalled, the resulting feature candidates are provided as output.

The presented approach has been supported by a tool within the Eclipse environment and applied to the product models obtained from our industrial partner BSH, one of the largest manufacturers of home appliances in Europe. Its induction division has been producing Induction Hobs (sold under the brands of Bosch and Siemens) for the last 15 years. The firmware for their products is generated from models using a model-based approach. The application of the approach shows that the values of recall and precision higher than 85%. Finally, we provide some recommendations on how to provide the input to the approach to improve the location of features over the models.

The rest of the paper is structured as follows: Section 2 provides some background. Next, Section 3 provides the details of the presented approach. Section 4 presents the evaluation performed. Then, Section 5 discusses the approach. Finally, Section 6 presents some related work and the paper is concluded.

2. BACKGROUND

This section presents the Domain Specific Language (DSL) used by our industrial partner to formalize their products, the IHDSL. It will be used through the rest of the paper to present a running example. Then, the Common Variability Language (CVL) is presented, CVL is the language used by our approach (FLM) to formalize the model fragments used

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

MODELS '16, October 02-07, 2016, Saint-Malo, France

© 2016 ACM. ISBN 978-1-4503-4321-3/16/10...$15.00

DOI: http://dx.doi.org/10.1145/2976767.2976789

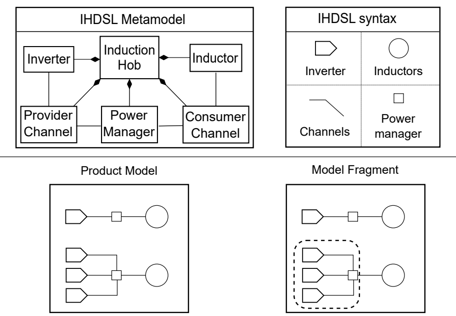


Figure 1: IHDSL product model and model fragment formalization

as feature candidates.

The newest Induction Hobs (IHs) feature full cooking surfaces, where dynamic heating areas are automatically generated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. In addition, 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 energy consumption of the IH. All of these changes are being possible at the cost of increasing the software complexity and thus require several modifications into the models used to formalize the products.

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 paper we use a simplified subset of the IHDSL (see top part of Figure 1).

Inverters are in charge of transforming the input electric supply to match the specific requirements of the IH. Then, the energy is transferred to the inductors 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 the energy is transformed into an electromagnetic field.

Bottom left part of Figure 1 depicts an example of a product model specified with the IHDSL. The product model contains four inverters used to power two different inductors. The upper inductor is powered by a single inverter while the lower inductor is powered by the combination of three different inverters. Power managers act as hubs to perform the connection between the inverters and the inductors.

To formalize the model fragments used by the approach we use the Common Variability Language (CVL) [13, 24], given its capabilities to formalize the feature realizations in terms of model fragments. CVL defines variants of a base model (conforming to MOF, the OMG metalanguage for defining modeling languages) by replacing variable parts of the base model (the features) by alternative model replacements (the feature realizations) found in a library.

Bottom right part of Figure 1 shows an example of a model fragment of the product model (bottom-left part). The model fragment includes the three inverters in charge of powering the lower inductor along with the three provider channels and the power manager used to aggregate and manage the power provided by those inverters. This model fragment is the realization of the feature "triple inverter".

3. FEATURE LOCATION IN MODELS (FLM)

This section presents FLM, the proposed approach for feature location in product models. The objective of the approach is to provide the subset of elements from a given product model that realize a particular feature being requested by the user. To do so, the approach relies on a Genetic Algorithm that iterates a population of model fragments and evolves them using genetic operations. As output the approach provides a list of feature candidates that might be realizing the feature. This list is ranked taking into account the information provided by the user as input.

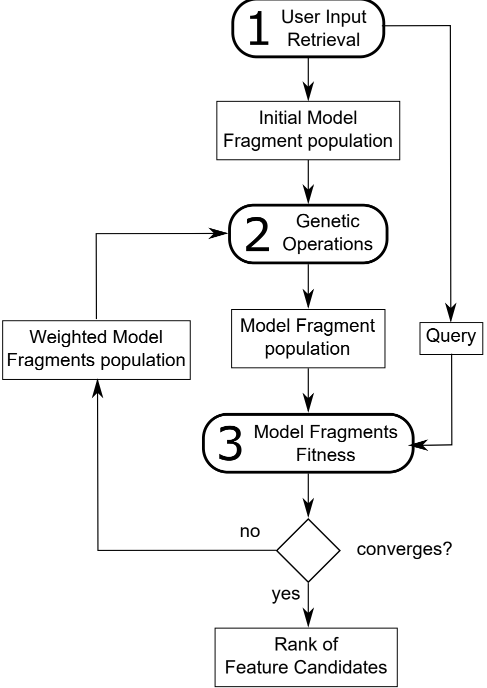


Figure 2: Feature Location in Models Overview

Figure 2 presents an overview of the approach. Rounded boxes represent the different steps of the approach while rectangular boxes represent the inputs and outputs of each of the steps. Lines indicate that an element is an input or output of one of the steps.

The input of the approach is the product model where the feature is going to be located. Then, the user provides a description of the target feature in terms of an initial seed

fragment and a textual description of the feature. The initial seed and the product model are used to generate some candidate fragments. Then, those candidates are assessed taking into account the textual description of the feature being located. These two steps (generation and assessment) are repeated until some stop condition is met. When the stop condition is fullfilled the process returns the list of fragment candidates ranked according to the assessments.

3.1 User Input Retrieval

The first step is to gather input for the feature location. The input will consist of the product model and information about the target feature provided by the user. In particular, the user will provide a seed fragment and a textual description of the feature.

A **seed** fragment of the target feature is an element or set of elements that the engineer believes that could be part of the feature being located. To do so, the engineer 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.

A **feature description** of the target feature, using natural language. Typically these descriptions can come from textual documentation of the products, comments in the code, bug reports or oral descriptions from the engineers. Therefore, the query will include some domain specific terms similar to those used when specifying the product models. The knowledge of the engineers about the domain and the product models will be useful to select the textual description from the sources available.

Figure 3 presents an example of input for the approach. Left part presents the seed fragment proposed by the user (a model fragment of the product model where the feature is going to be located). The user believes that the selected inductor is going to be part of the feature realization. Then, the right part of the figure shows a textual description for the feature being located, the hotplate. It is a simplified version of a text description that has been extracted from the internal documentation used by our industrial partner to describe their products.

The textual description provided by the user is turned into a query by using some well stablished IR techniques:

- First, the textual description is tokenized (divided into words). Usually a white space tokenizer can be applied (that splits the strings whenever it finds a white space) but for some sources 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 can be following CamelCase naming.
- Secondly, we apply the Parts-of-Speech (POS) tagging technique. POS tagging analyses the words grammatically and infers the role of each word into the text provided. As a result, each word is tagged enabling 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 commonly used and do not contribute with relevant information that describe the feature, so they are removed.
- Thirdly, stemming techniques are applied to unify the language used in the text. This technique consists of reducing each word to its roots enabling that different words referring to similar concepts can be grouped together. For instance, plurals are turn into singulars (inductors to inductor) or verbs tenses are unified (using and used are turn into use).

Fragment Seed

Feature Description

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.

Figure 3: Input provided to the approach

The User Input Retrieval step generates as a result an initial population of fragments (that contain the model fragment provided as seed) and the query that will be used for the comparisons (obtained from the textual description). Then, the model fragment from the initial population will be evolved into several model fragments through the use of the genetic operations.

3.2 Genetic Operations

The second step is to generate a set of model fragments that could be realizing the feature. The generation of model fragments is done by applying genetic operators adapted to work over model fragments. That is, new fragments based on the existing ones (the seed fragment during the first execution) are generated through the use of three genetic operators: the selection of parents, the mutation and the crossover.

In order to apply the genetic operators, it is first necessary to apply the **selection operator** that selects the best candidates from the population to be the input for the rest of operators. There are different methods that can be used to perform the selection of the parents, but one of the most spread choices is to follow the wheel selection mechanism [4]. That is, each model fragment from the population has a probability of being selected proportional to their fitness score. Therefore, candidates with high fitness values will have higher probabilities of being chosen as parents for the next generation. Top part of Figure 4 shows an example of application of the selection operator.

The **crossover operator** 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 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 operator applied to model fragments takes as input two model fragments and a randomly generated mask 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 on the parent or not). A model fragment is a subset of the elements present in a product model. As both model fragments have

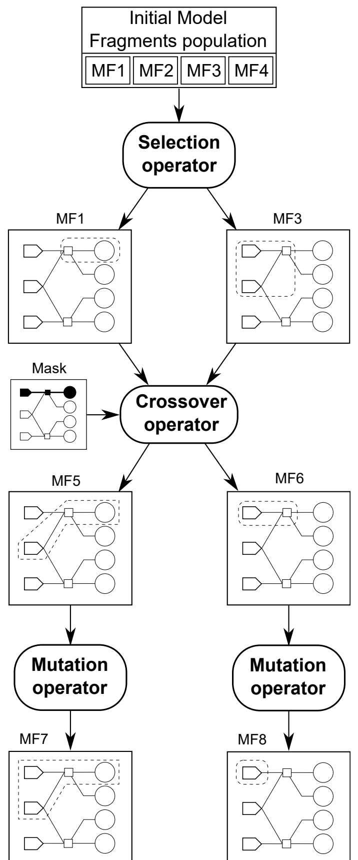


Figure 4: Genetic Operators: Mutation and Crossover over Model Fragments

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 applying directly the mask and another one by applying the inverse of the mask as it is usually done in genetic algorithms [5].

Figure 4 shows an example of application of the crossover operator. The input of the operator 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 we interpret 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 middle-right part of Figure 4), 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 target feature.

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 a 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 kind of modifications, addition of elements to the fragment, or removal of elements from the model fragment.

Bottom part of Figure 4 shows two examples of application of the mutation operator. Left part shows the first example, MF5 is used as input of the operator that produces M7 as output. In this example, the mutation operation has added some elements (a new inverter connected to the power manager). The resulting model fragment remains being part of the product model that is driving the mutation, so it is a candidate as realization of the feature. Right part shows the second example, where MF6 is used as input and MF8 is produced as output. In this example the mutation operator has removed an element (the power manager).

3.3 Model Fragment Fitness

The third step of the process consists of the assessment of each candidate fragment produced and the ranking of them according to a fitness function. The fitness function is used to imitate the different degrees of adaptation to the environment that different individuals have. Therefore, individuals that result of mutations and crossovers that contribute to their adaptation to the environment will have higher chances of survival that others.

Following this idea, the fitness function is used to determine the suitability of each candidate as solution to the problem, enabling to rank them from the best candidate to the worst. The fitness function is based on the comparison between the feature description query and the identi-

fier names and other natural language items present in the model fragments. The input of this step is a population of candidate fragments, and the feature description query; the output produced is a ranking where each candidate has been assigned with a fitness value.

However, 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. Therefore, as part of our fitness function we will follow two steps: (1) the population of fragments will be grouped according to their similarities in terms of the domain, then (2) each one of these groups will receive a fitness value obtained using the feature description obtained from the user as part of step one.

3.3.1 Grouping of Model Fragments into Feature Candidates through FCA

To perform the grouping of model fragments into feature candidates we rely on Formal Concept Analysis (FCA) [11], 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 built.

Top of Figure 5 shows an example of a formal context relating model fragments and the metamodel elements used to build them. Columns show each of the attributes 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 every one is a maximal collection of model fragments that share common metamodel elements. Bottom of Figure 5 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.

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.

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

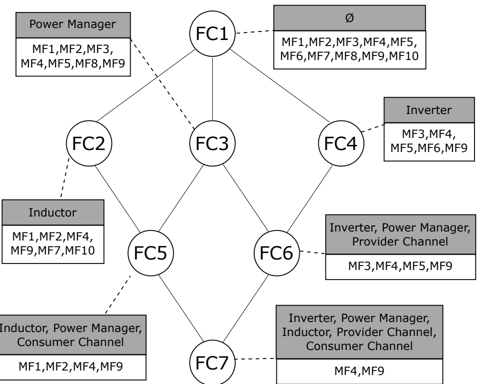


Figure 5: Formal context between model fragments and metamodel elements and Lattice obtained through FCA

3.3.2 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 are going to apply methods based on Information Retrieval (IR) techniques. In particular 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.

LSA constructs vector representations of a query and a

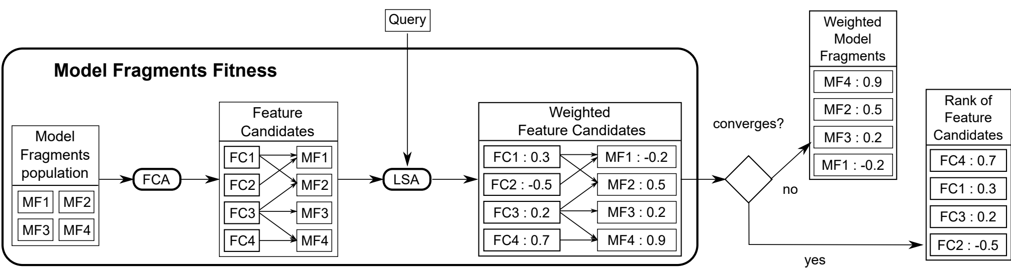


Figure 6: Model Fragment Fitness

corpus of text documents by encoding them as a term-by-document co-occureence matrix. That is, a matrix where each row corresponds to terms and each column corresponds to documents, followed by the query in the last column. Then, each cell holds the number of occureences of a term (row) inside a document or the query (column).

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) [17]. One vector that represents the latent semantic is obtained for each document and the query. Finally, the similarities between the query and each document are calculated as the cosine between both vectors, obtaining values between -1 and 1.

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 contained by 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 performed with the textual description provided in the first step, see Section 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.

	FC1	FC2	FC3	FC4	FC5	FC6	FC7	Q
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 7: Term-by-document co-occurrence matrix for Feature Candidates

Figure 7 shows an example of co-occurrence matrix for our running example. Each column is one of the Feature Candidates obtained through the application of FCA. Then, the last column is the query provided by the user as part of the input of the process. Each row is one of the terms extracted from the corpuses of text conformed by all the feature candidates and the query itself (we show the terms before the stemming process to improve the readability). Each cell shows the number of occurrences of each of the terms into the feature candidates.

3.3.3 Loop

The next step is to spread the similarity values obtained by each feature candidate to the model fragments contained by 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 6 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 met yet, the genetic algorithm will keep its execution one 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.

The process of generation of fragments, extraction of feature candidates and assessment of those candidates is repeated until the stop condition is met. Usually, the stop condition can be a time slot, a fixed number of generations or a trigger value of the fitness that makes the process finish when reached. In addition, it also possible to monitor the fitness values and determine when they are converging and no further improvements are being made by new generations. The stop condition highly depends on the domain and the problem being solved; therefore, it is adjusted depending on the results being outputted by the process.

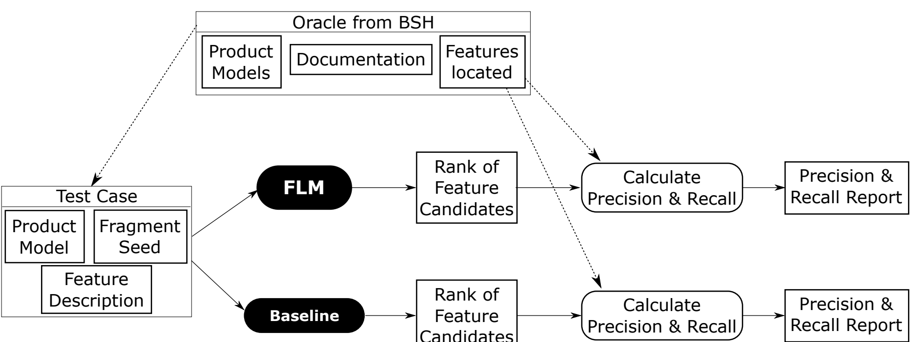


Figure 8: Evaluation Overview

4. EVALUATION

To evaluate the approach we applied it to a case study extracted from our industrial partner BSH, the leading manufacturer of home appliances in Europe. Their induction division has been producing Induction Hobs under the brands of Bosch and Siemens for the last 15 years. The firmware of the different induction hobs is generated following a model-based Software Product Line approach.

We are going to use the product models from BSH as an oracle to evaluate the presented approach. That is, we make use of a set of products models whose feature realizations are known beforehand and will be considered as the ground truth, enabling us to compare the results provided by the approach with the oracle.

4.1 Setup

Figure 8 shows an overview of the process followed to evaluate the presented approach. Top part shows the oracle for the case study, a set of product models, features located over those product models, and documentation obtained from the model-based SPL of our industrial partner. Those features correspond to products that are currently being sold or will be released to the market in the near future. This oracle will be considered the ground truth and will be used to evaluate the presented approach.

The oracle is composed of 46 induction hob models where each product model is composed of more than 500 elements on average. For each of the 96 features used to build the product models we got a test case including a product model, a fragment seed (extracted from the located feature in the oracle) and a feature description (obtained from the documentation of the features).

Then, each test case was fed as input for two different executions: the presented approach where the Genetic Algorithm fitness is performed through FCA and LSI (FLM); the same Genetic Algorithm but using a random fitness function to assign random values to each model fragment instead of using IR techniques (Baseline). As a result we got a pair (one for FLM and one for Baseline) of Feature Candidates rankings for each of the test cases.

Finally, we computed the precision, recall and F-measure values (three of the most common measures for information retrieval methods [23]) for each of the Feature Candidates rankings, obtaining a precision and recall report. Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle), while recall measures the number of elements of the solution that are retrieved by the proposed solution. F-measure combines precision and recall into a single value and corresponds to the harmonic mean of precision and recall.

To calculate the precision and recall we need to compute the true positives (TP); the number of elements in the solution that are actually correct according to the ground truth (the oracle). That is, the number of elements that are present in both, the solution and the ground truth. The precision is calculated dividing the TP by the total number of elements in the solution. The recall is calculated dividing the TP by the total number of elements in the ground truth.

In our case, each feature candidate from the rankings is a model fragment composed of a subset of the model elements present in the product model (where the feature is being located). The granularity will be at the level of model elements, so each model element present in both (the solution feature candidate and the located feature from the oracle) will be a TP.

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.

The presented approach has been implemented within the Eclipse environment. We have used Eclipse Modeling Framework (EMF) to manipulate the models from our industrial partner, and the Common Variability Language to manage the fragments of models. The genetic algorithm is built upon Watchmaker Framework for Evolutionary Computation [7] that enable us to implement our own genetic operators. Regarding the IR techniques, we have used colibri-java [12] to

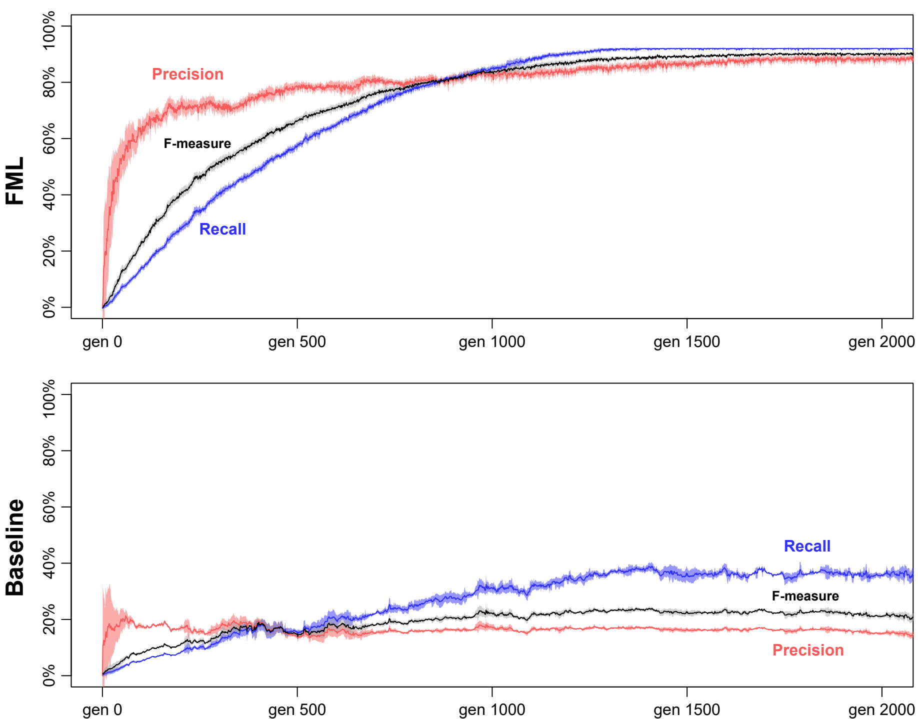


Figure 9: Mean Precision, Recall and F-measure for FLM and the Baseline

implement the FCA. The IR techniques used to process the language have been implemented using OpenNLP [2] for the POS-Tagger and Snowball [3] for the stemming. Finally, the LSI has been implemented using the Efficient Java Matrix Library (EJML [1]).

The evaluation has been executed using a Dell XPS with a processor Intel(R) Core(TM) i7-2670QM @2.2GHz with 8 GB or RAM and running Windows 10 Pro N 64 bits as the hosting Operative System. The approach has been executed under Java(TM) SE Runtime Environment (build 1.8.0_73-b02).

4.2 Results

Figure 9 shows the mean precision and recall values measured for the 96 features located by both executions (the presented approach and the Baseline). Top chart shows the results for the execution of the presented approach while 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 genetic 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 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 the 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 the 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 (Base-

line). 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.

5. DISCUSSION

The evolution of the recall and precision values over the generations suggests that the proposed fitness function is performing well and guiding the algorithm to find feature realization candidates close to the target feature.

5.1 Input data

The presented approach relies on two pieces of information given by the engineer performing the feature location, the **seed** fragment and the **query**. These two elements will have an impact on the ranking of feature candidates produced and must be chosen carefully by the engineer performing the feature location.

To test out the impact of the **seed** fragment in the results, we have executed the approach with different kind of seed fragments containing one element (single element belonging to the feature being located or single element not belonging to the feature being located). But those executions did not produce noticeable differences in the resulting ranking of feature candidates or in the number of generations needed to converge.

However, when selecting seed fragments of sizes closer to the size of the feature being located, the effect is noticeable. The number of generations needed by the GA to converge was reduced when a seed fragment close to the feature being located was chosen. In particular, when the fragment seed contained about 50% of the elements belonging to the feature being located, the number of generations needed for the GA to converge was reduced up to 15%.

Overall, when the engineer provides a seed that is not related with the target feature, the recall and precision of the feature candidates returned is not affected. Therefore, our recommendation is that when the engineer is pondering whether to include a model fragment seed or not, he should do it. The approach is capabale of accepting more than one model fragment as seed at the same time and we have performed some test (up to 10 related and unrelated seeds) that reveals that there is no negative impact when unrelated seeds are included.

To test out the impact of the **query** in the results, we have also executed the approach varying the text description used as input (using longer and smaller queries by subsetting the original description, including more or less domain terms and including more or less meta-element terms).

The search **query** used to locate the feature is in charge of driving the search and greatly impacts on the precision and recall results. In fact, depending on the level of detail of the query, the recall and precision values obtained will change. When the query provided is too broad, the precision decreases as there are several model elements matching the query not belonging to the target feature. Anyhow, the elements belonging to the feature will be also matched positively so the recall value will be high. However, when the query provided is too specific, some of the elements relevant for the feature being located can be missed out. Thus, the recall value is decreased although the precision values remain high.

To achieve good precision and recall values, it is important to avoid the usage of words included into the meta-elements of the model elements. That is, if we refer to the metaclass name of one of the model elements, all instances of this class will match to that word (e.g. any inductor class model element will match the query "inductor"). By contrast, by using words specific for the model element (as the value of the name property or values of some of the parameters contained in those model elements), those model elements (and not others with the same class) will be included into the feature candidates, affecting positively to the precision values (e.g. only some inductors will match the query "doubleTwistedCoil" as it is the value of a property of the inductor class). In fact, when removing the usage of meta-element names in the queries, the approach obtained similar values of recall but the precision raised up to a 20% for best cases.

It is important to notice that we have made use of an oracle (obtained from our industrial partner model-based SPL and considering the ground truth) to evaluate the approach using test cases where the expected solution was known beforehand. By doing so, we were able to compute the recall, precision and F-measure for the feature candidates rankings provided by the approach. However, when applying the presented approach to locate features (and thus not having an oracle), the approach should be used iteratively, refining the query and the seed fragment as described above.

5.2 Scalability

The presented approach has been executed to locate 96 features over 46 product models of sizes ranging from around 400 elements to around 600 elements (around 500 elements on average). Each feature being located has a relative size in the range between 3% and 10% of the product model where the feature is being located. The mean number of generations needed to locate those features (when the genetic algorithm converges) is about 1500 generations.

The time needed to locate the features ranged between 12 seconds and 26 seconds. That is, the 1.500 generations where generated in an average time of 19 seconds. Most of the time (around 85%) was spent on the execution of the fitness function while the rest was used to process the query (3%) and execute the genetic operations (12%). The approach is able to reach a million of generations within less than 5 minutes when locating features over models with dimensions similar to the models of our industrial partner.

The implementation of the approach is far from being optimized. Furthermore, the computer used to run the case study is a four years old laptop. Therefore, the performance of the approach could be increased by some means if necessary.

5.3 Generalization

The presented approach has been designed to be applied, not only to our industrial partner domain, but to any domain. The only requisite to apply the approach is that the set of models where features have to be located conform to MOF (the OMG metalanguage for defining modeling languages). The query must be provided as a textual description.

The generation and management of fragments is performed using the Common Variability Language (CVL), which can be applied to any MOF-based languages. With the use of CVL, the approach is able to work with the model fragments

provided as seed and evolve them applying the genetic algorithms. As output, the approach produces a set of feature candidates rankings in the form of CVL model fragments.

Furthermore, the fitness function can also be applied to any MOF-based model. The text elements associated to the models are extracted automatically by the approach using the reflective methods provided by the Eclipse Modeing Framework. That is, there is no need of knowledge about the domain of application in order to extract the relevant terms.

However, the approach can be tailored to fit the needs of different domains if necessary. For instance, the naming conventions used by companies for model elements, properties and functions can follow different formats, but the approach can be tailored to handle them. In our case study some model elements follow the CamelCase convention while others follow the Underscore convention. To address that, we applied different tokenizers in order to obtain the terms properly. Similarly, the Part-of-Speech tagger that is used to eliminate non-relevant words based on their grammatical category is language dependant, but can be configured to other languages when necessary.

In summary, the approach can be applied to locate features on any MOF-based model from any domain. If necessary, some tweaks and modifications can be applied to tailor the approach to particular needs of the domains, but the core of the approach will remain unchanged.

6. RELATED WORK

Some works report their industrial experiences in a wide range of fields transforming legacy products into Product Line assets [15, 16, 18]. These approaches focus on capturing guidelines and techniques for manual transformations. In contrast, our approach introduces automation into the process while taking advantage from the knowledge of the domain experts.

Some works [25, 14, 26, 27, 20, 10, 8] focus on the location of features over models by comparing the models with each other to formalize the variability among them in the form of a Software Product Line.

Wille et al. [25] present an approach where the similarity between models is measured following an exchangeable metric, taking into account different attributes of the models. Then, the approach is further refined [14] to reduce the number of comparisons needed to mine the family model.

The authors in [26] propose a generic approach to automatically compare products and locate the feature realizations in terms of a CVL model. In [27] the approach is refined to automatically formalize the feature realizations of new product models added to the system. A similar approach is proposed in [10] where the feature location results is validated against an industrial environemnt.

Martinez et al. [20] 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.

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 features realizations. In contrast, our work is applied to a single product model, so it does not rely on model comparisons to locate the features but in comparisons with a textual description of the target feature.

Font et al. [8] propose a generic approach to locate features among a family of product models based on a human-in-the-loop process. The features are located by comparison of models and the interaction of engineers that provide their knowledge of the domain. The approach is further refined in [9] and generalized through the use of a genetic algorithm to locate features among a family of models in the form of a variation point (model placements and a set of corresponding replacements).

However, the work from [9] cannot be applied to a single model as the fitness function is based on the number of occurrences of one model fragment among the family of models. In the present work, the feature location is performed over a single model; no family of models is required to apply the approach. We introduce a new crossover operator (that applies to model fragments extracted from the same product model) and a fitness function that combines FCA and LSA to determine the similarity between the query provided and the "evolving" model fragments. The approaches in [9] and the present work differ on (1) the scenarios where they can be applied (variability formalization of a family of products for SPL creation VS isolation of a feature realization from a single product for maintenance purposes), (2) the models that can be fed to the approach (family of models VS single model with meaningful identifier names and other natural language items present in the models), (3) the fitness function (occurrences of model fragment VS textual similarity between a search query and the model elements performed through FCA+LSA) and (4) the evaluation performed (impact of model fragments seed VS impact of the query used as input).

7. CONCLUSION AND FUTURE WORKS

As part of this work we have presented a Genetic Algorithm to Feature Location that target models as the feature realization artifacts. We propose a new crossover operator that combines two model fragments extracted from the same product model and generates a new individual that contains elements from both parents. We propose a fitness function that clusters model fragments into feature candidates and then assigns them a fitness value based on their similarity with a query. As a result, the features located by using the approach shows a recall and precision measures of around 90% while the baseline remains below 40%. Finally, we discuss the approach and provide recommendations on how to provide the input to the approach to improve the location of features over the models.

Our next steps involve the application of the presented approach to other domains. In particular we plan to apply the approach to locate the features on train models from CAF[1], an international company that builds and deploys railways solutions around the world.

Acknowledgment

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R).

[1] http://www.caf.net/en

8. REFERENCES

[1] Efficient java matrix library. http://ejml.org/. [Online; accessed 7-April-2016].

[2] Apache opennlp: Toolkit for the processing of natural language text. https://opennlp.apache.org/, 2016. [Online; accessed 7-April-2016].

[3] Snowball : Snowball is a small string processing language designed for creating stemming algorithms for use in information retrieval. http://snowball.tartarus.org/, 2016. [Online; accessed 7-April-2016].

[4] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham. *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications.* Chapman & Hall/CRC, 1st edition, 2009.

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

[6] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. Feature location in source code: a taxonomy and survey. *Journal of Software: Evolution and Process*, 25(1):53–95, 2013.

[7] D. Dyer. The watchmaker framework for evolutionary computation (evolutionary/genetic algorithms for java). http://watchmaker.uncommons.org/, 2016. [Online; accessed 7-April-2016].

[8] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Building software product lines from conceptualized model patterns. In *Proceedings of the 19th International Conference on Software Product Line (SPLC)*, pages 46–55, 2015.

[9] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Feature location in model-based software product lines through a genetic algorithm. In *15th International Conference on Software Reuse*, ICSR 2016, Limassol, Cyprus, Jun 2016.

[10] J. Font, M. Ballarín, Ø. Haugen, and C. 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)*, pages 411–418, 2015.

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

[12] D. Götzmann. Formal concept analysis implemented in java (colibri-java). https://code.google.com/archive/p/colibri-java/, 2016. [Online; accessed 7-April-2016].

[13] Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen. Adding standardized variability to domain specific languages. In *Software Product Line Conference, 2008. SPLC '08. 12th International*, pages 139–148, Sept 2008.

[14] S. Holthusen, D. Wille, C. Legat, S. Beddig, I. Schaefer, and B. Vogel-Heuser. Family model mining for function block diagrams in automation software. In *Proceedings of the 18th International Software Product Line Conference: Volume 2*, pages 36–43, 2014.

[15] K. Kim, H. Kim, and W. Kim. Building software product line from the legacy systems "experience in the digital audio and video domain". In *Software Product Line Conference, 2007. SPLC 2007. 11th International*, pages 171–180, Sept 2007.

[16] R. Kolb, D. Muthig, T. Patzke, and K. 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, Mar. 2006.

[17] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[18] H. Lee, H. Choi, K. Kang, D. Kim, and Z. Lee. Experience report on using a domain model-based extractive approach to software product line asset development. In *Formal Foundations of Reuse and Domain Engineering*, volume 5791 of *Lecture Notes in Computer Science*, pages 137–149. Springer Berlin Heidelberg, 2009.

[19] M. M. Lehman, J. Ramil, and G. Kahen. A paradigm for the behavioural modelling of software processes using system dynamics. Technical report, Imperial College of Science, Technology and Medicine, Department of Computing, Sep 2001.

[20] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon. Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th International Conference on Software Product Line (SPLC)*, pages 101–110, 2015.

[21] J. Rubin and M. Chechik. Combining related products into product lines. In *Fundamental Approaches to Software Engineering*, volume 7212, pages 285–300. Springer Berlin Heidelberg, 2012.

[22] J. Rubin and M. Chechik. A survey of feature location techniques. In I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, editors, *Domain Engineering*, pages 29–58. Springer Berlin Heidelberg, 2013.

[23] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[24] A. Svendsen, X. Zhang, R. Lind-Tviberg, F. Fleurey, Ø. Haugen, B. Møller-Pedersen, and G. K. Olsen. Developing a software product line for train control: a case study of cvl. In *14th international conference on Software product lines (SPLC)*, 2010.

[25] D. Wille, S. Holthusen, S. Schulze, and I. Schaefer. Interface variability in family model mining. In *Proceedings of the 17th International Software Product Line Conference: Co-located Workshops*, pages 44–51, 2013.

[26] X. Zhang, Ø. Haugen, and B. Moller-Pedersen. Model comparison to synthesize a model-driven software product line. In *Proceedings of the 2011 15th International Software Product Line Conference (SPLC)*, pages 90–99, 2011.

[27] X. Zhang, Ø. Haugen, and B. Møller-Pedersen. Augmenting product lines. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, volume 1, pages 766–771, Dec 2012.

11.5 TEVC'17 Paper

Title:	Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering.[1]
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Journal:	IEEE Transactions on Evolutionary Computation
Date:	September, 2017
DOI:	10.1109/TEVC.2017.2751100
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work.

[1] In Press

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

Achieving Feature Location in Families of Models through the use of Search-Based Software Engineering

Jaime Font, Lorena Arcega, Øystein Haugen and Carlos Cetina

***Abstract*—The application of Search-Based Software Engineering (SBSE) techniques to new problems is increasing. Feature location is one of the most important and common activities performed by developers during software maintenance and evolution. Features must be located across families of products and the software artifacts that realize each feature must be identified. However, when dealing with industrial software artifacts, the search space can be huge. We propose and compare five search algorithms to locate features over families of product models guided by Latent Semantic Analysis (LSA), a technique that measures similarities between textual queries. The algorithms are applied to two case studies from our industrial partners (leading manufacturers of home appliances and rolling stock) and are compared in terms of precision and recall. Statistical analysis of the results is performed to provide evidence of the significance of the results. The combination of an evolutionary algorithm with LSA can be used to locate features in families of models from industrial scenarios such as the ones from our industrial partners.**

***Index Terms*—Feature Location, Families of Models, Evolutionary Algorithm, Search-Based Software Engineering**

I. Introduction

SEARCH-based techniques have been applied successfully to a growing number of engineering problems. Software engineering is concerned with finding near optimal solutions or those that fall within a specified level of acceptable tolerance. It is precisely these factors which make search-based techniques readily applicable. Search-Based Software Engineering (SBSE) has had notable successes and there is an increasingly widespread application of SBSE across the full spectrum of Software Engineering activities and problems [1]. In addition, some research efforts demonstrate that SBSE, which was previously only applied to laboratory programs, can scale to real-world systems of tens of thousands of lines of code [2].

Feature location (FL) is one of the most important and common activities performed by developers during software maintenance and evolution [3]. FL is known as the process of finding the set of software artifacts that realize a specific feature, and it has received much attention during recent years [4], [5]. However, most of the research on FL targets code as the software artifacts that realize the feature [4], [5], neglecting other software artifacts such as the models. When performing FL over models, the set of possible realizations for a specific feature is too big to be evaluated exhaustively (a model of 500 elements can yield around 10^{29} potential fragments) and there is a need for search-based techniques to drive the process [6].

In this paper, we propose and compare five search algorithms to locate features over a family of models (MFL): Evolutionary Algorithm (EA-MFL), Random Search (RS-MFL), steepest Hill Climbing (HC-MFL), Iterated Local Search with restarts (IHL-MFL), and a hybrid between Evolutionary algorithm and Hill Climbing (EHC-MFL). The five algorithms rely on Latent Semantic Analysis (LSA) [7] as the fitness function for the evaluation of the solutions. LSA is an Information Retrieval technique that measures the similarity between two textual queries, and, in this paper, it is applied to compare the feature realization solutions with the search query that describes the feature being located.

We have applied the five approaches to two different industrial domains: BSH, the leading manufacturer of home appliances in Europe; and CAF, a worldwide leading company that manufactures rolling stock. To compare the search algorithms, we extracted two case studies from our industrial partners that include the problem (the features to be located) and the oracle (the realization of those features validated by the company). Then we compared the results from the five algorithms with the oracle (which is considered to be the ground truth) in terms of precision, recall, the F-measure, and the Matthews Correlation Coefficient (MCC) [8], [9]. Finally, we performed a statistical analysis of the results (following the guidelines in [10]) in order to provide quantitative evidence of the impact of the five search algorithms and to show that this impact is significant.

The EHC-MFL algorithm performed better than the other algorithms in terms of the four performance indicators. For the best case study, up to 72.41% of the model elements that were expected to be in the features being located (according to the oracle) were present when the EHC-MFL was used (up to 67.32% for EA-MFL, 61.81% for ILS-MFL, 56.61% for HC-MFL, and 38.91% for RS-MFL). In addition, only 23.53% of the elements introduced by the EHC-MFL algorithm as part of the feature realization were misplaced and should not be part of it (29.32% for EA-MFL, 39.1% for ILS-MFL, 48.01% for HC-MFL, and 61.67% for RS-MFL). It turns out that the genetic operations performed by EHC-MFL and EA-MFL together with the fitness function are able to properly traverse search spaces that are originated when locating features in

J. Font, L. Arcega, and C. Cetina are with the SVIT Research Group, Universidad San Jorge, Spain, e-mail: (jfont,larcega,ccetina)@usj.es.
J. Font, and L. Arcega are with the Department of Informatics, University of Oslo, Norway.
Ø. Haugen is with the Department of Information Technology, Østfold University College, Norway.
Manuscript received April 19, 2005; revised August 26, 2015.

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

industrial models such as the ones from our industrial partners.

The rest of the paper is structured as follows: Section II provides some background. Section III provides an overview of the work. Section IV presents the encoding that was used for the model fragments. Section V shows the usage of LSA as fitness function. Section VI presents the five search algorithms. Section VII presents the evaluation that was performed with our industrial partners and the statistical analysis of the results. Section VIII presents some related work, and then we conclude the paper.

II. Background

This section presents the Domain-Specific Language (DSL) used by our industrial partner BSH to formalize their products, the IHDSL. It will be used throughout the rest of the paper to present a running example. Then the Common Variability Language (CVL) is presented. CVL is the language used by our MFL approaches to formalize the location of the features.

A. The Induction Hobs Domain-Specific Language (IHDSL)

The newest Induction Hobs (IHs) feature full cooking surfaces, where dynamic heating areas are automatically generated and activated or deactivated depending on the shape, size, and position of the cookware placed on top. In addition, 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 made possible at the expense of increasing the software complexity.

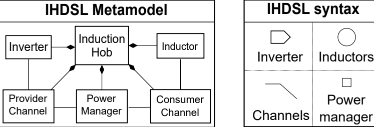
The Domain-Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references, and more than 180 properties. However, in order to increase legibility and due to intellectual property rights, we show a simplified subset of the IHDSL at the top of Fig. 1.

Inverters are in charge of transforming the input electric supply to match the specific requirements of the IH. Then, the energy is transferred to the inductors 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 run-time. The path followed by the energy through the channels is controlled by the power manager. Inductors are the elements where the energy is transformed into an electromagnetic field. Inductors can be organized into groups to heat larger cookware while sharing the user interface.

B. The Common Variability Language applied to IHs

Our MFL approaches use the Common Variability Language (CVL) [11] due to its expressiveness to properly formalize the feature realizations in terms of model fragments. CVL defines variants of a base model that conforms to Meta-Object Facility (MOF) [12] by replacing variable parts of the base model with alternative model replacements that are found in a library.

The base model is a model described by a given DSL (here, IHDSL) that serves as the base for different variants



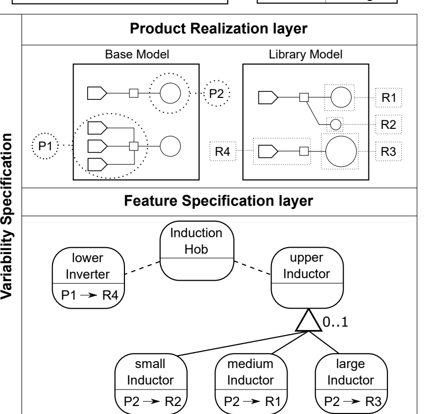


Fig. 1. CVL applied to IHDSL

defined over it. In CVL, the elements of the base model that are subject to variations are the placement fragments (hereafter *placement*). 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 that is described in the same DSL as the base model that will serve as a base to define alternatives for a placement. Each one of the alternatives for a placement is a replacement fragment (hereafter *replacement*). Similarly to placements, a replacement can be any element or set of elements that can be used as variation for a placement.

The bottom of Fig. 1 shows an example of a variability specification of an IH through CVL. In the product realization layer, two placements are defined over the IH base model (P1 and P2). Then, four replacements are defined over the IH library model (R1, R2, R3, and R4). In the feature specification layer, a Feature Model that formalizes the variability among the IH (based on the placements and replacements) is defined. For instance, P1 can only be substituted by R4 (which is optional), but P2 can be replaced by R1, R2, or R3.

III. Overview

In this paper, we present and compare five search algorithms for Model-based Feature Location (MFL). The objective of the approach is to find the model fragment (from a given set of product models) that realizes a specific feature being described by the user. This is one of the most important and

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

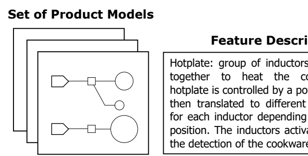


Fig. 2. Input provided to the approach

common activities performed by developers during software maintenance and evolution [3]. This activity takes up to 80% of the time spent on a system [13]. Therefore, increasing the automation level of the feature location activity will help in reducing the time spent on maintenance and evolution tasks.

The five MFL approaches are: Evolutionary Algorithm (EA-MFL); Random Search (RS-MFL); steepest Hill Climbing (HC-MFL); Iterated Local Search with Replacement (ILS-MFL) and a hybrid between Evolutionary Algorithm and Hill-Climbing (EHC-MFL). The same fitness operation is used by the five algorithms. The fitness value is calculated according to the similarity of the model fragment and a feature description that is provided by the user. The outputs of the five approaches are model fragments that may realize the feature being located.

The input for the proposed approaches consists of a set of product models and a textual description of the feature. With these, the engineer can embed their implicit knowledge of the domain into the feature location process.

Fig. 2 presents an example of input for the MFL approaches. The part on the left represents the set of product models. The part on the right shows a textual description for the feature to be located, the hotplate. This is a simplified version of a text description that has been extracted from the internal documentation used by one of our industrial partners to describe their products.

IV. Encoding

The features being located by the algorithms will be in the form of model fragments, which are a subset of the model elements present in a specific product model. Traditionally, evolutionary algorithms encode each possible solution of the problem as a string of binary values. Each position of the string has two possible values: 0 or 1. Therefore, each solution candidate of our proposed approaches will be a model fragment that is defined over one of the product models. Each model element will have a corresponding position in the binary string and the value of that position will indicate the presence or absence of that specific element in the encoded model fragment.

Fig. 3 shows two examples of our encoding of model fragments. We tag each model element of the product model with a letter. In the example of the upper part of Fig. 3, the letters A and F correspond to inverters, the letters B, D, G, and I correspond to channels, and the letters E and J correspond to inductors. The string of binary values that represents the model fragment from this product model has the positions that correspond to each letter with a value of 0 or 1. If the model element is part of the model fragment, the value will be 1; if the model element is not part of the model fragment, the value will be 0.

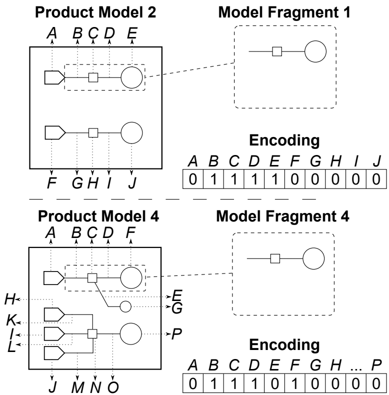


Fig. 3. Encoding of Individuals (Model Fragments)

Each model fragment representation depends on the product model that it came from. Both of the examples in Fig. 3 represent the same model fragment, but they come from different product models and thus have different representations. Throughout the rest of the paper, we will refer to each individual as a model fragment that is part of a product model.

V. Model Fragment Fitness

To assess the relevance of each model fragment in relation to the feature description provided by the user, we apply methods that are based on Information Retrieval (IR) techniques. Specifically, we apply Latent Semantic Analysis (LSA) [7] to analyze the relationships between the description of the feature provided by the user and the model fragments previously obtained. Recent studies have shown that there is not a statistically significant difference among different IR techniques [14], [15] when applied to software artifacts [16]. Hence, we chose LSA because it provides results that are similar 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 corresponds to the query). We use the *term-frequency (tf)* as the term weighting schema to construct the matrix. In other words, 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 the methods of each model fragment. The query is constructed from the terms that appear

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

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
Inductor	0	10	5	2	5	2	2	0
...	...	...	...	...	...	...	...	...

LSA Results

MF2
MF4
Q
MF1
MF3

Fitness

MF4 : 0.9
MF2 : 0.5
MF3 : 0.2
MF1 : -0.2

Fig. 4. Fitness Operation via Latent Semantic Analysis (LSA) - term-by-document co-occurrence matrix (left) and vector representation (right)

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:

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). However, 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.

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 proven the usefulness of POS-tagging techniques to remove textual noise in software documents [17]. In addition, the use of word-selection strategies [18], [19] can improve the results in feature location [20]. 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.

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 verbs tenses are unified (*using* and *used* are turned into *use*).

The union of all of the keywords extracted from the documents (model fragments) and from the query (feature description) are the terms (rows) used by our LSA fitness operation.

Fig. 4 (left) shows an example of the co-occurrence matrix for our running example. Each column is one of the model fragments. The last column is the query that is obtained from the feature description of the user. Each row is one of the terms extracted from the corpora of text, which is 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 of the terms in the model fragments.

Once the matrix is built, we normalize and decompose it into a set of vectors using a matrix factorization technique called Singular Value Decomposition (SVD) [7]. SVD projects 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. [21], which provides good results [22]. One vector that represents the latent semantics 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.

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 semantics of the query where the angle formed by the vectors A and B is θ. The fitness function can be defined as:

$$fitness(p_1) = \cos(\theta) = \frac{A \cdot B}{\|A\| \cdot \|B\|} \quad (1)$$

Fig. 4 (right) shows a three-dimensional graph of the LSA results. The graph shows the representation of each one of the vectors, which are labelled 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.

VI. Search Algorithms for FML

In this section, we present the five different search algorithms that will be evaluated. All of them will be guided using the same heuristic function, the fitness based on LSA presented in the previous section. Therefore, their differences with each other lie in how the solution space is traversed looking for the best solution.

A. Evolutionary Algorithm MFL (EA-MFL)

Our first search algorithm is an Evolutionary Algorithm that iterates a population of model fragments and evolves them using genetic operations. As output, the algorithm provides the model fragment that best realizes the feature according to the fitness function. The generation of model fragments is done by applying genetic operators that we have adapted to work with model fragments.

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

Fig. 5. Example of application of the Crossover Operation on Model Fragments

We develop our evolutionary algorithm as outlined in Algorithm 1 (available in the Appendix). The algorithm starts with a random initial population of model fragments that are obtained from the product models provided as input, (P, Lines 7-14). From this initial population, the offspring of model fragments is created by applying a selection operation (Line 18), a crossover operation (Line 19), and a mutation operation (Line 20). Then, the new offspring is added to the population (Line 21). When the creation of the new population has finished (Lines 17-22), it replaces the existing one (Line 4). This loop (Lines 2-5) is repeated until the $StopCondition$ is met. Below we describe the genetic operations used by this algorithm.

1) Selection of parents: The evolutionary algorithm uses the selection operator to select the best model fragments from the population to be used as the input for the following operations. There are different methods that can be used to perform the selection of the parents, but one of the most common choices is to follow the wheel selection mechanism [23]. In other words, each model fragment from the population has a probability of being selected that is proportional to its fitness score. Therefore, model fragments with high fitness values will have higher probabilities of being chosen as parents for the next generation.

2) Crossover: In our encoding, there are two elements that can be mapped across the different individuals: the model fragment and the referenced product model. 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, thereby preserving the basic mechanics of the crossover operation.

To achieve the above, our crossover operation is based on model comparisons. Fig. 5 shows an example of the application of the crossover operation on model fragments. First, we select the model fragment from the first parent. Then, we select the product model from the second parent. The model fragment (from first parent) is then compared with the product model (from the second parent). If the comparison finds the model fragment in the product model, the operation 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 the search space to be expanded to a different product model, i.e., both model fragments (the one from the first parent and the one from the new individual) will be the same. However, since each of them is referencing a different product model, they will mutate differently and provide different individuals in further generations.

3) Mutation: Fig. 6 shows an example of our mutation for model fragments. Each model fragment is associated to a product model, and the model fragment mutates in the context of its associated product model. In other words, the model fragment will gain or drop some elements, but the resulting model fragment will still be part of the referenced product model. The mutation possibilities of a given model fragment are driven by its associated product model. All the potential model fragments for a given product model can be achieved through the combination of additive and subtractive mutations (given that the model fragment is a subset of the product model elements); therefore there is no need for more complex mutation operations, such as the *change mutation* (a combination of a subtractive and an additive mutation).

To perform the mutation, the type of mutation that will occur (either the addition or removal of elements) is decided randomly:

Subtractive Mutation: This kind of mutation randomly removes an element from the model fragment. Since the resulting model fragment is a subset of the original model fragment and the original is part of the referenced product model, the resulting product model will always be part of the

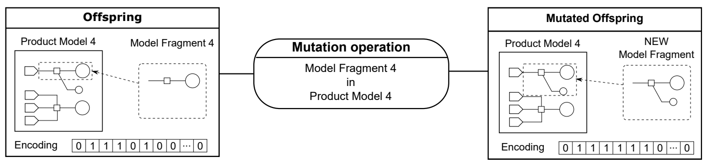


Fig. 6. Example of application of the Mutation Operation on Model Fragments

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

referenced product model.

Additive Mutation: This kind of mutation randomly adds some elements to the model fragment. The only constraint is that the resulting model fragment be part of the referenced product model. To achieve this, the boundaries of the model fragment with the rest of the product model are identified and a random element that is connected to any of those boundaries (and that is not currently part of the model fragment) is added to the resulting model fragment (See Fig. 6: the inverter and the inductor are connected to the boundaries and could be added as part of the additive mutation). By doing so, the mutated model fragment will be part of the referenced product model.

As a result, a new model fragment is created, but it still references the same product model. In other words, the individual represents other possible feature realizations (that are part of the product model) for the specific feature being located.

B. Random Search MFL (RS-MFL)

The second search algorithm that we use is a standard random search that is used as a sanity check. We want to determine if the presented algorithms perform better than mere chance (represented by this Random Search algorithm).

We used this algorithm as outlined in Algorithm 2 (available in the Appendix). The algorithm starts with a random initial model fragment, ($Best$, Line 1). Then, a new random model fragment is generated (Line 4). The search moves to a new model fragment if the fitness value is better than the current $Best$ model fragment (Lines 5-6). This loop (Lines 3-9) is repeated until the $StopCondition$ is met.

C. Hill Climbing MFL (HC-MFL)

The third search algorithm that we use is Steepest Ascent Hill Climbing with Replacement [24]. The algorithm starts with a random individual and then evaluates its neighbors (small modifications of the individual) looking for a better one (in terms of the fitness function). Those steps are repeated until the stop condition is met. The algorithm does not count with random restarts; therefore, the quality of the solution found by the algorithm greatly depends on the first individual created.

We used this algorithm as outlined in Algorithm 3 (available in the Appendix). The initial model fragment is generated randomly (S, Line 1). A neighborhood of size $NSize$ is created based on this initial model fragment by applying the *tweak* operation (Lines 7-13). This operation applies the mutation operation described in Section VI-A3 and keeps track of the best neighbor of the group (Lines 9-11). After exploring the neighborhood, if a neighbor has a better fitness value than the current state, the search moves to that model fragment (Lines 14-16). This algorithm is repeated until the $StopCondition$ is met.

D. Iterated Local Search with Random Restarts (ILS-MFL)

The fourth search algorithm that is used is the Iterated Local Search with Random Restarts. Different versions of this algorithm have been used since 1981 ([25], [26]) and can be seen as an evolution of the Hill-Climbing algorithm. The algorithm uses hill-climbing strategy to search for local optimum during a certain period of time. Then, it switches to a near hill (restart) and performs hill climbing on the new hill. The main particularity of this search algorithm is the selection of a new hill (which is performed by the $newHomeBase$ function).

We used this algorithm as outlined in Algorithm 4 (available in the Appendix). The initial model fragment is generated randomly ($Current$, Line 1). A random distribution of times is then generated (Line 2), splitting the budget into time slots for each local search. Then, solutions obtained through the $tweak$ function will be explored during the available time (Lines 7-12). When the time is over, we store the new $Best$ solution if it is better (Lines 13-15) and check if we have to switch to another $Home$. The $newHomeBase$ function will determine wether or not we need to move based on the fitness difference between the two individuals (Line 16). Then, the individual will undergo a big change ($perturb$ function), which will be achieved by applying the crossover operation described in Section VI-A2. The algorithm is repeated until the $StopCondition$ is met (Lines 5-18).

E. Hybrid between Evolutionary Algorithm and Hill-Climbing (EHC-MFL)

The fifth algorithm that we will compare is a hybrid between the hill-climbing and the evolutionary algorithms. It will take advantage of the hill-climbing capabilities of searching local optimum values, but it will also make use of the crossover operation to move to different hills (and then perform local searches again through the hill-climbing approach).

We used this algorithm as outlined in Algorithm 5 (available in the Appendix). The initial population of model fragments is generated randomly (P, Line 2). Then, hill-climb is performed for a fixed amount of iterations (Line 6) for each of the individuals in the population (Lines 5-9). If a better solution is found in this process, it is stored in the Best variable (Lines 7-9). Then, when the hill-climb part has finished, we move to a different hill through the $breedPopulation$ operation (Line 11) and repeat the process. The algorithm is repeated until the $StopCondition$ is met (Lines 3-12).

VII. EVALUATION

The goal of this paper is to provide answers to the following research questions:

RQ1: Can SBSE techniques driven by LSA be applied to locate features in product models from real industrial scenarios?

RQ2: If so, which evolutionary algorithm produces the best results in terms of solution quality?

This section presents the evaluation that was performed to answer the RQs. It includes a description of the experimental setup, a description of the case studies where we applied the five search algorithms, the results obtained, and the statistical analysis performed.

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

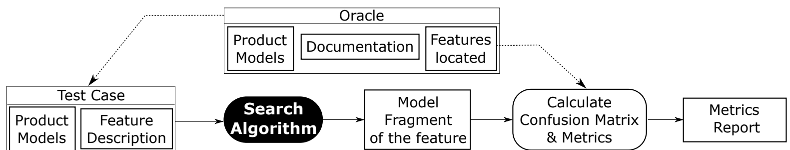


Fig. 7. Experimental Setup

A. Experimental Setup

We use the product models from the case studies as an oracle to evaluate our approach. In other words, we make use of a set of products models whose feature realizations are known beforehand and that will be considered as the ground truth, thus allowing us to compare the results provided by the five algorithms with the oracles.

Fig. 7 shows an overview of the process that was followed to evaluate the five search algorithms (EA-MFL, RS-MFL, HC-MFL, ILS-MFL, and EHC-MFL) for locating features in the two industrial case studies. The top part shows the oracle for the case study: a set of product models, the features located in those product models, and the documentation obtained from our industrial partners. The oracles will be considered the ground truth and will be used to evaluate the five algorithms in terms of MCC, precision, recall, and the F-measure.

First, we elaborated a test case for each feature present in the oracle, including the product models where the feature is present and a textual description obtained from the documentation for the feature. Then, each test case was fed as input for five different algorithms. As a result we obtained a solution in the form of a model fragment for each of the test cases for each algorithm. Those solutions were then compared to the features located from the oracle in order to obtain a confusion matrix.

A confusion matrix is a table that is often used to describe the performance of a classification model (in this case, our algorithms) on a set of test data (the resulting model fragments) for which the true values are known (from the oracle). In our case, each solution outputted by the algorithms is a model fragment that is 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, the presence or absence of each model element will be considered as a classification. The confusion matrix distinguishes between the predicted values and the real values by classifying them into four categories:

True Positive (TP): values that are predicted as true (in the solution) and are true in the real scenario (the oracle).

False Positive (FP): values that are predicted as true (in the solution) but are false in the real scenario (the oracle).

True Negative (TN): values that are predicted as false (in the solution) and are false in the real scenario (the oracle).

False Negative (FN): values that are predicted as false (in the solution) but are true in the real scenario (the oracle).

Then, some performance metrics are derived from the values in the confusion matrix. Specifically, we will create a report that includes four performance metrics (precision, recall, the F-measure, and the MCC) for each of the test cases for each search algorithm.

Precision measures the number of elements from the solution that are correct according to the ground truth (the oracle) and is defined as follows:

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

Recall measures the number of elements of the oracle that are correctly retrieved by the proposed solution and is defined as follows:

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

The F-measure corresponds to the harmonic mean of precision and recall and is defined as follows:

$$F - measure = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2TP + FP + FN} \quad (4)$$

Finally, the MCC is a correlation coefficient between the observed and predicted binary classifications that takes into account all of the observed values (TP, TN, FP, FN) and is defined as follows:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (5)$$

The presented approach has been implemented[1] within the Eclipse environment. We used the Eclipse Modeling Framework [27] to manipulate the models from our industrial partners and the CVL to manage the fragments of models. The evolutionary algorithm was built using the Watchmaker Framework for Evolutionary Computation [28], which allowed us to implement our own genetic operators. The IR techniques that were used to process the language were implemented using OpenNLP [29] for the POS-Tagger and the English (Porter2) [30] as stemming algorithm. Finally, the LSA was implemented using the Efficient Java Matrix Library (EJML [31]).

We performed the execution of the algorithms using an array of computers with processors ranging from 4 to 8 cores, clock speeds between 2.2 GHz and 4GHz, and 4-16 GB of RAM. All of them were running Windows 10 Pro N 64 bits as the hosting Operative System and the Java(TM) SE Runtime Environment (build 1.8.0_73-b02).

[1]The source code can be found here: www.jaimefont.com/flimea.html

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

TABLE I
AVERAGE TIME REQUIRED TO CONVERGE

	EA-MFL	RS-MFL	HC-MFL	ILS-MFL	EHC-MFL	Budget allocated
Time (s) $\pm$ (σ)	38.3 $\pm$ 10.3	19.9 $\pm$ 11.6	26.5 $\pm$ 9.1	33.7 $\pm$ 8.5	47.4 $\pm$ 12.4	80

1) The BSH case study: The first case study where we applied our approach was BSH (already presented in section II-A 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 each product model, on average, 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. For each of the 96 features, we created a test case that included the set of product models where that feature was used and a feature description; this information was obtained from the documentation of the features in the oracle.

For this case study, we executed 30 independent runs (as suggested by [10]) for each of the 96 test cases for each of the five algorithms (i.e., 96 (features) x 5 (algorithms) x 30 repetitions = 14400 independent runs).

2) The CAF case study: 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, 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 harvests 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 to achieve the train 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.

For instance, the high voltage connection sequence can be described using the DSL. This 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 harvest 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. For each of the 121 features, we created a test case that included the set of product models where that feature was used and a feature description.

For this case study, we executed 30 independent runs for each of the 121 test cases for each of the five algorithms (i.e., 121 (features) x 5 (algorithms) x 30 repetitions = 18150 independent runs). The sum for the two case studies presented is a total of 32550 independent runs.

3) Parameters and Budget: In general, there are two atomic performance measures for search algorithms: one regarding solution quality and one regarding algorithm speed or search effort. In this paper, we focus on the solution quality, trying to determine which algorithm provides solutions that are more similar to the one extracted from the oracle in terms of precision and recall.

Since we allocated a fixed amount of wall clock time for each of the runs of the algorithms, each algorithm has the same amount of time to traverse the search space and so the comparison is fair. First, we run some prior tests to determine the time needed to converge for each of the algorithms, and then we selected the budget time based on those tests.

Then, we use the allocated time to determine the $StopCondition$ of the algorithms (see Table VI and Algorithms in the Appendix). The allocated budget time was 80 seconds (adding a margin to ensure convergence). Although the focus of this paper is the solution quality and not the performance of each algorithm in terms of time, we include the times required by each algorithm to converge as an indication to practitioners when choosing which algorithm to use (see Table I).

As suggested by Arcuri and Fraser [32] and confirmed in Kotelysanskii and Kapfhammer [33], tuned parameters can outperform default values generally, but they are far from optimal in individual problem instances. The focus of this paper is not to tune the values to improve the performance of the algorithms when applied to a specific problem, but rather to compare the performance of the algorithms in terms of solution quality (precision and recall).

Therefore, we will use default parameter values that are commonly used in the literature [34] for the algorithms as described in the literature [32] (see Table VI available on the Appendix). The crossover operation is applied with a probability (p_c) of 0.75. The mutation operation is applied with a probability (mutation probability) of $1/n$ where n is the size of the individual (the number of bits needed to encode that product model). The population size $size$ for the algorithms based on a population will be 100 individuals. Given our crossover operation, the crossover will act over 2 parents (μ) and produce 1 offspring (λ).

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

TABLE II
MEAN VALUES AND STANDARD DEVIATIONS FOR PRECISION, RECALL, F-MEASURE AND MCC FOR EACH SEARCH ALGORITHM AND EACH CASE STUDY

	BSH				CAF			
	Precision $\pm$ (σ)	Recall $\pm$ (σ)	F-measure $\pm$ (σ)	MCC $\pm$ (σ)	Precision $\pm$ (σ)	Recall $\pm$ (σ)	F-measure $\pm$ (σ)	MCC $\pm$ (σ)
EA	70.68 $\pm$ 14.91	67.32 $\pm$ 14.32	67.34 $\pm$ 11.01	0.58 $\pm$ 0.16	68.80 $\pm$ 14.97	65.81 $\pm$ 14.35	65.91 $\pm$ 11.29	0.59 $\pm$ 0.16
RS	38.33 $\pm$ 16.61	28.12 $\pm$ 14.82	29.21 $\pm$ 13.64	0.20 $\pm$ 0.20	34.18 $\pm$ 14.39	38.91 $\pm$ 15.16	33.74 $\pm$ 12.09	0.28 $\pm$ 0.30
HC	47.07 $\pm$ 15.46	39.48 $\pm$ 13.06	40.26 $\pm$ 10.50	0.28 $\pm$ 0.17	51.99 $\pm$ 14.18	56.61 $\pm$ 16.38	51.97 $\pm$ 11.07	0.45 $\pm$ 0.15
ILS	60.90 $\pm$ 14.76	59.92 $\pm$ 14.07	58.91 $\pm$ 11.16	0.47 $\pm$ 0.19	58.86 $\pm$ 16.89	61.81 $\pm$ 16.57	58.23 $\pm$ 13.75	0.48 $\pm$ 0.22
EHC	76.47 $\pm$ 13.39	72.41 $\pm$ 13.79	72.99 $\pm$ 9.35	0.67 $\pm$ 0.13	71.75 $\pm$ 12.54	67.96 $\pm$ 15.07	68.34 $\pm$ 10.24	0.62 $\pm$ 0.13

B. Statistical Analysis

To properly compare the five algorithms, all of the data resulting from the empirical analysis was analyzed using statistical methods following the guidelines in [10].

In order to answer RQ2, we performed statistical analysis to: (1) provide formal and quantitative evidence (statistical significance) that the five search-based techniques do in fact have an impact on the comparison metrics (i.e., that the differences in the results were not obtained by mere chance); and (2) show that those differences are significant in practice (effect size).

1) Statistical significance: To enable statistical analysis, all of the algorithms should be run a large enough number of times (in an independent way) to collect information on the probability distribution for each algorithm. A statistical test should then be run to assess whether there is enough empirical evidence to claim (with a high level of confidence) that there is a difference between the two algorithms (e.g., A is better than B). In order to do this, two hypotheses, the null hypothesis H_0 and the alternative hypothesis H_1, are defined. The null hypothesis H_0 is typically defined to state that there is no difference among the algorithms, whereas the alternative hypothesis H_1 states that at least one algorithm differs from another. In such a case, a statistical test aims to verify whether the null hypothesis H_0 should be rejected.

The statistical tests provide a probability value, $p - value$. The $p - value$ receives values ranging between 0 and 1. The lower the $p - value$ of a test, the more likely that the null hypothesis is false. It is accepted by the research community that a $p - value$ under 0.05 is statistically significant [10], so the hypothesis H_0 can be considered false.

The test that we must follow depends on the properties of the data. Since our data does not follow a normal distribution in general, our analysis requires the use of non-parametric techniques. There are several tests for analyzing this kind of data; however, the Quade test is more powerful than the rest when working with real data [35]. In addition, according to Conover [36], the Quade test has shown better results than the others when the number of algorithms is low, (no more than 4 or 5 algorithms).

However, it is not possible to answer the following question with the Quade test: *Which of the algorithms gives the best performance?*. In this case, the performance of each algorithm should be individually compared against all of the other alternatives. In order to do this, we perform an additional post hoc analysis. This kind of analysis performs a pair-wise comparison among the results of each algorithm, determining whether statistically significant differences exist among the results of a specific pair of algorithms. Specifically, we apply the Holm Post Hoc procedure, as suggested by Garcia et. al. [35].

2) Effect size: When comparing algorithms with a large enough number of runs, statistically significant differences can be obtained even if they are so small as to be of no practical value [10]. Then it is important to assess if an algorithm is statistically better than another and to assess the magnitude of the improvement. *Effect size* measures are used to analyze this.

For a non-parametric effect size measure, we use Vargha and Delaney's $\hat{A}_{12}$ [37], [38]. $\hat{A}_{12}$ measures the probability that running one algorithm yields higher performance values than running another algorithm. If the two algorithms are equivalent, then $\hat{A}_{12}$ will be 0.5.

For example, $\hat{A}_{12} = 0.7$ means that we would obtain better results 70% of the times with the first of the two algorithms compared, and $\hat{A}_{12} = 0.4$ means that we would obtain better results 60% of the times with the second of the two algorithms. Thus, we have an $\hat{A}_{12}$ value for every pair of algorithms.

C. Results

Fig. 8 presents the mean values of precision, recall, the F-measure and the MCC for each feature located for the two case studies and the five algorithms. The first column of the charts (see Fig. 8a ,Fig. 8c, Fig. 8e, 8g, 8i) shows the results for the BSH case study, and the second column of the charts shows the results for the CAF case study (see Fig. 8b, 8d, 8f, 8h, 8j). The first row shows the results for EA-MFL, the second row shows the results for RS-FML, the third row shows the results for HC-MFL, the fourth row shows the results for ILS-MFL, and the fifth row shows the results for EHC-MFL. Each point in the charts represents the mean value (between the 30 independent runs) of the two performance indicators (precision on the x axis and recall on the y axis) for one of the features located for that case study and algorithm.

In Table II, we outline the results aggregated for each algorithm and case study. We also show the F-measure and the MCC performance indicators. The EHC-MFL algorithm achieves the best results for all the performance indicators, providing a precision value of 76.47% in the BSH case study and a precision value of 71.75% in the CAF case study. The Recall achieved is 72.41% for BSH and 67.96% for CAF. The combined F-measure is 72.99% for BSH and 68.34% for CAF. Finally, the MCC achieved is 0.67 for BSH and 0.62 for CAF.

The EA-MFL algorithm follows EHC-MFL and is the second best option, providing values around 10% lower than

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

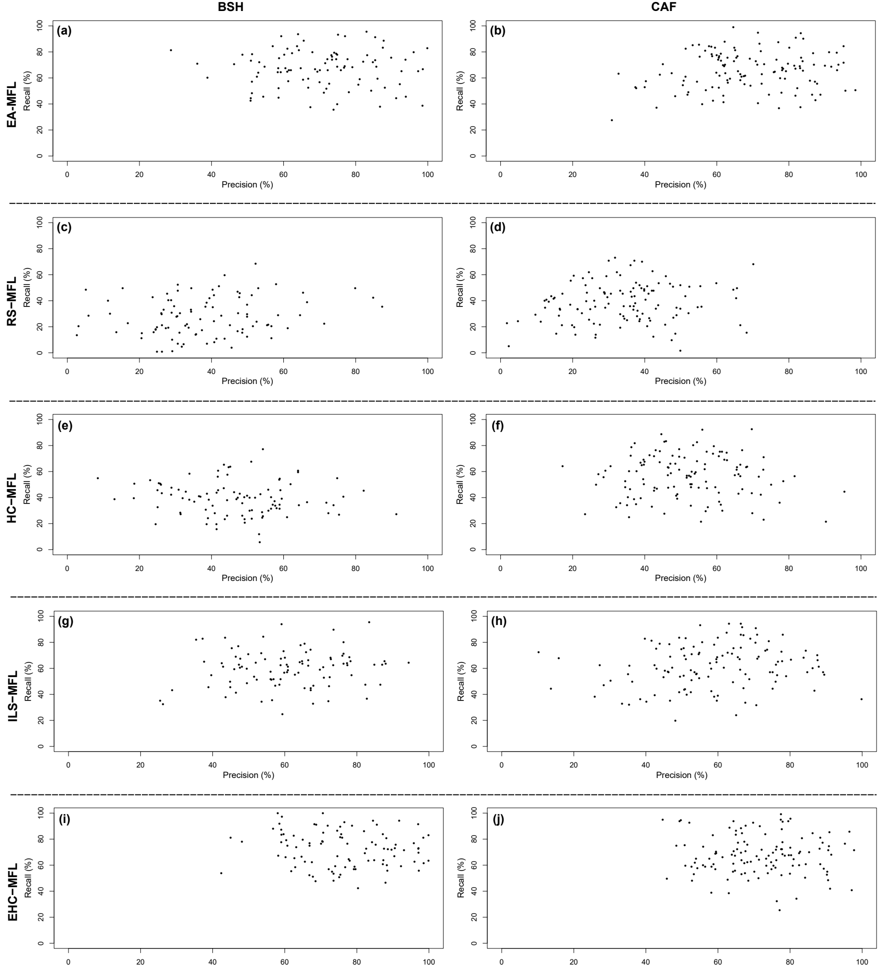


Fig. 8. Mean Precision, Recall, and the F-measure for the two case studies and the five algorithms

EHC-MFL. Then, ILS-MFL is the third option, with values around 20% lower than EHC-MFL. Finally, HC-MFL and RS-MFL are the fourth and fifth options, with values between 25% and 50% lower than the best option.

In response to RQ1, the search algorithms driven by LSA and applied to models from our industrial partners' scenarios have been capable of locating features, giving precision values of up to 76.47% and recall values of up to 72.41%.

1) Statistical significance: The $p - Values$ and statistics of this test are shown in Table III. Since the $p - Values$ shown in this table are smaller than $2x10^{-16}$ in all cases, we reject the null hypothesis. Consequently, we can state that there

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

TABLE III
QUADE TEST STATISTIC AND $p - Values$

	BSH				CAF			
	Precision	Recall	F-measure	MCC	Precision	Recall	F-measure	MCC
$p - Value$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
Statistic	72.319	100.27	132.15	84.132	70.181	50.389	99.646	57.504

TABLE IV
HOLM'S POST HOC $p - Values$

	BSH				CAF			
	Precision	Recall	F-measure	MCC	Precision	Recall	F-measure	MCC
EA vs HC	$1.0x10^{-14}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$2.7x10^{-12}$	$4.9x10^{-5}$	$1.2x10^{-15}$	$1.3x10^{-10}$
EA vs RS	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
EA vs ILS	0.0056	0.0017	$4.2x10^{-5}$	0.00014	$7.7x10^{-5}$	0.3453	$9.9x10^{-5}$	$6.0x10^{-5}$
EA vs EHC	0.0056	0.1274	0.004	0.00975	0.466	0.3453	0.11	0.060
HC vs RS	0.0020	$1.3x10^{-5}$	$4.2x10^{-5}$	0.00975	$3.9x10^{-9}$	$1.2x10^{-10}$	$4.9x10^{-12}$	$1.7x10^{-5}$
HC vs ILS	$7.7x10^{-7}$	$3.9x10^{-11}$	$6.6x10^{-13}$	$9.5x10^{-8}$	0.003	0.0081	$3.5x10^{-5}$	0.023
HC vs EHC	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.9x10^{-14}$	$1.9x10^{-7}$	$\ll 2x10^{-16}$	$3.2x10^{-16}$
RS vs ILS	$5.3x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$2.4x10^{-15}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$1.7x10^{-11}$
RS vs EHC	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$	$\ll 2x10^{-16}$
ILS vs EHC	$2.2x10^{-8}$	$5.8x10^{-6}$	$6.9x10^{-12}$	$8.0x10^{-11}$	$3.6x10^{-6}$	0.0359	$7.6x10^{-8}$	$6.3x10^{-9}$

TABLE V
$\hat{A}_{12}$ STATISTIC FOR EACH PAIR OF ALGORITHMS

	BSH				CAF			
	Precision	Recall	F-measure	MCC	Precision	Recall	F-measure	MCC
EA vs HC	0.867241	0.917806	0.962674	0.894640	0.791408	0.657674	0.812649	0.751383
EA vs RS	0.92551	0.965061	0.984809	0.886393	0.947067	0.898094	0.971553	0.898777
EA vs ILS	0.676595	0.655653	0.701931	0.685981	0.668602	0.568540	0.655727	0.637457
EA vs EHC	0.388129	0.410862	0.352431	0.336914	0.446418	0.464244	0.446896	0.447715
HC vs RS	0.663466	0.713487	0.743001	0.601129	0.811591	0.780445	0.864832	0.754457
HC vs ILS	0.254178	0.142687	0.111599	0.217665	0.366949	0.414692	0.348064	0.404003
HC vs EHC	0.072103	0.043240	0.009657	0.034722	0.151083	0.313401	0.140291	0.192268
RS vs ILS	0.153103	0.064290	0.050456	0.197917	0.130524	0.159996	0.095144	0.219589
RS vs EHC	0.041829	0.012967	0.003147	0.069119	0.027013	0.086060	0.014343	0.057646
ILS vs EHC	0.222277	0.272461	0.166721	0.168945	0.272625	0.400792	0.291374	0.308790

are differences in the algorithms for all of the performance indicators evaluated.

Table IV shows the $p-Values$ of Holm's post hoc analysis for each pair of algorithms, case study, and performance indicator. The majority of the $p-Values$ shown in this table are smaller than their corresponding significance threshold value (0.05), indicating that the differences of performance between those algorithms are significant. However, when comparing EA-MFL and EHC-MFL (fourth row), the values for some performance indicators are greater than the threshold, indicating that the differences between those algorithms could be due to the stochastic nature of the algorithms and are not significant.

2) Effect size: Table V shows the values of the effect size statistics. In general, the largest differences were obtained between the EHC-MFL and RS-MFL algorithms (where EHC-MFL achieves better precision than RS-MFL 95% of the times, better recall 98% of the times, better F-measure 99% of the times and better MCC 93% of the times). When comparing EA-MFL and EHC-MFL, the differences are not so big, and the EHC-MFL outperforms EA-MFL around 55% of the times.

In response to RQ2, the EHC-FML algorithm obtained the best performance results among the five algorithms evaluated (see Table II). The statistical analysis performed indicated that EHC-FML will outperform the rest of the algorithms in terms of the metrics analyzed (around 60% of the times when compared to EA-MFL, 78% of the times when compared to ILS, 92% of the times when compared to HC-MFL, and almost all of the times when compared to RS-MFL).

D. Threats to validity

In this section, we present some of the threats to validity. We follow the guidelines suggested by De Oliveira et. al [39] to identify those are applicable to this work.

Conclusion validity threats. We identify four threats of validity of this type. The first threat is the *not accounting for random variation*. To address this threat, we considered 30 independent runs for each feature with each algorithm. The second threat is the *lack of meaningful comparison baseline*. Because we used random search as a standard comparison baseline, this threat is addressed. The third threat is the *lack of formal hypothesis and statistical tests*. In this paper, we employed standard statistical analysis following accepted guidelines [32] to avoid this threat. The fourth threat is the *lack of good descriptive analysis*. In this work, we have used precision, recall, the F-measure and the MCC metrics to

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

analyze the confusion matrix obtained from the experiments; however, other metrics could be applied. In addition, some works argue that the use of the Vargha and Delaney A12 metrics can be miss-representative [40] and that the data should be pre-transformed before applying them. We did not find any use case for data pre-transformation that applies to our case study.

Internal validity threats. We identify two threats of validity of this type. The first threat is *the poor parameter settings*. In this paper, we used standard values for the algorithms. As suggested by Arcuri and Fraser [32], default values are good enough to measure the performance of search-based techniques in the context of testing. These values have been tested in similar algorithms for feature location [41]. In addition, the choice of the k value in the application of SVD can produce sub-optimal accuracy when using LSA for software artifacts [42]. Nevertheless, we plan to evaluate all of the parameters of our algorithms in a future work. The second threat is the *lack of real problem instances*. The evaluation of this paper was applied to two industrial case studies from two of our partners, BSH and CAF.

Construct validity threats. We identify one threat of validity of this type. The threat is the *lack of assessing the validity of cost measures*. To address this threat, we performed a fair comparison among the algorithms by generating the same number of model fragments and using the same number of fitness evaluations.

External validity threats. We identify two threats of validity of this type. The first threat is the *lack of a clear object selection strategy*, and the second threat is the *lack of evaluations for instances of growing size and complexity*. Both threats are addressed by using two industrial case studies from two of our partners, BSH and CAF. Our instances are collected from real-world problems. In addition, we have two different domains (induction hobs and trains) with different sizes and complexity.

VIII. Related Work

Some works report their industrial experiences transforming legacy products into Product Line assets in a wide range of fields [43], [44], [45]. They focus on capturing guidelines and techniques for manual transformations. In contrast, our approach performs search-based software engineering while taking advantage of the knowledge of the domain experts.

Some works focus on the location of features over models by comparing the models with each other to formalize the variability among them in the form of a Software Product Line:

Wille et al. [46] present an approach where the similarity between models is measured following an exchangeable metric, taking into account different attributes of the models. Then the approach is further refined [47] to reduce the number of comparisons needed to mine the family model.

The authors in [48] propose a generic approach to automatically compare products and locate the feature realizations in terms of a CVL model. In [49], the approach is refined to automatically formalize the feature realizations of new product models that are added to the system. A similar approach is proposed in [50] where the feature location results are validated in an industrial environment.

Martinez et al. [51] propose an extensible approach that is based on comparisons to extract the feature formalization over a family of models. In addition, they provide the means to extend the approach to locate features over any kind of asset based on comparisons. The MoVaPL approach [52] considers the identification of variability and commonality in model variants as well as the extraction of a Model-based Software Product Line (MSPL) from the features identified in these variants. MoVaPL builds on a generic representation of models, making it suitable for any MOF-based models.

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 features realizations. In contrast, our work does not rely on model similarity to locate the features; it relies on comparisons with a textual description of the target feature. Specifically, humans are involved in the search by means of the fitness function. 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. Model fragments that are obtained mechanically are less recognizable by software engineers than those obtained with the participation of software engineers [6].

Lopez-Herrejon et al. [41] evaluate three standard search-based techniques with three objective functions in order to calculate the relationships of a feature model. Their results are slightly better for hill climbing than for the evolutionary algorithm, but they are not statistically significant when the first two objective functions are applied. The authors do not address how each feature is materialized. Our work focuses on the extraction of model fragments that correspond to a feature. Therefore, both works are complimentary: [41] calculates the feature relationships of the feature specification layer, while our work locates the model fragments of the product realization layer (see Fig. 1).

Harman et al. [53] performed a survey on the topic of search-based software engineering applied to SPLs. They present an overview of recent articles that are classified according to themes such as configuration, testing, or architectural improvement. Lopez-Herrejon et al. [54] performed a preliminary systematic mapping study at the connection of search-based software engineering and SPL. They categorized the articles using a known framework for SPL development. These two surveys indicate that search-based software engineering is 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.

Font et al. [6] propose a generic approach to locate features among a family of product models based on a human-in-the-loop process. The features are located by the comparison of models and the interaction of engineers that provide their knowledge of the domain. The approach is further refined in [55] and generalized through the use of a genetic algorithm to create the model fragments. They introduce a genetic operator for mutation that can work with a single model fragment and a crossover operator that combines two different product models.

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

The results show that the use of a genetic algorithm allows the approach to provide accurate location of features in spite of inaccurate information on the part of the user.

However, since the work in [55] is designed to locate features by comparisons among the members of a family, the participation of the software engineers is limited and the resultant model fragments are less recognizable to them. In contrast, in this paper, we present five algorithms (EA-MFL, HC-MFL, RS-MFL, ILS-MFL, and EHC-MFL), which are addressed by a feature description given in natural language. Our fitness function makes use of LSA to measure the similarity with the description provided and to store the model fragments.

IX. Conclusions

This work proposes and compares five search algorithms to locate features over a family of models (MFL): Evolutionary Algorithm (EA-MFL), Random Search (RS-MFL) used as a sanity check, steepest Hill Climbing (HC-MFL), Iterated Local Search with random restarts (ILS-MFL), and a hybrid between Evolutionary and Hill Climbing (EHC-MFL). We apply Latent Semantic Analysis (LSA) as the fitness function.

In this work, we address two research questions: (RQ1) Can SBSE techniques driven by LSA be applied to locate features in product models from real industrial scenarios?; (RQ2) If so, which evolutionary algorithm produces the best results in terms of solution quality? To do so, we conducted an evaluation in BSH (the manufacturer of home appliances) and in CAF (the manufacturer of rolling stock). We report our evaluation, including the experimental setup, the results, the statistical analysis, and the threats to validity identified.

The results show that SBSE techniques can be applied to locate features in product models. Specifically, the use of genetic operations for models in combination with the EHC-MFL algorithm provided the best results in our study. This demonstrates that SBSE for feature location at the model level can be applied in real world environments.

Acknowledgments

This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R). The authors are very grateful to the anonymous reviewers for their valuable suggestions and comments to improve the quality of this paper.

References

[1] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," in *Proceedings of the 13th International Conference on Fundamental Approaches to Software Engineering*, ser. FASE'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 1–12.

[2] W. B. Langdon and M. Harman, "Optimizing existing software with genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb 2015.

[3] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: A taxonomy and survey," in *Journal of Software Maintenance and Evolution: Research and Practice*, 2011.

[4] J. Rubin and M. Chechik, "A survey of feature location techniques," in *Domain Engineering*, I. Reinhartz-Berger, A. Sturm, T. Clark, S. Cohen, and J. Bettin, Eds. Springer Berlin Heidelberg, 2013, pp. 29–58.

[5] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013.

[6] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Building software product lines from conceptualized model patterns," in *Proceedings of the 19th International Conference on Software Product Line (SPLC)*, 2015, pp. 46–55.

[7] T. K. Landauer, P. W. Foltz, and D. Laham, "An introduction to latent semantic analysis," *Discourse processes*, vol. 25, no. 2-3, pp. 259–284, 1998.

[8] C. D. Manning, P. Raghavan, H. Schütze *et al.*, *Introduction to information retrieval*. Cambridge university press, 2008, vol. 1, no. 1.

[9] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820 – 827, 2012, special Issue: Voice of the Editorial BoardSpecial Issue: Voice of the Editorial Board.

[10] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verif. Reliab.*, vol. 24, no. 3, pp. 219–250, May 2014. [Online]. Available: http://dx.doi.org/10.1002/stvr.1486

[11] Ø. Haugen, B. Moller-Pedersen, J. Oldevik, G. Olsen, and A. Svendsen, "Adding standardized variability to domain specific languages," in *Software Product Line Conference, 2008. SPLC '08. 12th International*, Sept 2008, pp. 139–148.

[12] "Meta object facility (mof) version 2.4.1," 2013, object Management Group (OMG) Specification.

[13] M. M. Lehman, J. Ramil, and G. Kahen, *A paradigm for the behavioural modelling of software processes using system dynamics*. Citeseer, 2001.

[14] M. Gethers, R. Oliveto, D. Poshyvanyk, and A. D. Lucia, "On integrating orthogonal information retrieval methods to improve traceability recovery," in *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, Sept 2011, pp. 133–142.

[15] A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Labeling source code with information retrieval methods: an empirical study," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1383–1420, 2014.

[16] A. Hindle, E. T. Barr, Z. Su, M. Gabel, and P. Devanbu, "On the naturalness of software," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 837–847.

[17] S. Gupta, S. Malik, L. Pollock, and K. Vijay-Shanker, "Part-of-speech tagging of program identifiers for improved text-based software engineering tools," in *2013 21st International Conference on Program Comprehension (ICPC)*, May 2013, pp. 3–12.

[18] G. Capobianco, A. D. Lucia, R. Oliveto, A. Panichella, and S. Panichella, "On the role of the nouns in ir-based traceability recovery," in *Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on*, May 2009, pp. 148–157.

[19] ——, "Improving ir-based traceability recovery via noun-based indexing of software artifacts," *Journal of Software: Evolution and Process*, vol. 25, no. 7, pp. 743–762, 2013.

[20] S. Zamani, S. P. Lee, R. Shokripour, and J. Anvik, "A noun-based approach to feature location using time-aware term-weighting," *Information and Software Technology*, vol. 56, no. 8, pp. 991 – 1011, 2014.

[21] A. Kuhn, S. Ducasse, and T. Gírba, "Semantic clustering: Identifying topics in source code," *Inf. Softw. Technol.*, vol. 49, no. 3, pp. 230–243, Mar. 2007.

[22] P. van der Spek, S. Klusener, and P. van de Laar, *Complementing Software Documentation*. Dordrecht: Springer Netherlands, 2011, pp. 37–51.

[23] M. Affenzeller, S. Winkler, S. Wagner, and A. Beham, *Genetic Algorithms and Genetic Programming: Modern Concepts and Practical Applications*, 1st ed. Chapman & Hall/CRC, 2009.

[24] S. Luke, *Essentials of Metaheuristics*, 2nd ed. Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[25] J. Baxter, "Local optima avoidance in depot location," *The Journal of the Operational Research Society*, vol. 32, no. 9, pp. 815–819, 1981.

[26] H. R. Lourenço, O. C. Martin, and T. Stützle, *Iterated Local Search*. Boston, MA: Springer US, 2003, pp. 320–353.

[27] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF: Eclipse Modeling Framework 2.0*, 2nd ed. Addison-Wesley Professional, 2009.

[28] D. Dyer, "The watchmaker framework for evolutionary computation," http://watchmaker.uncommons.org/, 2016, [Online; accessed 7-April-2016].

[29] "Apache opennlp: Toolkit for the processing of natural language text," https://opennlp.apache.org/, 2016, [Online; accessed 7-April-2016].

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

[30] "The english (porter2) stemming algorithm," http://snowball.tartarus.org/algorithms/english/stemmer.html, 2016, [Online; accessed 7-April-2016].

[31] "Efficient java matrix library," http://ejml.org/, [Online; accessed 7-April-2016].

[32] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[33] A. Kotelyanskii and G. M. Kapfhammer, "Parameter tuning for search-based test-data generation revisited: Support for previous results," in *2014 14th International Conference on Quality Software*, Oct 2014, pp. 79–84.

[34] A. S. Sayyad, J. Ingram, T. Menzies, and H. Ammar, "Scalable product line configuration: A straw to break the camel's back," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 465–474.

[35] S. García, A. Fernández, J. Luengo, and F. Herrera, "Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power," *Inf. Sci.*, vol. 180, no. 10, pp. 2044–2064, May 2010.

[36] W. J. Conover, *Practical Nonparametric Statistics, 3rd Edition*. Wiley, 1999.

[37] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[38] R. J. Grissom and J. J. Kim, *"Effect sizes for research: A broad practical approach*. Mahwah, NJ: Earlbaum, 2005.

[39] M. de Oliveira Barros and A. C. Dias-Neto, "0006/2011-threats to validity in search-based software engineering empirical studies," *RelaTeDIA*, vol. 5, no. 1, 2011.

[40] G. Neumann, M. Harman, and S. Poulding, *Transformed Vargha-Delaney Effect Size*. Cham: Springer International Publishing, 2015, pp. 318–324. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-22183-0_29

[41] R. E. Lopez-Herrejon, L. Linsbauer, J. A. Galindo, J. A. Parejo, D. Benavides, S. Segura, and A. Egyed, "An assessment of search-based techniques for reverse engineering feature models," *Journal of Systems and Software*, vol. 103, pp. 353 – 369, 2015.

[42] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia, "Parameterizing and assembling ir-based solutions for se tasks using genetic algorithms," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, vol. 1, March 2016, pp. 314–325.

[43] K. Kim, H. Kim, and W. Kim, "Building software product line from the legacy systems "experience in the digital audio and video domain"," in *Software Product Line Conference, 2007. SPLC 2007. 11th International*, Sept 2007, pp. 171–180.

[44] R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi, "Refactoring a legacy component for reuse in a software product line: A case study: Practice articles," *J. Softw. Maint. Evol.*, vol. 18, no. 2, pp. 109–132, Mar. 2006.

[45] H. Lee, H. Choi, K. Kang, D. Kim, and Z. Lee, "Experience report on using a domain model-based extractive approach to software product line asset development," in *Formal Foundations of Reuse and Domain Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5791, pp. 137–149.

[46] D. Wille, S. Holthusen, S. Schulze, and I. Schaefer, "Interface variability in family model mining," in *Proceedings of the 17th International Software Product Line Conference: Co-located Workshops*, 2013, pp. 44–51.

[47] S. Holthusen, D. Wille, C. Legat, S. Beddig, I. Schaefer, and B. Vogel-Heuser, "Family model mining for function block diagrams in automation software," in *Proceedings of the 18th International Software Product Line Conference: Volume 2*, 2014, pp. 36–43.

[48] X. Zhang, Ø. Haugen, and B. Moller-Pedersen, "Model comparison to synthesize a model-driven software product line," in *Proceedings of the 2011 15th International Software Product Line Conference (SPLC)*, 2011, pp. 90–99.

[49] X. Zhang, Ø. Haugen, and B. Møller-Pedersen, "Augmenting product lines," in *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, vol. 1, Dec 2012, pp. 766–771.

[50] J. Font, M. Ballarín, Ø. Haugen, and C. 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)*, 2015, pp. 411–418.

[51] J. Martinez, T. Ziadi, T. F. Bissyandé, J. Klein, and Y. L. Traon, "Bottom-up adoption of software product lines: a generic and extensible approach," in *Proceedings of the 19th International Conference on Software Product Line (SPLC)*, 2015, pp. 101–110.

[52] J. Martinez, T. Ziadi, T. F. Bissyand, J. Klein, and Y. l. Traon, "Automating the extraction of model-based software product lines from model variants (t)," in *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, Nov 2015, pp. 396–406.

[53] M. Harman, Y. Jia, J. Krinke, W. B. Langdon, J. Petke, and Y. 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*, ser. SPLC '14. New York, NY, USA: ACM, 2014, pp. 5–18.

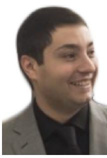
[54] R. E. Lopez-Herrejon, J. Ferrer, F. Chicano, L. Linsbauer, A. Egyed, and E. Alba, "A hitchhiker's guide to search-based software engineering for software product lines," *CoRR*, vol. abs/1406.2823, 2014.

[55] J. Font, L. Arcega, Ø. Haugen, and C. Cetina, "Feature location in model-based software product lines through a genetic algorithm," in *15th International Conference on Software Reuse*, ser. ICSR 2016, Limassol, Cyprus, Jun 2016.

Jaime Font is a PhD student in computer science at the University of Oslo and a researcher in the SVIT Research Group at San Jorge University. His research interests include reverse engineering, evoluationary computation, and variability modeling. He received an MSc in Advanced Software Technologies from San Jorge University. Contact him at jfont@usj.es

Lorena Arcega is a PhD student in computer science at the University of Oslo and a researcher in the SVIT Research Group at San Jorge University. Her research interests are software evolution, variability modeling, and models at run-time. She received an MSc in Advanced Software Technologies from San Jorge University. Contact her at larcega@usj.es

Øystein Haugen is a professor at Østfold University College and senior researcher at SINTEF ICT. His research focuses on cyber-physical systems, variability modeling and software product lines, object oriented languages and methods, software engineering, and practical formal methods. Contact him at oystein.haugen@hiof.no



Carlos Cetina is an associate professor at San Jorge University and the head of the SVIT Research Group. His research focuses on software product lines, variability modeling, and model-driven development. Cetina received a PhD in computer science from the Universitat Politècnica de València. Contact him at ccetina@usj.es

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TEVC.2017.2751100, IEEE Transactions on Evolutionary Computation

APPENDIX
ALGORITHMS PSEUDOCODES AND PARAMETERS

TABLE VI
PARAMETERS FOR THE ALGORITHMS

Parameter	Description	Value
Size	Size of the population	100
StopCondition	Budget allocated to run the algorithm (s)	80
μ	Number of parents for the crossover	2
λ	Number of offspring from μ parents	1
p_c	Probability of crossover	0.75
p_m	Probability of mutation, where n is the length of the chromosome being mutated	1/n

Algorithm 1 Evolutionary Algorithm

```
1:  P ← initPopulation(inputData, size)
2:  while (!StopCondition) do
3:      Bestevaluatefitness(P)
4:      P ← breedPopulation(P)
5:  end while
6:  return Best

7:  function initPopulation(inputData,size)
8:      P ← []                                  ▷ Initial population empty
9:      for i = 1 to size do
10:         F ← randomFragment(inputData)
11:         P ← P + F                           ▷ Add the new individual
12:     end for
13:     return P
14: end function

15: function breedPopulation(P,size)
16:     P_0 ← []                                ▷ empty population
17:     for i = 1 to size do
18:         parents ← selectionParents(P)
19:         offspring ← crossover(parents, p_c)
20:         offspring ← mutation(offspring, p_m)
21:         P_0 ← P_0 + offspring               ▷ Add the new offspring
22:     end for
23:     return P_0
24: end function
```

Algorithm 2 Random Search

```
1:  Best ← randomFragment(inputData)
2:  I ← 0
3:  while (!StopCondition)) do
4:      S ← randomFragment(inputData)
5:      if (fitness(S) > fitness(Best)) then
6:          Best ← S
7:      end if
8:      I ← I + 1
9:  end while
10: return Best
```

Algorithm 3 Steepest Ascent Hill Climbing with Replacement

```
1:  S ← randomFragment(inputData)
2:  NSize ← number of neighbors desired
3:  I ← 0
4:  Best ← S
5:  while (!StopCondition) do
6:      X ← 0
7:      while X < NSize do
8:          S' ← tweak(Best)
9:          if (fitness(S') > fitness(S)) then
10:             S ← S'
11:         end if
12:         X ← X + 1
13:     end while
14:     if (fitness(S) > fitness(Best)) then
15:         Best ← S
16:     end if
17:     I ← I + 1
18: end while
19: return Best
```

Algorithm 4 Iterated Local Search (ILS) with Random Restarts

```
1:  Current ← randomFragment(inputData)
2:  Times ← distribution of time intervals
3:  Home ← Current
4:  Best ← Current
5:  while (!StopCondition) do
6:      time ← random time chosen from Times
7:      while (!StopCondition && time ≠ 0) do
8:          Aux ← tweak(Current)
9:          if (fitness(Aux) > fitness(Current) then
10:             Current ← Aux
11:         end if
12:     end while
13:     if (fitness(Current) > fitness(Best) then
14:         Best ← Current
15:     end if
16:     Home ← newHomeBase(Home, Current)
17:     Current ← perturb(Home)
18: end while
19: return Best
```

Algorithm 5 Hybrid between Evolutionary and Hill-Climbing

```
1:  HCIter ← number of iterations to Hill-Climb
2:  P ← initPopulation(InputDataSize)
3:  while (!StopCondition) do
4:      fitness(P)
5:      for P_i in P do
6:          P_i ← Hill − Climb(P_i)forHCIter
7:          if (fitness(P_i) > fitness(Best)) then
8:              Best ← P_i
9:          end if
10:     end for
11:     P ← breedPopulation(P)
12: end while
13: return Best
```

1089-778X (c) 2017 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

12

EVOLUTION OF MODEL FRAGMENTS

Contents

12.1 GPCE'15 Paper 228
12.2 COMLAN'17 Paper 239

12.1 GPCE'15 Paper

Title:	Addressing Metamodel Revisions in Model-based Software Product Lines.
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Proceedings:	Proceedings of the 2015 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '15)
Location:	Pittsburgh, PA, USA - October 26 - 27, 2015
Publisher:	ACM New York, NY, USA
Pages:	161-170
DOI:	`http://doi.acm.org/10.1145/2814204.2814214`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work. He was also responsible for the oral presentation of the work which took place during the conference.

Addressing Metamodel Revisions in Model-Based Software Product Lines *

Jaime Font[1,2] Lorena Arcega[1,2] Øystein Haugen[2,3] Carlos Cetina[1]

San Jorge University, SVIT Research Group, Spain[1]
University of Oslo, Department of Informatics, Norway[2]
Østfold University College, Department of Information Technology, Norway[3]
jfont@usj.es larcega@usj.es oysteinh@ifi.uio.no ccetina@usj.es

Abstract

Metamodels evolve over time, which can break the conformance between the models and the metamodel. Model migration strategies aim to co-evolve models and metamodels together, but their application is not fully automatizable and is thus cumbersome and error prone. We introduce the Variable MetaModel (VMM) strategy to address the evolution of the reusable model assets of a model-based Software Product Line. The VMM strategy applies variability modeling ideas to express the evolution of the metamodel in terms of commonalities and variabilities. When the metamodel evolves, the models continue to conform to the VMM, avoiding the need for migration. We have applied both the traditional migration strategy and the VMM strategy to a retrospective case study that includes 13 years of evolution of our industrial partner, an induction hobs manufacturer. The comparison between the two strategies shows better results for the VMM strategy in terms of model indirection, automation, and trust leak.

Categories and Subject Descriptors D.2.13 [*Software Engineering*]: Reusable Software—Reuse Models

Keywords Model-based Software Product Lines, Variability Modeling, Model and Metamodel Co-evolution

1. Introduction

Model-Driven Development aims to shift the focus of software development from coding to modeling. Metamodels are used to formalize a set of concepts and the relationships among those concepts. A model conforms to a metamodel if it is expressed by the terms that are encoded in the metamodel.

Model-based Software Product Lines enable a planned reuse of software components in products that are within the same scope [14]. Commonalities and variabilities among the products are formalized into a set of models (and metamodels) using a variability language; either feature models [2] (the de facto standard for variability modeling) or Common Variability Language (CVL) [8], (recommended for adoption as a standard by the Architectural Board of the Object Management Group). Although the details are different, all share the idea of modeling commonalities and variabilities among the different products.

Similar to other software components, metamodels evolve over time [7]; however, changes that are introduced in the evolved metamodel can invalidate the models that conform to the previous version of the metamodel. To address this issue, migration strategies [3, 10, 12, 15, 17] propose co-evolving models and metamodels together to maintain consistency.

However, even though migration strategies have proven to be successful in model-based approaches, their application is not fully automatizable and can be cumbersome and error prone in large systems. Evolution is particularly critical for a successful adoption of model-based Software Product Lines (SPLs) [16].

We believe that the ideas of variability modeling can also be applied at the metamodel level to address the evolution of SPLs and at the same time avoid the issues involved with migration strategies. Our contribution is the Variable MetaModel (VMM) strategy, which enables the evolution of the metamodel without breaking conformance. In VMM, each metamodel evolution is expressed in terms of metamodel commonalities and variabilities. As a result, already existing models continue to conform to the created VMM, avoiding the need for migration and its related issues.

First, we build a retrospective case study of the evolution undergone by our industrial partner (BSH) over the last 13 years regarding the evolution of their models and metamodels. BSH is the leading manufacturer of home appliances in Europe and its induction department produces induction hobs (under the brands of Bosch and Siemens) following an MDD approach [9].

We then apply a migration strategy to the case study, migrating the models whenever a metamodel change that breaks the conformance between models and metamodels arises. Migration strategies involve the following three issues: 1) model migration introduces indirection to the models; 2) some of the steps of the migration strategy need human assistance; 3) the trust gained by models (over years of use) is lost when they are migrated.

Finally, we also apply the VMM strategy to the retrospective case study and compare both strategies (VMM and migration). The comparisons shows that the VMM strategy achieves better results than migration in terms of the three issues related to migration: 1) VMM avoids the need for migration (and the indirection introduced); 2) some of the steps of the migration strategy require human assistance while in the VMM strategy those steps are auto-

* Project carried out by the Research Group SVIT T92 (Consolidated Group for Applied Research). Partially supported by the Aragonian Government and the European Social Fund "Building Europe from Aragon".

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

GPCE'15, October 26–27, 2015, Pittsburgh, PA, USA
© 2015 ACM. 978-1-4503-3687-1/15/10...$15.00
http://dx.doi.org/10.1145/2814204.2814214

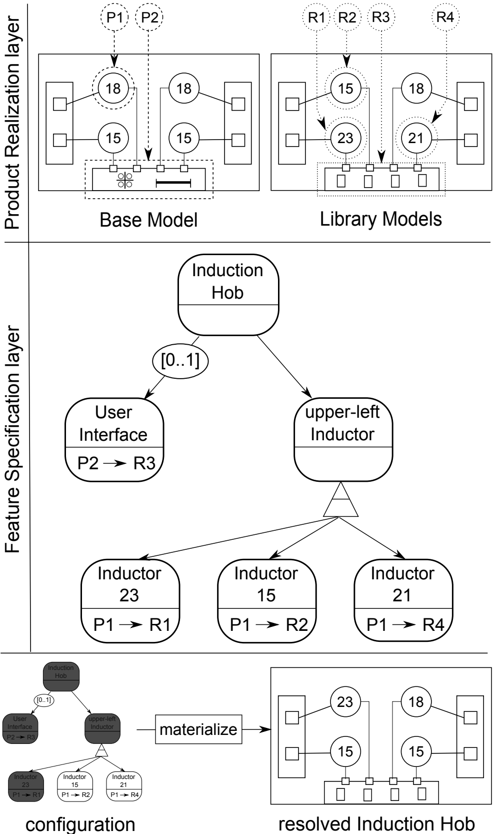


Figure 1. CVL applied to IH-DSL

matic; 3) the trust gained by models remains the same in the VMM strategy (since the model does not need to change).

2. The Induction Hobs Domain and CVL

Induction hobs use electromagnetism to generate heat that is transferred to the cookware. Traditionally, stoves feature four rounded areas that become hot when turned on. Therefore, the first Induction Hobs (IHs) created provided similar capabilities. However, the induction hob domain is constantly evolving due to the possibilities provided by the induction phenomena and the electronic components that are present.

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 real-time measurements of the consumption of the IH.

The Domain Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references among the meta-classes and more than 180 properties. However, in order to gain legibility and due to intellectual property rights concerns, in this paper we use a simplified subset of the IHDSL (top-left corner of Figure 2).

The bottom-right corner of Figure 1 shows an Induction Hob with the graphical representation of the IHDSL. It is composed of two power modules (vertical rectangles on both sides of the IH). Each of them holds two inverters (squares), which are in charge of providing the electrical supply required to generate the magnetic field. Inverters are connected to the inductors (circles), which are the elements where the magnetic field is generated. The number inside each inductor represents the diameter of the inductor. The line that connects inverters and inductors represents the channel, which transfers energy from the inverter to the inductor. The user interface of an IH has controllers to configure the power level of each inductor (the horizontal rectangle at the bottom of the IH).

The Common Variability Language (CVL) is a DSL for modeling variability in any model of any DSL based on Meta-Object Facility (MOF), which is an OMG specification to define a universal metamodel for describing modeling languages. CVL defines variants of the base model by replacing parts of the base model with model replacements that are found in a library.

The variability specification in CVL is divided across two different layers: the feature specification layer (where variability is specified following a feature model syntax [2]); and the product realization layer (where the 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) that serves as the base for different variants defined over it. The top-left corner of Figure 1 shows the Base Model, which is a complete IH model with four inductors and a slider user interface.

The elements of the base model subject to variations are the **placement fragments** (hereinafter placements). In the top-left corner of Figure 1 there are two placements defined over the Base Model: P1, which is defined over the top-left inductor; and P2, which is defined over the user interface.

To define alternatives for a placement, we use a replacement library, which is a model described in the same DSL as the Base Model. Each alternative for a placement in the Base Model is a **replacement fragment** (hereinafter replacement). In the top-right corner of Figure 1 there are four replacements that are defined over the library model: three inductor replacements (R1, R2 and R4) and a user interface replacement (R3).

CVL defines variants of the base model by means of **fragment substitutions** (i.e. the substitution of placements by replacements). The middle part of Figure 1 shows the Feature specification layer with the substitutions. P2 can be substituted by R3 (this substitution is optional) and P1 can be substituted by R1, R2, or R4. The materialization operation of CVL executes the substitutions that are selected and produces a variation of the base model where the placements have been substituted by the replacements selected. The bottom part of Figure 1 shows an example of configuration (over the feature specification) and materialization where P1 has been substituted by R1 and P2 has been substituted by R3.

For simplicity throughout the rest of the paper, we will show the placements superimposed on the base model, even though they are defined in a separate model. Likewise, the replacements defined in the replacements library will be shown separately from the rest of the model where they are defined.

3. SPL Evolution Formalized by CVL

This section presents the retrospective case study that was extracted from the evolution of our industrial partner's 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

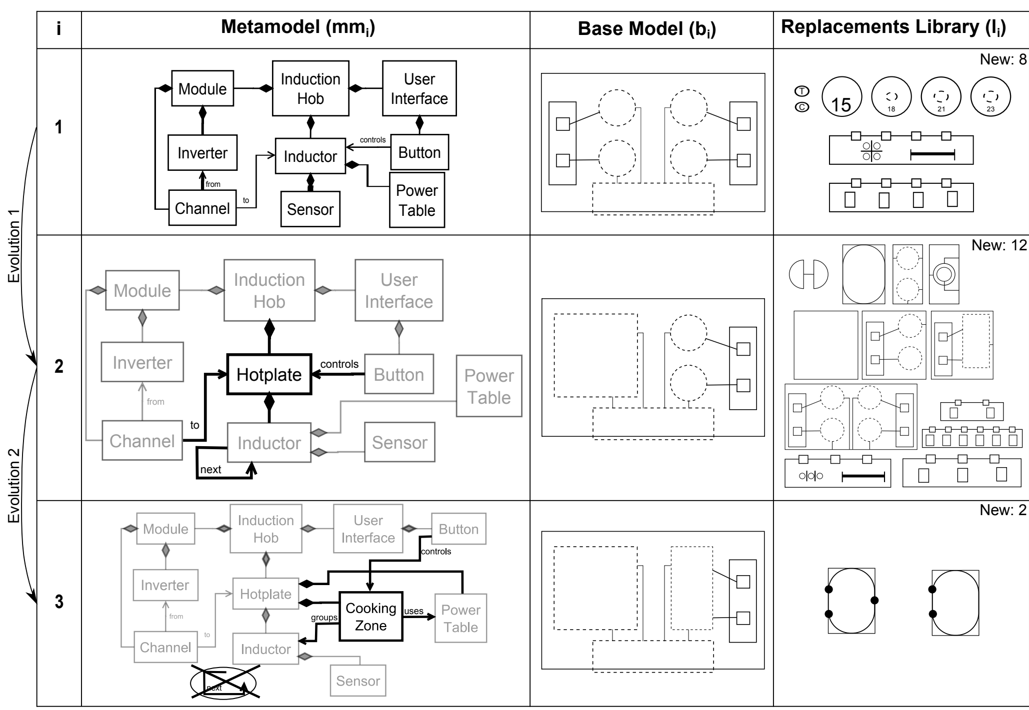


Figure 2. Model generations of the CVLSPL

due to intellectual property rights concerns, 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). Let M be the set of all models. If m_i is in M and mm_i is in MM then $C(m_i, mm_i)$ means that m_i conforms to mm_i. Let $CVLSPL$ be the set of all CVL-based product lines. One such product line, $cvlspl_i$, is denoted as follows:

$$\begin{aligned} CVLSPL =& \ MM \times M \times M \\ cvlspl_i =& <mm_i \ , \ b_i \ , \ l_i> \end{aligned} \tag{1}$$

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 library 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. That is, 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.

We perform a $CVLSPL$ evolution (shift from one $cvlspl_i$ generation to the next generation, $cvlspl_{i+1}$) whenever there is a *breaking and unresolvable change* (hereinafter *breaking change*) [3] in the metamodel. Breaking changes break the conformance of models and metamodel in a way that cannot be resolved by automatic means [3] (e.g., the addition of a mandatory meta-class or a restriction in the multiplicities). There are other meta-model changes that do not break the conformance of models and metamodel (e.g., the addition of an optional class) or metamodel changes that can be resolved automatically by existing approaches [3, 10, 12, 15, 17] (e.g., eliminating a property). However, in this work we will focus on the evolution triggered by breaking changes.

Figure 2 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; the third row shows $cvlspl_3$, which includes the concept of cooking zone. The figure shows the breaking changes that were overcome by our industrial partner, such as the addition or removal of meta-elements.

Evolution 1 (from $cvlspl_1$ to $cvlspl_2$) is triggered by a new concept called Hotplate (see the first and second rows of Figure 2). 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. Therefore, we need to control the whole Hotplate (two inductors) with just one user interface controller, so the controller will now act over hotplates instead of inductors. This is reflected in the metamodel mm_2 (see the second row, first column).

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 auxil-

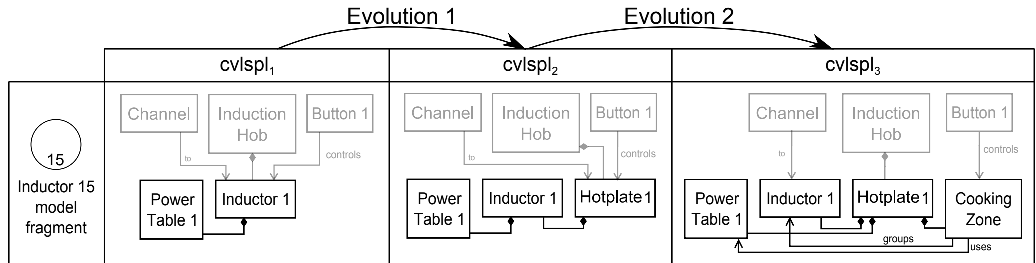


Figure 3. Migration Strategy steps

iary) 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 2). 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. As 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).

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.

4. Motivation of the Approach

The evolution presented in Section 3 needs to be properly supported by the metamodels that are used by our industrial partner to formalize their SPL. Some of the changes presented can be addressed without breaking the conformance between the models and the metamodel, such as the creation of new model fragments or the addition of new optional elements to the metamodel. However, when we perform a breaking change to the metamodel (e.g., the hotplate and cooking zone concepts), the conformance between the models and the metamodel is lost.

Traditional migration strategies [3, 10, 12, 15, 17] propose migrating all of the models to conform to the new version of the metamodel. The migration of the SPL can be achieved by the following steps: Given a metamodel change, 1) the metamodel is upgraded to a new version introducing the new concept; 2) a model-to-model (M2M) transformation that migrates models from one version to another is created (by means of one of the existing approaches in the literature: manual specification [15], operator-based [12, 17] or metamodel matching [3, 10]); 3) existing replacements are migrated (by executing the M2M transformation obtained from step 2) to conform to the new generation of the metamodel; 4) if some common parts that are present in the Base Model have become variable, the user creates placements over the base model and extracts the model fragments as replacements; 5) new replacements are created to instantiate the new concepts that have been incorporated into the metamodel.

Let E_{mig} be the operation used to evolve a $cvlspl_i$ from a given generation i to the next generation $(i + 1)$ following the migration strategy. The operation is defined as follows:

$$\begin{array}{ll} E_{mig}: & CVLSPL \longrightarrow CVLSPL \\ E_{mig}\,(< mm_i, b_i, l_i >) & = \; < mm_{i+1}, b_{i+1}, l_{i+1} > \\ & \text{where} \quad M2M(L_i) = L_{i+1} \end{array} \tag{2}$$

Figure 3 presents the evolution of a model fragment following a migration strategy. Each column shows the same fragment (Inductor 15) for each of the $cvlspl_i$ generations. 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.

Specifically, the migration of models from our industrial partner involves three related issues: **indirection**, where 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 make 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. 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 domain of our industrial partner is constantly evolving, but the original elements are still present in new IHs. New kinds 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.

5. The Variable MetaModel Strategy Applied to the Case Study

In order to avoid the need for migration when a new generation is created, we want to build a new metamodel that supports both generations: the Variable MetaModel (VMM). For instance, mod-

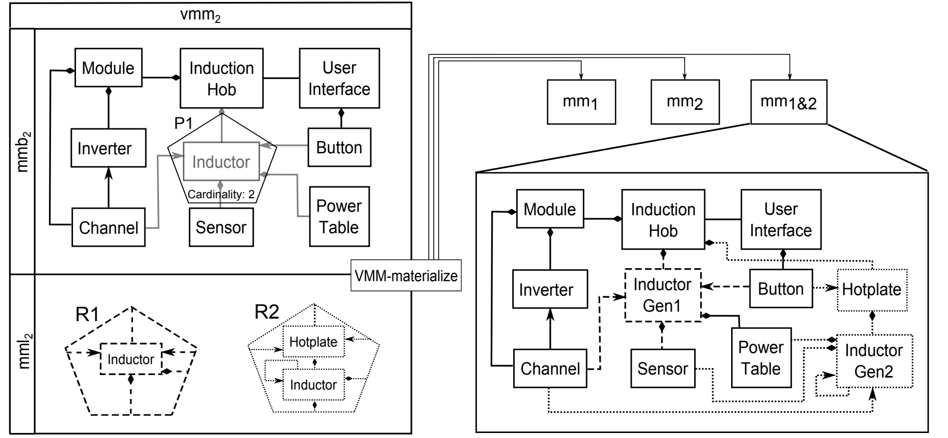


Figure 4. VMM and VMM-materialize

els 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.

The VMM is the result of applying CVL at the metamodel level; 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:

$$\begin{array}{l} VMM = \; MM \; \times MM \\ vmm_i = < mmb_i \; , mml_i > \end{array} \quad (3)$$

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

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:

$$\begin{array}{l} VMMmat: \quad VMM \quad \times \mathcal{P}(G) \longrightarrow MM \\ VMMmat(< mmb_i, mml_i >, \;\; g) \;\; = mm_g \\ \qquad\qquad\qquad\qquad where \; g \neq \emptyset \end{array} \quad (4)$$

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 4 (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. In addition, the bottom-left corner of Figure 4 shows the replacements library (mml_2), which contains two different replacements: R1 (in dashed lines) defined over the $cvlspl_1$ metamodel; R2 (in dotted lines) defined over the $cvlspl_2$ metamodel.

Figure 4 (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 produced models 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; 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, a placement can be substituted more than once. 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, new references that are analogous to the existing ones need to be created. For instance, the aggregation of Inductors reference in mm_i is duplicated into an aggregation of Inductor Gen1 (in dashed lines) and aggregation of Hotplate (in dotted lines) in the $mm_{1\&2}$.

The $mm_{1\&2}$ metamodels 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 $mm_{1\&2}$.

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 enable the creation of new replacements.

5.1 Steps of the VMM Strategy

The evolution of a $cvlspl_i$ following the VMM-strategy is denoted as follows:

$$\begin{array}{l} E_{VMM}: \quad CVLSPL \quad \longrightarrow \qquad VMM \\ \qquad < mm_i, b_i, l_i > \longrightarrow < mmb_{i+1}, mml_{i+1} > \end{array} \quad (5)$$

$VMMmat$ is used with the generated vmm_i to retrieve the different CVLSPL generations needed by the company.

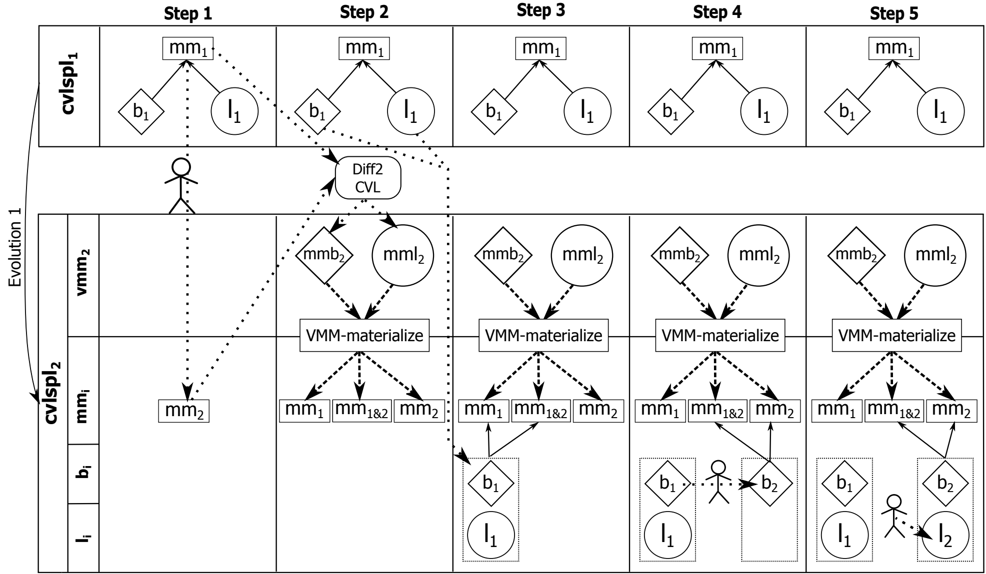


Figure 5. Steps of the VMM Strategy

Figure 5 shows the method to perform the Evolution 1 from $cvlspl_1$ to $cvlspl_2$. Each of the columns of the tables represent one step in the application of the VMM strategy. The top part shows the $cvlspl_1$ with its base model b_1 (depicted as a diamond), its metamodel mm_1 (depicted as a rectangle), and its fragment library l_i (depicted as a circle). The bottom part shows the $cvlspl_2$, the first row shows vmm_2, and the second row shows the $cvlspl_2$.

Step 1 shows the edition of the metamodel by the user. The mm_1 metamodel is edited to include the new concepts of the next generation (Hotplate), resulting in the mm_2 metamodel.

Step 2 shows our Diff2CVL operation, which is used to spot the differences between the two metamodels and to describe them in terms of a base model and replacements. Diff2CVL is built upon EMFCompare[1]. This is an eclipse plugin that provides generic support for any kind of metamodel in order to compare and merge models. The common parts of the two metamodels (mm_1 and mm_2) are included in the mmb_2 and placements are created over it for the differences between mm_1 and mm_2. Furthermore, replacements that contain these differences are created and included in the mml_2. The $VMMmat$ operation can be applied to vmm_i to obtain mm_1, mm_2, and $mm_{1\&2}$.

In Step 3, the l_1 and b_1 from $cvlspl_1$ are copied without any modification to be used in $cvlspl_2$. Both conform to the materialized mm_1, and they also conform to the materialized $mm_{1\&2}$.

In Step 4, some common parts of the base model (b_1) may become variable because of the new concepts introduced in Generation 2. In that case, the engineer edits the base model b_1 (that has been copied in the previous step) from the $cvlspl_2$ to extract the variable parts as replacements.

In Step 5, the engineer creates new replacements that instantiate the new concepts of this generation (Hotplate) and includes them in l_2. These new replacements conform to mm_2, and they also conform to $mm_{1\&2}$.

Following the above steps, we can evolve the SPL from one generation to the next, while avoiding the need for migrating existing fragments. Then, when the engineer wants to create new replacements, the engineer will be able to use the metamodel of just one generation and not the $mm_{1\&2}$. As a result, the engineer can create replacements for the most recent generation (using mm_2) to instantiate the new concepts of that generation. In contrast, the engineer can use the previous generation metamodel (mm_1) to create replacements that do not exercise the expressiveness provided by the new generation, avoiding the overcharge of the model (as the case of the motivating example, see Section 4). When materializing an IH model containing replacements from both generations (l_1 and l_2), the resulting IH model will conform to $mm_{1\&2}$.

In addition, the recursion capabilities of CVL enable us to create placements inside a replacement and hence apply the VMM strategy to further generations. That is, when creating the next generation, the step 2 of the process could end up in the creation of a new replacement that includes previously defined placements (if the replacement is not common for both metamodels).

5.2 Resulting Models after Applying VMM Strategy

Figure 6 shows an overview of our industrial partner's $CVLSPL$ models (rows) after applying the VMM strategy to manage Evolution 1 and Evolution 2 (columns).

In **Evolution 1** a new concept (hotplate) is introduced (see the first and second columns). This concept affects the inductor, which is now aggregated by the hotplate; therefore, we apply the method explained above to perform Evolution 1. Diff2CVL produces a base model (mmb_2) that contains a placement (P1) with cardinality 2 (i.e., it can be replaced up to two times). Diff2CVL also produces mml_2, which contains two replacements: R1 (which holds the

[1] https://www.eclipse.org/emf/compare/

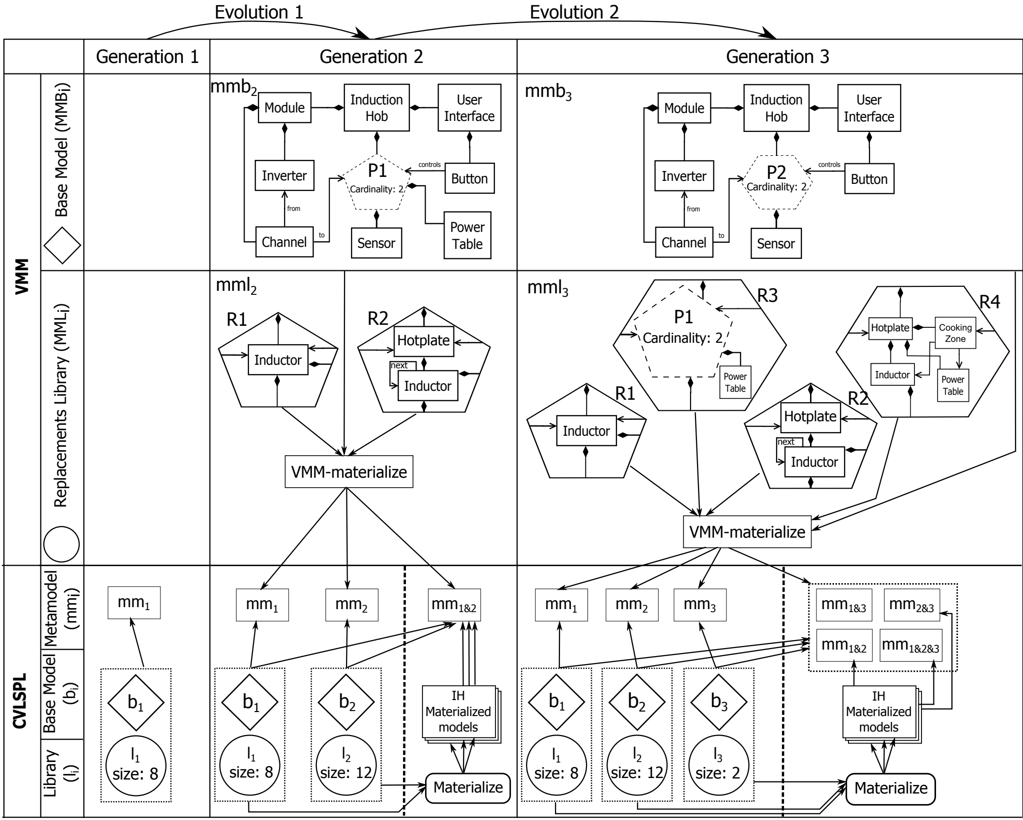


Figure 6. IHDSL Metamodel level for each generation of the CVLSPL

particularities of Gen1) and R2 (which holds the particularities of Gen2). $VMMmat$ operation can be applied to those models to produce three different metamodels: 1) the substitution of P1 by R1 produces mm_1; 2) the substitution of P1 by R2 produces mm_2; 3) the substitution of P1 twice, by R1 and by R2 produces $mm_{1\&2}$.

When creating new fragments, the engineer must stick to only one generation in order to create a valid fragment. In other words, fragments must conform to a specific metamodel generation, either mm_1 or mm_2. As a result the engineer can create replacements only using concepts from mm_1, avoiding the indirection introduced by the migration strategy (see Section 4).

When materializing an IH model that contains replacements from both generations (l_1 and l_2), the resulting IH model conforms to $mm_{1\&2}$. Overall, vmm_2 enables the materialization of IH models with replacements from both generations (l_1 and l_2), while at the same time allowing the creation of fragments pertaining to one generation (either conforming to mm_1 or to mm_2).

In **Evolution 2** a new *breaking change* that introduces the concept of cooking zones occurs (see the second and third columns). Similarly to Evolution 1, we apply the method to perform Evolution 2 (from Generation 2 to Generation 3).

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the placements applied to the metamodel level have proven to provide enough expressiveness to overcome all of the evolution situations of our industrial partner over 13 years. In addition, the VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution.

5.3 Derivation of SPL Products after Applying VMM

The VMM strategy has been tooled within the Eclipse environment and integrated into our industrial partner's SPL. The resulting tool is used by our industrial partner (BSH, the leading manufacturer of home appliances in Europe) to generate the firmware of their Induction Hobs (sold under the brands of Bosh and Siemens). An example of the resulting tool in action can be seen here [2]. This section present an example of using the SPL evolved with the VMM strategy: an engineer of our industrial partner deriving a new product.

[2] http://www.carloscetina.com/variablemetamodel.htm

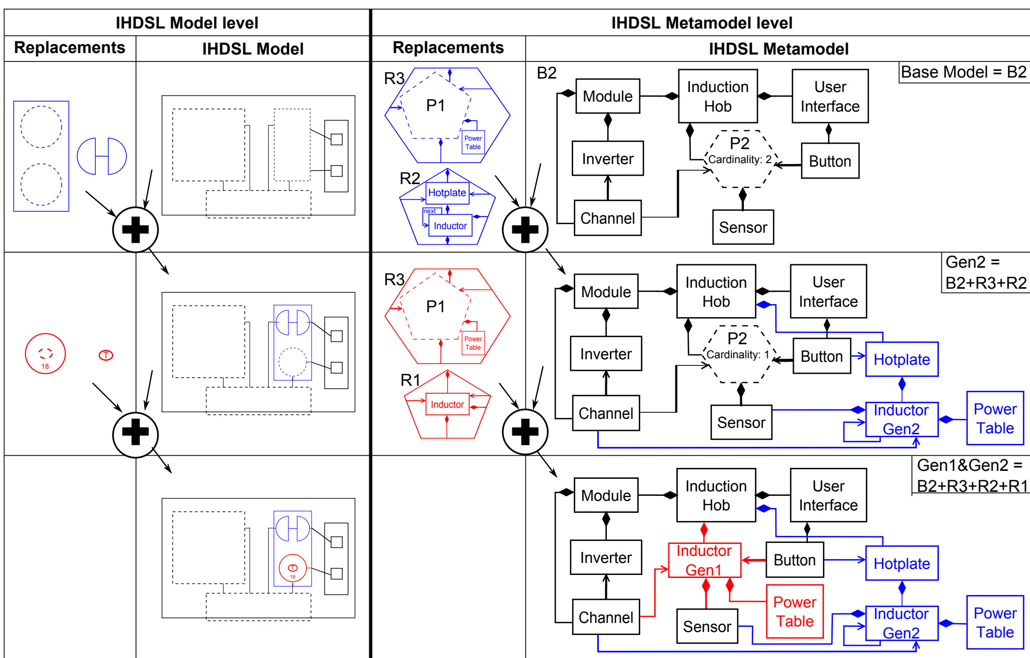


Figure 7. Fragment substitutions to derive a SPL product

The engineer will act at the model level, choosing which replacements should be substituted in the base model and building the Induction Hob model; in the meantime, at the metamodel level, the metamodel is built up automatically reflecting those model level substitutions. Each time a replacement is chosen by the engineer (at model level), the replacement (at metamodel level) corresponding to the replacement chosen by the engineer at the model level will be automatically substituted in the base model metamodel (only if it is the first occurrence of that generation).

Figure 7 show an example of the derivation when the SPL is in Generation 3. At the model level (the first and second columns), the engineer chooses the replacements (the first column) for the placements of the base model (the second column), while at the metamodel level (the third and fourth columns), the metamodel replacements (the third column) are automatically substituted for the placements of the Base Model (the fourth column). Note that the metamodel level elements presented in Figure 7 (the third and fourth columns) and the metamodel level elements presented in Figure 6 (the third column) are the same.

The first row in Figure 7 shows the first substitution of the product derivation: the engineer can use replacements from the three different generations available. In this case, the engineer is going to use replacements from the second generation (the first column). The base model of the current generation (the second column) is used. The metamodel level has the replacements R2 and R3 (third column) that correspond to the model level replacements, and the metamodel base B2 (fourth column) with all the common elements from all of the generations.

The second row in Figure 7 shows the result of the first fragment substitution. The fragments chosen by the engineer have been substituted at the model level (the second column). At metamodel level, corresponding fragments have been automatically substituted (the fourth column), resulting in the Generation 2 metamodel (Gen2). Now, if more model level replacements from Generation 2 are added, the metamodel does not vary (it only varies the first time that a generation is used). We repeat the operation with more replacements: this time they belong to Generation 1. At the metamodel level, the corresponding metamodel level replacements R1 and R3 are used.

The third row shows the results of the second fragment substitution. The model now has elements from two SPL generations; therefore, the metamodel has automatically been increased to be the combination of those two generations (Gen1&Gen2) maintaining the conformance between the model (the second column) and the metamodel (the fourth column). The engineer then performs more fragment substitutions until all the placements of the IH model are substituted; the metamodel is automatically increased as necessary.

The VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution. The following Section discusses the advantages and disadvantages of each of the strategies, taking into account the experience acquired from our industrial partner.

6. Discussion

We have applied both strategies to the retrospective of 13 years of our industrial partner's SPL models. In this paper, 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

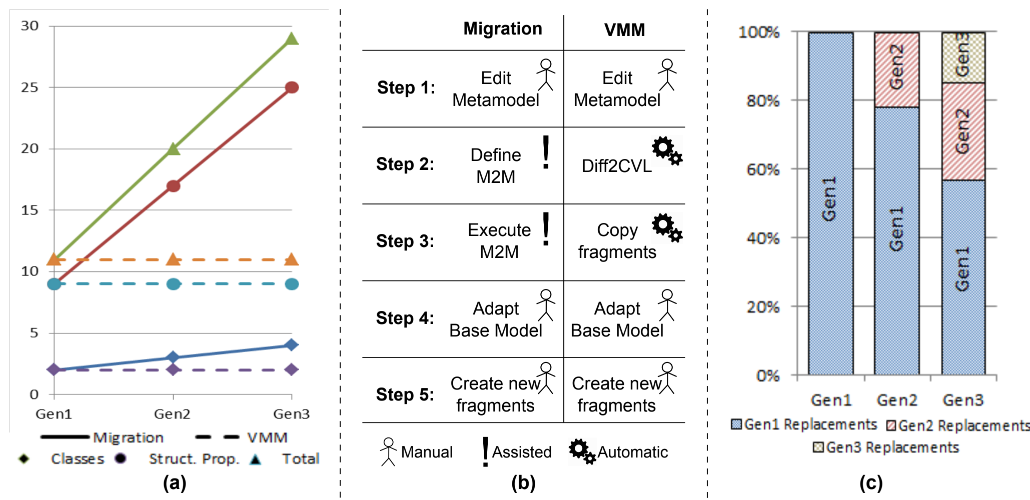


Figure 8. Comparison between Migration and VMM Strategy

32 different IH models composed of approximately 72 different model replacements (each of them composed of multiple model elements). The average number of model elements of a fragment replacement is 43, while the average number of elements of an IH model is about 470. Figure 8 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) indirection, (b) automation and, (c) trust leak.

6.1 Indirection

Indirection 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 4).

Figure 8 (a) shows the comparison of both strategies in terms of the indirection 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 indirection 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.

6.2 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 user must help in the process; and it is manual when the whole process is performed by the user.

Figure 8 (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 [15], operator-based [12, 17] or metamodel matching [3, 10]), the process is, at best, assisted [3, 11]. In contrast, in the VMM Strategy Step 2 (Diff2CVL) is fully automatizable, (CVL applied to the model and the metamodel level enabled us to resolve all kind of changes presented by [3] 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 ([3, 11]), so the step needs to be assisted. In contrast, in the VMM strategy replacements are used "as is": 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.

6.3 Trust Leak

Models are used to produce code: once they have been used repeatedly on many IHs, they gain the trust of our industrial partner's engineers. However, when the replacements are modified, there is a loss in this trust on the part of the engineers, which has been reported as *trust leak*.

Figure 8 (c) shows a comparison of both strategies in terms of *trust leak*. The graph shows the weight of the replacements of each generation in relation to the total number of products created with the SPL (i.e., average percentage of replacements from each generation present in the induction hobs taking into account all the IHs derived from the SPL). For instance, using the migration strategy for Evolution 2 (from Gen2 to Gen3), the replacements that represent 83% of the total products built need to be migrated (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, they are the replacements that have the highest level of trust leak. In contrast, when using the VMM strategy there is no need to migrate replacements, thus avoiding the trust leak.

7. Related Work

To the best of our knowledge, there are no works that address the evolution of SPLs using variability modeling ideas at the metamodel level; However, there are research efforts on SPL evolution that can complement model-based SPL evolution.

In [1] Batory et al. present the AHEAD model, based on the step-wise refinement paradigm, enables the synthesization of multiple complex programs from a simple program. In AHEAD the software is expresed as nested sets of equations describing feature refinements. The composition function (specific for each kind of asset) is used to stack the refinements applied to the base program to produce the different variants. However we do not focus on how to specify variants of the base product, the main focus in our approach is to avoid the migration of the models from one generation to the next by applying variability at the metamodel level.

Dhungana et al. [6] present an approach that is based on model fragments applied at the model level. They provide tool support for the automated detection of changes, facilitating metamodel evolution and propagating changes in the domain to already existing variability models. However, in contrast to our approach, they do not use fragments at the metamodel level having to update their fragments when changes occur at the metamodel level.

Deng et al. [5] 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. Although the details are different, their approach is similar to the *migration strategy* with support of model transformations. However, our work shows that, in the case of a CVLSPL, the *VMM strategy* offers the best results.

Creff et al. [4] propose an incremental evolution by extension of the product line. They aim to benefit from the investments supplied during the product derivation and *re-invest* them into the SPL models. Specifically, they introduce an assisted feedback algorithm to extend the SPL to emerging product derivation requirements. We believe that their feedback algorithm could be tailored to help in the detection for the need of new metamodel changes (new SPL Generations) when product derivations occur, triggering our *VMM strategy* to address the evolution at the metamodel level.

Passos et al. [13] 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. In our work, the SPL is specified by means of base models, fragment substitution and metamodel expressiveness. However, we can represent the variability model of our industrial partner's SPL by means of a feature model, therefore strategy can benefit from the analysis and traceability of the work of Passos et al. [13].

8. Conclusions

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the placements applied to the metamodel level have proven to provide enough expressiveness to overcome all the evolution situations of our industrial partner over 13 years. In addition, the VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution.

This work indicates that the VMM achieves better results than the migration strategy in domains like the domain of our industrial partner in terms of indirection, automation, and trust leak. Furthermore, using already existing variability management approaches (like CVL) enables us to bring efforts from the variability research community to address the evolution challenge. Nevertheless, there are still open issues (e.g., evolutions that turn variabilities into commonalities) that will be addressed in our work in the future.

References

[1] D. Batory, J. N. Sarvela, and A. Rauschmayer. Scaling step-wise refinement. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 187–197, Washington, DC, USA, 2003. IEEE Computer Society.

[2] D. Benavides, S. Segura, and A. R. Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems Journal*, 35(6):615–636, 2010.

[3] A. Cicchetti, D. Di Ruscio, R. Eramo, and A. Pierantonio. Automating co-evolution in model-driven engineering. In *Enterprise Distributed Object Computing Conference, 2008. EDOC '08. 12th International IEEE*, pages 222–231, 2008.

[4] S. Creff, J. Champeau, J.-M. Jézéquel, and A. Monégier. Model-based product line evolution: an incremental growing by extension. In *16th International Software Product Line Conference*, SPLC '12, NY, USA, 2012. ACM.

[5] G. Deng, D. C. Schmidt, A. Gokhale, J. Gray, Y. Lin, and G. Lenz. Evolution in model-driven software product-line architectures. *Designing Software-Intensive Systems: Methods and Principles*, 2008.

[6] D. Dhungana, P. Grünbacher, R. Rabiser, and T. Neumayer. Structuring the modeling space and supporting evolution in software product line engineering. *Journal of Systems and Software*, 83(7):1108–1122, 2010.

[7] J.-M. Favre. Meta-model and model co-evolution within the 3d software space. In *In Procedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA) at ICSM*, pages 98–109, Amsterdam, Netherlands, September 2003.

[8] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang. A generic language and tool for variability modeling. *Technical Report SINTEF A13505*, 2009.

[9] J. Font, L. Arcega, Ø. Haugen, and C. Cetina. Building software product lines from conceptualized model patterns. In *Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20-24, 2015*, pages 46–55, 2015.

[10] K. Garcés, F. Jouault, P. Cointe, and J. Bézivin. Managing model adaptation by precise detection of metamodel changes. In R. F. Paige, A. Hartman, and A. Rensink, editors, *Model Driven Architecture - Foundations and Applications*, volume 5562 of *Lecture Notes in Computer Science*, pages 34–49. Springer Berlin Heidelberg, 2009.

[11] B. Gruschko, D. Kolovos, and R. Paige. Towards synchronizing models with evolving metamodels. In *Proceedings of the International Workshop on Model-Driven Software Evolution*, 2007.

[12] M. Herrmannsdoerfer, S. Benz, and E. Juergens. Cope - automating coupled evolution of metamodels and models. In S. Drossopoulou, editor, *ECOOP 2009 Object-Oriented Programming*, volume 5653 of *Lecture Notes in Computer Science*, pages 52–76. Springer Berlin Heidelberg, 2009.

[13] L. Passos, K. Czarnecki, S. Apel, A. Wasowski, C. Kästner, J. Guo, and C. Hunsen. Feature-oriented software evolution. In *7th International Workshop on Variability Modelling of Software-intensive Systems*, Italy, 2013. ACM.

[14] K. Pohl, G. Böckle, and F. Van Der Linden. *Software product line engineering: foundations, principles, and techniques*. Springer, 2005.

[15] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack. An analysis of approaches to model migration. In *Proc. Joint MoDSE-MCCM Workshop*, pages 6–15, 2009.

[16] M. Svahnberg and J. Bosch. Evolution in software product lines: Two cases. *Journal of Software Maintenance*, 11(6):391–422, Nov. 1999.

[17] G. Wachsmuth. Metamodel adaptation and model co-adaptation. In E. Ernst, editor, *ECOOP 2007 Object-Oriented Programming*, volume 4609 of *Lecture Notes in Computer Science*, pages 600–624. Springer Berlin Heidelberg, 2007.

12.2 COMLAN'17 Paper

Title:	Leveraging variability modeling to address metamodel revisions in Model-based Software Product Lines.
Authors:	Jaime Font, Lorena Arcega, Øystein Haugen, Carlos Cetina.
Journal:	Computer Languages, Systems & Structures
Date:	June, 2017
DOI:	`https://doi.org/10.1016/j.cl.2016.08.003`
Contribution:	Jaime Font is the main author of the paper and is responsible for 90% of the work.

Computer Languages, Systems & Structures 48 (2017) 20–38

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Leveraging variability modeling to address metamodel revisions in Model-based Software Product Lines ☆

Jaime Font [a,b,*], Lorena Arcega [a,b], Øystein Haugen [c], Carlos Cetina [a]

[a] Universidad San Jorge, SVIT Research Group, Spain
[b] University of Oslo, Department of Informatics, Norway
[c] Østfold University College, Department of Information Technology, Norway

A R T I C L E I N F O

Article history:
Received 1 January 2016
Accepted 8 August 2016
Available online 11 August 2016

Keywords:
Model-based Software Product Lines
Variability Modeling
Model and metamodel co-evolution

A B S T R A C T

Metamodels evolve over time, which can break the conformance between the models and the metamodel. Model migration strategies aim to co-evolve models and metamodels together, but their application is currently not fully automatizable and is thus cumbersome and error prone. We introduce the Variable MetaModel (VMM) strategy to address the evolution of the reusable model assets of a model-based Software Product Line. The VMM strategy applies variability modeling ideas to express the evolution of the metamodel in terms of commonalities and variabilities. When the metamodel evolves, changes are automatically formalized into the VMM and models that conform to previous versions of the metamodel continue to conform to the VMM, thus eliminating the need for migration. We have applied both the traditional migration strategy and the VMM strategy to a retrospective case study that includes 13 years of evolution of our industrial partner, an induction hobs manufacturer. The comparison between the two strategies shows better results for the VMM strategy in terms of model indirection, automation, and trust leak.
© 2016 Elsevier Ltd. All rights reserved.

1. Introduction

Model-Driven Development aims to shift the focus of software development from coding to modeling. Metamodels are used to formalize a set of concepts and the relationships among those concepts. A model conforms to a metamodel if it is expressed by the terms that are encoded in the metamodel.

Model-based Software Product Lines enable a planned reuse of software components in products that are within the same scope [1]. Commonalities and variabilities among the products are formalized into a set of models (and metamodels) using a variability language: either feature models [2,3] (the de facto standard for variability modeling) or Common Variability Language (CVL) [4], (recommended for adoption as a standard by the Architectural Board of the Object Management Group). Although the details are different, all share the idea of modeling commonalities and variabilities among the different products.

☆ This work has been partially supported by the Ministry of Economy and Competitiveness (MINECO) through the Spanish National R+D+i Plan and ERDF funds under the project Model-Driven Variability Extraction for Software Product Line Adoption (TIN2015-64397-R).
* Corresponding author at: Universidad San Jorge, SVIT Research Group, Autovía A-23 Zaragoza-Huesca Km. 299. 50.830 Villanueva de Gállego, Zaragoza, Spain.
E-mail addresses: jfont@usj.es (J. Font), larcega@usj.es (L. Arcega), oystein.haugen@hiof.no (Ø. Haugen), ccetina@usj.es (C. Cetina).

http://dx.doi.org/10.1016/j.cl.2016.08.003
1477-8424/© 2016 Elsevier Ltd. All rights reserved.

Similar to other software components, metamodels evolve over time [5]; however, changes that are introduced in the new metamodel revision can invalidate the models that conform to the previous revision of the metamodel. To address this issue, migration strategies [6–10] propose co-evolving models and metamodels together in order to maintain consistency.

However, even though migration strategies have proven to be successful in model-based approaches, their application is not fully automatizable and can be cumbersome and error prone in large systems. Evolution is particularly critical for a successful adoption of model-based Software Product Lines (SPLs) [11].

We believe that the ideas of variability modeling can also be applied at the metamodel level to address the evolution of SPLs and at the same time avoid the issues involved with migration strategies. Our contribution is the Variable MetaModel (VMM) strategy, which enables the evolution of the metamodel without breaking model conformance. In VMM, each metamodel evolution is expressed in terms of metamodel commonalities and variabilities. As a result, already existing models continue to conform to the created VMM, thus eliminating the need for migration and its related issues.

First, we build a retrospective case study of the evolution undergone by our industrial partner (BSH) over the last 13 years regarding the evolution of their models and metamodels. BSH is the leading manufacturer of home appliances in Europe and its induction department produces induction hobs (explained in Section 2) following an MDD approach [12].

We then apply a migration strategy to the case study, manually migrating the models (as described in Section 4) whenever a metamodel change that breaks the conformance between models and metamodels arises. Migration strategies involve the following three issues: (1) model migration introduces indirection to the models; (2) some of the steps of the migration strategy need human assistance; and (3) the trust gained by models (over years of use) is lost when they are migrated.

Finally, we also apply the VMM strategy to the retrospective case study and compare both strategies (VMM and migration). The comparison shows that the VMM strategy achieves better results than migration in terms of the three issues related to migration: (1) VMM eliminates the need for migration (and the indirections introduced); (2) some of the steps of the migration strategy require human assistance while in the VMM strategy those steps are automatic; (3) the trust gained by models remains the same in the VMM strategy (since the model does not need to change).

This paper is an extended and revised version of our paper published at GPCE 2015 [13]. Apart from revisions throughout the article, in this version we have improved the motivation of the approach and included details about the core operations of the VMM approach (InitVMM and addGen). We have also added some lessons learned from the application of the approach to our industrial partner, information which may be valuable for practitioners that want to apply the ideas of VMM to manage metamodel revisions.

2. Background

This section presents the Domain Specific Language (DSL) used by our industrial partner to formalize their products, the IHDSL. It will be used throughout the rest of the paper to present a running example. Then, the Common Variability

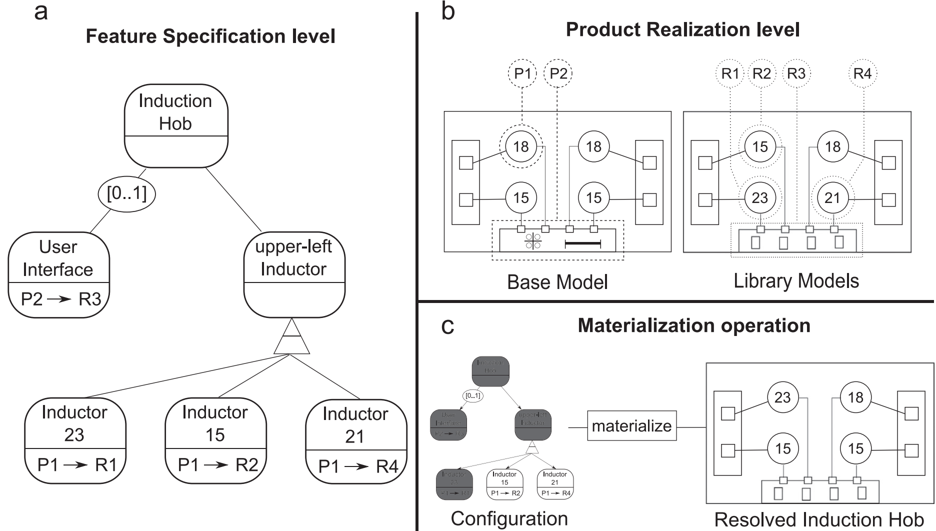


Fig. 1. CVL applied to IH-DSL.

Language (CVL) is presented. CVL is the language used by our VMM approach to formalize the differences between metamodel revisions.

2.1. The induction hobs domain

Induction cookers or hobs use electromagnetism to generate heat that is transferred to the cookware. Traditionally, stoves feature four rounded areas that become hot when turned on. Therefore, the first Induction Hobs (IHs) created provided similar capabilities. However, the induction hob domain is constantly evolving due to the possibilities provided by the induction phenomena and the programmable microcontrollers that are present in the IHs.

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 real-time measurements of the consumption of the IH.

The Domain Specific Language used by our industrial partner to specify the Induction Hobs (IHDSL) is composed of 46 meta-classes, 74 references among the meta-classes, and more than 180 properties. However, in order to gain legibility and due to intellectual property rights, in this paper we use a simplified subset of the IHDSL (the top-left corner of Fig. 2 shows the metamodel for this DSL).

The bottom-right corner of Fig. 1 shows an Induction Hob (Resolved Induction Hob) with the graphical representation of the IHDSL. It is composed of two power modules (vertical rectangles on both sides of the IH). Each of them holds two inverters (squares), which are in charge of providing the electrical supply required to generate the magnetic field. Inverters are connected to the inductors (circles), which are the elements where the magnetic field is generated. The number inside each inductor represents the diameter of the inductor. The line that connects inverters and inductors represents the power channel, which transfers energy from the inverter to the inductor. The user interface of an IH has controllers to configure the power level of each inductor (the horizontal rectangle at the bottom of the IH).

2.2. The Common Variability Language (CVL)

The Common Variability Language (CVL) is a DSL for modeling variability in any model of any DSL based on Meta-Object Facility (MOF, the OMG specification to define a universal metamodel for describing modeling languages). CVL defines variants of the base model by replacing parts of the base model with model replacements that are found in a library of models. The base model and the library of models must be specified using the same DSL.

Despite the fact that CVL is currently frozen by intellectual property rights issues, the proposed ideas have been recommended for adoption by the architectural board of the OMG and we decided to use it for technical reasons (given its expressiveness for realizing the variability specification over models). However, any other variability specification mechanism with capabilities similar to the ones of CVL could be used to materialize our approach.

The variability specification in CVL is divided across two different levels:[1] the feature specification level (where variability is specified following a feature model syntax [2,3]) and the product realization level (where the variability specified in terms of features is linked to the actual models in the form of variation points, replacements, and substitutions). Finally, the materialization operation takes a configuration of the feature specification and executes the substitutions defined by the product realization level to obtain a new variant.

Fig. 1 shows an example of specification of variability through CVL applied to the induction hobs models. Fig. 1(a) shows the feature specification level, Fig. 1(b) shows the product realization level, and Fig. 1(c) shows the materialization operation.

In Fig. 1(a), the bubbles represent features which are organized hierarchically. Some features can be optional (denoted by the range [0..1] like the *User Interface*). In addition, an OR operation between multiple choices of features can be defined (denoted by the triangle above the three inductors features, which are children of the *upper-left inductor* feature). Some features are only used to improve the readability and structure of the tree (e.g., *the upper-left inductor*) while others are linked with the substitutions (defined in the product realization level) that must be performed when materializing those products.

In Fig. 1(b), the substitutions of variation points by replacements are defined. The *base model* is a model described by a given DSL (here, IHDSL) that serves as the base for different model variants defined over it. The elements of the base model that are subject to variations are the placement fragments in CVL; however, we will refer to them with the more common term *variation points* (hereinafter VPs). It is important to remark that the term VP is also used in CVL and its definition includes not only the part of the model subject to variations but also the set of elements that can be used to substitute them. Therefore, the reader used to CVL terminology needs to be aware of the use of VP instead of placement in the context of this paper.

[1] Layers in the CVL terminology.

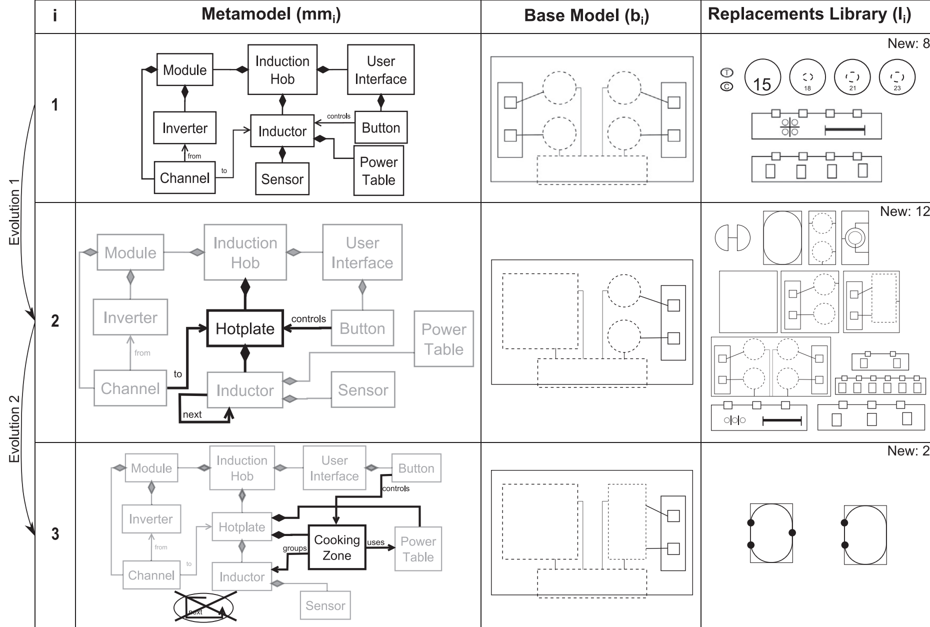


Fig. 2. Model generations of the CVLSPL.

In the base model, there are two VPs defined over it: P1, which includes the top-left inductor; and P2, which includes the user interface. To define replacements for a VP, CVL uses a library of models described in the same DSL as the Base Model. Each alternative for a VP in the Base Model is a *replacement* fragment (hereinafter replacement). In the library model, there are four replacements defined: three inductor replacements (R1, R2, and R4) and a user interface replacement (R3).

Notice the correspondence between the substitutions linked to the features from the feature specification level in Fig. 1(a) and the VP and replacements defined in the product realization level in Fig. 1(b).

CVL defines variants of the base model by means of *fragment substitutions* (i.e., the substitution of VPs by replacements). Fig. 1(a) shows the Feature specification level with the substitutions linked to some features. First, the *Induction Hob* feature is the root and must be present in all IHs. Then, we can have buttons for the *User Interface* substituting P2 by R3 (this substitution is optional). We also have different alternatives for the *upper-left inductor* (sizes 23, 15, or 21) substituting P1 by R1, R2, or R4.

In order to specify a product model from the set of models described by the feature specification model, a *configuration*[2] of the feature specification model is used. In other words, the choices about the features that will or will not be included in the product are resolved by the user. Fig. 1(c) shows the materialization operation of CVL, which executes the substitutions that are linked to the features present in a configuration and produces a variant of the base model where the VPs have been substituted by the replacements selected. As a result of the materialization operation, a resolved induction hob is produced where P1 has been substituted by R1 and P2 has been substituted by R3 (following features from the configuration and the linked substitutions).

When VPs are defined through CVL, there are two characteristics that make the language really flexible: recursion and cardinality. Recursion enables the definition of VPs inside other VPs in a recursive manner. The cardinality enables the substitution of one VP by several replacements at the same time. CVL is currently frozen by non-technical reasons, the recursion and cardinality capabilities when defining VPs are leveraged by our approach, and, therefore, CVL is used by our approach as the variability specification language.

For simplicity throughout the rest of the paper, we will show the VPs superimposed on the base model, even though they are defined in a separate model. Also, the replacements defined in the library models will be shown separately from the rest of the model where they are defined.

[2] Resolution model in the CVL terminology.

3. SPL evolution formalized by CVL

This section presents the retrospective case study that was extracted from the evolution of our industrial partner's 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 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:

$$CVLSPL = MM \times M \times M$$
$$cvlspl_i = \langle mm_i, b_i, l_i \rangle \quad (1)$$

where mm_i is the metamodel of the DSL (conforming to MOF), b_i is the base model (over which VPs for the variable parts are defined), l_i is the library of replacements for those VPs, 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* (hereinafter *breaking change*) [9] in the metamodel. Breaking changes break the conformance of models and the metamodel in a way that cannot be resolved by automatic means [9] (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 [6–10] (e.g., eliminating a property). However, in this work we will focus on the evolution triggered by breaking changes.

Fig. 2 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 VP in the base model and shape of each replacement in the replacements library are indicators of what VPs can be substituted by what replacements.

Evolution 1 (from $cvlspl_1$ to $cvlspl_2$) is triggered by a new concept called Hotplate (see the first and second rows of Fig. 2). 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).

There are also modifications at the model level. A new VP 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 Fig. 2). 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).

A new VP 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.

As any other model change, the evolutions presented so far can be seen as graph transformations. For instance, the changes could be expressed as extensions and derivations like it is done in the work of Gonçalves et al. [14]. Although the presented changes could be formalized using graph transformations, the changes come from pragmatic development, and graph rewrites have not been the basis for the developments.

Sections 4 and 5 show the application of both Migration and VMM strategies to address the evolution presented in this section.

4. Motivation of the approach

The evolution presented in Section 3 needs to be properly supported by the metamodels that are used by our industrial partner to formalize their SPL. Some of the changes presented can be addressed without breaking the conformance between the models and the metamodel, such as the creation of new model fragments or the addition of new optional elements to the metamodel. However, when we perform a breaking change to the metamodel (e.g., the hotplate and cooking zone concepts), the conformance between the models and the metamodel is lost.

Traditional migration strategies [6–10] propose migrating all of the models to conform to the new version of the metamodel. Given a metamodel change, the migration of the SPL can be achieved by the following steps: (1) the metamodel is upgraded to a new version introducing the new concept; (2) a model-to-model (M2M) transformation that migrates models from one version to another is created (manual specification [6], operator-based [7,8], or metamodel matching [9,10]); (3) existing replacements are migrated (by executing the M2M transformation obtained from Step 2) to conform to the new generation of the metamodel; (4) if some common parts that are present in the base model have become variable, the user creates VPs over the base model and extracts the model fragments as replacements; and (5) new replacements are created to instantiate the new concepts that have been incorporated into the metamodel.

Let E_{mig} be the operation used to evolve a $cvlspl_i$ from a given generation i to the next generation $(i+1)$ following the migration strategy. The operation is defined as follows:

$$
\begin{aligned}
&E_{mig}: CVLSPL \longrightarrow CVLSPL\\
&E_{mig}(\langle mm_i, b_i, l_i\rangle) = \langle mm_{i+1}, b_{i+1}, l_{i+1}\rangle\\
&\quad \text{where } M2M(L_i) = L_{i+1}
\end{aligned} \tag{2}
$$

Fig. 3 shows three *CVLSPL generations* that are managed using the migration strategy. We start from $cvlspl_1$, which is shown in the first column. The square depicts the metamodel (mm_1) (which includes the inductor concept), the diamond depicts the base model (b_1), and the circle depicts the library of replacements (l_1). Both b_1 and l_1 conform to mm_1.

Given a breaking change (hotplate concept), we perform the five steps of the migration strategy. In Step 1 (Fig. 3, column one), we modify the metamodel to include the new concept. Then, in Step 2, we define the $M2M_1$ transformation, which migrates models that conform to mm_1 into models that conform to mm_2. In Step 3, we apply the $M2M_1$ transformation to the library of replacements l_1 (8 replacements). This step requires assistance on the part of the user to apply the transformation associated to breaking changes. Step 4 consists of the extraction of common parts from the base model b_1 that have turned into variables parts (as required by the new generation). Since the changes of $cvlspl_2$ include bigger hotplates in the form of new replacements that cannot be placed in existing VPs, we create new VPs and substitutions over the base model b_2. In addition, we extract those old VPs from the base model and include them in new replacements (three new replacements in l_2). Finally, in Step 5, nine new replacements that instantiate the new concept are created in l_2.

We have created the $cvlspl_2$ of the SPL by following the above steps. We follow the same steps to create the $cvlspl_3$ with this strategy. This time, when the metamodel is edited (Step 1), the relationship *next*, (created in generation 2), is eliminated since its functionality is now provided by the cooking zone. Note that this time we need to migrate both replacement libraries, the library previously migrated from $cvlspl_1$ (8 migrated replacements) and the library created during $cvlspl_2$ generation (9 + 3 new replacements).

Fig. 4 presents the evolution of a model fragment following a migration strategy. Each column shows the same fragment (Inductor 15) for each of the $cvlspl_i$ generations. 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

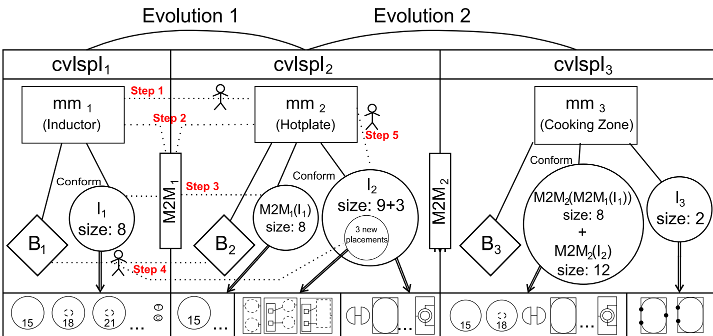


Fig. 3. The steps of the migration strategy.

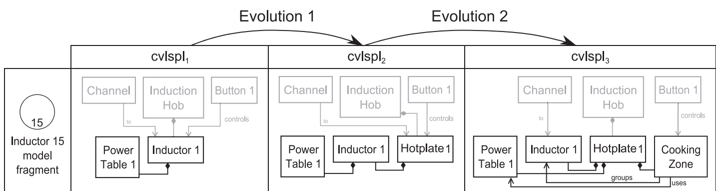


Fig. 4. Evolution of model fragment through migrations.

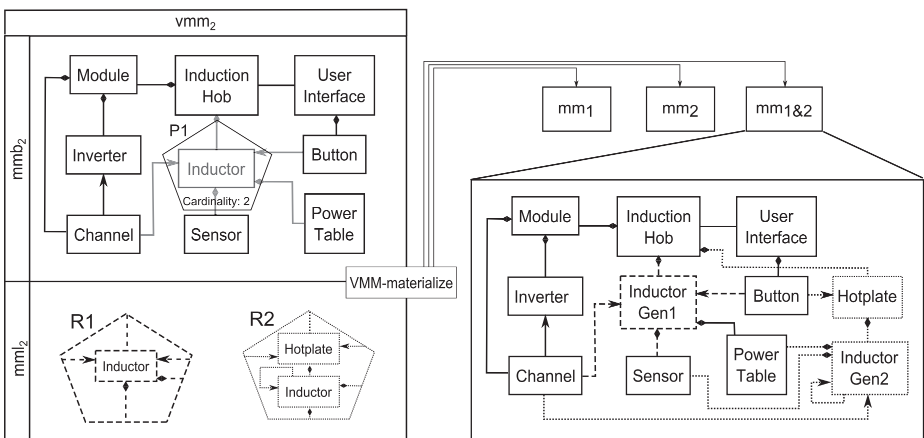


Fig. 5. VMM and VMM-materialize.

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.

Specifically, the migration of models from our industrial partner involves three related issues: (1) *indirection*, where there is an increase in the number of elements used to model the same element of the induction hob (as in this example); (2) *automation*, since the migration of the models cannot be performed automatically, an engineer needs to generate the M2M transformation and make decisions when applying it; and (3) *trust leak*, the modification of the model fragments (through the migrations) decreases the trust gained by those models during that generation. 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 domain of our industrial partner is constantly evolving, but the original elements are still present in new IHs. New kinds 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.

5. 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 is automatically built: the Variable MetaModel (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 the generations of the CVLSPL will exist.

The *VMM* is the result of applying CVL at the metamodel level; we have a base model in a given DSL (in this case, MOF) with VPs defined over it and a library of replacements. *VMM* is defined as follows:

$$VMM = MM \times MM$$
$$vmm_i = \langle mmb_i, mml_i \rangle \tag{3}$$

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

The vmm_i holds all metamodel variations from starting generation (generation 1) to generation i. Similar 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:

$$VMMmat: VMM \times \mathcal{P}(G) \longrightarrow MM$$
$$VMMmat(\langle mmb_i, mml_i \rangle, g) = mm_g \quad \text{where } g \neq \emptyset \tag{4}$$

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.

Fig. 5 (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 VP (P1) defined over the inductor. The bottom-left corner of Fig. 5 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.

Fig. 5 (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 VPs in CVL enables the creation of $mm_{1\&2}$. In other words, one VP can be substituted more than one time (the number of times can be specified). The first time that a VP is substituted, the existing references of the VP are replaced. The second time that the same VP is substituted, new references that are analogous to the existing ones need to be created. In Fig. 5, 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 $mm_{1\&2}$ are 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 [15]); 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 enable the creation of new replacements.

6. 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

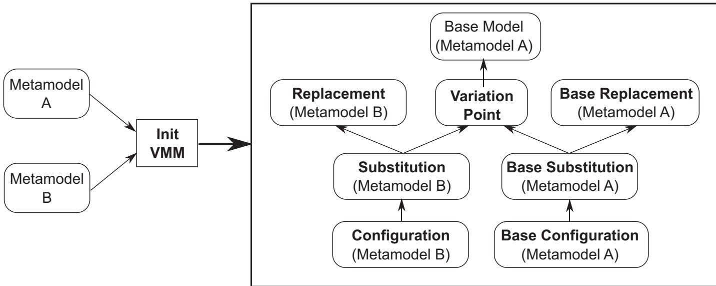


Fig. 6. InitVMM operation.

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.

6.1. InitVMM operation

Fig. 6 shows an example of the initVMM operation. InitVMM receives two metamodel revisions (e.g., Metamodel A and Metamodel B) as an input and produces a VMM that includes both generations as an 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 [12], 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).

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 (VPs, 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) *Variation Point*: A VP is created over the base model (if that VP does not previously exist). Using the information from the comparison, the boundaries of the VP 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 VP, we use the information from the comparison to determine the boundaries of the replacement.
   (c) *Substitution*: Once a VP and a replacement have been created the process generates a substitution. That is, the boundaries of the VP and the replacement are mapped accordingly, so the VP 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 VP 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 VP 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 VP 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 (in spite of whether or not that revision is used as the base model).

In summary, the initVMM operation formalizes a metamodel revision in terms of CVL, generating VPs, 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 the base model.

Fig. 5 (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 the base model (mmb_2), a new VP 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 VP was updated accordingly as there were two substitutions using that VP (the configurations do not have a graphical representation in the figure).

6.2. 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 VPs. In other words, when creating a new VP as part of Step 1.(b) Variation Point, the VP may already exist and there is no need to create a new one. The same VP 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 VPs, 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 [16]. 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).

7. Application of the VMM approach

The previous sections have presented the VMM approach and the operations needed to materialize it. In this section, we describe how to apply the VMM strategy to manage the metamodel revisions of a model-based SPL. Section 7.1 presents the steps necessary to build new generations. Section 7.2 shows the resulting VMM when applied to our case study. Finally, Section 7.3 shows the usage of the resulting SPL to derive new products from the SPL.

7.1. The steps of the VMM strategy

This section presents the steps performed to include a new CVLSPL generation into the VMM. Some steps need to be manually performed by the engineer while others are performed automatically by applying the operations described previously. In this example we show the inclusion of the first generation of CVLSPL, so the initVMM operation will be used

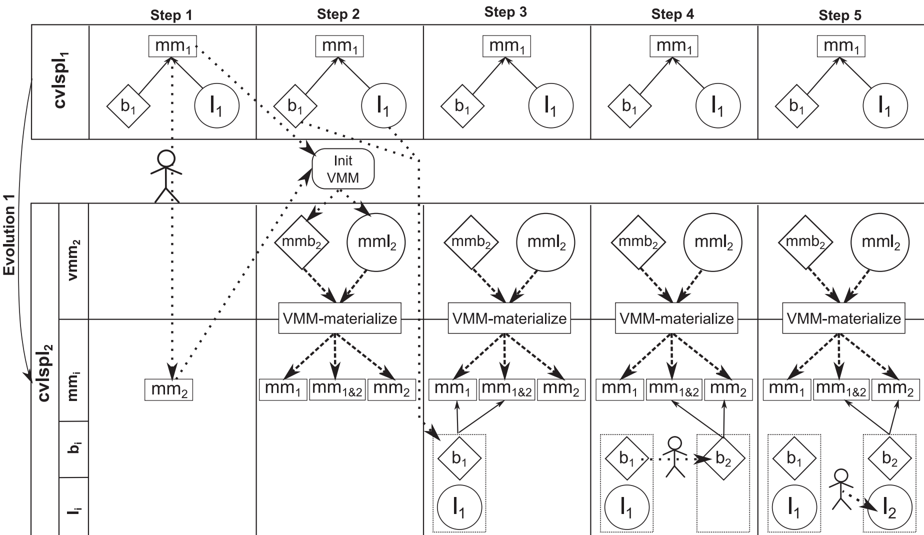


Fig. 7. The steps of the VMM strategy.

(the addition of further generations follows the same steps). The evolution of a $cvlspl_i$ following the VMM-strategy is denoted as follows:

$$E_{VMM}: \; CVLSPL \longrightarrow VMM$$
$$\langle mm_i, b_i, l_i \rangle \longrightarrow \langle mmb_{i+1}, mml_{i+1} \rangle \tag{5}$$

VMMmat is used with the generated vmm_i to retrieve different CVLSPL generations needed by the company.

Fig. 7 shows the method for performing Evolution 1 from $cvlspl_1$ to $cvlspl_2$. Each of the columns of the tables represents one step in the application of the VMM strategy. The top part shows $cvlspl_1$ with its base model b_1 (depicted as a diamond), its metamodel mm_1 (depicted as a rectangle), and its fragment library l_i (depicted as a circle). The bottom part shows $cvlspl_2$: the first row shows vmm_2, and the second row shows $cvlspl_2$.

Step 1 shows the edition of the metamodel by the user. The mm_1 metamodel is edited to include the new concepts of the next generation, resulting in the mm_2 metamodel. In this example the engineer has modified the mm_1 (see first row first column of Fig. 2) into the mm_2 (see second row first column of Fig. 2), including the concept of Hotplate.

Step 2 shows our initVMM operation, which is used to spot the differences between the two metamodels, to describe them in terms of a base model and replacements, and to initialize the VMM. The common parts of the two metamodels (mm_1 and mm_2) are included in the mmb_2 and VPs are created over it for the differences between mm_1 and mm_2. Furthermore, replacements that contain these differences are created and included in the mml_2. This operation has been explained in detail in Section 6.1. Then, the *VMMmat* operation can be applied to vmm_i to obtain mm_1, mm_2, and $mm_{1\&2}$ as explained in Section 5.

In Step 3, the l_1 and b_1 from $cvlspl_1$ are copied without any modification to be used in $cvlspl_2$. Both conform to the materialized mm_1, and they also conform to the materialized $mm_{1\&2}$. That is, model replacements from $cvlspl_1$ (see first row third column of Fig. 2) are copied to $cvlspl_2$.

In Step 4, some common parts of the base model (b_1) may become variable because of the new concepts introduced in generation 2. In that case, the engineer edits the base model b_1 (that has been copied in the previous step) from the $cvlspl_2$ to extract the variable parts as replacements. Notice the modifications performed to b_1 (see first row second column of Fig. 2) in order to obtain b_2 (see second row second column of Fig. 2)

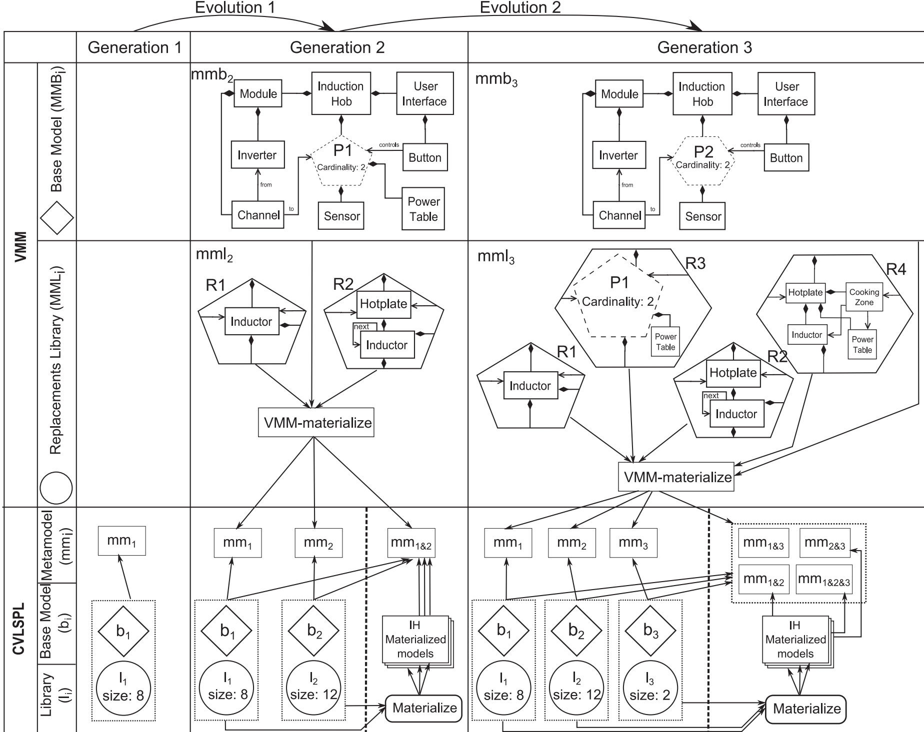


Fig. 8. The IHDSL metamodel level for each generation of the CVLSPL.

In Step 5, the engineer creates new replacements that instantiate the new concepts of this generation (see second row third column of Fig. 2) and includes them in l_2. These new replacements conform to mm_2, and they also conform to $mm_{1\&2}$.

Following the above steps, we can evolve the SPL from one generation to the next, while eliminating the need for migrating existing fragments. Then, when the engineer wants to create new replacements, the engineer will be able to use the metamodel of just one generation and not the $mm_{1\&2}$. As a result, the engineer can create replacements for the most recent generation (using mm_2) to instantiate the new concepts of that generation. In contrast, the engineer can use the previous generation metamodel (mm_1) to create replacements that do not exercise the expressiveness provided by the new generation, thus avoiding the overcharge of the model (as in the case of the motivating example, see Section 4). When materializing an IH model containing replacements from both generations (l_1 and l_2), the resulting IH model will conform to $mm_{1\&2}$.

In addition, the recursion capabilities of CVL enable us to create VPs inside a replacement and hence apply the VMM strategy to further generations. In other words, when creating the next generation, Step 2 of the process could end up in the creation of a new replacement that includes previously defined VPs (if the replacement is not common for both metamodels).

7.2. The resulting models after applying the VMM strategy

Fig. 8 shows an overview of our industrial partner's *CVLSPL* models (rows) after applying the VMM strategy to manage Evolution 1 and Evolution 2 (columns).

In *Evolution* 1, a new concept (hotplate) is introduced (see the first and second columns). This concept affects the inductor, which is now aggregated by the hotplate; therefore, we apply the method explained above to perform Evolution 1. InitVMM produces a base model (mmb_2) that contains a VP (P1) with cardinality 2 (i.e., it can be replaced up to two times). InitVMM also produces mml_2, which contains two replacements: R1 (which holds the particularities of Gen1) and R2 (which holds the particularities of Gen2). The *VMMmat* operation can be applied to those models to produce three different metamodels: (1) the substitution of P1 by R1 produces mm_1; (2) the substitution of P1 by R2 produces mm_2; and (3) the substitution of P1 twice (by R1 and by R2) produces $mm_{1\&2}$.

When creating new fragments, the engineer must stick to only one generation in order to create a valid fragment. In other words, fragments must conform to a specific metamodel generation, either mm_1 or mm_2. As a result, the engineer can create replacements using only concepts from mm_1, thereby eliminating the indirection introduced by the migration strategy (see Section 4).

When materializing an IH model that contains replacements from both generations (l_1 and l_2), the resulting IH model conforms to $mm_{1\&2}$. Overall, vmm_2 enables the materialization of IH models with replacements from both generations (l_1 and l_2), while at the same time allowing the creation of fragments pertaining to one generation (either conforming to mm_1 or to mm_2).

In *Evolution* 2, a new *breaking change* that introduces the concept of cooking zones occurs (see the second and third columns). Similar to Evolution 1, we apply the method to perform Evolution 2 (from generation 2 to generation 3).

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the VPs applied to the metamodel level have proven to provide enough expressiveness to overcome all of the evolution situations of our industrial partner over 13 years.

7.3. The derivation of SPL products after applying VMM

The VMM strategy has been tooled within the Eclipse environment and integrated into our industrial partner's SPL. The resulting tool is used by our industrial partner (BSH, the leading manufacturer of home appliances in Europe) to generate the firmware of their Induction Hobs (sold under the brands of Bosh and Siemens). An example of the resulting tool in action can be seen here[3]. This section presents an example of using the SPL evolved with the VMM strategy: an engineer of our industrial partner deriving a new product.

The engineer will act at the model level, choosing which replacements should be substituted in the base model and building the Induction Hob model; in the meantime, at the metamodel level, the metamodel is built up automatically reflecting those model level substitutions. Each time a replacement is chosen by the engineer (at the model level), the replacement (at the metamodel level) corresponding to the replacement chosen by the engineer at the model level will be automatically substituted in the base model metamodel (only if it is the first occurrence of that generation).

Fig. 9 shows an example of the derivation when the SPL is in generation 3. At the model level (the first and second columns), the engineer chooses the replacements (the first column) for the VPs of the base model (the second column); at the same time, at the metamodel level (the third and fourth columns), the metamodel replacements (the third column) are automatically substituted for the VPs of the base model (the fourth column). Note that the metamodel level elements presented in Fig. 9 (the third and fourth columns) and the metamodel level elements presented in Fig. 8 (the third column) are the same.

[3] http://www.carloscetina.com/variablemetamodel.htm

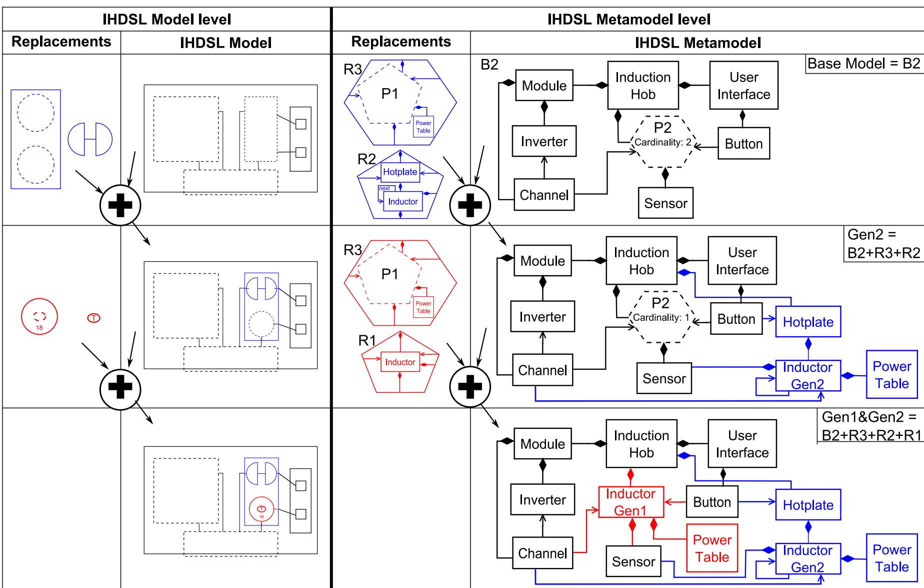


Fig. 9. Fragment substitutions for deriving SPL products.

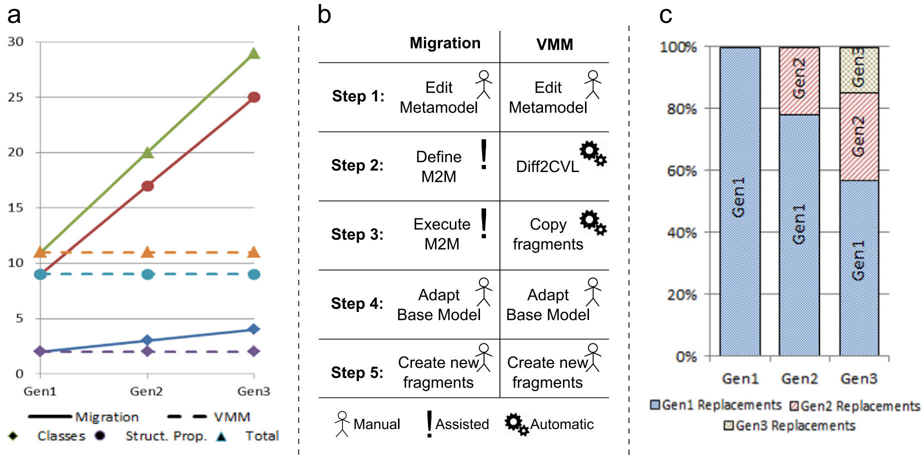


Fig. 10. Comparison between migration and VMM strategy.

The first row in Fig. 9 shows the first substitution of the product derivation: the engineer can use replacements from three different generations available. In this case, the engineer is going to use replacements from the second generation (the first column). The base model of the current generation (the second column) is used. The metamodel level has the replacements R2 and R3 (third column) that correspond to the model level replacements, and the metamodel base B2 (fourth column) with all the common elements from all of the generations.

The second row in Fig. 9 shows the result of the first fragment substitution. The fragments chosen by the engineer have been substituted at the model level (the second column). At the metamodel level, corresponding fragments have been automatically substituted (the fourth column), resulting in the generation 2 metamodel (Gen2). Now, if more model level replacements from generation 2 are added, the metamodel does not vary (it only varies the first time that a generation is

used). We repeat the operation with more replacements: this time they belong to generation 1. At the metamodel level, the corresponding metamodel level replacements R1 and R3 are used.

The third row shows the results of the second fragment substitution. The model now has elements from two SPL generations; therefore, the metamodel has automatically been increased to be the combination of those two generations (Gen1&Gen2) maintaining the conformance between the model (the second column) and the metamodel (the fourth column). The engineer then performs more fragment substitutions until all the VPs of the IH model are substituted; the metamodel is automatically increased as necessary.

The VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution. Section 8 discusses the advantages and disadvantages of each of the strategies, taking into account the experience acquired from our industrial partner.

8. Discussion

We have applied both strategies to the retrospective of 13 years of our industrial partner's SPL models. In this paper, 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 fragment replacement is 43, while the average number of elements of an IH model is about 470. Fig. 10 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) indirection, (b) automation, and (c) trust leak.

8.1. Indirection

Indirection 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 4).

Fig. 10(a) shows the comparison of both strategies in terms of the indirection 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 indirection 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.

8.2. 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.

Fig. 10(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 [6], operator-based [7,8], or metamodel matching [9,10]), the process is, at best, assisted [17,9]. 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 [9] 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 [17,9], 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.

8.3. Trust leak

Models are used to produce code; once they have been used repeatedly on many IHs, they gain the trust of our industrial partner's engineers. 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*.

Fig. 10(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. 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 usage of 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 id 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.

9.1. 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 usage, we realized that some false revisions were being created in the VMM.

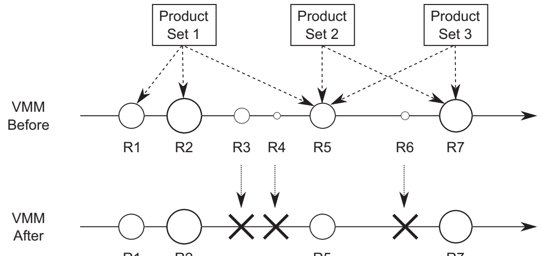


Fig. 11. False revisions.

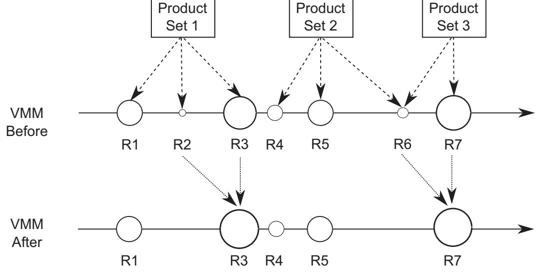


Fig. 12. Revision folding.

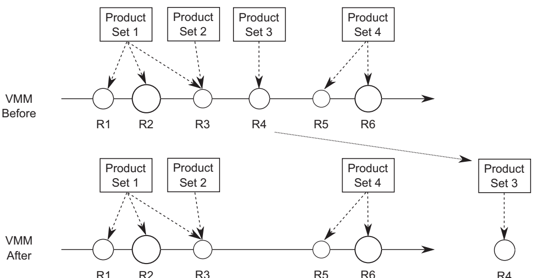


Fig. 13. Isolated revisions.

Fig. 11 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 turns 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 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 usage 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 (as in Section 7.3).

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 uses 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 not considered anymore a false revision.

9.2. 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 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.

Fig. 12 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 that belong to the unused metamodel revision.

When the engineer decides to fold two revisions, the traditional migration strategy is followed. That is, the steps presented in Section 4 are followed to migrate the fragments 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 among the two revisions.

9.3. 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 the complexity.

Fig. 13 shows an example of an isolated revision. The VMM *before* shows four products 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 notice that, although at that point in time 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 extracted.

10. Related work

To the best of our knowledge, there are no works that address the evolution of SPLs using variability modeling ideas at the metamodel level. However, there are research efforts on SPL evolution that can complement model-based SPL evolution.

There are some approaches that rely on model comparisons to formalize the variability that exists among a set of models [18–21]. However, those operations are applied at the model level (not at the metamodel level) and the aim of the operations is to be capable of recreating the products used as input. In other words, they produce the fragments necessary to recreate the models, but they do not capture each model's particularities in separate fragments. In contrast, the initVMM operation from our approach is designed to capture the particularities of each metamodel revision provided as input in separate fragments, thus enabling the possibility of materializing metamodels that combine several generations.

In [22], Batory et al. present the AHEAD model, which is based on the step-wise refinement paradigm and enables the synthesization 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. However, we do not focus on how to specify variants of the base product; the main focus in our approach is to avoid the migration of the models from one generation to the next by applying variability at the metamodel level.

Dhungana et al. [23] 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 already existing variability models. However, in contrast to our approach, they do not use fragments at the metamodel level, requiring their fragments to be updated when changes occur at the metamodel level.

Deng et al. [24] 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. Although the details are different, their approach is similar to the migration strategy with the support of model transformations. However, our work shows that, in the case of a CVLSPL, the *VMM strategy* offers better results.

Creff et al. [25] 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. We believe that their feedback algorithm could be tailored to help in the detection of the need for new metamodel changes (new SPL generations) when product derivations occur, triggering our *VMM strategy* to address the evolution at the metamodel level.

All of these works are based on address the evolution of a software. However, these approaches do not take into account the problems that can arise from the migration of the deprecated software of the system (in our case model fragments). We focus our work on the evolution that incorporates new versions of products without damaging the rest of the product that are already developed and in use. Anyway, all of them are using different strategies and, although we base our evolution strategy in the ideas of CVL, we can adapt their ideas to use them with our evolution strategy.

There are much more research efforts in categorization and analysis of changes that can trigger the need of evolving a system. These works combine empirical studies and analysis in order to obtain a better understanding of the changes that occur in the software life cycle.

Lotufo et al. [26] 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 investigation, they identify six categories of reasons for changes. Although we do not use feature models, we can consider their categories to categorize the changes that occur in the evolution of our CVLSPL.

Passos et al. [27] 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. In our work, the SPL is specified by means of base models, fragment substitution, and metamodel expressiveness. However, since we can represent the variability model of our industrial partner's SPL by means of a feature model, our strategy can benefit from the analysis and traceability of the work of Passos et al. [27].

Similar to us, they are focus on software evolution in the real-world. Although they studies are based on different techniques, some of the ideas and categorizations could be applied to the evolution of our CVLSPL to obtain a more accurate evolution strategy.

11. Conclusions

The CVL capabilities of recursion (placements inside replacements) and cardinalities over the VPs applied to the metamodel level have proven to provide enough expressiveness to overcome all the evolution situations of our industrial partner over 13 years. In addition, the VMM strategy of this work enables our industrial partner's engineers to derive products by means of replacements from any generation, while avoiding the disadvantages of migrating the replacements after each evolution.

This work indicates that the VMM achieves better results than the migration strategy in domains like the domain of our industrial partner in terms of indirection, automation, and trust leak. Furthermore, using already existing variability management approaches (like CVL) enables us to bring efforts from the variability research community to address the evolution challenge. Nevertheless, there are still open issues (e.g., evolutions that turn variabilities into commonalities) that will be addressed in our work in the future.

The VMM strategy avoids the need for migration of model fragments when new metamodel revisions arise, but at the expense of introducing the need of managing the VMM. Such management is needed to avoid creation of false revisions, provides means for folding revisions and properly extracts isolated revisions to reduce the complexity of the VMM. However, the management of the VMM is performed at a higher abstraction level than the migration tasks that have been replaced.

References

[1] Pohl K, Böckle G, Van Der Linden F. Software product line engineering: foundations, principles, and techniques. Springer, Springer-Verlag New York, Inc. Secaucus, NJ, USA ©2005; 2005.
[2] Kang KC, Cohen SG, Hess JA, Novak WE, Peterson AS. Feature-oriented domain analysis (foda) feasibility study. Technical report. DTIC Document; 1990.
[3] Benavtides D, Segura S, Ruiz-Cortés A. Automated analysis of feature models 20 years later: a literature review. Inf Syst 2010;35(6): 615–36, http://dx.doi.org/10.1016/j.is.2010.01.001, URL ⟨http://www.sciencedirect.com/science/article/pii/S0306437910000025⟩.
[4] Fleurey F, Haugen Ø, Møller-Pedersen B, Olsen GK, Svendsen A, Zhang X. A generic language and tool for variability modeling. Technical report SINTEF A13505, 2009.
[5] Favre J-M. Meta-model and model co-evolution within the 3d software space. In: Proceedings of the international workshop on evolution of large-scale industrial software applications (ELISA) at ICSM. Amsterdam, Netherlands; 2003. p. 98–109.
[6] Rose LM, Paige RF, Kolovos DS, Polack FA. An analysis of approaches to model migration. In: Proceedings of the joint MoDSE-MCCM workshop; 2009. p. 6–15.
[7] Herrmannsdoerfer M, Benz S, Juergens E. Cope - automating coupled evolution of metamodels and models. In: Drossopoulou S, editor, ECOOP 2009 object-oriented programming. Lecture notes in computer science, vol. 5653. Berlin, Heidelberg: Springer; 2009. p. 52–76. http://dx.doi.org/10.1007/978-3-642-03013-0_4.
[8] Wachsmuth G. Metamodel adaptation and model co-adaptation. In: Ernst E, editor, ECOOP 2007 object-oriented programming. Lecture notes in computer science, vol. 4609. Berlin, Heidelberg: Springer; 2007. p. 600–624. http://dx.doi.org/10.1007/978-3-540-73589-2_28.
[9] Cicchetti A, Di Ruscio D, Eramo R, Pierantonio A. Automating co-evolution in model-driven engineering. In: Enterprise distributed object computing conference, 2008. EDOC'08. 12th International IEEE; 2008. p. 222–31. http://dx.doi.org/10.1109/EDOC.2008.44.
[10] Garcés K, Jouault F, Cointe P, Bézivin J. Managing model adaptation by precise detection of metamodel changes. In: Paige RF, Hartman A, Rensink A, editors, Model driven architecture–foundations and applications. Lecture notes in computer science, vol. 5562. Berlin, Heidelberg: Springer; 2009. p. 34–49. http://dx.doi.org/10.1007/978-3-642-02674-4_4.
[11] Svahnberg M, Bosch J. Evolution in software product lines: two cases. J Softw Maint 1999;11(6):391–422.
[12] Font J, Arcega L, Haugen Ø, Cetina C. Building software product lines from conceptualized model patterns. In: proceedings of the 19th international conference on software product line, SPLC 2015, Nashville, TN, USA; July 20–24, 2015. p. 46–55. http://dx.doi.org/10.1145/2791060.2791085.
[13] Font J, Arcega L, Haugen Ø, Cetina C. 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. p. 161–70. http://dx.doi.org/10.1145/2814204.2814214.
[14] Gonçalves RC, Batory D, Sobral JL, Riché TL. From software extensions to product lines of dataflow programs. Softw Syst Model 2015: 1–19, http://dx.doi.org/10.1007/s10270-015-0495-8.
[15] Apel S, Kolesnikov S, Siegmund N, Kästner C, Garvin B. 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. New York, NY, USA: ACM; 2013. p. 1–8. http://dx.doi.org/10.1145/2528265.2528267.
[16] Foundation E. Eclipse modeling framework compare (emfcompare) website. ⟨http://wiki.eclipse.org/index.php/EMFCompare⟩; 2008.
[17] Gruschko B, Kolovos D, Paige R. Towards synchronizing models with evolving metamodels. In: Proceedings of the international workshop on model-driven software evolution; 2007.

[18] Zhang X, Haugen Ø, Moller-Pedersen B. Model comparison to synthesize a model-driven software product line. In: Proceedings of the 2011 15th international software product line conference, SPLC'11. Washington, DC, USA: IEEE Computer Society; 2011. p. 90–99. http://dx.doi.org/10.1109/SPLC.2011.24.
[19] Martinez J, Ziadi T, Bissyandé TF, Klein J, Traon YL. Bottom-up adoption of software product lines: a generic and extensible approach. In: Proceedings of the 19th international conference on software product line, SPLC 2015, Nashville, TN, USA; July 20–24, 2015. p. 101–110. http://dx.doi.org/10.1145/2791060.2791086.
[20] Font J, Ballarín M, Haugen Ø, Cetina C. 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 2015, Nashville, TN, USA; July 20–24, 2015. p. 411–18. http://dx.doi.org/10.1145/2791060.2793678.
[21] Rubin J, Chechik M. Combining related products into product lines. in: de Lara J, Zisman A, editors, Fundamental approaches to software engineering. Lecture notes in computer science, vol. 7212. Berlin, Heidelberg: Springer; 2012. p. 285–300. http://dx.doi.org/10.1007/978-3-642-28872-2_20.
[22] Batory D, Sarvela JN, Rauschmayer A. Scaling step-wise refinement. In: Proceedings of the 25th international conference on software engineering, ICSE'03. Washington, DC, USA: IEEE Computer Society; 2003. p. 187–97. URL ⟨http://dl.acm.org/citation.cfm?id=776816.776839⟩.
[23] Dhungana D, Grünbacher P, Rabiser R, Neumayer T. Structuring the modeling space and supporting evolution in software product line engineering. J Syst Softw 2010;83(7):1108–22, http://dx.doi.org/10.1016/j.jss.2010.02.018, URL ⟨http://www.sciencedirect.com/science/article/pii/S0164121210000506⟩.
[24] Deng G, Schmidt DC, Gokhale A, Gray J, Lin Y, Lenz G. Evolution in model-driven software product-line architectures. In: Designing software-intensive systems: methods and principles. http://dx.doi.org/10.4018/978-1-59904-699-0.ch005, http://www.igi-global.com/chapter/evolution-model-driven-software-product/8235.
[25] Creff S, Champeau J, Jézéquel J-M, Monégier A. Model-based product line evolution: an incremental growing by extension. In: 16th international software product line conference, SPLC'12. New York, USA: ACM; 2012. http://dx.doi.org/10.1145/2364412.2364430.
[26] Lotufo R, She S, Berger T, Czarnecki K, Wasowski A. Evolution of the linux kernel variability model. In: Proceedings of the 14th international conference on software product lines: going beyond, SPLC'10. Berlin, Heidelberg: Springer-Verlag; 2010. p. 136–50. URL ⟨http://dl.acm.org/citation.cfm?id=1885639.1885653⟩.
[27] Passos L, Czarnecki K, Apel S, Wasowski A, Kästner C, Guo J, et al. Feature-oriented software evolution. In: 7th international workshop on variability modelling of Software-intensive Systems. Italy: ACM; 2013.