A SCALABLE DESIGN FRAMEWORK FOR VARIABILITY MANAGEMENT IN LARGE-SCALE SOFTWARE PRODUCT LINES

MUHAMMAD GARBA

MARCH 2016

A thesis submitted in partial fulfilment of the requirements of the University of East London for the Degree of Doctor of Philosophy

Abstract

Variability management is one of the major challenges in software product line adoption, since it needs to be efficiently managed at various levels of the software product line development process (e.g., requirement analysis, design, implementation, etc.).

One of the main challenges within variability management is the handling and effective visualization of large-scale (industry-size) models, which in many projects, can reach the order of thousands, along with the dependency relationships that exist among them. These have raised many concerns regarding the scalability of current variability management tools and techniques and their lack of industrial adoption.

To address the scalability issues, this work employed a combination of quantitative and qualitative research methods to identify the reasons behind the limited scalability of existing variability management tools and techniques. In addition to producing a comprehensive catalogue of existing tools, the outcome form this stage helped understand the major limitations of existing tools.

Based on the findings, a novel approach was created for managing variability that employed two main principles for supporting scalability. First, the separation-of-concerns principle was employed by creating multiple views of variability models to alleviate information overload. Second, hyperbolic trees were used to visualise models (compared to Euclidian space trees traditionally used). The result was an approach that can represent models encompassing hundreds of variability points and complex relationships. These concepts were demonstrated by implementing them in an existing variability management tool and using it to model a real-life product line with over a thousand variability points.

Finally, in order to assess the work, an evaluation framework was designed based on various established usability assessment best practices and standards. The framework was then used with several case studies to benchmark the performance of this work against other existing tools.

Acknowledgements

I would like to thank Dr. Rabih Bashroush, my Director of Studies, Reader in Distributed Systems and Software Engineering, School of Architecture, Computing and Engineering (ACE), University of East London, for his intellectual guidance and constant support that has led to the completion of this thesis. Thanks also to my second supervisor, Dr. Usman Naeem, for his positive feedback and assistance in this research.

I am grateful to all people who provided their valuable insight to me, at meetings, or via email. I am particularly thankful to Dr. Adel Noureddine and Dr. Rick Rabiser for their incisive feedback. Thank you, my colleagues in the Research Centre, School of Architecture Computing and Engineering, for the many discussions that have contributed to my work in this thesis.

Finally, a special thanks to Alh. Wada Sani, for his moral and financial support. My sincere gratitude to my family and friends for their constant caring and support– this work would have not been possible without their sustained understanding.

Contents

List of Figures

List of Tables

Glossary

ASADAL	A System Analysis and Design Aid tooL
BVR	Base-Variation-Resolution
CASE	Computer-Aided Software Engineering
ConIPF	Configuration in Industrial Product Families
CPP	C-preprocessor
DOORS	Rational Dynamic Object Oriented Requirements System
DOPLER	Decision Oriented Product Line Engineering for Effective Re-use
EC	Exclusion Criteria
EMF	Eclipse Modelling Framework
EWSA	European Workshop on Software Architecture
FODA	Feature Oriented Domain Analysis
FORM	Feature Oriented Reuse Method
GEMS	Generic Eclipse Modelling Framework
GMF	Graphical Modeling Framework
HCI	Human Computer Interaction
IC	Inclusion Criteria
ISMT4SPLs	Integrated Software Management Tool for Adopting Software Product Lines
LVAT	Linux Variability Analysis Tools
MUSA	Multitouch Variability Modelling Solution for Software Product Lines
NUI	Natural User Interface
OVM	Orthogonal Variability Model
PLM	Product Line Management
PLUSEE	Product Line UML-Based Software Engineering Environment
RCP	Rich Client Platform
RSEB	Reuse-Driven Software Engineering Business
SLR	Systematic Literature Review
SPL	Software Product Line

SPLC	Software Product Line Conference
SPLE	Software Product Line Engineering
SPLOT	Software Product Lines Online Tools
SXFM	Simple XML Feature Model
UML	Unified Modelling Language
VaMoS	Workshop on Variability Modelling of Software Intensive Systems
VARMA	VARiability Modelling and Analysis
VisPLE	International Workshop on Visualisation in Software Product Line Engineering
VM	Variability Management
VP	Variation Points
WICSA	Working International Conference on Software Architecture
WPF	Windows Presentation Foundation
XML	Extensible Markup Language
XSL	EXtensible Stylesheet Language

Chapter 1

Introduction

Software Product Line Engineering (SPLE) is a paradigm of software engineering for creating a portfolio or a collection of similar software products with variations in their features and functions. The products can be software, such as home automation system, as well as systems with software inside. Typical example of these include; airplanes, automobiles, ships, cameras, mobile phones, computers and tablets, among others (Krueger, 2007).

The SPLE technique provides a systematic way to reuse software assets. These assets are the software artefacts or resources associated with your products. The artefacts include, but are not limited to requirements analysis, design specifications, software implementation, configuration, test plans, test cases, etc. The assets are then engineered to be shared across the entire product line, i.e., to be used in multiple products. Therefore, SPLE is a technique that optimizes the reuse of existing software assets by creating multiple applications that share many features, while still exhibiting certain differences (Clements and Northrop, 2002, K. C. Kang et al., 2002). SPLE allows for the planned reuse of artefacts among the software systems under development.

Some of the key advantages of Software Product Line (SPL) development over “one at a time” system development include: productivity gains (the core assets and architecture are reused), decreased time-to-market of products, large-scale productivity, low-cost production, increased product quality and reliability, and increased customer satisfaction (Clements and Northrop, 2001).

Over the last two and a half decades, SPLE has increasingly gained the attention of researchers and practitioners alike. This is due to the potential economic advantages and business competitiveness the SPLE process can bring (Clements and Northrop, 2002, Van der Linden et al., 2007). The benefits can range from cutting the development cost and increasing software quality, to enabling mass customisation, market dominance, and reduced time to market (Clements and Northrop, 2002, Pohl et al., 2005).

In traditional software development, individual software systems are developed from scratch, i.e., one software at a time. Typical software development process requires going through stages such as requirements, analysis, design, implementation and testing to be performed. In contrast, SPLE is centered around multiple developments of similar software systems from a common core asset (Clements and Northrop, 2002, Pohl et al., 2005). This is achieved by explicitly capturing the commonalities and variabilities in the family of systems that forms the product line (Gomaa, 2005).

In addition, the market benefits encourage both, the software as well as the hardware industry, to recognise the significance of transitioning from single software development to a product line approach. Various terminologies are used to refer to SPL, such as software product families, system families, or family of systems.

The SPLE process (Pohl et al., 2005, Bachmann and Clements, 2005) involves studying and managing the common and varied features of the different product line members, a process usually referred to as domain engineering or development for reuse. Core (shared) assets – e.g., requirements, architecture, code, test cases – are then used as a basis to derive products from the product line, a process usually referred to as application engineering or development with reuse.

Correspondingly, defining and managing commonalities and variability in software product lines is widely referred to as variability management and is a key step of the SPL engineering process (Van Gurp et al., 2001). The variability management process guides the construction of product line variability models.

A lot of work has been conducted in the area which resulted in many approaches including various techniques, methods, and tools. Typical Examples of these include early methods: FODA (Kang et al., 1990) (Feature Oriented Domain Analysis) for discovering and identification of prominent distinctive features of software systems in a domain as well as presenting commonalities among related software systems. FORM (Kang et al., 1998) (Feature Oriented Reuse Method) a method that searches and captures common and different features of an application and using the analysis results to develop domain architecture and components.

Others are, FeatRSEB (Griss et al., 1998) (combination of the FODA method and the Reuse-Driven Software Engineering Business method (RSEB) (Jacobson et al., 1997)), this method includes domain engineering and feature modelling into RSEB by extending the original feature diagram in FODA into a network of features linked together using a unified modelling language (UML) refinements. The method allows explicit representation of variation points.

Other approaches are: Decision-oriented modelling technique (Atkinson et al., 2002), in which a set of questions (variation points) are described and a set of possible answers or decisions to be choose from. This method offers invaluable guidance to the development of product line variants using UML within a Model Driven Architecture.

Furthermore, OVM (Pohl et al., 2005) (Orthogonal Variability Model) contains the description of variation points (a representation of variability subject from which possible selection can be made), variants (an identification of a single option of variation point), and variability dependencies (constraints on variants selection) and models the variability as a separate concern in a specific OVM notations. ConIPF techniques pioneer by Bosch et al (Van Gurp et al., 2001), which uniformly models variability in all abstraction layers of product families, i.e. in the features, the architecture and component implementation layers.

Also, a number of tools evolved such as PLUSEE (Gomaa and Shin, 2007) (Product Line UML Based Software Engineering Environment), provide an automated product line engineering tool where a multiple view model of the product line architecture and components are developed and stored in a product line repository. Feature Mapper (Heidenreich, 2009), an eclipse plug-in software product line tool that provides support for mapping features from feature models to subjective modelling artefacts that are expressed by using an Ecore-based languages such as UML2 and DSLs. DOPLER meta-tool (Dhungana et al., 2011) (Decision Oriented Product Line Engineering for Effective Re-use) supports variability modelling that helps define variability of core assets, such as features, architectural elements or resources.

View Infinity (Stengel et al., 2011), is a Zoomable Interface for Feature-Oriented Software   Development. This tool offers seamless and semantic zooming from the feature model level to file structure and the source code level of different abstraction layers of SPL. ISMT4SPLs (Park et al., 2012) is an Integrated Software Management Tool for Adopting Software Product Lines that can provide traceability among the artefacts created at domain engineering and application engineering stages.

BigLever Gears (BigLever), a commercial tool that allows defining arbitrary reusable software assets and a product feature profile that describes products in terms of features. Gears can be tailored to different environments with parameter sets representing different kinds of variability. Pure::Variant (Levent V, 1998, Beuche, 2008), is a tool that supports variant management and product configuration based on feature models and has a strong focus on interoperability and extensibility, among others. Further information and a good overview of existing modelling approaches can be found in Czarnecki et al. (Czarnecki et al., 2012), (Chen et al., 2009) and Sinnema et al. (Sinnema and Deelstra, 2007).

However, surprisingly, very few of these approaches have actually made it to industry. These are the BigLever Gears and Pure::Variants, and both of these tools/companies were university spin-outs based on the work of two PhD students in America and German respectively. A recent study shown that 71.43% of these approaches have never been evaluated against industrial settings (Chen and Ali Babar, 2011).

1.2 Problem Statement

Variability models define the commonalities and variability of the product line from a problem space (e.g., features, decisions, or variation points) and a solution space (e.g., the reusable assets or variants) perspective along with the relationships that exist between these two spaces and among the elements in these spaces. Example of relationships include exclusivity (when two features cannot exist in one product at the same time); inclusivity (when the existence of one feature depends on another); and alternatives (when only one of a number of alternative features can be supported), to name a few.

Variability models tend to be very large in size, in many cases comprising thousands of features, and complex in nature due to the myriad of relationships that could exist among the features. This makes the construction of variability models manually a very tedious and error-prone process.

Accordingly, one of the major challenges within variability models of large-scale (industry-size) is the handling and effective visualisation of the models, which usually encompass a very large number of variation points as well as the dependency relationships that exist among them (Bashroush, 2010, Bashroush et al., 2011, Botterweck et al., 2008, Nestor et al., 2007, Pleuss and Botterweck, 2012, Heuer et al., 2010). Product line developers are facing problems with dependency management within variability models. An excessive amount of time and effort is being spent on fixing dependencies to ensure valid derivation of products (Berger et al., 2013, Sinnema et al., 2006, Daizhong and Shanhui, 2009).

However, for more than two decades, numerous variability management tools and techniques have been proposed and introduced from both academia and the industry. The main goal of all these research works is to help practitioners in the industry deal with variability management-related complexities (Chen and Babar, 2010, Sinnema and Deelstra, 2007). In spite of all these significant efforts, most of these approaches do not scale well when visualizing large-scale variability models, besides, they offer limited or no mechanism for managing dependency relationships that exist within the models. These have raised many concerns regarding the feasibility and scalability of current variability management tools and techniques.

As such, there has been an increasing demand for focus on making variability management tools and techniques more scalable to handle the complexity of real world industrial product lines (Chen and Babar, 2009).

The key scalability challenges are summarised below:

- ***Challenge 1:*** Creating and visualising of large scale and complex product line models (industry size models). Currently, existing approaches focus on ad hoc software product line variability, and often do not fully address real life product line variability required by SPL practitioners.
- ***Challenge 2:*** Visualising of hundreds of variants and their variation points in a large scale product line model.
- ***Challenge 3:*** Defining and visualising of constraints and dependency relationships (such as variation point to variation point, variant to variation point, or variant to variant) in a large scale product line model.
- ***Challenge 4:*** Proper arrangement of constraints and dependency relationships for better visualisation.
- ***Challenge 5:*** Effective visualisation of the effect of constraints and dependency relationships such as (inclusivity, alternativeness, or multiplicity).
- ***Challenge 6:*** Clear information to differentiate as to whether a feature is mandatory, optional, or alternative.

1.3 Research Context

This thesis investigated the reasons behind the lack of scalability in current variability management tools and techniques. Using a rigorous approach, we have examined the types of tools developed and the characteristics of these tools (visualisation techniques deployed, platform used, interoperability, etc.), in order to understand the main challenges of the problem. We have also explored the limitations faced by the current Product Line Management (PLM) tools and techniques.

The overall goal of this research is to improve the scalability of modelling variability by employing the idea of separation-of-concerns design principle, in order to show how the dependency relationships (such as variation point to variation point, variant to variation point, or variant to variant) of variability models, can be captured and managed independently from the actual variability representation.

This thesis introduced a new solution for capturing and managing dependencies using logic circuit. A separate view is proposed (i.e., dependency view), for managing dependencies separately, in order to reduce the problem of information overloading when viewing and managing large-scale variability points from one view. Support for this was implemented by redesigning and creating a new version of a Multitouch Variability Modelling Solution for Software Product Lines (MUSA) tool suite (a proof-of-concept variability management tool and framework that was developed within our research group), that can address these challenges, and thus, lend itself to industrial large-scale applications. This latest version of MUSA provides better means to represent, visualise, and manage the variability of large and complex product line models. This solution has been evaluated using a large-scale, multifaceted case study.

1.4 Research Aims

One of the challenges with SPLE is the scalability of variability management techniques. This has limited the adoption of SPLE to specific application domains. The main reason behind this challenge is attributed to the inability of current tools and techniques to scale to industry size applications. In this research, we aim to:

1. Closely examine the current literature to identify the main reasons behind the current limited scalability of variability management tools and techniques.
2. Identify the barriers to adoption of current tools and techniques.
3. Based on the findings of 1 and 2, design a tool and framework that addresses the shortcomings identified.
4. Implement a working prototype of the system.

1.5 Research Questions

The thesis answered the following research questions:

1. What are the key limiting factors affecting the scalability of existing variability management tools and techniques?
2. What are the barriers to industrial adoption of the current variability management tools?
3. What can be done to address these limitations in current tools?

1.6 Research Methodology

A combination of qualitative and quantitative research methodologies were used to address the research questions identified above over three parts or stages of the project.

In the first part, a Systematic Literature Review (SLR) approach was used to identify gaps in the body of knowledge and answer Q1/Q2. An SLR is a formal and rigorous way used to carefully examine, evaluate and interpret identified research evidence based on research questions or a particular research area (Kitchenham and Charters, 2007). The aim was to systematically review the reported literature on variability management tools in software product lines (known as primary studies). The process of SLR involved three main phases which are:

1) Planning the Review:- which has three stages:
   - Identifying the need for a review (importance)
   - Indicating the research question(s)
   - Developing and evaluating a review protocol
2) Conducting the Review:- which consists of four stages:
   - Identification of primary studies
   - Selection of primary studies based on clear criteria
   - Assessing the quality of primary studies
   - Data extraction and synthesis

3) Reporting the Review: - which involves:

- Writing and Formatting the main report and
- Evaluating / drawing conclusions based on the findings

In the second part, a new version of the MUSA tool suit (Bashroush, 2010) was built based on the findings of the first two stages in a way that addresses the identified shortcomings. The tool has been ported to Java technology from WPF (Windows Presentation Foundation). Among other features that have been introduced to MUSA is an innovative visualisation technique based on mind-mapping which is to replace the traditional tree structure for representing variability models, and the use of logic circuit design to graphically represent the dependency relationships.

After the completion of the tool's redesign and the MUSA framework, in the third part, we have evaluated the tool using multiple case studies, which ranges from small, medium and large-scale. However, these case studies were used as a basis to assess the scalability of MUSA as compared to other tools such as Pure::variants, one of the most popular commercial tools that we have access to. Among them, the largest sample is a case study for a Frequency Power Drives product line, and was acquired from Danfoss Power Electronics. Others include a case for Library Services product line representing the variability modelling of a wide range of services offered by a library to provide smooth and effective services to customers.

Further to that, is a case of a house automation system product line that provides basic security, alarm, lighting, communication, and agenda services, to mention a few. Chapter 8 provides detail description of the case studies used in the evaluation.

1.7 Summary of Contributions

The main contributions of this thesis are summarized below:

C1 A systematic investigation and understanding of the state of the art tools that can be utilised in contemporary software product line development: This study is a contribution to knowledge, as it conducts a systematic review of Variability Management tools according to the chronological order of development, and provides a conclusive evaluation of these tools. The results are intended to assist practitioners in selecting the best available tools, based on their suitability for a particular industrial task. The analysis also identifies gaps in the field that should be addressed through further research of product line tools. Moreover, the analysis identifies gaps in research that should be addressed in more studies. Based on these results, we have collected the data and necessary requirements for the development of our new MUSA tool.

C2 Redesign of MUSA framework to improve the scalability of visualizing and representing variability models: Although scalability was the main motivation for developing the early version of MUSA, redesigning and enhancing its capability to add more innovative visualisation techniques will increase productivity, time-to-market and allow for the creation and management of larger and more complex product families; hence, improving its scalability.

C3 An additional view for capturing and managing dependency relationships that exist within the model separately: Using principle of separation-of-concerns, we have proposed a separate view called, "dependency view" to capture and manage dependency interaction independently from the actual representation of the models. This was achieved using a logic circuit. The main idea is to reduce the complexities, such as graphical overloading, when viewing and managing dependencies of large variability points all from one view.

C4 A complete working prototype system will be implemented as a new MUSA: Support for managing dependency relationships among variability models has been implemented by redesigning and extending the current version of the MUSA tool suite (a proof-of-concept variability management tool and framework). This will allow the creation and management of larger and more complex product families.

C5 The new version of MUSA will be available as a multi-platform application: To make it more generic and maximise its functionality, the new version of MUSA has been ported from Windows Presentation Foundation-WPF to Java technology. This has solved the main problem of platform dependency suffered by the existing version of MUSA.

C6 A benchmark for evaluating our approach: In order to evaluate the MUSA tool in comparison with other tools, we developed a benchmark for evaluating the quality attributes, important for practical use of SPL engineering tools, which has been applied in the evaluation process. The benchmark focused on measuring the four quality attributes: Usability, Performance, Scalability, and Integration. In addition, an evaluation study was conducted experimentally, and involved 10 feature-modelling tools. In order to know and get an insight on how well, and to what extent these tools satisfy these quality attributes, four case studies of different sizes were used as the basis for the experiment.

C7 Literature review process (Chapter 2): This process contributes to knowledge by providing empirical step-by-step guidelines to identify, collect, and review papers with: 1) a scope of the review clearly identified in advance; 2) a comprehensive search conducted to find all relevant studies; 3) the use of explicit criteria to include or exclude studies; 4) the establishment of standards to critically appraise study quality; and 5) the provision of explicit methods for extracting and synthesizing study findings. This process will benefit both new and experienced researchers by helping them avoid what is regarded as author's bias in research, while also providing a reliable basis for making decisions.

C8 Benchmarking process: The results of this will contribute to knowledge, as it will assist both practitioners and researchers alike by providing a standard and empirical approach to evaluating product line tools in the future. It also helps to identify and recommend areas that require attention in future tool design.

C9 The Context of Research: The distribution of the research context presented in Figure 4.8 of Chapter 4 indicates that there is a need to bridge the gaps between research in academia and industry through collaborative efforts. The figure shows that most studies (68%) have been conducted in an academic context, whereas only 16% of the studies are joint industrial academic endeavours. In 16% of the studies, no information was provided on the research context. Table 4.11 presents the list of all the studies with their research context. Please refer to Chapter 4 for details on this contribution.

1.8 Thesis Structure

This thesis is structured in three parts and nine chapters.

1.8.1 Part I: State of The Art

- ***Chapter 2*** is organised in two main sections. We present and discuss the research methodology used to collect data in the first part. This includes the study’s research questions, search protocol, inclusion and exclusion criteria, quality criteria, and the data extraction and synthesis process. Section 2 provides an overall meta-analyses of the primary studies identifying trends and developments in the field.
- ***Chapter 3*** introduces tools’ supporting variability management, and discusses their usable functionality, i.e., the approach it uses in tackling variability issue— the environment, or a platform and technology, based on which a tool was developed and implemented, respectively. It also identifies the notations type (graphical, textual, or a combination of both), employed by a tool, and the category to which a tool belonged to; whether, commercial, academic, or both. The chapter also assesses the possibility of obtaining an evaluation copy.

- ***Chapter 4*** is a thorough analysis of the state-of-the-art technology in the field, essentially for tools supporting variability management, to understand the tools' characteristics, maturity, and the challenges it might be exposed to, in the field. Consequently, two parts were formed during the analysis: in the first part, different tools were identified and assigned with a unique ID, then their analysis was carried out, based on certain key topics that were recognised as follows: development environment; support for transformations (between different formats); management of constraints and reasoning on variability models, and; their proposed graphical and textual notations. In the second part, an analysis was made, based on the quality of the research conducted in the reported approaches, as well as the research context of the studies as they have been conducted.

1.8.2 Part II: Early Version of Musa Framework versus New Version

- ***Chapter 5*** presents the early version of MUSA (A Multitouch Variability Modelling Solution for Software Product Lines) tool and its theoretical foundation, upon which it was designed and developed. MUSA was implemented as a proof-of-concept over the state-of-the-art Human Computer Interaction (HCI), the Microsoft Surface and Windows 7 Multitouch platform.
- ***Chapter 6*** Introduces the new version of MUSA (i.e., version 2) tool, and a framework that exhibits a number of features (multi-platform support, dependency management, innovative visualisation technique,

etc.), for dealing with large-scale software product line models. This version adopts the separation-of-concerns design principles and provides multiple perspectives to the model, each of which conveys a different set of information.

1.8.3 Part III: Validation

- ***Chapter 7*** presents a benchmark that focuses on two major aspects: measuring the four quality attributes (usability, scalability, performance, and integration), identified as important for practical use of SPL, and the use of this benchmark as a basis to assess the scalability of our (MUSA) approach, as compared to other variability management tools.
- ***Chapter 8*** describes the four case studies of varying sizes and data elements that are used in the experimentation. Also, the results of the experimental evaluation are presented.

1.8.4 Part IV: Conclusion and Future Research Work

- ***Chapter 9*** summarises and concludes this thesis and describes the further work that could be conducted to improve the framework and tool supporting it.

1.9 Bibliographical Notes

Some of the material presented in this thesis reuses and extends publications of the author in the following papers:

Conference

1. Garba, M., Noureddine, A. and Bashroush, R. (2016) 'MUSA: A Scalable Multi-Touch and Multi-Perspective Variability Management Tool'. *13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Venice, Italy: IEEE Computer Society. (**Chapter 6**)

Under Review

2. Garba, M., Bashroush, R., Rabiser, R., Groher, I. and Botterweck, G. (2015) 'CASE Tool Support for Variability Management in Software Product Lines', *ACM Computing Survey.* (**Chapter 2, 4**)

3. Garba, M., Bashroush, R. and Naeem, U. (2016) 'Towards Bridging the Gap Between Industry and Academia in CASE Tool Support for Software Product Lines', *IEEE Transactions on Systems, Man, and Cybernetics.* **(Chapter 7, 8)**

PART I: STATE OF THE ART

Chapter 2

Literature Review Methodology

2.1 Introduction

Now that we have introduced the main context of the thesis, it is time to put these concepts into play. Earlier in this work, we have studied all the published literature on Computer-Aided Software Engineering (CASE) tool support for variability management over the last two decades, using a systematic literature review as inspired by (Kitchenham and Charters, 2007). The objective was to understand what tools have been produced, the characteristics of these tools, their context, and the challenges and limitations they faced.

This chapter presents and explains in its first part, the research method used to collect and review papers, as well as the trend of analysis from the results of the extracted data. The second section provides overall meta-analyses of the primary studies, identifying trends and developments in the field.

The results of the study will: i) give practitioners access to a catalogue of published tools and guide them in selecting the best tool for a given task enhancing the accessibility of the published tools; ii) provide researchers in the field with the main challenges and limitations that require further investigation, and; iii) provide new researchers with a good understanding of the state-of-the-art in tool support for variability management in SPL engineering.

2.2 Research Method

To achieve the objectives of this study, a SLR approach was adopted to conduct the survey. An SLR, as stated in section 1.5 of chapter one is a rigorous method for examining, evaluating, and interpreting all available research evidence based on research question(s) or particular research topic(s) (Kitchenham and Charters, 2007).

The study examines current literature on variability management tools in SPLE engineering (known as primary studies) published over the last two decades. Throughout the research study, the guidelines for SLRs were followed as provided in (Kitchenham and Charters, 2007). This involves three main phases: (1) Planning the review; (2) Conducting the review, and; (3) Reporting the review. Figure 2.1 depicts the stages of SLRs, adapted from (Brereton et al., 2007).

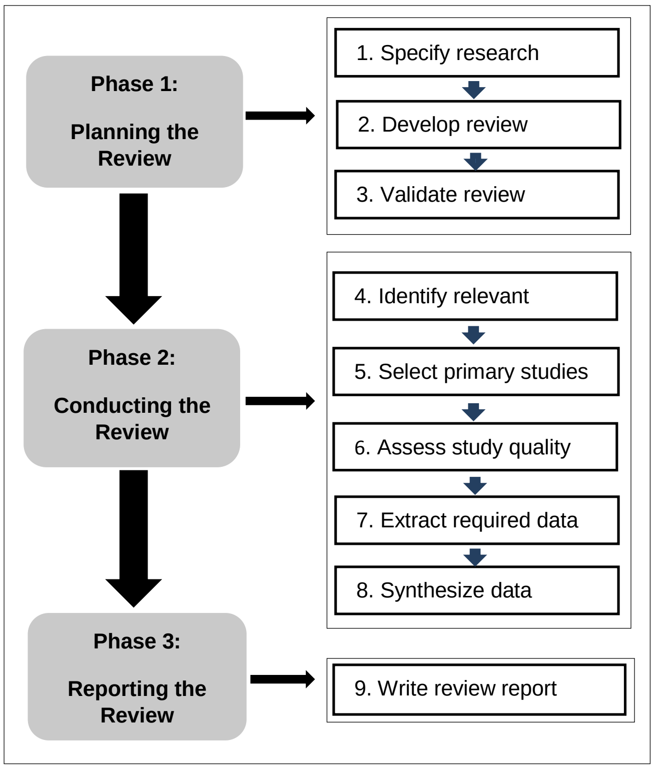


Figure 2.1: Systematic literature review process

An important element in SLRs is the development of a review protocol (Figure 2.2). This protocol specifies the background and procedures to be used by researchers to ensure rigor while conducting the review and reduces the possibility of researchers' bias throughout the review process.

The systematic review protocol begins by defining research questions to be answered followed by the search strategy to be followed to identify the primary studies (described in Sections 2.2.1 and 2.2.2). Then, the study selection criteria for determining which studies should be included or excluded from the surveyed literature is defined (Section 2.2.3). Then, quality assessment criteria are defined. These are used to assess the quality of the primary studies (Section 2.2.4). Finally, procedures for extracting and synthesizing data reported from primary studies are defined (Section 2.2.5).

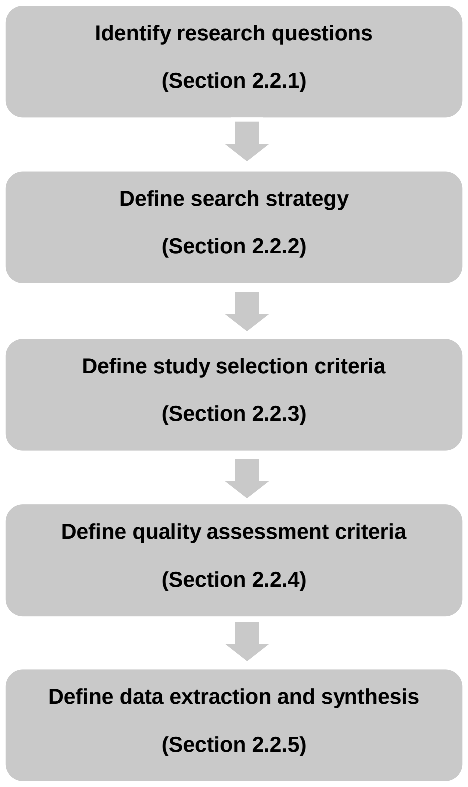

Identify research questions
(Section 2.2.1)
Define search strategy
(Section 2.2.2)
Define study selection criteria
(Section 2.2.3)
Define quality assessment criteria
(Section 2.2.4)
Define data extraction and synthesis
(Section 2.2.5)

Figure 2.2: SLR review protocol process

2.2.1 Research Questions

In order to achieve the research aim and objectives of this study, we defined the following 5 research questions.

RQ1: What tools have been developed to manage variability in software product lines?

RQ2: What are the characteristics of these tools?

RQ3: What is the quality of the research conducted in the reported approaches?

RQ4: What is the context of research?

RQ5: What are the main challenges faced by current Product Line Management (PLM) tools?

2.2.2 Search Strategy

Following Kitchenham's guidelines (Kitchenham and Charters, 2007), we constructed a search string to help us identify the relevant primary studies to answer our 5 research questions.

The guidelines followed were as follows:

- Derive main terms from the topic being researched and research questions;
- Determine and include synonyms, related terms and alternative spellings for major terms;
- Check the keywords in all relevant papers researchers already knew and those returned by initial searches on relevant databases;
- Include other relevant terms that increase the possibility of identifying further related material;

Use logical operators such as "OR" and "AND" to link alternative spellings and to join the synonym words or phrases to create one search string.

After constructing various search strings based on the guidelines above and performing a series of test searches in diverse digital libraries and analysing the outcome, the following search string was constructed:

<<Variability AND (Product Line OR Software Product Lines OR Software Product Family OR Software Product Families OR Product Family OR Product Families* OR Systems Family OR Family of Systems) AND (Variability OR Variability Management OR Variant OR Variation Point OR Feature Model OR Feature Modelling or Feature Modelling) AND (Tool OR Tools OR Approach, Approaches, Method* OR Methods)>>*

Although it was not possible to apply only one search string for all the electronic data sources, when varying the string for different sources we ensured that if the syntactic nature of the strings were not the same, they were all comparable semantically.

We also performed manual searches on different sources where SPL researchers were known to publish their findings, this included conferences and workshops. We searched for papers published between 1990 (i.e., when the first Feature-Oriented Domain Analysis [FODA] technical report was published (Kang et al., 1990)) up until February 2014 inclusive (when the search stage of this study was completed). Although only data reported in peer-reviewed published material was used in the analyses, we also attempted to acquire the identified tools. Where the tools weren't available for download or use online, the respective authors were contacted.

Our search covered 11 digital data sources as shown in Table 2.1. The manual search covered the proceedings of the following conferences and workshops:

- SPLC (Software Product Line Conference)

- VaMoS (Workshop on Variability Modelling of Software Intensive Systems)
- VisPLE (International Workshop on Visualisation in Software Product Line Engineering)
- WICSA (Working International Conference on Software Architecture)
- EWSA (European Workshop on Software Architecture)

Table 2.1: Electronic databases used for searching for primary studies

S/No	**Data Source Names**
1.	IEEEXplore
2.	ACM Digital Library
3.	SpringerLink
4.	ScienceDirect
5.	CiteSeerXLibrary
6.	Microsoft Academic Search
7.	Scopus
8.	IEEE Computer Society Digital Library
9.	EBSCOhost E-Journal Services
10.	Google Scholar
11.	Web of Science

Finally, forward and backward reference checking ("snowballing") was conducted on the identified primary studies. Search engines were used to find citations of the primary studies identified that could be of relevance to the review (forward reference checking). The reference lists of the primary studies were then checked for any potential relevant studies missed (backward reference checking).

2.2.3 Study Selection Criteria

This section explains the study selection process and lists the inclusion and exclusion criteria.

Inclusion Criteria (IC):

- **IC1:** The primary study is a peer-reviewed, scientific paper rather than a PowerPoint presentation or a short/extended abstract paper.
- **IC2:** The primary study discusses a variability management tool.
- **IC3:** When several reports of the same study existed in different sources, the most complete and recent version of the study was included in the review.
- **IC4:** The paper was written in English.

Exclusion Criteria (EC):

- **EC1:** The primary study does not address variability management tools.
- **EC2:** The papers were published before January 1991 and after February 2014.
- **EC3:** It is a short paper, PowerPoint file, poster presentation or consists of lecture notes.
- **EC4:** The primary study consists of a compilation of work, for instance, from a conference or workshop.

We found a total of 556 papers from different initial searches covering digital libraries, manual searches, and the works of known authors.

After the initial screening of paper abstracts, in which papers addressing non-SPL related topics were excluded by one researcher, 113 publications were selected. The full papers were then acquired and four independent researchers reviewed the studies. 47 publications were then selected through voting and discussions among the four researchers in a first step. Finally, and after another round carefully considering the inclusion and exclusion criteria, again through voting and discussions in case of disagreements, 37 studies were selected. Figure 2.3 below show a summary of the study selection process.

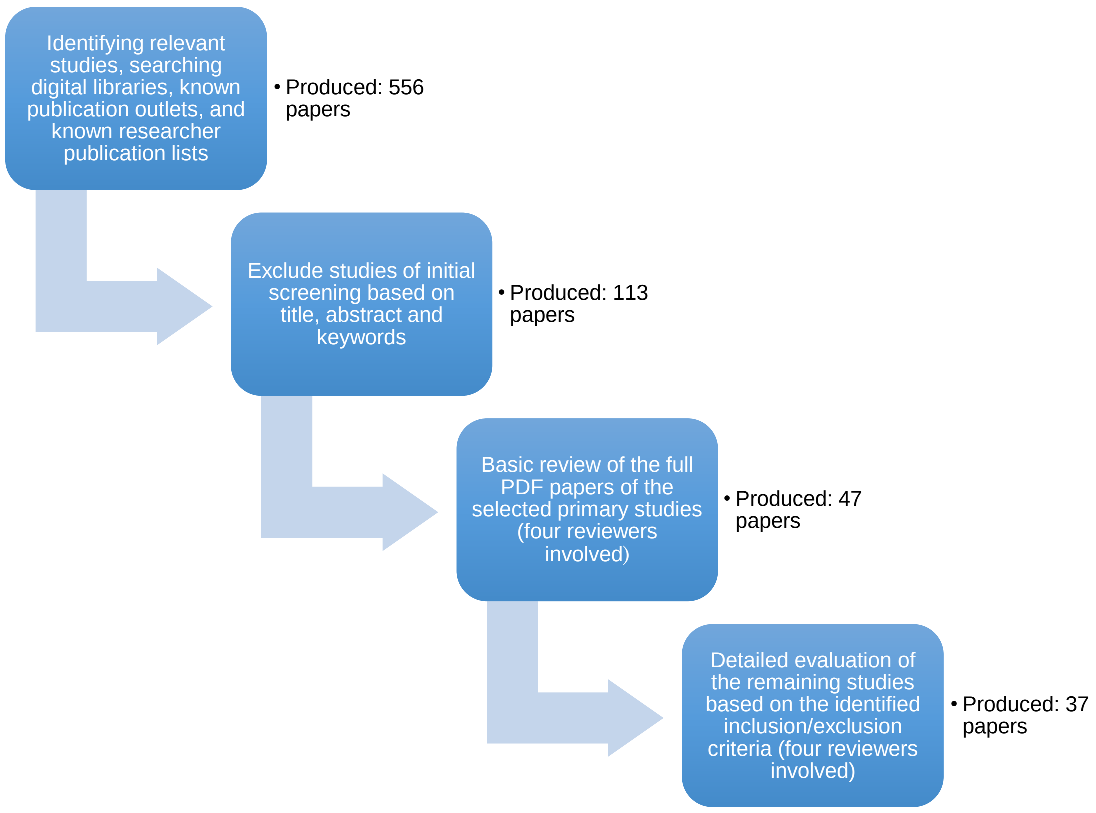

Identifying relevant studies, searching digital libraries, known publication outlets, and known researcher publication lists
• Produced: 556 papers
Exclude studies of initial screening based on title, abstract and keywords
• Produced: 113 papers
Basic review of the full PDF papers of the selected primary studies (four reviewers involved)
• Produced: 47 papers
Detailed evaluation of the remaining studies based on the identified inclusion/exclusion criteria (four reviewers involved)
• Produced: 37 papers

Figure 2.3: Study selection process

Finally, thirty-seven primary studies were analysed (see Table 2.3).

2.2.4 Quality Assessment Criteria

The quality of the reported research in the selected 37 papers was assessed based on the eight-quality assessment questions listed in Table 2.2 below. These were based on the quality assessment strategy defined in (Kitchenham and Charters, 2007). The studies were assessed using a ternary scale where each question was given a score of 1 (for Yes), 0.5 (for Perhaps) and 0 (for No). This system allowed us some flexibility when answering some of the questions that were difficult to judge as Yes or No from the information provided in the primary study. Once scores were allocated to questions, an aggregate mark was then given to each study. This data was also used to answer RQ3 (discussed in chapter 4).

Table 2.2: Quality assessment criteria

	Questions
QA.Q1	Is there a rationale for why the study was undertaken?
QA.Q2	Is there a description of the context (e.g., industry, laboratory setting, products used, etc.) in which the research was carried out?
QA.Q3	Did the paper present enough details about the variability management tool to enable us conduct the required analysis?
QA.Q4	Did the paper present an evaluation of the tool? If yes, did it include feedback from end users?
QA.Q5	Are the substantive claims in the paper supported by reliable evidence?
QA.Q6	Do the authors compare and evaluate their own results against related work?
QA.Q7	Do the authors discuss the credibility of their findings?
QA.Q8	Are limitations of the study discussed explicitly?

2.2.5 Data Extraction and Synthesis

Following the selection process, the 37 primary studies identified are shown in Table 2.3 below.

Table 2.3: Studies included in the final review

Study ID	Paper Title	Year of Publication	Author(s)	Reference
[S1]	DARE-COTS A Domain Analysis Support Tool	1997	Frakes, W., Priet-Diaz, R., and Fox, C.	(Frakes et al., 1997)
[S2]	Intelligent Design of Product Lines in Holmes	2001	Succi, G., et al.	(Succi et al., 2001)
[S3]	Scaling Step-Wise Refinement	2004	Batory, D., et al.	(Batory et al., 2004)
[S4]	XVCL: a mechanism for handling variants in software product lines	2004	Zhang, H. and Jarzabek, S.	(Zhang and Jarzabek, 2004)
[S5]	Tool Support for Software Variability Management and Product Derivation in Software Product Lines	2004	Gomaa, H. and Shin, M., E.	(Gomaa and Shin, 2004b)
[S6]	XML-Based Feature Modelling	2004	Cechticky, V., et al.	(Cechticky et al., 2004)
[S7]	On the Implementation of a Tool for Feature Modelling with a Base Model Twist	2006	Shakari, P. and Møller-Pedersen, B.	(Shakari and Møller-Pedersen, 2006)
[S8]	COVAMOF: A Framework for Modelling Variability in Software Product Families	2004	Sinnema, M., et al.	(Sinnema et al., 2004)
[S9]	Towards Systematic Ensuring Well-Formedness of Software Product Lines	2009	Heidenreich, F.	(Heidenreich, 2009)
[S10]	Odyssey: A Reuse Environment based on Domain Models	1999	Braga, R., M., M., Werner, C., M., L., and Mattoso, M.	(Braga et al., 1999)
[S11]	A NUI Based Multiple Perspective Variability Modelling CASE Tool	2010	Bashroush, R.	(Bashroush, 2010)
[S12]	The DOPLER meta-tool for decision-oriented variability modelling: a multiple case study	2011	Dhungana, D., Grünbacher, P., and Rabiser, R.	(Dhungana et al., 2011)
[S13]	XToF – A Tool for Tag-based Product Line Implementation	2010	Gauthier, C., et al.	(Gauthier et al., 2010)
[S14]	View Infinity: A Zoomable Interface for Feature-Oriented Software Development	2011	Stengel, M., et al.	(Stengel et al., 2011)
[S15]	FeatureIDE: An Extensible Framework for Feature-Oriented Software Development	2014	Thüm, T., et al.	(Thüm et al., 2014)
[S16]	FeaturePlugin: Feature Modelling Plug-In for Eclipse	2004	Antkiewicz, M. and Czarnecki, K.	(Antkiewicz and Czarnecki, 2004)

[S17]	An Integrated Software Management Tool for Adopting Software Product Lines	2012	Park, K., et al.	(Park et al., 2012)
[S18]	Kumbang Configurator – A Configuration Tool for Software Product Families	2005	Myllärniemi, V., et al.	(Myllärniemi et al., 2005)
[S19]	Towards a Model-Driven Product Line for Web systems	2009	Martinez, J., et al.	(Martinez et al., 2009)
[S20]	PuLSE-BEAT – A Decision Support Tool for Scoping Product Lines	2000	Schmid, K. and Schank, M.	(Schmid and Schank, 2000)
[S21]	Moskitt4SPL: Tool Support for Developing Self-Adaptive Systems	2012	Gómez, M., et al.	(Gómez et al., 2012)
[S22]	BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models	2012	Segura, S., et al.	(Segura et al., 2012)
[S23]	An Analysis of Variability Modelling and Management Tools for Product Line Development	2007	Capilla, R., et al.	(Capilla et al., 2007)
[S24]	Visualisation of variability and configuration options	2012	Pleuss, A. and Botterweck, G.	(Pleuss and Botterweck, 2012)
[S25]	ASADAL: A Tool System for Co-Development of Software and Test Environment based on Product Line Engineering	2006	Kim, K., et al.	(Kim et al., 2006)
[S26]	RequiLine: A Requirements Engineering Tool for Software Product Lines	2003	von der Maßen, T. and Lichter, H.	(von der Maßen and Lichter, 2004)
[S27]	ToolDAy: A Tool for Domain Analysis	2011	Lisboa, L., B., et al.	(Lisboa et al., 2011)
[S28]	The Linux Kernel Configurator as a Feature Modelling Tool	2008	Sincero, J. and Schröder-Preikschat, W.	(Sincero and Schroder-Preikschat, 2008)
[S29]	Automating Product-Line Variant Selection for Mobile Devices	2007	White, J., et al.	(White et al., 2007)
[S30]	Managing Feature Models with FAMILIAR: a Demonstration of the Language and its Tool Support	2011	Acher, M., et al.	(Acher et al., 2011)
[S31]	Easy-Producer – Product Line Development for Variant-Rich Ecosystems	2014	Eichelberger, H., et al.	(Eichelberger et al., 2014)

[S32]	OPTI-SELECT: an interactive tool for user-in-the-loop feature selection in software product lines	2014	El Yamany, A. E. Shaheen, M. and Sayyad, A.	(Yamany et al., 2014)
[S33]	MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing]	2014	Samih, H. and Bogusch, R.	(Samih and Bogusch, 2014)
[S34]	Variability code analysis using the VITAL tool	2014	Zhang, B. and Becker, M.	(Zhang and Becker, 2014)
[S35]	ViViD: a variability-based tool for synthesizing video sequences	2014	Acher, M., et al.	(Acher et al., 2014)
[S36]	VMC: recent advances and challenges ahead	2014	Ter Beek, M. H. and Mazzanti, F.	(Ter Beek and Mazzanti, 2014)
[S37]	WebFML: synthesizing feature models everywhere	2014	Bécan, G., et al.	(Bécan et al., 2014)

Beside the 37 primary studies included in the study, we identified further 13 tools that did not meet the inclusion/exclusion requirements. These are shown in Table 2.4 below, along with the criteria they didn't meet.

Table 2.4: Studies excluded in the final review

Reasons for Exclusion	Paper Title	Year of Publication	Author(s)	Reference
EC3	FAMA Framework	2008	Trinidad, P., Benavides, D., Ruiz-Cort´es, A., Segura, S., Jimenez, A.	(Trinidad et al., 2008)
EC1	Development of a Feature Modelling Tool using Microsoft DSL Tools	2009	Fernández, R., Laguna, M. A., Requejo, ,J., Serrano, N.	(Fernández et al., 2009)
EC3	S.P.L.O.T. - Software Product Lines Online Tools	2009	Mendonca, M.,	(Mendonca et al., 2009)
EC3	V-Manage	2002	European Software Institute (ESI)	(*SAP Configurator*)
EC2	PACOGEN : Automatic Generation of Pairwise Test Configurations from Feature Models	2011	Hervieu, A., Baudry B., Gotlieb, A.	(Hervieu et al., 2011)

EC1	Variability Modelling in the Real: A Perspective from the Operating Systems Domain	2010	Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.	(Berger et al., 2010)
EC1	MetaProgramming Text Processor		Campbell, G.	(Campbell)
EC1	An Algorithm for Generating t-wise Covering Arrays from Large Feature Models	2012	Johansen, F., M., Haugen, Ø., Fleurey, F.	(Johansen et al., 2012)
EC2&EC3	Varmod-Tool-Environment	2005	Pohl, K., Böckle, G., van der Linden, F.	(Klaus et al., 2005)
EC3	Linux Variability Analysis Tools (LVAT)		She, S.	(She)
EC2	VARMA--VARiability Modelling and Analysis Tool	2012	Russell, G., Burns, F., Yakovlev, A.	(Russell et al., 2012)
EC3	ZIPC SPLM	2009	NTTDaTa MSE Corporation	(Gauthier et al., 2010)
EC3	Hydra Tool	2009	Jose R. Salazar	(Modeling)

Upon the completion of the primary study selection phase, and the primary study quality assessment step, data extraction commenced. In order to answer the research questions, the following data was extracted from every primary study (see Table 2.5). The data extraction form below also shows the relevance of each of the extracted data elements to the study research questions.

Table 2.5: Data extraction form

Data Field	**Related Concern/Research Question**
DE.Q1 Paper title	Documentation
DE.Q2 Year of publication	Documentation
DE.Q3 Type of publication (e.g. Journal, Conference, Workshop, etc.)	Reliability of Review
DE.Q4 Publication outlet (conference name, etc.)	Reliability of Review
DE.Q5 Paper brief description (synopsis)	RQ1, RQ3

DE.Q6 The research rationale, challenges or problems as reported in the paper	RQ3, RQ5
DE.Q7 Research Context (e.g. industry, academic, product, etc.)	RQ4
DE.Q8 Tool Performance and Stability	RQ2, RQ5
DE.Q9 Visualisation technique	RQ2
DE.Q10 Textual notation	RQ2
DE.Q11 Usability	RQ2
DE.Q12 Tool environment/Platform	RQ2
DE.Q13 Integration (e.g. with DOORS, etc.)	RQ2
DE.Q14 Scalability (ability to deal with large-scale models)	RQ2
DE.Q15 Relevance (Research or Practice)	RQ4
DE.Q16 The research limitations as reported in the paper	RQ5

2.3 Data Extraction Results

The next step after the data extraction step was the data synthesis and analysis step. In this section, we provide meta-analyses of the primary studies relating to their publication types, venues, trends and overall characteristics. We analyse the collected data to address the 5 main research questions of the study.

Based on the data collected, the research question one (RQ1) is then addressed in details in chapter 3. In chapter 4, the remaining four questions (RQ2, RQ3, RQ4, and RQ5) were then answered. In addition, the chapter 3 also discusses additional findings on commercial tools and tool adoption in industry. Chapter 4 discusses the study limitations and threats to validity. And finally, it rounds off the analysis with summary and conclusions.

2.3.1 Trend Analysis

The first search of the systematic literature review resulted in 556 papers. The application of inclusion/exclusion criteria in several iterations resulted in 37 papers for the final review.

The primary studies included 18 conference papers, 6 journal papers, and 13 workshop papers. Figure 2.4 presents a pie chart showing the percentage for each publication outlet. From the chart, it can be seen that conferences are more prominent venues for research on variability management tools followed by workshops, whereas journals seem to be less attractive outlets for research on tools. The 37 papers are scattered over 24 different venues (see Table 2.3). This distribution further highlights the importance of this systematic review as a manual search of well-known conferences or journals could not possibly identify all the relevant literature.

Figure 2.5 shows the distribution of studies over time; combined in 5-year intervals (to avoid influence of events occurring every 18 or 24 months). The chart shows that there has been considerable surge in new tools over the past 5 years. Our search did not identify any relevant paper published before 1997. The figure shows that there was a peak in publishing research on variability management tools from 2011 to 2015. There was a gradual uprising from 1996 to 2000, then a steady uptrend from 2001 to 2005 and 2006 to 2010. The shape of the curve (spike, followed by trough, then slow pickup) aligns nicely with Gartner's technology maturity Hype Cycle model (Linden and Fenn, 2003). Comparing the publication timeline (Figure 2.5) with Gartner's

Hype Cycle (Figure 2.6), it can be deduced that variability management tools have now entered the slope of enlightenment/plateau of productivity stage. This indicates that the benefits of variability management tools to the enterprise are starting to become widely understood, while conservative companies remain cautious.

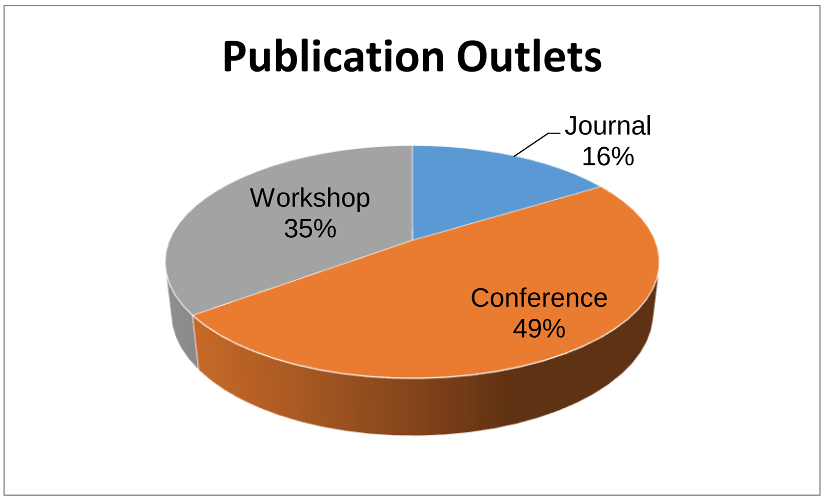


Figure 2.4: Percentage of each publication type

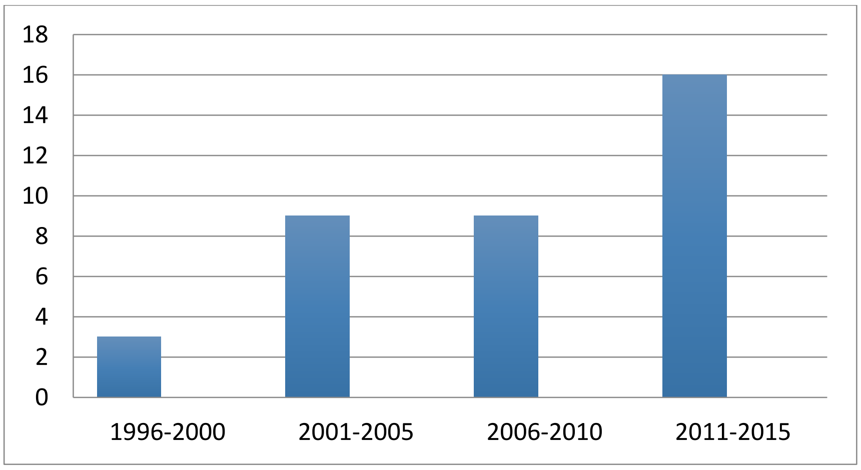
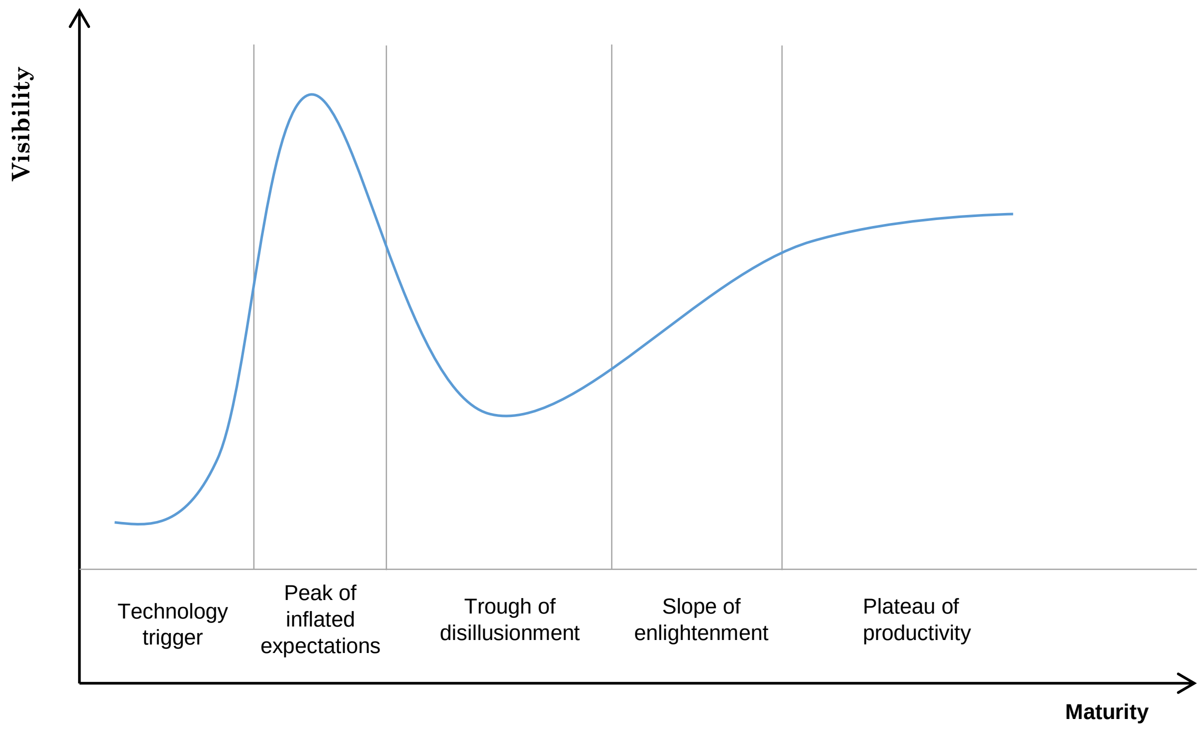


Figure 2.5: Distribution of primary studies over time



Each Hype Cycle drills down into the five key phases of a technology's life cycle.

Technology Trigger: A potential technology breakthrough kicks things off. Early proof-of-concept stories and media interest trigger significant publicity. Often no usable products exist and commercial viability is unproven.

Peak of Inflated Expectations: Early publicity produces a number of success stories—often accompanied by scores of failures. Some companies take action; many do not.

Trough of Disillusionment: Interest wanes as experiments and implementations fail to deliver. Producers of the technology shake out or fail. Investments continue only if the surviving providers improve their products to the satisfaction of early adopters.

Slope of Enlightenment: More instances of how the technology can benefit the enterprise start to crystallize and become more widely understood. Second- and third-generation products appear from technology providers. More enterprises fund pilots; conservative companies remain cautious.

Plateau of Productivity: Mainstream adoption starts to take off. Criteria for assessing provider viability are more clearly defined. The technology's broad market applicability and relevance are clearly paying off.

Figure 2.6: Gartner Hype Cycle reproduced based on (Linden and Fenn, 2003)

2.3.2 Phrase Map Analysis

Phrase maps were used to conduct a thematic contextual analysis of the complete text of the primary studies (excluding lists of references, author descriptions and the most commonly occurring non-technical words like introduction, figure, etc.) to try and identify evolving trends in the field. These maps visualise two main features of the text: (1) connections between terms are depicted by the grey lines, where a thicker line corresponds to a stronger relationship between the terms; and (2) the centrality of the terms which are portrayed by their font size (the bigger the font, the more frequently a term appears in the text).

The primary studies were divided into two batches, the first batch contained studies published between 1997 and 2006 inclusive, and the second batch contained studies published between 2007 and 2015 inclusive. Then, phrase maps were created to show the frequency and relationship of the most common keywords in these groups of papers. The phrase maps showed the top 40 occurring keywords (ignoring common and connecting words such as ‘and’, ‘the’, ‘of’ etc.) and used one place space between terms to determine the connections between the terms. The phrase maps for the two batches can be seen in Figure 2.7 and Figure 2.8.

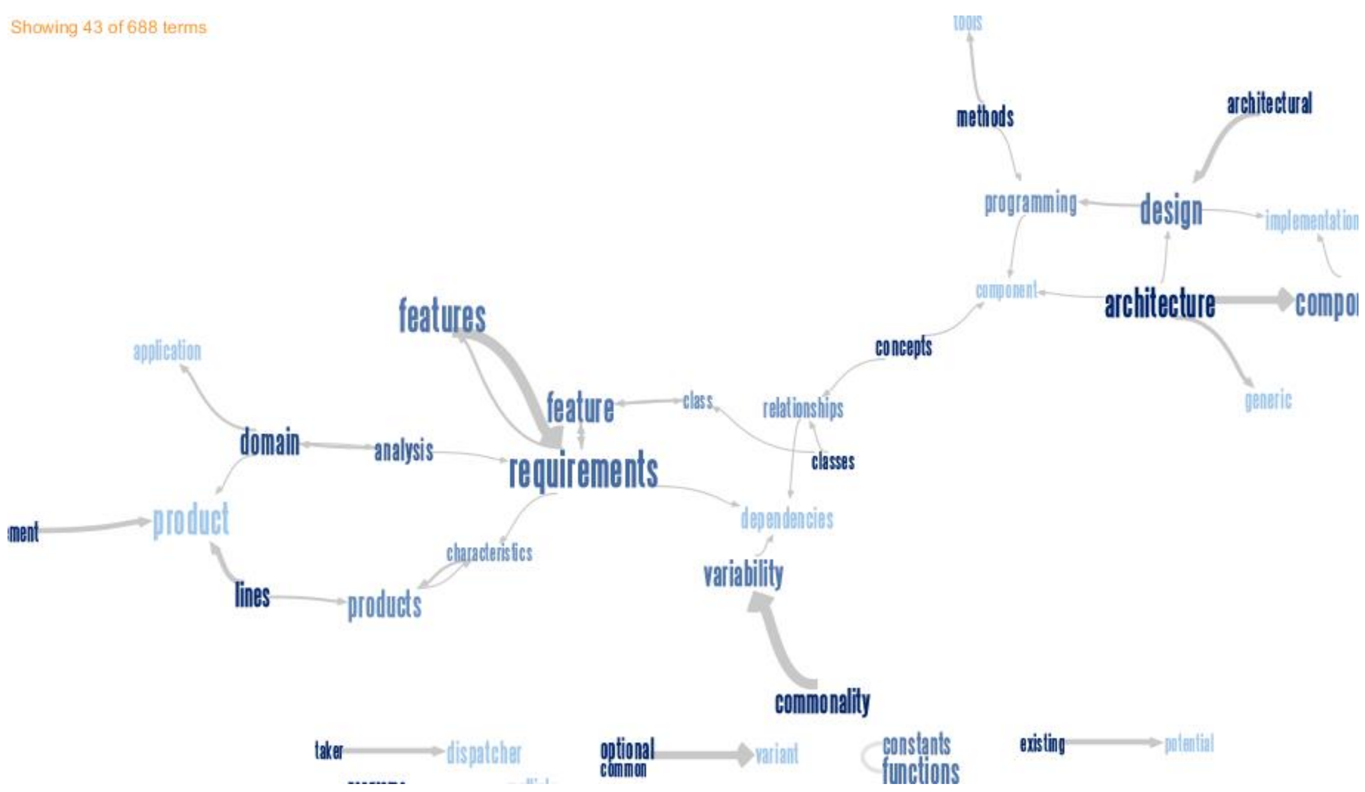


Figure 2.7: Phrase Map of studies published between 1997-2006

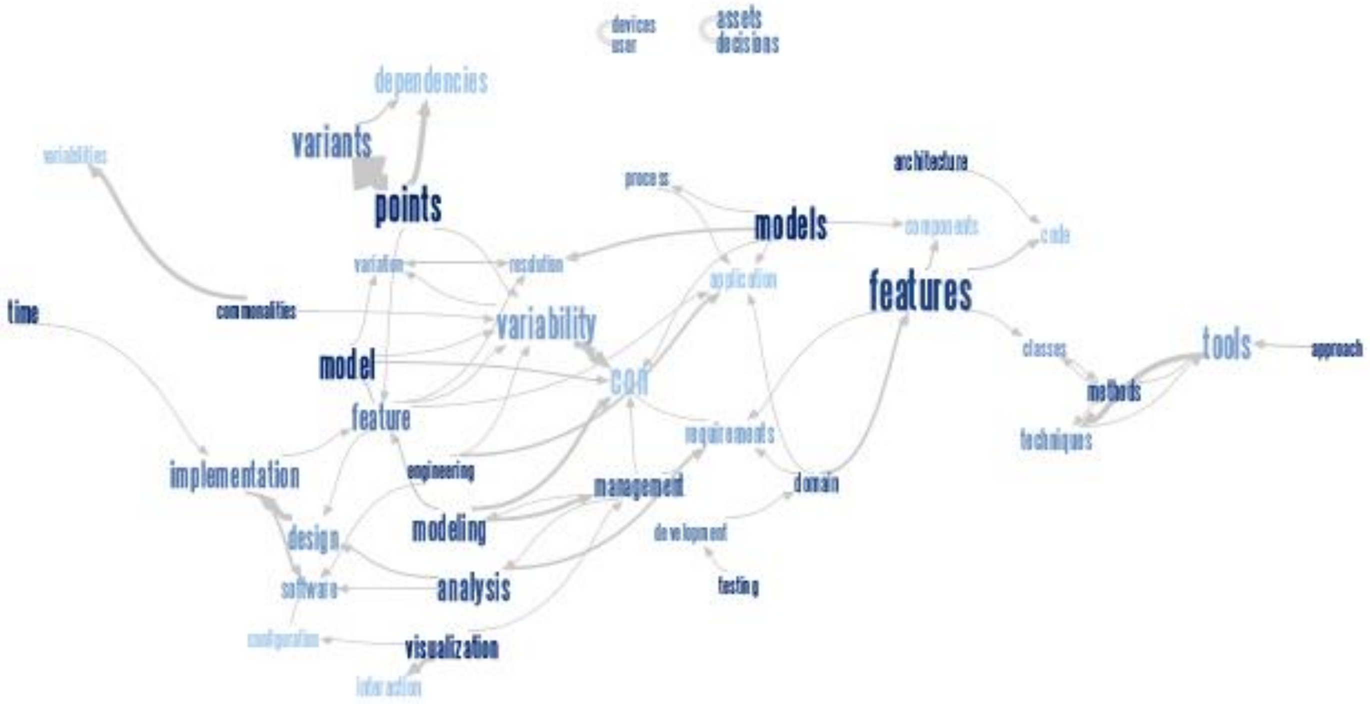
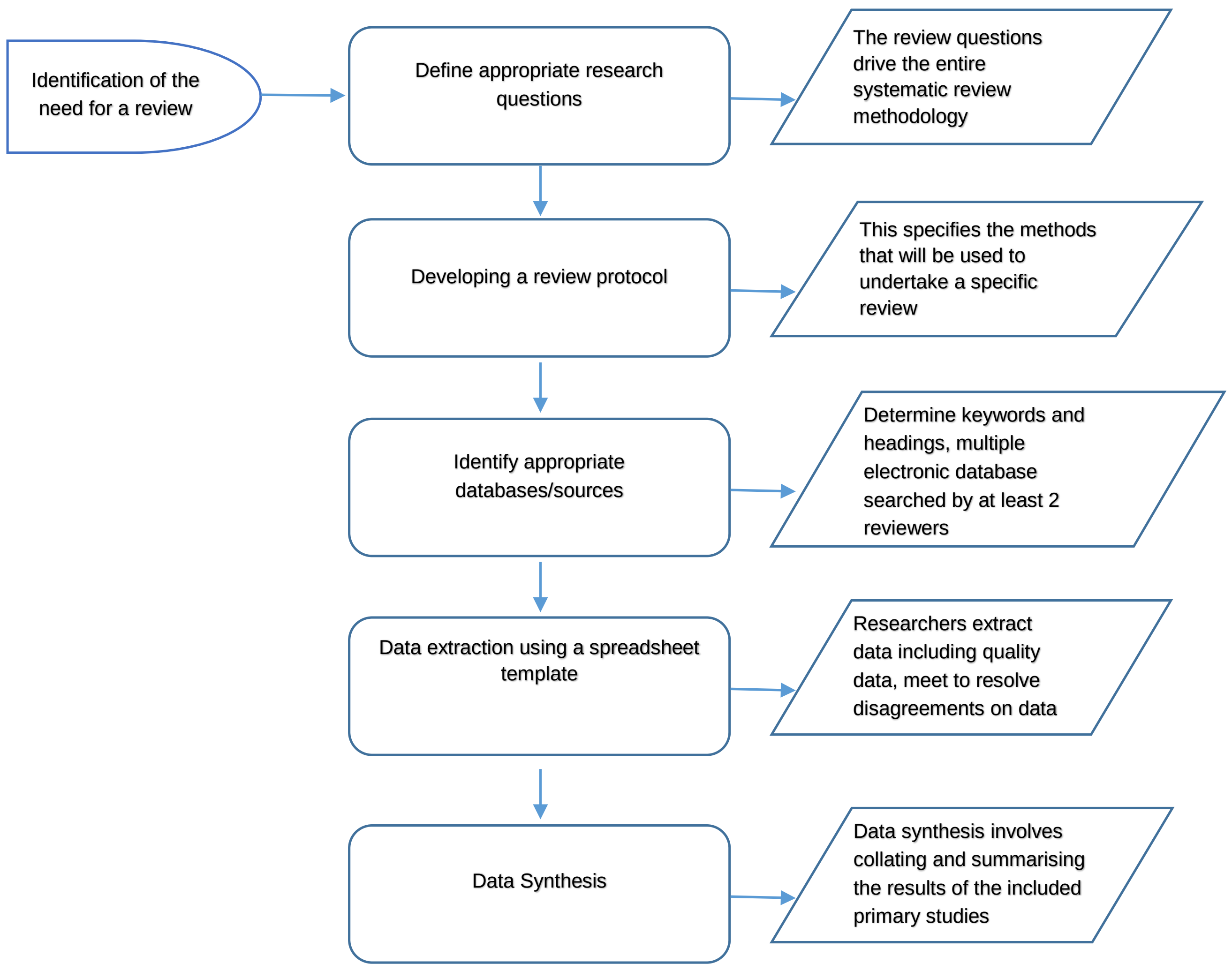


Figure 2.8: Phrase Map of studies published between 2007-2015

The first observation that can be made when looking at the two phrase maps is that in the first batch (1997-2006) fewer and simpler relationships existed among words. There were classical associations between terms such as features-requirements, commonality-variability, architecture-design, etc. However, looking at the second batch (2007-2015), the associations between words became more integrated and diverse.

The thematic characteristic of the 1997-2006 text shows several scattered connections between two and three words (seen at the bottom of the map in Figure 2.7). Figure 2.8, however, shows a tighter network of terms with diverse relationships (and no isolated connections). This could be an indication of the level of consolidation and maturity in the domain. Moreover, in the first batch, basic variability management concerns seemed to dominate (e.g., requirements, commonality, architecture, etc.). In the second batch, there was an emergence of terms such as visualisation and analysis, which emerged as key focus areas for tool developers. Figure 2.9 presents a summary of the entire SLR review process.



Identification of the need for a review
Define appropriate research questions
The review questions drive the entire systematic review methodology
Developing a review protocol
This specifies the methods that will be used to undertake a specific review
Identify appropriate databases/sources
Determine keywords and headings, multiple electronic database searched by at least 2 reviewers
Data extraction using a spreadsheet template
Researchers extract data including quality data, meet to resolve disagreements on data
Data Synthesis
Data synthesis involves collating and summarising the results of the included primary studies

Figure 2.9: Summary of SLR review process

2.4 Summary

This chapter describes the method used to identify, collect, and review papers, in the systematic literature review (SLR) process, using SLR to collect data in order to achieve the objective of this research. This method involves: identifying of the key research questions to be answered; the search strategy used to identify the relevant primary studies; the study selection criteria used in the process of inclusion or exclusion a paper, and; the quality assessment criteria used to assess the quality of the selected papers together with the data extraction and synthesis. The chapter also presents the overall meta-analysis of the selected primary studies based on the data extraction results.

Chapter 3

Existing Variability Management Tools in Software Product Lines

3.1 Introduction

This chapter introduces the concept of variability management, which is the central activity used to manage the commonality and variability that provides the ability to adapt and customise software artefacts for a particular context or setting. The chapter also discusses the locations within software artefacts at which variability actually occurs, usually referred to as variation points, as well as their number of possible occurrences, known as the variants. It also presents an overview of dependency management.

However, based on the information provided by the authors in the literature, the chapter discusses on a number of tools supporting variability management in terms of their usable functionality, i.e. the approach it uses in tackling variability issues, the technology used and environment or a platform based on which a tool was developed and, implemented respectively. The type of notation (graphical, textual or a combination of both) employed by a tool and the category to which a tool belonged to as commercial, academic, or both, are also explicated (see Appendix A for details). Finally, the chapter assesses whether a tool is an open source or if an evaluation copy could be obtained, in addition to a discussion on commercial tools and tool adoption in industry.

3.2 Variability Management

Variability management remains the main challenge in software product line (SPL) adoption, as it needs to be efficiently managed at different levels of the SPL development process (for example, requirements analysis, software design, implementation, etc.). However, effective management of variability is essential for successful product line development, as it determines and enables the creation of different products in a product line.

Variability has been defined in (Van Gurp et al., 2001) as the ability of a software system or artefact to be changed or customised for use in a specific context. This means that good variability for a software system should expect changes and allow for the implementation of those changes over time throughout the life cycle. Due to the large number of variability points within a real-life industrial product line, some variation points depend on other variation points. For instance, a variation point cannot be selected unless another variation point is implemented (requires dependency). On the other hand, it is possible that some variation points cannot be supported in the same product at the same time (excludes dependency).

3.2.1 Variation Point

Variation points are locations in the design or implementation at which changes occur (Jacobson et al., 1997), such as the type of screen size that a mobile phone can offer (large size, medium size, and small size). Therefore, variation points provide a description of existing differences. Hence, the variability of a product line is defined by variation points. The variability of features is usually represented as a tree in which variation points consist of a parent feature, a group of child features called variants, and multiplicity that specifies the possible number of variants that can be selected from the variation point during the configuration of a product (Pohl et al., 2005).

For a better understanding of variability, we used a variability meta-model based on the ideas in (Moon et al., 2005) and (Thiel and Hein, 2002) (see Figure 3.1). The variability of SPL is represented by a variation point. Associated with each variation point, there is one or more description(s) of possible choices, called variants, to replace the variation point.

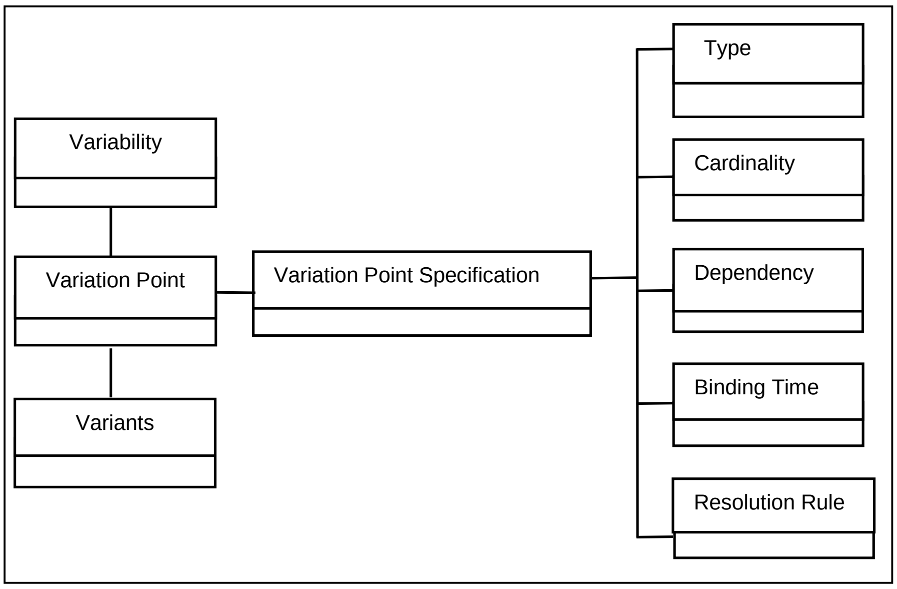


Figure 3.1: Variability Meta-model for representing concepts in variability (reproduced from (Kadir and Mohammad, 2008))

However, for easy selection and adaptation, variation points need to be specified as follows: (1) Variation type, which clarifies what varies and how it varies, and has been divided into four types: computational, external, control, and data types (Moon et al., 2005); (2) Variation point cardinality specifies the minimum and maximum number of variants that can be selected for a variation point (Halmans and Pohl, 2003); (3) Dependency represents the relationships and constraints at one or more variation points (Sinnema et al., 2006); (4) Binding time, which is the time when a variation point is bound to a chosen variant (Krueger, 2006). Finally, resolution rules, according to (Thiel and Hein, 2002) are the applied strategies when binding a variation point with a conflict, and that must be resolved as part of the product architecture design.

3.2.2 Variant

Variability enables the choice between different possibilities through variation points; each of these various options is referred to as a variant. The variant pinpoints a single choice of a variation point (Bachmann and Clements, 2005). Using the same example given in Section 3.1, every different choice of screen size for a particular mobile phone (big, medium or small) is represented by a variant. Therefore, through the different choice of variants, one mobile phone can differ from another in terms of size. Therefore, variants provide different possibilities to satisfy variation points.

3.2.3 Dependency Management

Variability dependencies are the constraints on the variant selection at one or more variation points (Sinnema et al., 2006). Effective and scalable ways of representing variability dependencies in a large-scale product line is a challenge and of primary concern in software product line engineering. As highlighted in Section 3, for instance, it is possible that Feature B must be present if Feature A is selected (inclusivity), while the choice of Feature A is based on a different condition.

3.3 Variability Management Tools in Software Product Lines

The efficient management of variability can give rise to the successful customisation of software products, which can result in high market success (Van Gurp et al., 2001). However, according to (Beuche et al., 2007), variability in today's software product lines has such complexity that the use of appropriate tools to support it has become crucial. Therefore, the management of variability needs sufficient tool support (Beuche and Spinczyk, 2003).

In their work, (Zhang and Jarzabek, 2001) asserts that for the effective handling of a scalability problem, a tool capable of interpreting and manipulating domain models is necessary to provide analysts with customised and simple domain views. (Jaring and Bosch, 2002) believe that tool support is especially necessary when developing a large-scale system. (Djebbi and Salinesi, 2006) ascertain that many notations can only be scalable if supported by an appropriate tool. Hence, tool support is of paramount importance for variability management, since without proper automation or tool support, the modelling variability of a large-scale model is boring, error-prone and difficult to conduct (Ferber et al., 2002).

Researchers in the community, from both academia and industry, devoted large amounts of time and resources in trying to find efficient and effective ways to deal with variability-related challenges. As a result, a wide variety of tools, approaches, techniques and methods were proposed.

Table 3.1 provides a list of all the tools identified in the SLR in chronological order. For details on each of the tools listed in the table see Appendix A.

Table 3.1: Identified tools with their year of introduction

Tools Name	**Technology/Implementation**	**Notations Supported**	**Availability Free/Evaluation Copy**	**Year of Introduction**
DARE-COT	C-language on a UNIX workstation	Textual notation	Open source	1997
Odyssey	Java technology	UML notations	Open source	1999
PuLSE	Visual Basic	Textual notation	Not available	2000
Holmes	Java language	Textual notations	Not available	2001
RequiLine	Java language	Graphical notation		2003
COVAMOF	Java technology	Graphical notation	Not available	2004
Feature Modelling Plug-In	Java language	Graphical and textual	Open source	2004
PLUSEE	Rational Rose and Rational Rose RT	Graphical notation	Not available	2004
XML-Based Feature Model	XML technology	Textual notation	Not available	2004
AHEAD	Java language	Textual notation	Open source	2004
XVCL	Java and XML technology	Textual notation	Open source	2004
KUMBANG	Java language	UML-like notations	Open source	2005
BVR: Base-Variation-Resolution	Java Technology	Graphical notation	Open source	2006
ASADAL (A System Analysis and Design Aid tooL)	Java Technology	Graphical notation	Not available	2006

Scatter Tool	Eclipse Generative Modelling Technology	Graphical notation	Not available	2007
VMWT	PHP and Ajax	Textual notation	Open source	2007
L K C- Feature Modelling Tool	Linux technology	Graphical and textual	Open source	2008
FeatureMapper	Eclipse Modelling Framework	Graphical notation	Open source	2009
PLUM	Eclipse Modelling Framework	Graphical notation	Open source	2009
MUSA	Java and XML	Graphical notation	Not available	2010
XToF – A Tool for Tag-based Product Line Implementation	Java and C languages, XML	Textual notation	Open source	2010
ToolDay	Eclipse's Graphical Modelling Framework	Graphical notation	Not available	2011
View Infinity	Java technology	Graphical notation	Open source	2011
FAMILIAR	Java technology	Graphical and textual	Open source	2011
DOPLER	Java technology	Graphical and textual	Not available	2011
FeatureIDE	Java technology	Graphical and textual	Open source	2012
ISMT4SPL	Java technology	Graphical and textual	Not available	2012
BeTTy	Java technology	Graphical notation	Open source	2012
MOSKitt4SPL	Eclipse Modelling Framework	Graphical notation	Open source	2012
S2T2 Configurator	Java technology	Graphical notation	Open source	2012
Easy-Producer	Java technology	Graphical and textual	Open source	2014
OPTI-SELECT	Simple XML Feature Model	Textual notation	Open source	2014
MPLM-MaTeLo product line manager	Eclipse Rich Client Platform	tree-like notations	Not available	2014
Variability code analysis using the VITAL tool	C-Preprocessor	Textual notation	Not available	2014
ViViD: a variability-based tool for synthesizing video sequences	Xtext language workbench	Textual notation	Not available	2014
VMC: recent advances and challenges ahead	HTML technology	Graphical notation	Not available	2014
WebFML: synthesizing feature models everywhere	JavaScript technology	Graphical notation	Not available	2014

3.4 Commercial Tools and Tool Adoption in Industry

In addition to our SLR, we conducted a web search on commercially available variability management tools as well as studies on tool adoption in industry/practice. In this section, we briefly discuss the commercial tools we identified and the findings of these studies.

3.4.1 Commercial Variability Management Tools

We explicitly focus on tools developed to support variability management in software product line engineering, thus leaving out commercial tools developed in other communities such as the CWAdvisor (Felfernig et al., 2001) or the SAP Configurator (*SAP Configurator*), which follow an AI-based process or MetaEdit+ (Tolvanen and Kelly, 2009), which is a domain-specific language and code generation environment. Industry also has extended other commercial tools with support for variability management (Berger et al., 2013), typically without following a particular product line engineering process. For example, IBM Rational DOORS (*IBM Rational DOORS*) comes with a requirements management add-on that allows to define variability in requirements documents.

SparxSystems Enterprise Architect (*SparxSystems Enterprise Architect*) has also been extended with variability management support. A very common approach followed by industry is to use Microsoft Excel or Microsoft Word to document the variability of their software systems.

All these commercial or industry solutions to variability management work very well for the context they have been developed for, but do not follow any particular product line engineering approach. We could only identify two commercial tools developed for product line variability management, more specifically, pure::variants (Beuche, 2008) and Gears (Krueger and Clements, 2014).

pure::variants (Beuche, 2008) is developed by pure-systems GmbH in Magdeburg, Germany. The tool supports variant management and product configuration based on feature models and has a strong focus on interoperability and extensibility. For example, the tool can be integrated in the Eclipse IDE, used with a web browser, as a command line client, and even in a custom application. Several extensions to existing commercial-off-the-shelf tools exist, e.g., to DOORS or SAP. Four types of models can be created and managed with pure::variants:

(1) Feature Models represent the variability of a system. (2) Family Models represent the variants of assets that can be selected. (3) Variant Description Models are used to store the selected features and their values. (4) Result Models based on 1-3 represent one concrete instance derived from a product line. Constraints on model elements can be defined in a self-defined dialect of the language prolog. A prolog-based constraint solver allows validating selected configurations. The main benefits of pure::variants are (i) the strong focus on interoperability and extensibility, (ii) the high number of available extensions, and (iii) the comprehensive support for model checking and validation (also during product configuration).

The main drawbacks are that (i) the tool has mainly been designed for engineers and (ii) the representation of features (tree-structure) does not scale well for very large systems. pure::variants is well-suited for use in industry as demonstrated by the various successful application in industry.

Gears (Krueger and Clements, 2014) is a commercial tool developed by BigLever Software Inc., Austin, Texas, USA. The tool has been developed in Java and supports the three-tiered methodology proposed by Krueger (Krueger, 2007). The tool allows defining arbitrary reusable software assets and a product feature profile that describes products in terms of features. Gears focuses on products: feature profiles define the products that can be built from assets and the optional and alternative choices that can be made for each product. Product configuration is supported by the Gears Configurator which automatically assembles and configures assets to produce products based on feature choices made using feature profiles. Gears can be tailored to different environments with parameter sets representing different kinds of variability. Dependencies are modelled as global constraints that are checked during configuration.

The main benefits of Gears are (i) its strong focus on producing products, (ii) the possibility to use arbitrary assets, and (iii) its methodological foundation given by the three-tiered methodology. The main drawback of Gears is that applying the tool to a concrete industrial case requires significant tailoring of the tool depending on the used assets and the environment to integrate it with.

3.4.2 Tool Adoption in Industry

Djebbi et al. (Djebbi et al., 2007) report findings of a study on the ability of product line management tools to answer industry needs. They identified 12 tools through an unsystematic search (we cover most of them on our SLR) but only analysed four tools in detail based on their availability. These four tools were RequiLine [S26], pure::variants (Beuche, 2008), XFeature [S6] and DOORS-TREK (an add-on to IBM Rational DOORS)(*DOORS-TREK*). Djebbi et al. describe these tools and discuss the support of these tools for variability modelling as well as the support for management (such as reporting capabilities) they provide. They conclude that tools developed in industry or in industry projects work well for the context they have been developed for but are hard to apply in other contexts.

Berger et al. (Berger et al., 2013) report the results of a survey on variability modelling in industrial practice. Among other questions, they asked industrial practitioners what variability modelling tools they use. Respondents could select from 10 particular tools or specify an open answer. pure::variants (Beuche, 2008) was the most used tool, followed by Gears (Krueger and Clements, 2014). From the tools we identified in our SLR, FeatureIDE [S15], DOPLER [S12], X-Feature [S6], and AHEAD [S3] were the only ones mentioned by respondents. This confirms our findings on the difficulty of research tool adoption in industry.

As Berger et al. conclude “all other tools play only a minor role in the participating projects” and were only reported as being used once or twice.

The answers of the 42 survey respondents were analysed in detail and it was found that many respondents use “other open source tools”, “other commercial tools”, or “home-grown domain-specific tools”. A key finding regarding variability modelling tool support of the survey was that there exists a wide variety of home-grown solutions developed in industry that are unknown to researchers. Our SLR would allow industrial practitioners to check what research tools are available before implementing their own solution.

Lettner et al. (Lettner et al., 2013) confirm the findings of Berger et al.’s survey when they report that industry often develops custom solutions to automate the configuration process of their variable software systems. These solutions are often not based on variability models but describe configuration knowledge directly in code or in simple XML files. Comparing a custom-developed with a model-based configuration approach leads them to the conclusion that using a model-based solution could be very beneficial for industry. For instance, it would help to decouple configuration UI and variability information and make the approach more adaptable and extensible. Again, our SLR could be an important first source of information for industrial practitioners thinking of implementing a variability management tool.

3.5 Summary

This chapter presents and discusses the concept of variability management, variation points, variants and the dependency relationships that exist among them. Finally, the chapter discusses an overview of 37 tools for managing variability in software product lines, along with a number of commercial tools and tools that have been adopted in industry. Such an overview mainly focused on tool functionality, technology used for development and implementation, and type of notations supported. Other key characteristics being considered are whether a tool can be found for free or if an evaluation copy can be obtained.

Chapter 4

Critical Analysis of Existing Approaches

4.1 Introduction

After having each of the 37 variability management tools identified in the survey examined in terms of their functionality and the platform on which they were implemented, along with the commercial tools and tool adoption in industry in the previous chapter, this chapter presents a detailed analysis of the state-of-the-art for the research field, particularly in terms of tools supporting variability management to understand the tools' characteristics, maturity, and the challenges in the field.

In the first half of the chapter, we begin by identifying the different tools (assigning a unique ID for each of the 37 tools identified) based on how they will be studied (see Table 4.1). The tools were then analysed in terms of the following topics: development environment; support for transformations (between different formats); management of constraints and reasoning on variability models; and their proposed graphical and textual notations. The second half of the chapter presents an analysis based on the quality of the research conducted in the reported approaches. We also present the research context of the studies that have been conducted. This includes whether studies are academically-based, joint academic-industrial endeavours, or if no information was provided on the research context. Finally, we discuss the main challenges faced by current Product Line Management (PLM) tools.

Table 4.1: Identified tools with assigned ID and their technical details

Tools Name	Technology/Implementation	Notation Supported	Availability Free/Evaluation Copy	Study ID
DARE-COT	C-language on a UNIX workstation	Textual notation	Open source	[S1]
Odyssey	Java technology	UML notations	Open source	[S10]
PuLSE	Visual Basic	Textual notation	Not available	[S20]
Holmes	Java language	Textual notations	Not available	[S2]
RequiLine	Java language	Graphical notation		[S26]
COVAMOF	Java technology	Graphical notation	Not available	[S8]
Feature Modelling Plug-In	Java language	Graphical and textual	Open source	[S16]
PLUSEE	Rational Rose and Rational Rose RT	Graphical notation	Not available	[S5]
XML-Based Feature Model	XML technology	Textual notation	Not available	[S6]
AHEAD	Java language	Textual notation	Open source	[S3]
XVCL	Java and XML technology	Textual notation	Open source	[S4]
KUMBANG	Java language	UML-like notations	Open source	[S18]
BVR: Base-Variation-Resolution	Java Technology	Graphical notation	Open source	[S7]
ASADAL (A System Analysis and Design Aid tooL)	Java Technology	Graphical notation	Not available	[S25]
Scatter Tool	Eclipse Generative Modelling Technology	Graphical notation	Not available	[S29]
VMWT	PHP and Ajax	Textual notation	Open source	[S23]
L K C- Feature Modelling Tool	Linux technology	Graphical and textual	Open source	[S28]
FeatureMapper	Eclipse Modelling Framework	Graphical notation	Open source	[S9]
PLUM	Eclipse Modelling Framework	Graphical notation	Open source	[S19]
MUSA	Java and XML	Graphical notation	Not available	[S11]
XToF – A Tool for Tag-based Product Line Implementation	Java and C languages, XML	Textual notation	Open source	[S13]
ToolDay	Eclipse's Graphical Modelling	Graphical notation	Not available	[S27]

	Framework			
View Infinity	Java technology	Graphical notation	Open source	[S14]
FAMILIAR	Java technology	Graphical and textual	Open source	[S30]
DOPLER	Java technology	Graphical and textual	Not available	[S12]
FeatureIDE	Java technology	Graphical and textual	Open source	[S15]
ISMT4SPL	Java technology	Graphical and textual	Not available	[S17]
BeTTy	Java technology	Graphical notation	Open source	[S22]
MOSKitt4SPL	Eclipse Modelling Framework	Graphical notation	Open source	[S21]
S2T2 Configurator	Java technology	Graphical notation	Open source	[S24]
Easy-Producer	Java technology	Graphical and textual	Open source	[S31]
OPTI-SELECT	Simple XML Feature Model	Textual notation	Open source	[S32]
MPLM-MaTeLo product line manager	Eclipse Rich Client Platform	tree-like notations	Not available	[S33]
Variability code analysis using the VITAL tool	C-Preprocessor	Textual notation	Not available	[S34]
ViViD: a variability-based tool for synthesizing video sequences	Xtext language workbench	Textual notation	Not available	[S35]
VMC: recent advances and challenges ahead	HTML technology	Graphical notation	Not available	[S36]
WebFML: synthesizing feature models everywhere	JavaScript technology	Graphical notation	Not available	[S37]

4.2 Key Characteristics of the Different Tools

4.2.1 Development Environment

The described tools are based on different development environments. The most frequently named platform is Eclipse (16 studies), which includes tools based on the Generic Eclipse Modelling Framework, GEMS (1 study); Eclipse Rich Client Platform RCP application development (1study); and the Eclipse Modelling Framework, EMF (9 studies). Within the latter group, two studies reported usage of textual modelling frameworks, i.e., EMFText (Heidenreich et al., 2009) and Xtext (Eysholdt and Behrens, 2010), and three reported usage of graph-oriented UI frameworks, i.e., GMF (Eclipse) and prefuse (Heer et al., 2005).

Two studies reported on tools based on commercial-off-the-shelf software, such as Microsoft Excel or Word. Six tools directly support the usage of UML, out of which two are based on commercial modelling tools, i.e., IBM Rational Rose and Rhapsody. Additionally, one study reported on a tool based on C-preprocessor (CPP) code parser. Finally, three studies were web-based.

In terms of implementation languages, tools in 14 studies are based on Java, one tool is implemented in C# (RequiLine [S26]) and one in C (the Linux Kernel Configurator [S19]). The remaining tools either do not state an implementation language or are realized as extensions of existing tools.

4.2.2 Transformation

Twelve studies reported the usage of some transformation mechanism, e.g., to support generating output. Two used XSL ([S6] and [S22]); one used dynamic loading of Simple XML Feature models (SXFM) [S32]; another used XML and Java source files [S31]; and one used the DIMACS format (a widely used standard for Boolean formulas in CNF) [S37].

4.2.3 Constraints and Reasoning

Fifteen studies reported on the usage of constraint languages or the usage of automated reasoning based on constraints in the wider sense. SAT solvers are used for instance by the S2T2 Configurator ([S24] and FAMILIAR [S30]), a CSP solver is for instance used by Scatter [S29] and [S35]; SAT solvers by [S36] and propositional formulas by [S37].

4.2.4 Graphical and Textual Notations

Among the thirty-seven tools identified in the primary studies, some supported graphical notations only (15 tools), others textual notations only (13 tools), and few supported multiple notations and views (9 tools). Additionally, there were some that did not provide enough details on the notations supported. Figure 4.1 summarises the breakdown of these notations based on the type of notation supported. These are discussed in details in the following sub-sections.

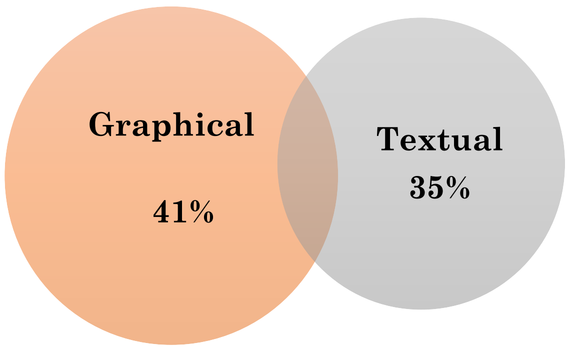


Figure 4.1: Breakdown of tools based on the type of notation supported

4.2.4.1 Graphical Notations

The graphical notations adopted by the tools reported in the primary studies can be classified under the following six visualisations:

- FODA

- File trees (vertical trees)
- Graphs
- Hyperbolic trees
- Logic diagrams (logic gates)
- UML

The figure below (Figure 4.2) shows the number of tools supporting each visualisation type. The figure clearly shows that FODA and File tree representations are still the most popular approaches.

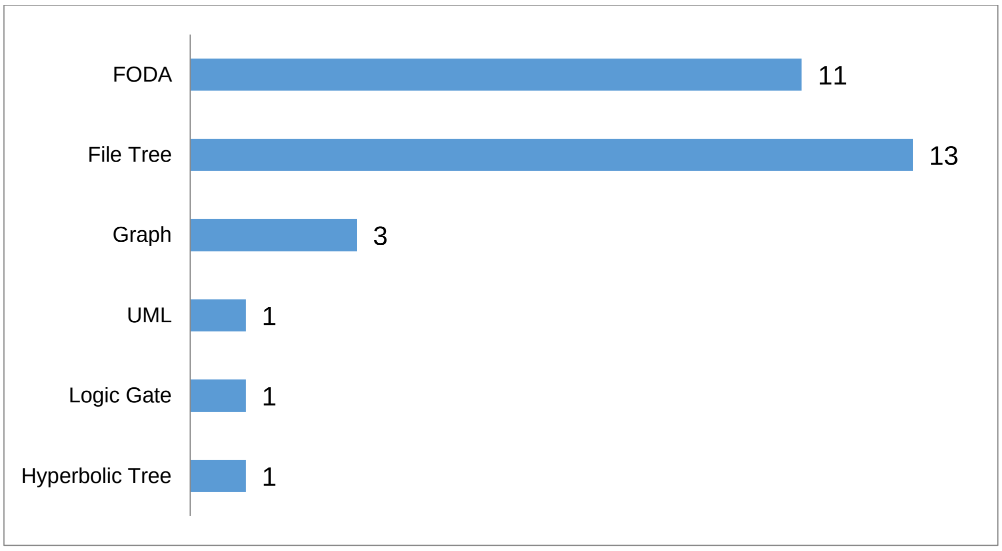


Figure 4.2: Number of tools supporting each visualisation type

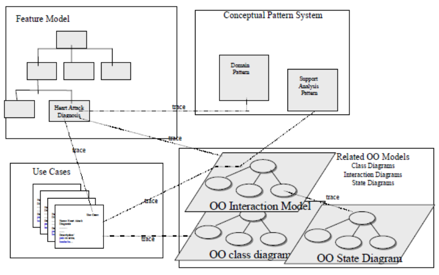
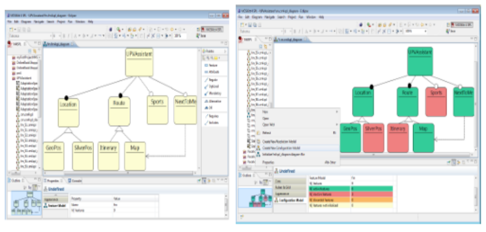
Tools in eleven studies are based on the FODA (Feature-Oriented Domain Analysis (Kang et al., 1990)) approach. These are:

- [S10], FODA with UML
- [S11], FODA, hyperbolic trees, logic diagrams and file tree
- [S14], FODA, Zoomable interface to colour coded source code

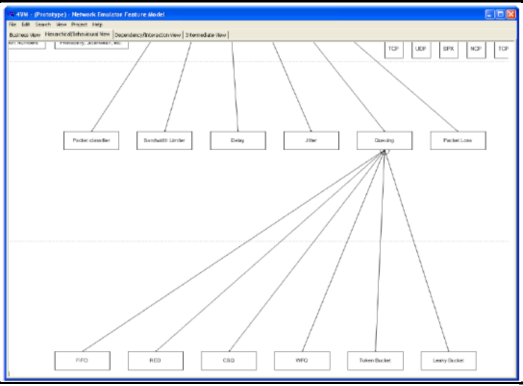
- [S15] and [S25], FODA with colour coding
- [S17], FODA multiple trees per feature model
- [S21], FODA with colour coding and basic file tree
- [S22], FODA basic feature tree with attributes
- [S27], FODA, UML and basic file tree
- [S30], FODA, basic file tree and coding area
- [S37], FODA and basic file tree

Examples of these notations are shown in Table 4.2 below (snapshots taken from the corresponding primary studies). Larger screenshots of these examples are presented in Appendix B. As can be seen in the table, different tools use different parts of the interface to display the FODA-like feature model. As such, they are all prone to graphical overloading issues, where once the feature model size gets into the hundreds, it becomes cumbersome to browse and manage.

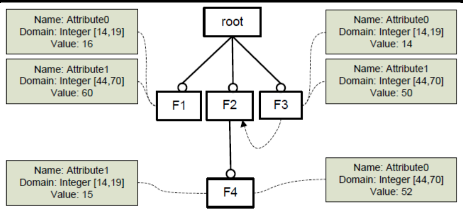
Table 4.1: Tools with FODA-like visual notations

Study	Example Snapshot	Study	Example Snapshot
[S10]	 Feature Model Conceptual Pattern System Domain Pattern Support Analysis Pattern Heart Attack Diagnosis trace Use Cases Related OO Models Class Diagrams Interaction Diagrams State Diagrams OO Interaction Model OO class diagram OO State Diagram	[S21]	

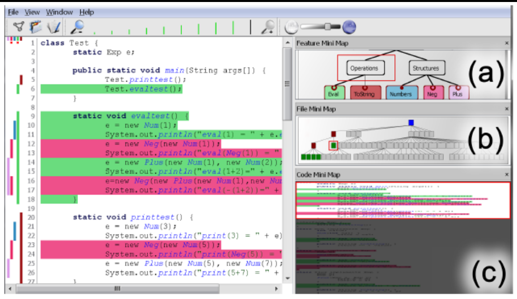
[S11]



[S22]

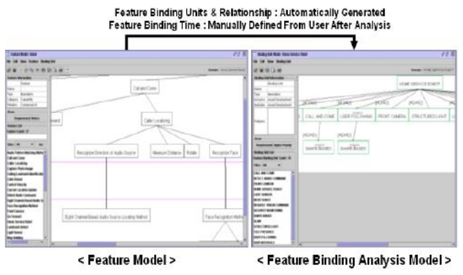


[S14]

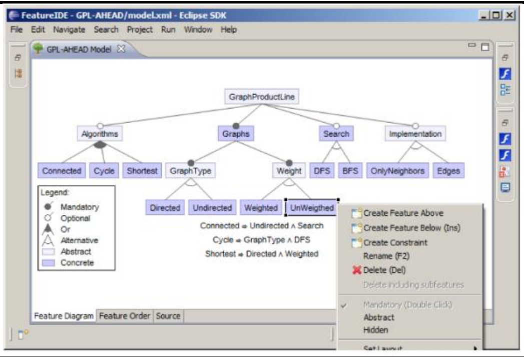


```
class Test {
        static Exp e;

        public static void main(String args[]) {
                Test.printtest();
                Test.evaltest();
        }

        static void evaltest() {
                e = new Num(1);
                System.out.println("eval(1) = " + e.
                e = new Neg(new Num(1));
                System.out.println("eval(Neg(1)) = "
                e = new Plus(new Num(1), new Num(2));
                System.out.println("eval(1+2)=" + e.
                e=new Neg(new Plus(new Num(1),new Num
                System.out.println("eval(-(1+2))=" +
        }

        static void printtest() {
                e = new Num(3);
                System.out.println("print(3) = " + e)
                e = new Neg(new Num(5));
                System.out.println("print(Neg(5)) = "
                e = new Plus(new Num(5), new Num(7));
                System.out.println("print(5+7) = " +
        }
```

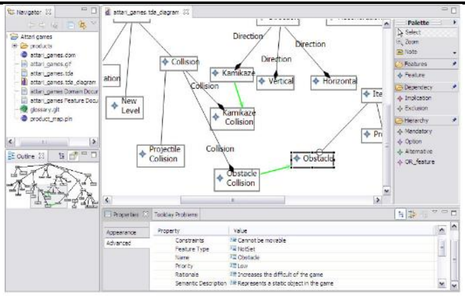
[S25]


Feature Binding Units & Relationship : Automatically Generated
Feature Binding Time : Manually Defined From User After Analysis
< Feature Model >
< Feature Binding Analysis Model >

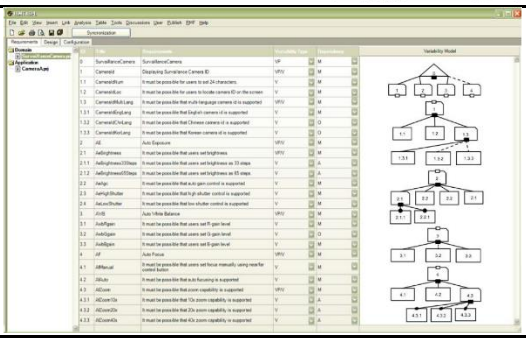
[S15]



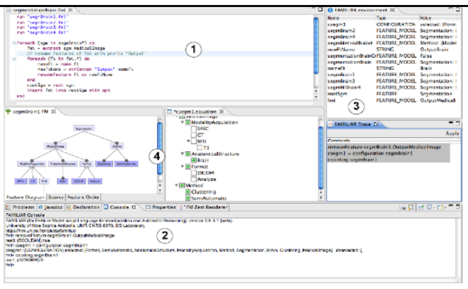
[S27]



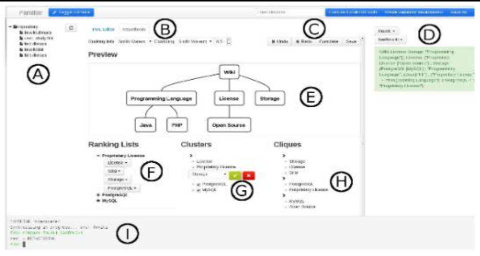
[S17]



[S30]



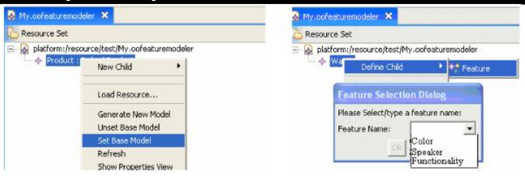
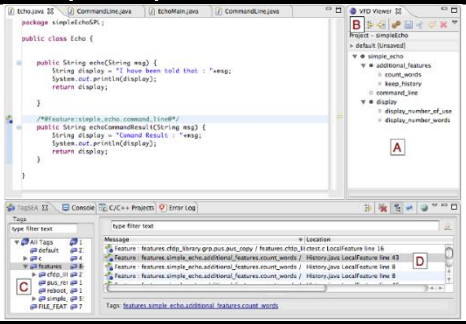
[S37]



Thirteen tools adopt file tree approaches of which eight-used basic right click functionality to access information (tools reported in studies [S7], [S9], [S13], [S26], [S28], [S31], [S32] and [S33]). Two studies are based on advanced customization (colour, shapes, etc.) of feature icons (tools in studies [S12] and [S16]). One-study reports file trees with semi circles representing relationships among different features [S8]. Flow maps are also used in [S24].

A summary of these notations is shown in the Table 4.3 (see Appendix B for larger screenshots). As can be seen in the table below, this family of tools tends to be more scalable due to the inherent nature of the file tree navigation mechanism. However, they are not as good as FODA-like tools in enabling better intellectual control over the model (textual abstraction vs graphical abstraction).

Table 4.3: Tools with file Tree-like visualisation

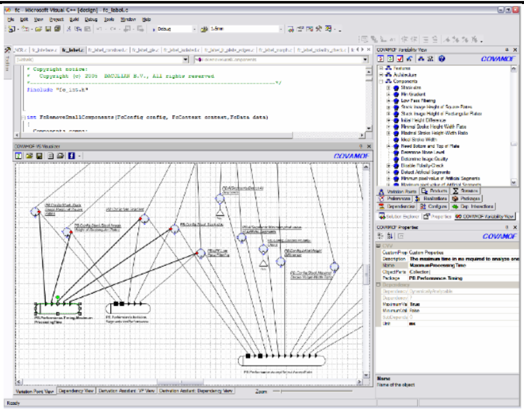
Study	Example Snapshot	Study	Example Snapshot
[S7]		[S13]	

```
package simpleEchoSPL;

public class Echo {

    public String echo(String msg) {
        String display = "I have been told that : "+msg;
        System.out.println(display);
        return display;

    }

    /*@feature:simple_echo.command_line@*/
    public String echoCommandResult(String msg) {
        String display = "Comand Result : "+msg;
        System.out.println(display);
        return display;
    }

}
```

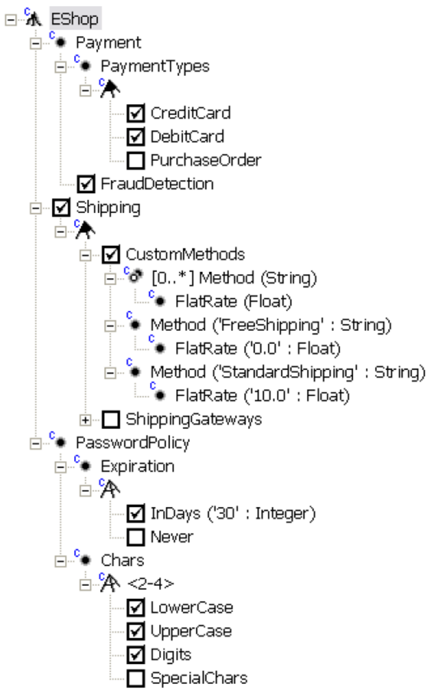
Message	Location
Feature : features.cfdp_library.grp.pus.pus_copy / features.cfdp_li	ctest.c LocalFeature line 16
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 43
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 8
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 8

Tags: features.simple_echo.additional_features.count_words

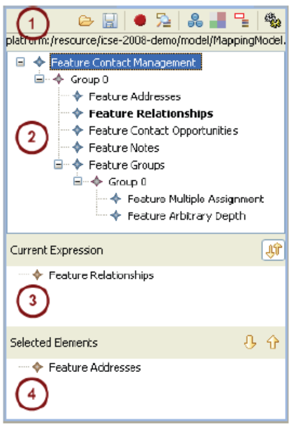
[S8]



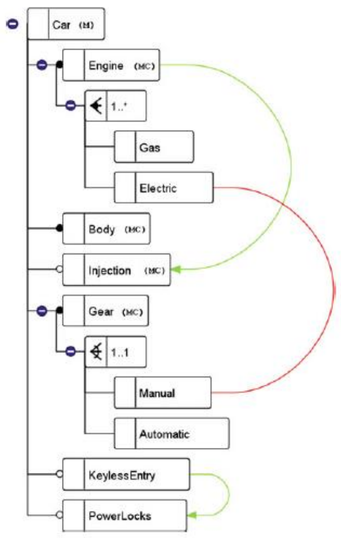
[S16]



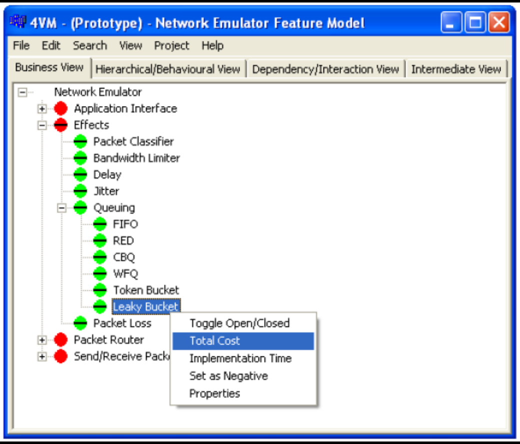
[S9]



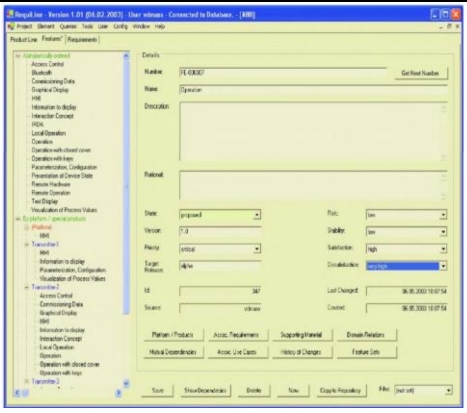
[S24]



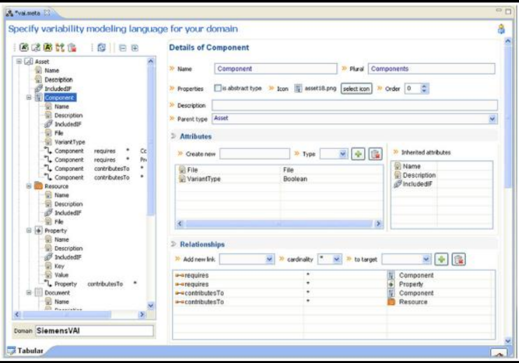
[S11]



[S26]



[S12]



*val.meta

Specify variability modeling language for your domain

- Asset
  - Name
  - Description
  - IncludedIF
  - Component
    - Name
    - Description
    - IncludedIF
    - File
    - VariantType
    - Component requires *
    - Component requires *
    - Component contributesTo *
    - Component contributesTo *
  - Resource
    - Name
    - Description
    - IncludedIF
    - File
  - Property
    - Name
    - Description
    - IncludedIF
    - Key
    - Value
    - Property contributesTo *
  - Document
    - Name

Domain: SiemensVAI

Details of Component

Name: Component — Plural: Components

Properties: is abstract type — Icon: asset18.png (select icon) — Order: 0

Description:

Parent type: Asset

Attributes

Create new — Type — Inherited attributes

File	File
VariantType	Boolean

Inherited attributes: Name, Description, IncludedIF

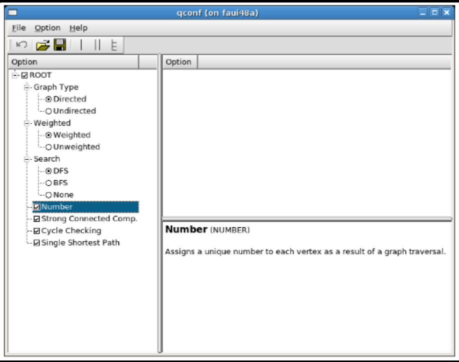
Relationships

Add new link — cardinality * — to target

requires	*	Component
requires	*	Property
contributesTo	*	Component
contributesTo	*	Resource

Tabular

[S28]



qconf (on faui48a)

File Option Help

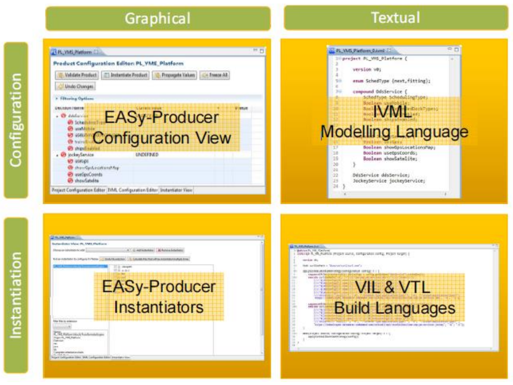
Option

- ☑ ROOT
  - Graph Type
    - ◉ Directed
    - ○ Undirected
  - Weighted
    - ◉ Weighted
    - ○ Unweighted
  - Search
    - ◉ DFS
    - ○ BFS
    - ○ None
  - ☑ Number
  - ☑ Strong Connected Comp.
  - ☑ Cycle Checking
  - ☑ Single Shortest Path

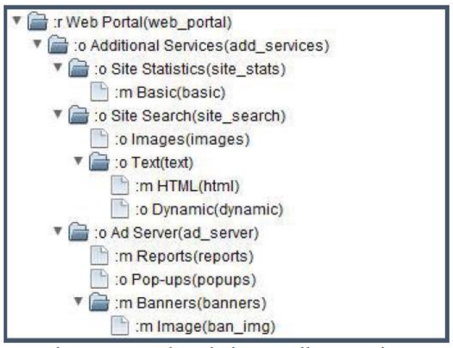
Number (NUMBER)

Assigns a unique number to each vertex as a result of a graph traversal.

[S31]

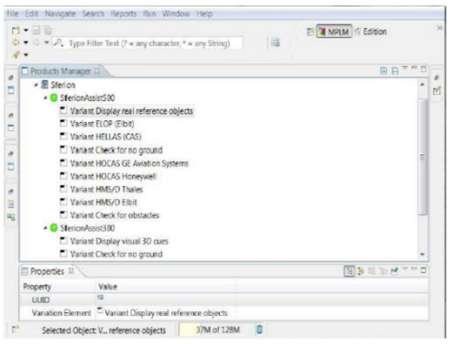

Graphical
Textual
Configuration
Instantiation
EASy-Producer Configuration View
IVML Modelling Language
EASy-Producer Instantiators
VIL & VTL Build Languages

[S32]



- :r Web Portal(web_portal)
  - :o Additional Services(add_services)
    - :o Site Statistics(site_stats)
      - :m Basic(basic)
    - :o Site Search(site_search)
      - :o Images(images)
      - :o Text(text)
        - :m HTML(html)
        - :o Dynamic(dynamic)
    - :o Ad Server(ad_server)
      - :m Reports(reports)
      - :o Pop-ups(popups)
      - :m Banners(banners)
        - :m Image(ban_img)

[S33]



File Edit Navigate Search Reports Run Window Help

Type Filter Text (? = any character, * = any String)

MPLM Edition

Products Manager

- Sferion
  - SferionAssist500
    - Variant Display real reference objects
    - Variant ELOP (Elbit)
    - Variant HELLAS (CAS)
    - Variant Check for no ground
    - Variant HOCAS GE Aviation Systems
    - Variant HOCAS Honeywell
    - Variant HMS/D Thales
    - Variant HMS/D Elbit
    - Variant Check for obstacles
  - SferionAssist300
    - Variant Display visual 3D cues
    - Variant Check for no ground

Properties

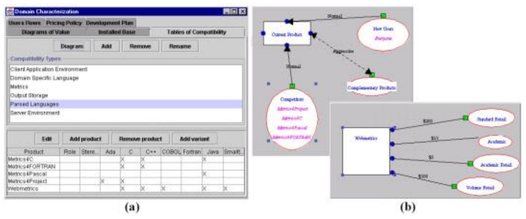
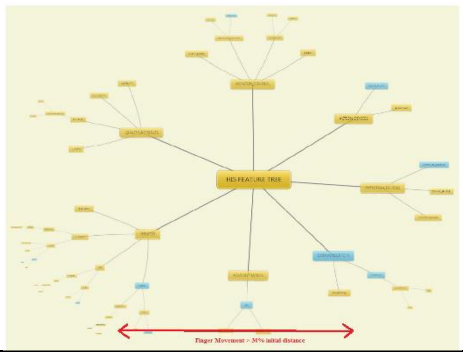
Property	Value
UUID	
Variation Element	Variant Display real reference objects

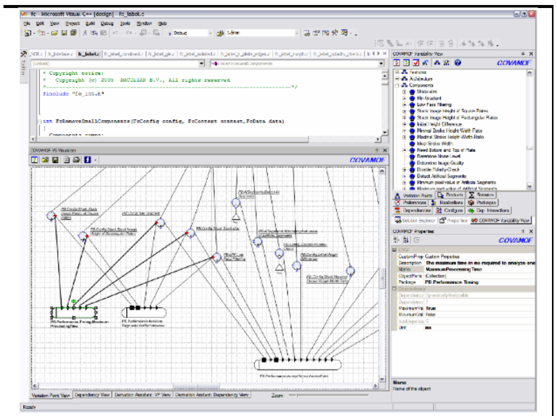
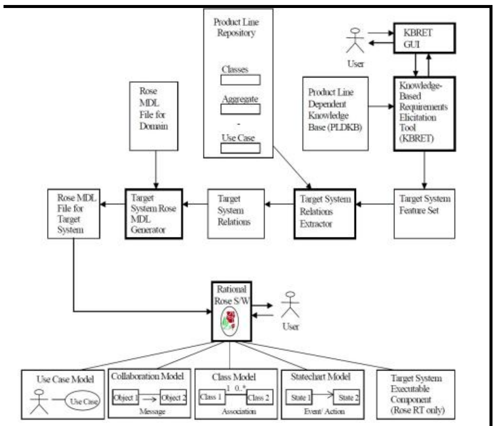
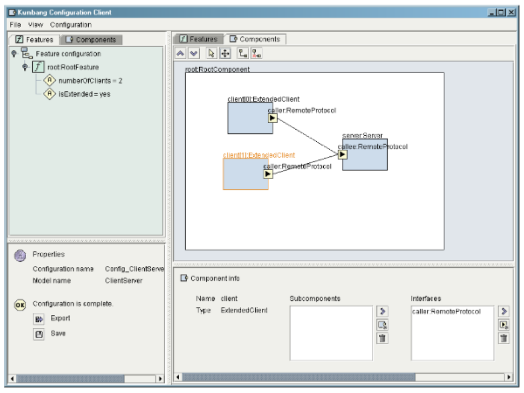
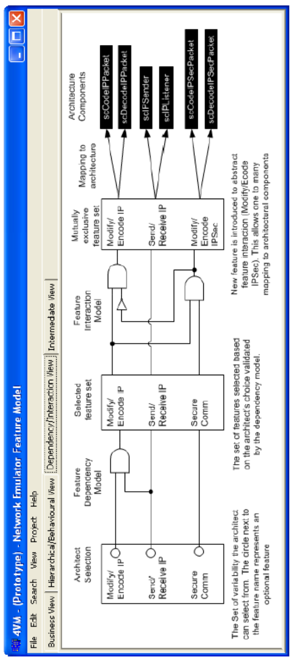
Selected Object: V... reference objects — 37M of 128M

Three tools support graph-based visualisations. One includes a configuration interface using simple node-link graphs (user flows) with different objects [S2]; another supports the use of different objects for dependencies (circles, triangles, etc.), file tree, and coding area [S8]; and one tool is based on KOALA (Van Ommering et al., 2000) like graph visualisation, i.e., architecture centric [S18]. Finally, one tool adopts a logic diagram (schematics) visualisation approach [S11]; another provides an UML-based visualisation [S5]; and one adopts hyperbolic tree visualisation [S11].

Examples of these visualisations are shown in 4.4 below. Larger screenshots of these visualisations are given in Appendix B. Looking at the table below, it can be seen that notations that adopt hyperbolic views tend to have the best balance between scalability and intellectual control (abstraction). While managing to display the structure of the complete feature model, hyperbolic trees allow for browsing the model by displaying more details about nodes that are centered in the middle of the screen, allowing for smoother navigation capabilities, especially when paired with Natural User Interface (NUI) capabilities (e.g. pinching for zooming, etc.).

Table 4.4: Tools with Graph, Logic Diagrams, UML and Hyperbolic Tree visualisations

	Study	Example Snapshot		Study	Example Snapshot
Graph	[S2]		Hyperbolic Tree	[S11]	
	[S8]			[S5]	

Graph		UML	 Product Line Repository Classes Aggregate Use Case User KBRET GUI Rose MDL File for Domain Product Line Dependent Knowledge Base (PLDKB) Knowledge-Based Requirements Elicitation Tool (KBRET) Rose MDL File for Target System Target System Rose MDL Generator Target System Relations Target System Relations Extractor Target System Feature Set Rational Rose S/W User Use Case Model Collaboration Model Class Model Statechart Model Target System Executable Component (Rose RT only)
[S18] Graph	 Kumbang Configuration Client File View Configuration Features Components Feature configuration root:RootFeature numberOfClients = 2 isExtended = yes root:RootComponent client[0]:ExtendedClient caller:RemoteProtocol server:Server client[1]:ExtendedClient Properties Configuration name Config_ClientServer Model name ClientServer Configuration is complete. Export Save Component info Name client Type ExtendedClient Subcomponents Interfaces caller:RemoteProtocol	[S11] Logic Diagrams	 AVM - [Prototype] - Network Emulator Feature Model File Edit Search View Project Help Business View Hierarchical/Behavioural View Dependency/Interaction View Intermediate View Architect Selection Feature Dependency Model Selected feature set Feature Interaction Model Mutually exclusive feature set Mapping to architecture Architecture Components Modify/Encode IP Send/Receive IP Secure Comm Modify/Encode IPSec scCodeIPPacket scDecodeIPPacket scIPSender scIPListener scCodeIPSecPacket scDecodeIPSecPacket

There are studies that do not provide enough details on the graphical notation used in the tools described ([S19] and [S29]).

Overall, seven tools supported multiple views of the feature model, where combination of a graph, a file tree, and a coding area are used by [S8]; Koala and file tree is reported in [S18]; a file tree and a coding area are used in [S13] and [S31]; FODA and basic file trees are used in [S21] and [S37];

FODA, a basic file tree and a coding area are reported in [S30], FODA, UML and a basic file tree are used by [S27]; and FODA, hyperbolic trees, logic gates and a file tree are reported in [S11] as summarized in Figure 4.3 below.

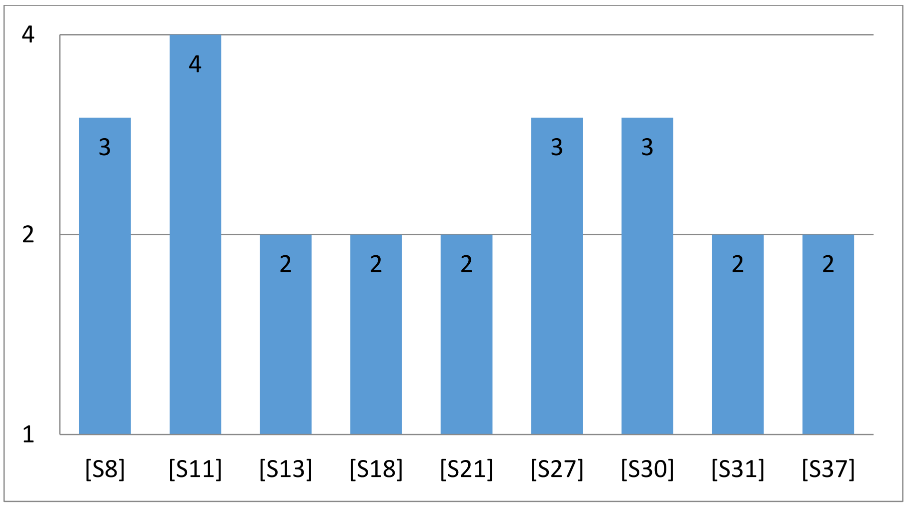


Figure 4.3: Number of views per tool for tools with more than one view

4.2.4.2 Textual Notations

For the textual notations, tools in thirteen studies reported the use of textual notations. These can be classified under three different categories:

- Code-like: with syntax similar to programming languages
- XML-based: notations that are based on XML
- Code-based: notations that embed variability representation within source code

Figure 4.4 below shows the number of tools supporting each textual notation type.

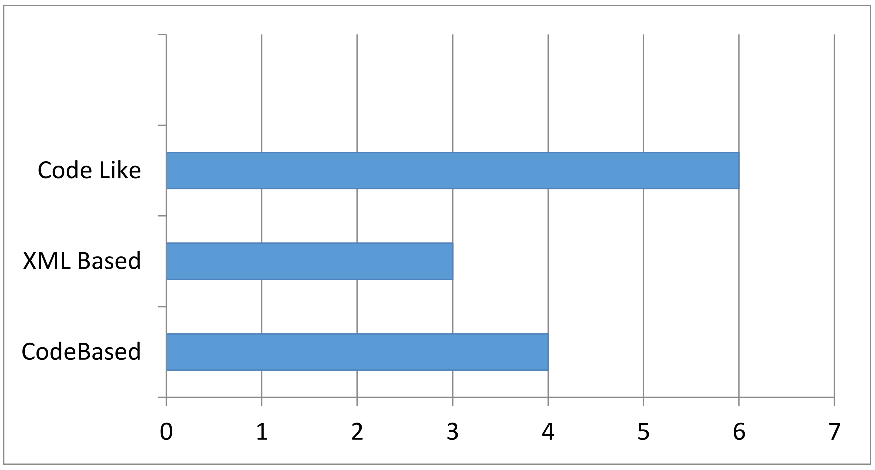


Figure 4.4: Tools with various textual notations

Code-like notations can be found in the tools described in [S3], [S18], [S28], and [S30], [S34] and [S36]. Example snapshots of these notations can be found in the Table 4.5 below.

Table 4.5: Tools with Code-like textual notations

Study	Example Snapshot	Study	Example Snapshot
[S3]	gui: compile G lnk: gui main link gui main common: compile X2 clean: delete *.gif super.clean main:common compile A compile B compile C common: compile X2 gui: compile G lnk: gui main link gui main clean: delete *.gif delete *.class	[S28]	config GPL boolean "ROOT" select M1 choice depends on GPL prompt "Graph Type" config DIRECTED boolean "Directed" config UNDIRECTED boolean "Undirected" endchoice config NUMBER default y if GPL requires (BFS ‖ DFS) boolean "Number" ---help--- Assigns a unique number to each vertex as a result of a graph
[S18]	Kumbang model KumbangExample root feature FSystem; root component CSystem feature FSystem { subfeature (FeatureA, FeatureB) f;	[S30]	GraphicCard: DirectX Bus [Vertex]; // Vertex is optional DirectX: (v10 | v10.1)+; // Or-group

	`  implementation`
	`      instance_of(f, FeatureA) <==>`
	`value($, attr) = a;`
	`      instance_of(f, FeatureB) <==>`
	`value($, attr) = b;`
	`}`
	
	`feature FeatureA, feature FeatureB only`
	`are Lee EE usually really really hello`
	`there closing are are are`
	
	`component CSystem {`
	`   attributes`
	`      ABBalue attr;`
	`}`
	
	`attribute type ABValue = {a, b}`
			`Bus: (n64 | n128);`
			`// Alternative-group`
			
			`n64 -> Vertex;`
			`// Constraints`

[S34]	`Variability Code Metrics Supported in`
	` VITAL`
	
	`  Metric              Description`
	
	`VP Nesting Degree    #ifde nesting level`
	`                      of a given VP`
	
	`Var Tangling Degree  #Vars used in a`
	`                      given VP`
	
	`Var Fan-out on VPG   #VPGs that contain`
	`                      a given Var`
	
	`Var Fan-out on File  #files that contain`
	`                        a given Var`
	
	`  Var Fan-in on File  #Vars included in`
	`                       a given File`
	
	`  VP Fan-in on File   #VP included in a`
	`                       given file`
		[S36]	`Station(I,N,J,M) =`
			`( [N = 0]`
			`nobike(I).Station(I,N,J,M) +`
			`[N > 0] bike(I).Station(I,N-1,J,M)`
			`) +`
			`return(I).Station(I,N+1,J,M) +`
			`redistribute(may,?FROM,?TO,?K).`
			`( [TO = I] Station(I,N+K,J,M) +`
			`[TO /= I] Station(I,N,J,M) ) +`
			`[N > M] redistribute(may,I,J,N-`
			`M).Station(I,M,J,M)`
			`Users(I,J) =`
			`request(I).`
			`( bike(I).return(J).Users(I,J) +`
			`nobike(I).Users(I,J) )`

XML-based notations are supported in [S4], [S8], and [S22]. Samples of these notations are presented in the Table 4.6 below.

Table 4.6: Tools with XML-based textual notations

Study	Example Snapshot	Study	Example Snapshot

[S4]	`<<x-frame  name=„Being‟  language=‟java‟>`
	`<set  var=‟BEING_CLASS‟  value=‟Being‟/>`
	`<break name=‟BEING_PARAMETERS‟/>`
	
	`class <value-of expr=‟?@BEING_CLASS?‟/>{`
	`String Name;`
	`int Age;`
	`double Weight;`
	`double Height;`
	`<break name=‟BEING_BODY‟/>`
	`public String getName(){return Name;}`
	`public int getAge(){return Age;}`
	`public double getWeight(){return`
	`Weight;}`
	`public double getHeight(){return`
	`Height;}`
		[S22]	`//STEP 1: Specify the user's`
			`preferences for the generation`
			`(characteristics)`
			`GeneratorCharacteristics`
			`characteristics = new`
			
			`GeneratorCharacteristics();`
			`//number of features`
			`characteristics.setNumberOfFeatures`
			`(30);`
			`//percentage of constraints`
			`characteristics.setPercentageCTC(10`
			`);`
			`//Max number of products of the`
			`feature model to be generated`
			`characteristics.setMaxProducts(1000`
			`);`
			`//STEP 2: Generate the model with`

	`<break name="BEING_NEW_METHODS"/>`
	`};`
	`</x-frame>`
			`the specific characteristics (FaMa`
			`metamodel is used)`
			`IGenerator generator = new`
			`MetamorphicFMGenerator(`
			`new`
			`FMGenerator());`
			`FaMaFeatureModel fm =`
			`(FaMaFeatureModel)generator.generat`
			`eFM(characteristics);`
			`System.out.println("Number of`
			`products of the feature`
			`model generated: " +`
			`generator.getNumberOfProducts());`
			`//STEP 3: Save the model and the`
			`products`
			`FMWriter writer = new FMWriter();`
			`writer.saveFM(fm, "./model.xml");`
			`//FaMa XML format`
			`writer.saveFM(fm, "./model.afm");`
			`//FaMa textual format`

[S8]	`<variationpoint id="[id]">`
	`  <artefact>`
	`    [artefact identifier]`
	`  </artefact>`
	`  <abstractionlayer>`
	`    [abstraction layer]`
	`  </abstractionlayer>`
	`  <description>`
	`    [description]`
	`  </description>`
	`  <type>`
	`    optional | alternative | optional`
	`    variant |variant | value`
	`  <type>`
	`  <variants> <!-- if not type=value -->`
	`    <variant id="[id]">`
	`    . . .`
	`    <variant id="[id]">`
	`  </variants>`
	`  <range>`
	`    [range specification]`
	`  </range> <!-- if type=value -->`
	`  <state>`
	`    open | closed`
	`  </state>`
	`  <mechanism>`
	`    [mechanism]`
	`  </mechanism>`
	`  <bindingtime>`
	`    [bindingtime]`
	`  </bindingtime>`
	`  <rationale>`
	`    [rationale]`
	`  </rationale>`
	`</variationpoint>`		

Finally, Code-based notations are found in [S7], [S13], [S14] and [S35]. These are demonstrated in the Figure 4.7 below.

Table 4.7: Tools with code based textual notations

Study	Example Snapshot	Study	Example Snapshot

[S7]

```
class Watch {
Color color;
Waterproof waterproof;
Depth depth;
...
}
class Color {...}
class Yellow extends Color {...}
class Metallic extends Color {...}
class Depth {...}
class 50m extends Depth {...}
class 100m extends Depth {...}
```

[S14]

```
class Test {
        static Exp e;
        public static void main
(String args[]) {
                Test.printtest();
                Test.evaltest();

        }

        static void evaltest() {
                e = new Num(1);

        System.out.println("eval (1)
= " + e.e
                e = new Neg (new Num
(1);
                System.out.println("
        eval(Neg(1) ="
                e = new Plus (new
        Num(1), new Num(2));
                System.out.println("
        eval(1+2)=" + e.e
                e = new Neg(new
Plus(new Num(1), new Num

        System.out.println("eval(-
(1+2))=" +
        }

        static void printtest() {
                e = new Num(3);

        System.out.println("print(3)
= " + e)
                e = new Neg(new
Num(5));

        System.out.println("print(Ne
g(5)) = "
                e = new  plus(new
Num(5), new Num(7));

        System.out.println("print
(5+7) = " +
        }
```

[S13]

```
// The syntax of feature tags is:

<fcomment> ::= "/*@feature:" <flist>
"@*/" [<filetag>]
<flist> ::= <featurename> ( ":" <flist>
) *
<filetag> ::= "/*@!file_feature!@*/"

// where <featurename> identifies a
// feature of the FD.
```

[S35]

```
Relationships: 2 sequence { 3
signal_quality 4 cloneBetween 0 and
5 vehicle 5 //. . . 6 } 7 8
Attributes: 9 @NT string
sequence.comment 10 @RT int
vehicle.speed [0..130] delta 5
default 4 0 11 @ND int *.cost [0 ..
1000] default 150 12 real
signal_quality.luminance_mean 13
[0.0 .. 3 2.0] delta 2.0 14 [3 2.0
.. 224.0] delta 8.0 15 [224.0 ..
255.0] delta 2.0 16 default 7 2.5 5
17 //. . . 18 19 Descriptions: //.
. . 20 21 Constraints: //. . . 22
23 Objectives: 24 objective
generate_low_cost_configurations {
25 min (sum (*.cost)) 26 } 27
Configurations: //. . .
```

Looking at the samples in the three tables above, it can be said that it is equally difficult for humans to read these descriptions, whether they are written in code-like, code-based or XML format. Accordingly, it would be best to choose the format that makes it easier for machines to parse textual notations, and focus on providing GUI based access to feature information for human use. As such, there is a need to develop a standardised description format to allow better exchange of information among the different tools. Competition between different tools would then be based on the quality of presentation and intuitiveness of navigation of such information by end-users.

Finally, there were six further notations that did not provide enough details about the textual notations they support, namely [S1], [S6], [S15], [S20], [S23], and [S24].

4.3 Quality of the Research Conducted in the Reported Approaches

We analysed the quality of research using the quality scores (0, 0.5, 1) for the eight quality questions (cf. section 2.2.4 of Chapter 2) and also assessed how the studies address four different quality attributes important for tools usability, integration, scalability, and performance.

Table 4.8 presents the results of the quality assessment of the 37 studies included in the final review according to the quality questions. A frequency analysis of the scores for each quality question is presented in Figure 4.5. Most studies provide a rationale for why the study was undertaken (Q1). Almost half of the studies describe the context in which the research was carried out (Q2). More than half of the papers described the variability management tool in enough detail to be able to perform an in-depth analysis of the capabilities of the tool (Q3). Very few studies present an evaluation of their proposed tools including feedback from end users (Q4). This could be one of the main factors limiting the industrial adoption of these tools. Less than a third of the studies support substantive claims made in the paper with reliable evidence (Q5). Less than a third of the studies compare and evaluate their own results against related work (Q6). Finally, very few studies discuss the credibility of their findings (Q7) and limitations (Q8).

Figure 4.6 shows the distribution of total quality scores. The maximum possible total score is 8 (a score of 1 across all quality questions). Most studies received scores around 3 and 4. The total average was 4.05 with a standard deviation of 1.84. This indicates that although the minimum quality requirement is met, there is plenty of potential for improvement.

In general, the authors provided a motivation and a description of the research context but papers lacked data to support the claims and findings. Also, authors seldom provided critical reflection of their results. Most variability management tools presented were not well evaluated, especially with respect to feedback from end users.

Table 4.8: Results of the quality assessment of the primary studies

	No (0)	**Partial (0,5)**	**Yes (1)**	**Average Score**
Q1	1	2	34	0,95
Q2	12	10	15	0,54
Q3	1	14	22	0,78
Q4	23	12	2	0,22
Q5	11	15	11	0,50
Q6	19	7	11	0,39
Q7	16	15	6	0,36
Q8	23	8	6	0,27

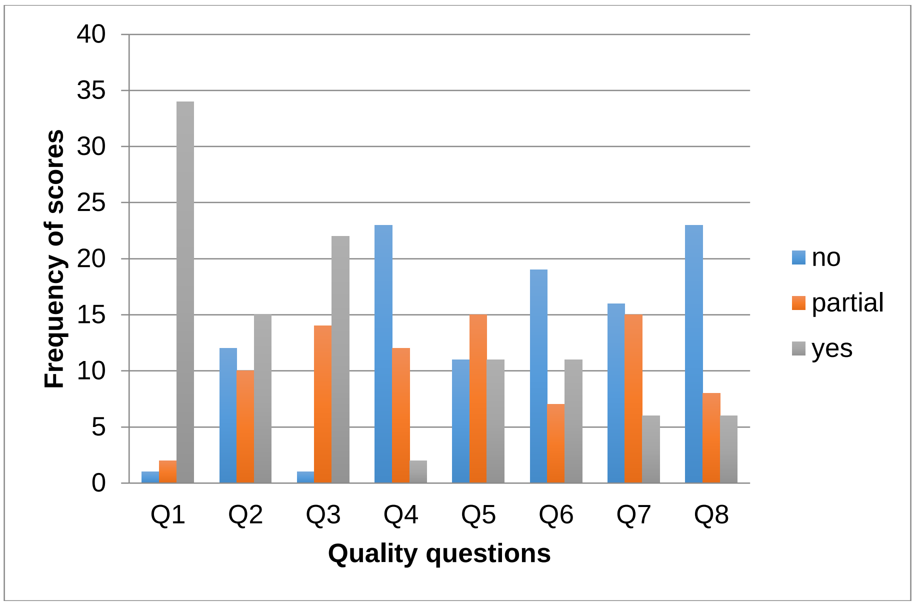


Figure 4.5: Frequency analysis of quality scores for each question

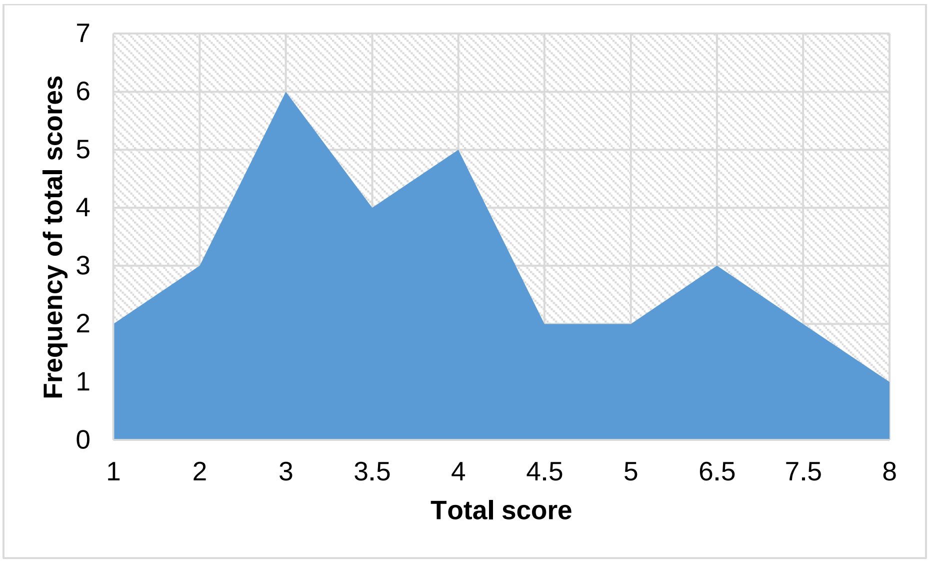


Figure 4.6: Distribution of total quality scores

Table 4.9 presents different quality attributes we focused on in our review (usability, integration, scalability, and performance) and how well they were addressed by the studies. The quality attributes were identified through an interview-based survey conducted with a number of SPL practitioners who were asked to list their five most important attributes of an SPL tool. Figure 4.7 shows the frequency analysis of the results for each quality attribute. As can be seen, most studies do not mention the attributes with only few studies providing contributions to the different areas of the quality attributes. The lack of attention of researchers to these quality attributes, which are high up in the priority list of practitioners, can be seen as another reason behind the very limited industrial adoption of these tools.

Table 4.9: Quality attributes addressed by studies

	Does Not Mention (0)	Mentions (1)	Contribution (2)	Contribution and Evaluation (3)	Average Score
Usability	19	8	10	0	0,76
Integration	21	0	13	3	0,97
Scalability	24	8	4	1	0,51
Performance	24	4	8	1	0,62

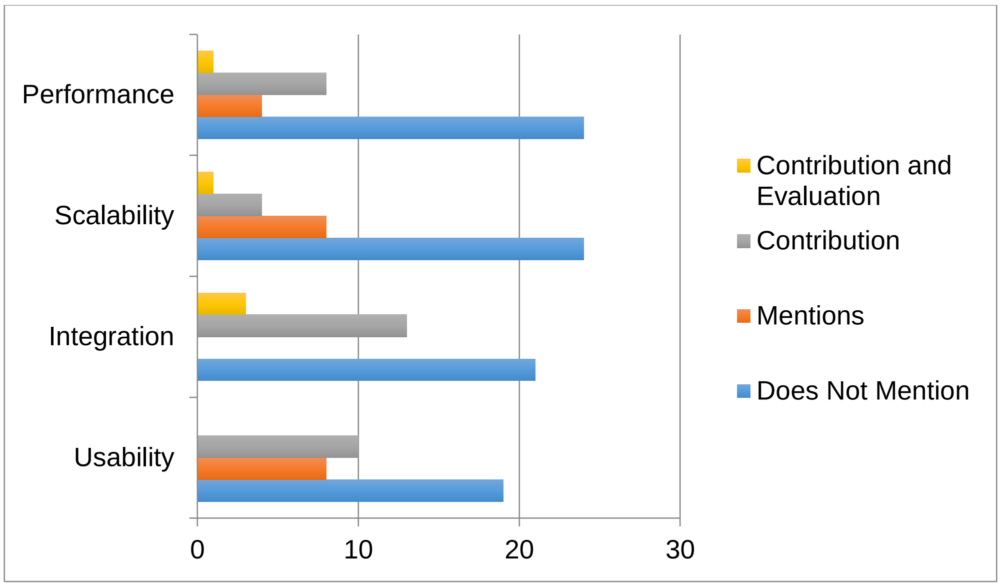


Figure 4.7: Frequency analysis of scores for each quality attribute

As per Table 4.9 and Figure 4.7, the assessment of how well the studies addressed the four different attributes (usability, integration, scalability, and performance) important for tools, was taken using the four levels distinguished as follows:

- Does not mention:
  - When a study does not mention the attribute to be satisfied at all.
- Mentions:
  - When a study mentions evaluation of the attribute as a challenge or research topic but includes no further discussion.
- Provides a contribution:
  - When a study provides contribution in the area.
- Provides a contribution and evaluation:
  - When a study provides contribution and evaluation in the area.

Table 4.10 provides a list of all the tools identified in the study in chronological order, along with the assessment results summary for how the studies addressed the four different attributes important for VM tools. An abbreviated symbol (CE) signifies “contribution and evaluation”, (C) denotes “contribution”, (blank space) indicates “does not mention”, and (E) only indicates “mentions of evaluation”.

Table 4.10: Identified tools with the assessment summary results

S/No	Tool Name	Usability	Integration	Scalability	Performance
1	DARE-COT				
2	Odyssey		C		C
3	PuLSE-BEAT	E	C		
4	Holmes				
5	RequiLine	E	C	E	E
6	COVAMOF				
7	Feature Modelling Plug-In	E			
8	PLUSEE		CE		
9	XML-Based Feature Model			C	
10	AHEAD			E	E
11	XVCL			E	E
12	KUMBANG			C	C
13	BVR: Base-Variation-Resolution				
14	ASADAL (A System Analysis and Design Aid tooL)	C			E
15	Scatter Tool			CE	CE
16	VMWT	C			C
17	L K C – Feature Modelling Tool				
18	FeatureMapper	C		E	C
19	PLUM				
20	MUSA	E		C	
21	XToF – A Tool for Tag-based Product Line Implementation		C		
22	ToolDay	E	C		
23	Zoomable	C		E	
24	FAMILIAR	E	C		
25	DOPLER	C		E	C
26	FeatureIDE	C	C		
27	ISMT4SPL				
28	BeTTy		C		
29	MOSKitt4SPL				
30	S2T2 Configurator		C	E	
31	Easy-Producer	C	C	CE	
32	OPTI-SELECT				C
33	MPLM-MaTeLo product line manager	C	C		C
34	Variability code analysis using the VITAL tool	C	C		
35	ViViD: a variability-based tool for synthesizing video sequences	C	E	C	C
36	VMC: recent advances and challenges ahead	C	C		
37	WebFML: synthesizing feature models everywhere	E	CE		

4.4 The Context of Research

The distribution of the research context of the studies is presented in Figure 4.8. The figure shows that most studies (68%) have been conducted in an academic context. Only 16% of the studies are joint industrial academic endeavours. In 16% of the studies, no information was provided on the research context. Table 4.11 presents a list of all studies with their research context.

Although the primary research context of some studies was academic, few still had practical relevance. Figure 4.9 shows the distribution of the relevance of the primary studies. Almost half of the studies (41%) are relevant to academia only. 36% of the studies are relevant to both academia and industry. Finally, 10% of the studies are relevant to practice only. While 13% provide no sufficient data to be judged.

Table 4.11: Research context of the primary studies

	academia	industry and academia	no information
S1	X		
S2	X		
S3	X		
S4	X		
S5	X		
S6	X		
S7	X		
S8	X		
S9			X
S10			X
S11	X		
S12		X	
S13		X	
S14	X		
S15	X		
S16	X		
S17	X		
S18	X		
S19		X	
S20	X		
S21			X
S22	X		
S23	X		
S24			X
S25			X

S26	X		
S27	X		
S28	X		
S29	X		
S30			X
S31	X		
S32	X		
S33		X	
S34		X	
S35		X	
S36	X		
S37	X		

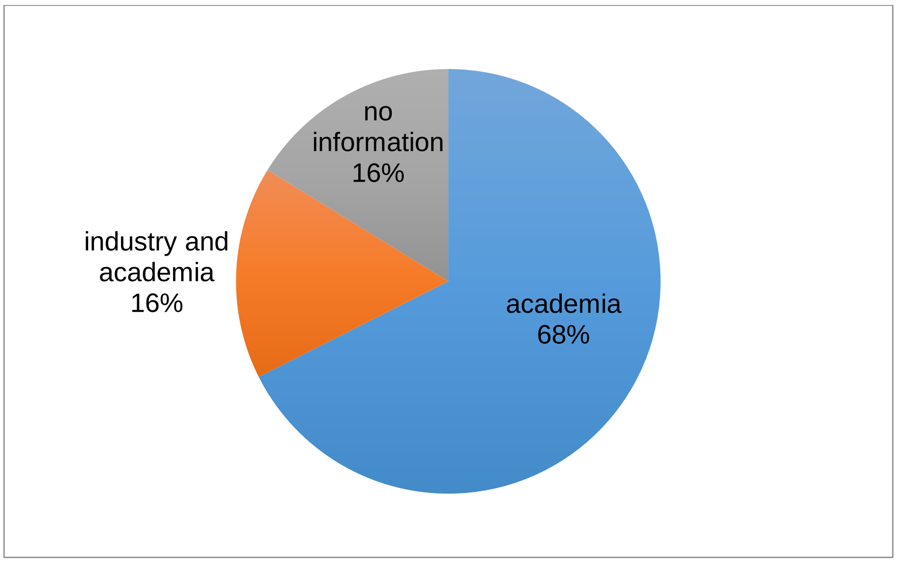


Figure 4.8: Research context of primary studies

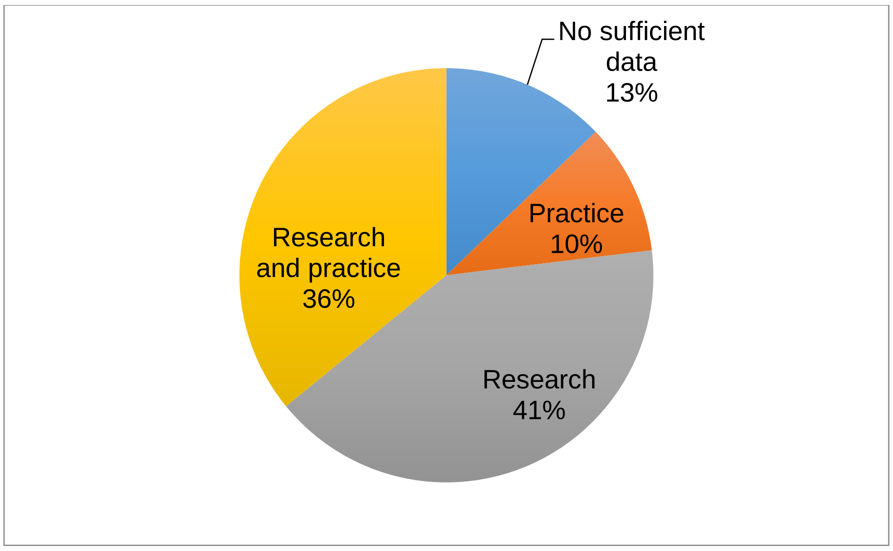


Figure 4.9: Relevance of primary studies

4.5 Main Challenges Faced by Current Product Line Management (PLM) Tools

Our last part of the analysis aimed at analysing the main challenges faced by current tools as well as limitations of the tools. We therefore analysed the 37 selected studies regarding the challenges and limitations of current variability management tools they discuss. Using the coding technique (Seaman, 1999), we first scanned the studies looking for keywords “challenge”, “issue”, “limitation”, and “drawback” and then extracted the related text (statements on challenges and/or limitations). This allowed us to find out which studies do not discuss any limitations or challenges (no statements extracted); which studies at least mention challenges or limitations (statements extracted list challenges or limitations, but do not discuss them); and which studies actually discuss challenges or limitations (statements extracted list and discuss challenges or limitations). 56% do not discuss limitations at all, 27% at least mention some limitations without further discussing them, and only about 17% actually discuss limitations. We find this a general weakness of publications on variability management tools, i.e., that they do not discuss their own limitations, which makes it hard to assess tools' usefulness.

Challenges are more frequently discussed (73% provide a discussion, 13% at least mention challenges, only 13% do not even mention challenges), i.e., authors mention what was the challenging part of implementing their tool and/or what challenges their tool addresses.

We eventually analysed the extracted statements and (through discussion and refinement among researchers) came up with ten categories for challenges and limitations, in which we could group the extracted statements on challenges and limitations discussed in detail below (ordered by the number of studies providing input to the category).

The key challenge of variability management tools is *scalability of models*, i.e., how to develop variability models that are still useful despite their size and complexity. 40% of the selected studies discuss this challenge and suggest different solutions as described above.

The second most discussed challenge is *checking models for consistency and correctness* (23%), especially how to keep the models consistent with the underlying architecture and check that the models represent the variability of the product line correctly. *Mapping problem and solution space* (20%) is also discussed as a key challenge to be addressed by variability management tools. Many tools only take care of creating and managing the variability models representing variability but not of how to map variability (e.g., represented by features or decisions) with the actual artefacts realizing this variability.

Visualisation/Graphical Overload is discussed as a challenge by 17% of the selected studies. Variability management tools must provide ways to cope with the size and complexity of variability models to help users suffering from graphical overload with visualisations. Other important challenges are *usability* and *maintenance and evolution of variability models* (both 13%). Addressing both challenges is essential for tools to be useful and successful in practice in the long run.

Integration of variability management and (legacy) software (development), i.e., the question of how to adopt a variability management tool in practice, is also still an important issue and discussed by 10% of the selected studies. *Process Improvement/Automation through variability management (7%)* is explicitly discussed by 2 selected studies, even though this is actually the key goal of variability management tools anyway.

Two further challenges, which are discussed by one study each, are supporting the modelling of *non-functional properties in variability management (e.g., resource consumption constraints) and compliance (with standards/quality policies/regulations).*

4.5.1 Scalability of (variability) Models (12 studies)

In an initial discussion, we had called this category “working with one large model vs. working with several separate models”. However, through our discussion we found out that the statements we categorized here actually are all about challenges regarding the scalability of (variability) models.

For instance, the authors of [S17] report experiences from empirical case studies that confirm that the complexity of variability management stems from the need to work with (too) large models. Study [S4] highlights the importance of compositional approaches to product line representation/implementation to address this challenge. Study [S21] report on a tool supporting variability management in self-adaptive systems, which again adds to the challenge of scalability of models.

As discussed by the authors of [S3] a key “*challenge is to show how scaling can be accomplished in a principled manner so that product line variability management tools are not just ad-hoc collections of tools using an incomprehensible patchwork of techniques*”. More specifically, they argue that “generators are a technological statement that the development of software in a domain is understood well enough to be automated. However, we must make the same claim for generators: The complexity of generators must also be controlled and must remain low as application complexity scales; otherwise, generator technology will unlikely have wide-spread adoption.”

The BVR tool [S7], for instance, proposes to have separate models related to a base model instead of one large model or completely separate models to allow working with product lines of a realistic size. DOPLER [S12] allows both, creating one big model and several small but related models. The DSL tool FAMILIAR [S30] suggests separating, relating, and composing several feature models while automating the reasoning on their compositions. FAMILIAR focuses mainly on textual representation because, as they claim, this favours readability of the specified operations and leads to more usability and productivity when dealing with compositional operations on feature models. They, however, also argue that graphical visualisation has proved to assist users, for example, during the configuration process. This is why they integrated their DSL with the Feature IDE tool.

The author of [S11] presents a NUI-based multiple perspective variability modelling tool to help working with large-scale models, i.e., multi-touch interfaces to allow working with large models (and their visualisations/different views) to address the scalability challenge.

ViewInfinity [S14] provides seamless and semantic zooming of different abstraction layers of an SPL. The tool described in [S5] provides multiple product line views (using the feature model as a unifying view). Study [S8] focuses on the hierarchical organization of variability, the first class representation of simple and complex dependencies ("*dependencies that affect the binding of a large number of variation points, e.g., quality attributes*" [S8]); and argues that relations between dependencies should be explicitly represented. The Odyssey Reuse environment [S10] specifies "*patterns based on both architectural styles and specific information from the* *application domain to create a complete reuse environment, which defines software architectures and conceptual model representations on a high level of abstraction*".

4.5.2 Checking Models for Consistency and Correctness (7 studies)

Checking the models underlying the variability management tools for consistency and correctness is considered as a key challenge by seven of the 30 studies. For instance, the authors of RequiLine [S26] argue that semantic information is needed for an automated consistency check in variability management tools. Study [S5] highlights that consistency checking among the multiple views in a product line (as provided by their tool) is essential. FeatureMapper [S9] provides diverse visualisations to support the SPL engineer in verifying the correctness of the models (feature models, mapping models, solution space models) and argues this is very important.

The authors of Odyssey [S10] suggest specifying the "*operations that will be performed on models, as well as to systematize these operations, to facilitate the consistent creation of models*". The DOPLER tools [S12] have an integrated consistency checking component that checks the consistency on different levels, i.e., in problem space, in solution space, and between problem and solutions space. ToolDAy [S27] is one of the few studies that discuss their limitations, i.e., that complex consistency rules cannot be described in their tool. The authors of study [S3] highlight the use of model checkers in their tool as important future work.

4.5.3 Mapping Problem and Solution Space (6 studies)

Six studies highlight the challenges and limitations of mapping problem and solution space, i.e., mapping the variability representation with the actual product line architecture. For instance, ISMT4SPL [S17] discusses “traceability between decisions in variability/feature models and the corresponding implementation artefacts” as a key challenge for variability management tools. The authors of study [S16] report about a limitation of their tool, i.e., that the support for mapping problem and solution space is missing.

FeatureMapper [S9] explicitly focuses on this aspect by introducing mapping models to map feature models and solution space models. Kumbang [S18] explicitly integrates architecture models (i.e., Koalish, an architecture description language/component model based on Koala ADL but adding variability concepts) with feature models within its tool support. DOPLER [S12] uses explicit asset models to represent the solution space and links these models with the problem space decision models via so-called inclusion conditions. Code tagging tools such as XToF [S13] do not map both spaces but rather integrate the representation of the problem space into the solution space, or, as could be argued, just represent solution space variability (i.e., variability in code).

4.5.4 Visualisation/Graphical Overload (5 studies)

Five studies argue that visualisation of variability easily leads to a graphical overload of the tool user and is a key challenge. For instance, the author of study [S11] argues that “*it is important for a variability management mechanism to be able to extract and present relevant information about a variability model in dedicated views for different groups of stakeholders (users, system analysts, developers, etc. to alleviate the graphical overload when showing all the information in one view.*”

ViewInfinity [S14] provides seamless and semantic zooming of different abstraction layers of an SPL. Study [S8] argues that variability models should “*represent variation points as first class entities in all abstraction layers (from features to code); provide a hierarchical organization of variability; focus on the first class representation of simple and complex dependencies (dependencies that affect the binding of a large number of variation points, e.g. quality attributes); and explicitly represent dependencies*”. ST2T [S24] provides sophisticated visualisation and interaction techniques to address the challenge that handling variability and configurations is hard due to the complexity on a cognitive level as human engineers reach their limits in identifying, understanding, and using all relevant details. Study [S16] highlights this as a key limitation of their tool, i.e., that a graphical representation missing.

4.5.5 Maintenance and Evolution of Variability (models) (4 studies)

Four studies report on the challenges and limitations regarding maintenance and evolution of variability (models). The BVR tool [S7] suggests to not use annotations of features but “*relations between feature models and elements of a base model*” to express/capture variability. Study [S30] confirms that with current technologies manipulating and evolving large-scale feature model is challenging and error-prone. Study [S29] argues that not all devices and their characteristics can be known in advance – "*their unique capabilities must be discovered and dealt with efficiently and correctly*". Study [S6] reports that ambiguities in existing feature meta-models negatively affect maintenance.

4.5.6 Usability (4 studies)

Only one study RequiLine [S26] mentions usability to be a limitation of their tool support. However, most tools suffer from this limitation in our own experience. The authors of [S4] admit that the understandability of their variability modelling language/tool must be improved. DOPLER [S12] puts a special emphasis on usability, however, only on the configuration side, i.e., the configuration tools are optimized to allow their use by sales staff. ST2T [S24] provides sophisticated visualisation and interaction techniques to make complex variability models usable by engineers.

4.5.7 Integration of Variability Management and Legacy Software (3 studies)

Three studies report about the challenge of integrating variability management support into legacy software. The development of XToF [S13], for instance, was motivated by industrial needs. One of the key goals was to develop support for variability management that does not require changing current development practices in the organization requesting support. Thus a code-tagging approach was applied. The authors argue that it is important to provide tool support for variability management, but this support must be nicely integrated with existing tools and processes. The development of FeatureIDE [S15] was challenged by the difficulty to integrate variability management and Eclipse. The author of the ToolDAy [S27] argues that supporting integration with tools like DOORS is essential (though not supported by ToolDAy).

4.5.8 Process Improvement/Automation through Variability Management (2 studies)

Two studies describe the challenge of improving development processes through automation provided by variability management tools. The authors of study [S19], for instance, argue that “on the one hand, the non-existence of a unified way to introduce the contents [leads to] an unnecessary waste of time for the employees to learn new technologies and feel comfortable with the new platforms. On the other hand, a rapid prototyping platform is also desirable for showing their customers a working prototype at an early stage.” The authors of study [S1] highlight the need for models that are expressive enough for automation.

4.5.9 Compliance (with standards/quality policies/regulations) (1 study)

Study [S13] stresses the need for compliance, i.e., they argue that it is also important that variability management/modelling tools do not violate with standards/quality policies/regulations in the organizations in which they are used.

4.5.10 Non-functional Properties in Variability Management (1 study)

Study [S29] argues that resource consumption constraints are not taken into account by existing configuration approaches and tools.

Table 4.12 below, presents a summary list all the challenges faced by current tools as well as limitations of the tools a long with number of studies that discusses each problem.

Table 4.12: Summary results of VM tools challenges

Challenge/problem	Number of studies	Total (%) of occurrence
Scalability of variability models	12	40%
Checking models for consistency and correctness	7	23%
Mapping problem and solution space	6	20%
Visualisation/graphical overload is discussed as a challenge	5	17%
Usability	4	13%
Maintenance and evolution of variability models	4	13%

Integration of variability management and legacy software	3	10%
Process improvement/automation through variability management	2	7%
Compliance (with standards/quality policies/regulations)	1	3.5%
Non-functional properties in variability management	1	3.5%

4.6 Summary

This chapter presents a critical analysis of the 37 variability management tools identified and reported in a survey, and contains a systematic literature review to understand the tools' characteristics and maturity, as well as the challenges in the field. The tools are based on diverse development environments, apply diverse technologies, and support different variability modelling approaches. Most tools support a feature modelling approach. Different graphical and textual notations are provided by the tools, with a focus on tree-based visualisations of features. Only few tools provide multiple views, e.g., a graphical view of features together with a text-based representation of source code variability.

While most studies about variability management tools provide a good motivation and a description of the research context they often lack data, e.g., from empirical studies with tool users, to support the claims made and the findings reported. Also, studies seldom provide a critical reflection of the presented tools and their limitations. Most variability management tools were not well evaluated, especially with respect to feedback from end users. Quality attributes important for the practical use of tools such as usability, integration, scalability, and performance are out of scope for most of the analysed studies. This might be explained by the fact that most studies have been conducted in an academic context. Only 6 of 37 studies are joint industrial academic endeavours.

Many studies discuss challenges, i.e., what was the challenging part of implementing the tool and/or what challenges related with variability management and SPL engineering the tool addresses. A detailed analysis of these challenges has been performed to guide future research.

The chapter concludes that the key challenge of variability management tools is scalability of models, i.e., how to support the development of variability models that are still useful despite their size and complexity. The second most discussed challenge is checking models for consistency and correctness, especially with regard to how to keep the models consistent with the underlying architecture and how to check that the models represent the variability of the product line correctly. This is also related with the third most important challenge, i.e., providing support for mapping problem and solution space. Visualisation of models and the resulting potential graphical overload of users are also recognized as important challenges. While these challenges, together with the importance of usability of variability management tools, are recognized as important challenges in many studies, only few actually address them or provide empirical proof that the reported tool helps to address the challenges.

Further challenges mentioned as important for variability management tools are support for the maintenance and evolution of variability models, integration of variability management and (legacy) software (development), process improvement and automation through variability management, managing non-functional properties (e.g., resource consumption constraints), as well as compliance with standards, quality policies, and regulations.

The analysis presented in this chapter do not only provides a good overview of existing variability management tools and the challenges for variability management tool support, but also establishes criteria and concepts for comparison of such tool support. The main hope is the study will encourage authors of approaches and tools to report on those aspects (particularly empirical studies on tool usefulness) and compare their tools with others.

PART II: MUSA 1 Vs MUSA 2

Chapter 5

Theoretical Foundation of MUSA

5.1 Introduction

So far, we have introduced a number of tools and techniques for managing variability in software product lines, together with a detailed analysis of the state-of-the-art of the research field. Within these techniques, feature modelling approach has been the most widely used, to represent, manage, and visualise the variability of product families and their configurations.

In chapter 4, we analysed and critically discussed about a number of variability management tools and modelling techniques, and the approaches they used in tackling variability-related challenges. We have also described the characteristics, maturity, and technology, based on which they were implemented. We have also discussed about their limitations and challenges in the field.

This chapter presents the early version of MUSA, implemented based on our theoretical foundation on multiple perspective-based Variability Management–the Four View Model (4VM)–which is aimed to alleviate the problem of information overloading. MUSA was implemented on Microsoft Surface and

Windows 7, with touch pack platforms. The chapter also describes the theoretical background as well as the technical background, which explained a series of funds received in order to implement MUSA as a proof of concept. Some of the functionalities of the early version of the MUSA tool were also presented.

5.2 Backgrounds and Motivation

In a real life project, software product lines can generate a large number of features that are extremely interconnected, both hierarchically, and in a non-hierarchical order; this is typically in the order of thousands in many cases (SCALE'09, 2009). The model usually goes beyond the control of human cognitive abilities and is too challenging for automated reasoning. Although feature modelling techniques are widely used to represent and visualise variability features, evidence from practice shows that this method has limited scalability (Reiser and Weber, 2006). These include, among others: (1) difficulties in providing effective supports of the artefacts representing different elements in the model, and (2) creating, editing, and interpreting specific features of interest.

However, other information visualisation techniques that focused on representing large and structured information were also explored. These are: (1) the node-link (Holten et al., 2011) – represented as a graph layout, in which a node represents the individual elements of the information and relationships between these elements, which are represented as edges, and (2) the treemap (Shneiderman and Plaisant, 2004) – a technique that provides a holistic visualisation of hierarchical data, using a set of nested rectangles, where each rectangle binds with smaller rectangles to form sub-branches.

These information visualisation techniques are effective and efficient to support a software product line development process by allowing large variability models to be represented and visualised appropriately. In addition to providing mechanisms for navigation within a large data, they also reduce the complexity of the data models, making them understandable for the stakeholders. Unfortunately, most of these techniques suffer from visual clutter when the number of child nodes grow exponentially in the order of 2^n, thus raising a scalability issue that requires an exponential amount of space for the data to be displayed more appropriately.

On the other hand, hyperbolic trees (Lamping et al., 1995) provide an adequate layout for visualising large scale data and hierarchies. Hyperbolic trees use hyperbolic space, which provides more room for appropriate representation of data as compared to other techniques such as Euclidean Geometry space. However, the focus of a hyperbolic tree is typically on contextual visualisation, helping users focus on a particular element of information. When applied to product line engineering, hyperbolic trees can offer more appropriate and clear representation of variability, variation points, and their variants, hierarchically.

5.3 Concept of Multitouch Technology

From a computing perspective, multi-touch is a technology that enables devices (touchscreen or trackpad) to recognize and respond to two or more simultaneous touch inputs, allowing one or more users to interact with computer applications through various gestures and pressure created by fingers on a surface. This is in contrast to single-point input devices, such as a mouse or a traditional touchpad, where users can select a single point, drag and drop, push and slide. Multi-touch technology allows users to swipe, pinch, rotate, and perform other actions that allow for richer, more immediate interaction with digital content.

Multi-touch technologies have a long history, but the first one designed for human input to a computer system began in 1982, when the University of Toronto introduced a system that used a frosted-glass panel with a camera placed behind the glass. When a finger or several fingers pressed on the glass, the camera would detect the action as one or more black spots on an otherwise white background, allowing it to be registered as an input (Mehta, 1982). Following this was the introduction of the first multi-touch screen capable of simultaneously capturing multiple touch-points on a display, which was developed by Bob Boie in 1984. This used a transparent capacitive array of touch sensors overlaid on a CR, and allowed for manipulating graphical objects with one’s fingers with excellent response time. This eventually led to the release of what has been considered the world’s first smartphone by IBM and Bell south in 1992 ('Bellsouth, IBM,' 1993).

5.3.1 The Benefits of Multi-touch over Single Touch

Multi-touch technology expands the functionality of traditional input devices, such as the keyboard, mouse and stylus, with new ways of interacting with information. For instance, two fingers can allow users to zoom in and out, or scale the display. The need for two activation points has been widely recognized in the industrial environment, that is, to have both the user's hands on the screen.

Furthermore, secure keyless entry to a room can be implemented with a fingerprint via touch display. Different security paradigms can be combined to implement a high level of security, e.g., unique gestures on the touchscreen display serve as the new password, while the meeting schedule further secures entry.

Another good example is building automation: imagine that your building is big enough that when the floor plan fills a display, the details are rendered too small to see. At this level, all you can do is get an overview, which may be enough for new visitors trying to find their way around, but proves insufficient for more specific needs.

Likewise, if a user needs to read a manual, multi-touch enables two-finger scrolling, pinching, spreading and rotating without a complicated learning curve.

5.4 MUSA Theoretical Background

MUSA (A Multi-touch Variability Modelling Solution for Software Product Lines) is designed to implement our theoretical work (Bashroush et al., 2008, Bashroush et al., 2011) on multiple perspective-based variability management, which provides a successful modelling framework while using the concept of separation-of-concerns to alleviate the problem of information overloading. As stakeholders have an interest in the different views of a product line variability model (Nuseibeh et al., 1994), it is important for a variability model to be able to represent and extract relevant information without overloading the graphical representation of the model.

The Four View Model for Variability Management (4VM) aims to alleviate this overload (Bashroush, 2010). The design and implementation of the MUSA tool was achieved by following the 4VM model. The model proposes the distribution of feature modelling information into four views, with each view dedicated to a particular theme and group of stakeholders. The views are:

- ***Business View****:* In this, the information associated to the project management, cost/benefit analysis, closed/open sets of features and others is presented. Project managers are the main targets with a view where they can specify feature costs, open and closed features, feature introduction time, etc.
- ***Hierarchical & Behavioural View***: This is where the different features are organised (usually presented in a tree structure), along with the behaviour attached to each feature is presented. The main concerns of this view are twofold: the software architects, and end users' requirements, need to be captured. This view is currently the most widely adopted by many feature-modelling techniques.
- ***Dependency & Interaction View***: Here the dependency and interaction among the features (e.g., inclusion, exclusion, etc.) are

presented. The focus of this view is towards architects, and offers a suitable basis for capturing feature dependency and feature interaction. The view is a complement of the Hierarchical & Behavioural View.

- ***Intermediate View***: Is where some design decisions are inserted into the feature model to take it one step further towards the architecture domain, in an effort to bridge the gap between the feature model and the system architecture. This view is centred towards architects, and provides a transition stage towards the architecture.

5.5 MUSA Technical Background

To demonstrate the theoretical groundwork in 4VM, the European RD Fund, through INI funded MUSA as a proof of concept project under the Proof of Concept funding scheme [2008-2010]. Further funding was received under the Challenge Fund scheme at the University of East London [2010-2011]. MUSA implements this theory using a mind-mapping modelling approach over the state-of-the-art in HCI (Human Computer Interaction), the multi-touch Microsoft Surface (Dietz and Eidelson, 2009). This offers a scalable solution that taps on the latest technology in Natural User Interface (NUI) (Microsoft, 2008) design, providing an intuitive and large display for Variability models. In addition, the MUSA provides solutions over the Windows 7 platform, using its native multi-touch pack.

As part of its innovative support for product line variability, MUSA provides a comprehensive collaborative interface for eliciting variability and requirements management from stakeholders, while at the same time allowing for suitable access to the variability model to different teams, such as requirement engineers, architects, implementation, testing and evaluation teams, etc.

MUSA provides end-to-end variability solution, in addition to automation of model verification using SAT solvers. It allows consistency between the different views to be maintained with the help of a centralised database (see Figure 5.1).

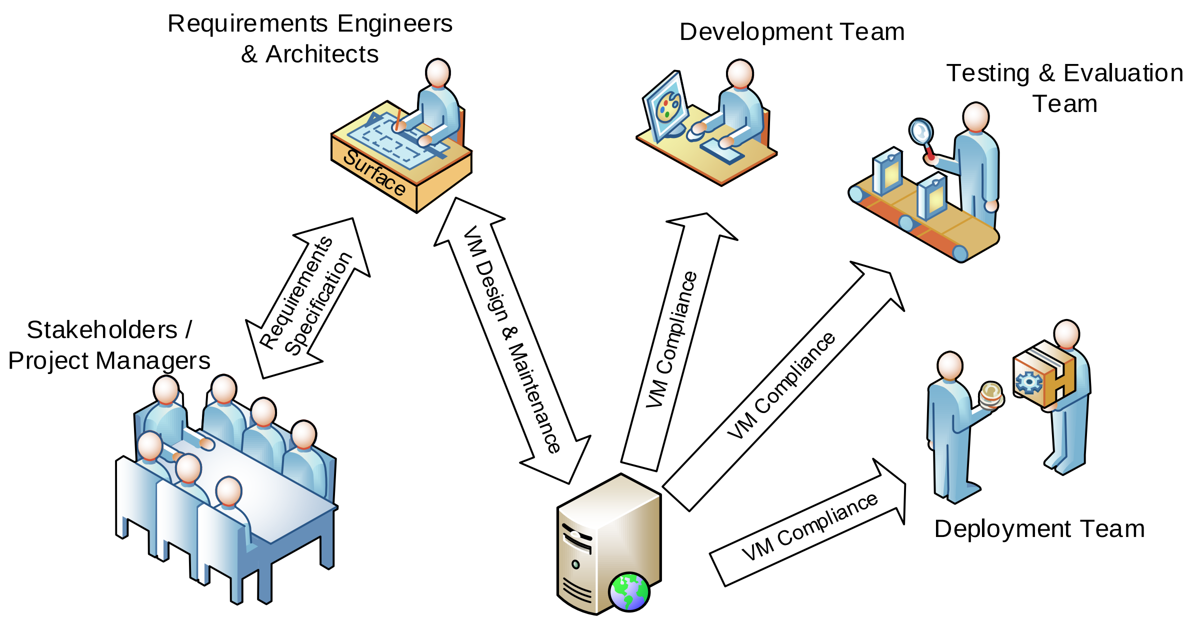

Requirements Engineers & Architects
Surface
Development Team
Testing & Evaluation Team
Stakeholders / Project Managers
Requirements Specification
VM Design & Maintenance
VM Compliance
VM Compliance
VM Compliance
Deployment Team

Figure 5.1: Theoretical Foundation (adapted from (Bashroush, 2010)

During the first official demonstration of the MUSA system, the focus was mainly on the interface that is used to manage variability and requirements elicitation, targeting mainly the architects/requirements engineers. The main functionalities are: (1) it provides large gesture-based interface for the modelling of variability in SPL. (2) It uses 360-D User Interface (UI) design principles and Natural User Interface (NUI) to provide a multi-user interface simultaneous interaction and collaboration, and (3) It uses mind-mapping techniques (hyperbolic tree) in the implementation of the variability model, providing a potential scalability in a large model.

5.6 Implementation of the Earlier Version of MUSA Tool

The MUSA tool suite was initially implemented on the Microsoft Surface platform and Windows 7, with a touch pack platform. It used hyperbolic trees and supporting gesture-based interaction (multi-touch interaction) for representing and visualising the variability models, which makes it a powerful solution for creating and managing large-scale product lines.

However, the initial version of MUSA was developed as a prototype due to some limitations with the surface platform, such as hardware issues inherited from surface technology, and software issues such as platform dependency. For this reason, many practitioners did not adopt MUSA; hence, there is need for making it more generic. Although Microsoft has recently rolled out cheaper and more portable versions of Surface, the earlier version of Surface was a bulky piece of hardware that came along with a table for it to be mounted upon. This made it very heavy and non-portable as a piece of hardware. In addition, there is also the fact that the Surface was too expensive when it initially hit the market. Figure 5.2 is a MUSA architect interface showing a hierarchical view, and displaying a set variability models on a MS-Surface.

Figure 5.3 is a Windows 7 interface, showing variability models displayed on a hierarchical view of the MUSA tool.

From these two figures, it can be noticed that different features are distinguished using colour coding; namely, optional (blue), and mandatory features (yellow). The existing Microsoft Surface-based MUSA system requires user access cards that can be placed over the surface interface and get recognised by the system. Appropriate access is then granted in accordance with user privileges. Once the user has successfully logged in, among others, he can select and load the existing feature trees that are structured in the hierarchical model and stored using xml file, from which the user can browse through the features, view the details of the features, and its sub-features.

The user can also recognise feature types (Mandatory and optional). However, depending on the user privilege, he can make changes to the feature model. This implementation of the MUSA system over Natural User Interface (NUI) was considered among the very first of its kind in order to overcome scalability issues. This, however, improves the interactivity and visualisation of the product line variability models. On the other hand, the Windows 7 platform login process does not require an access card as it does not support optical processing capabilities. Instead, it uses a standard login screen on which a user can login with valid credentials.

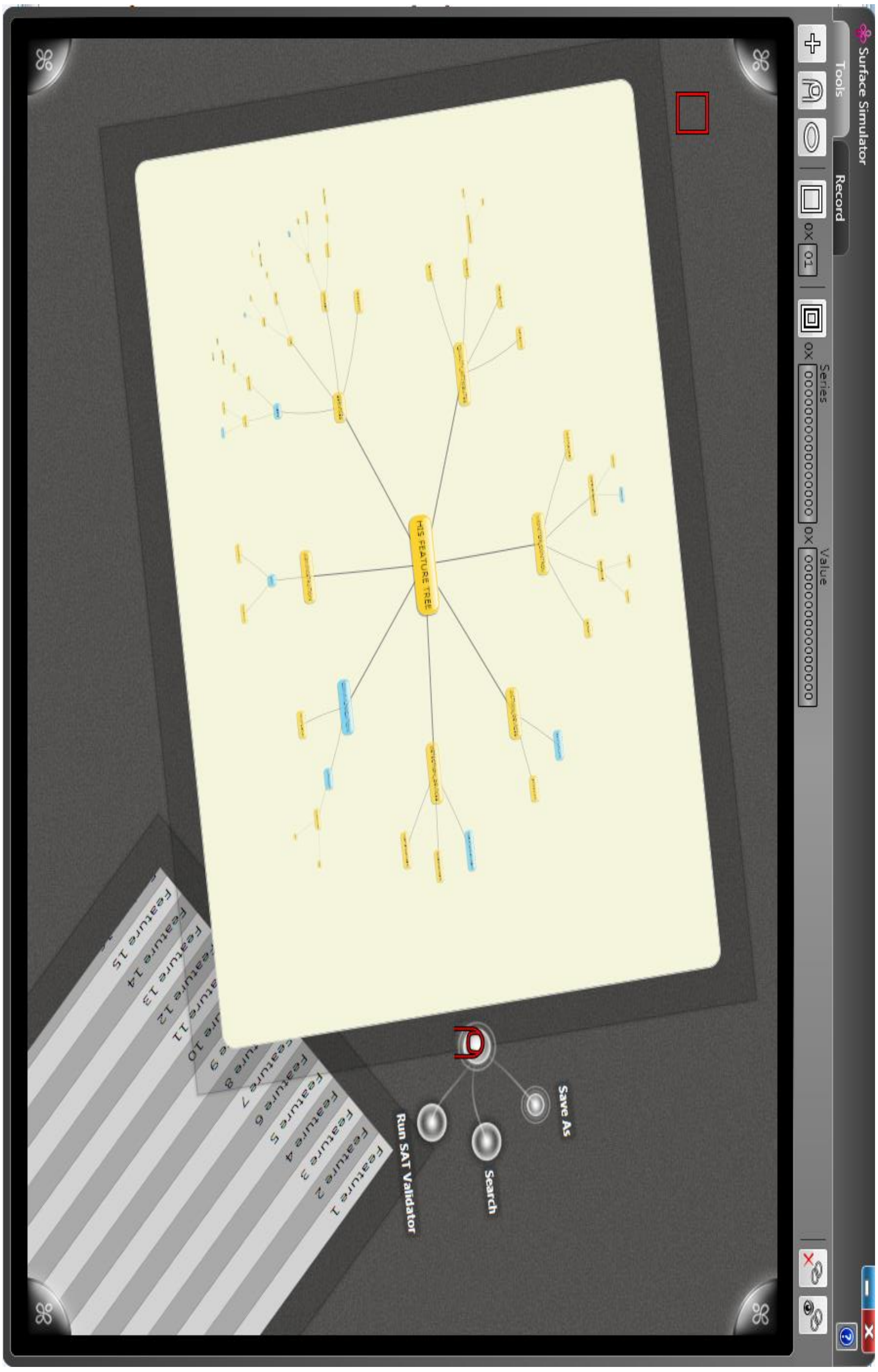
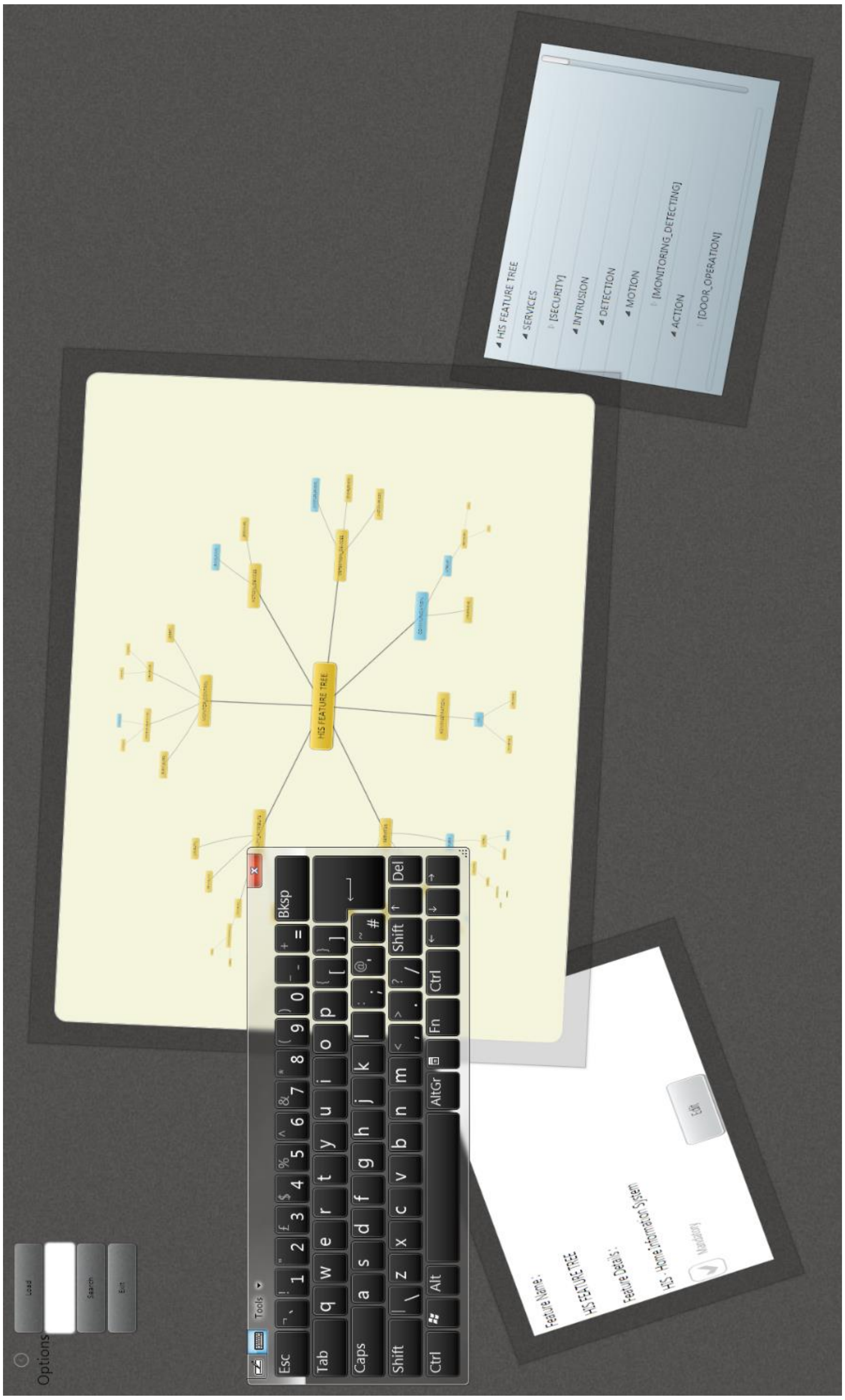


Figure 5.2: MUSA designed interface on MS-Surface showing the hierarchical view

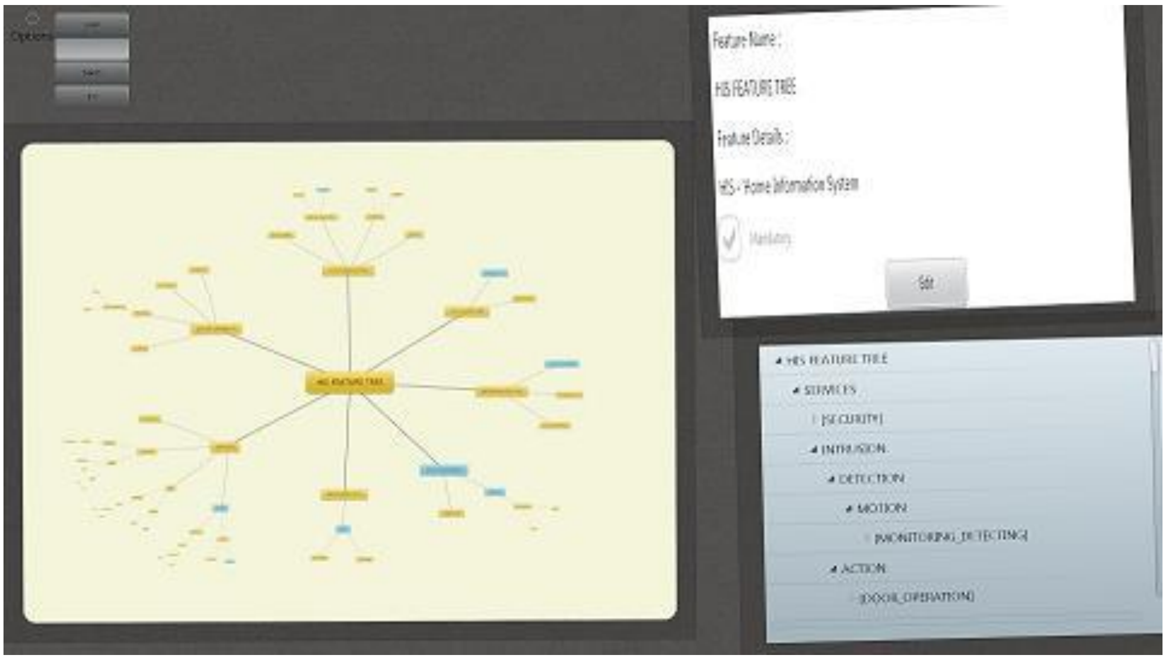


Feature Name :
HIS FEATURE TREE
Feature Details :
HIS - Home Information System

Figure 5.3: MUSA designed interface on Windows 7 showing the hierarchical view

5.7 Screenshots and Descriptions of the MUSA Tool Version One

The application loads a default tree structure as the main application screen, as shown in Figure 5.4. The user can also load and view a different feature tree by clicking or touching the load button, which opens up the Open-File-Dialog window, from which the user can select and load the needed feature file, as shown in Figure 5.5. Once the feature tree has been loaded into the application, the user can navigate through it, as well as view the details of its features. Touching a feature will automatically load its details, as shown in Figure 5.6. Other functionalities include editing features (Figure 5.7), distinction between mandatory and optional features (Figure 5.8), placing the nodes in focus in the centre, searching for a feature to locate its position in the feature tree, etc.



Feature Name :

HIS FEATURE TREE

Feature Details :

HIS - Home Information System

✓ Mandatory

- HIS FEATURE TREE
  - SERVICES
    - [SECURITY]
    - INTRUSION
      - DETECTION
        - MOTION
          - [MONITORING_DETECTING]
      - ACTION
        - [DOOR_OPERATION]

Figure 5.4: Main application window after successful log in

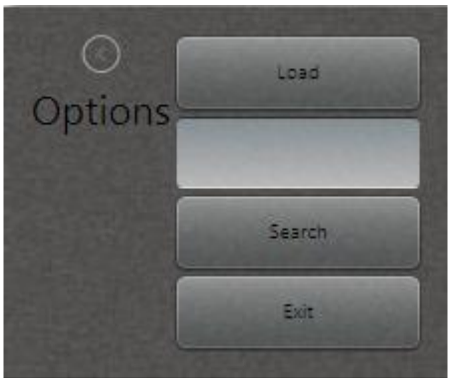
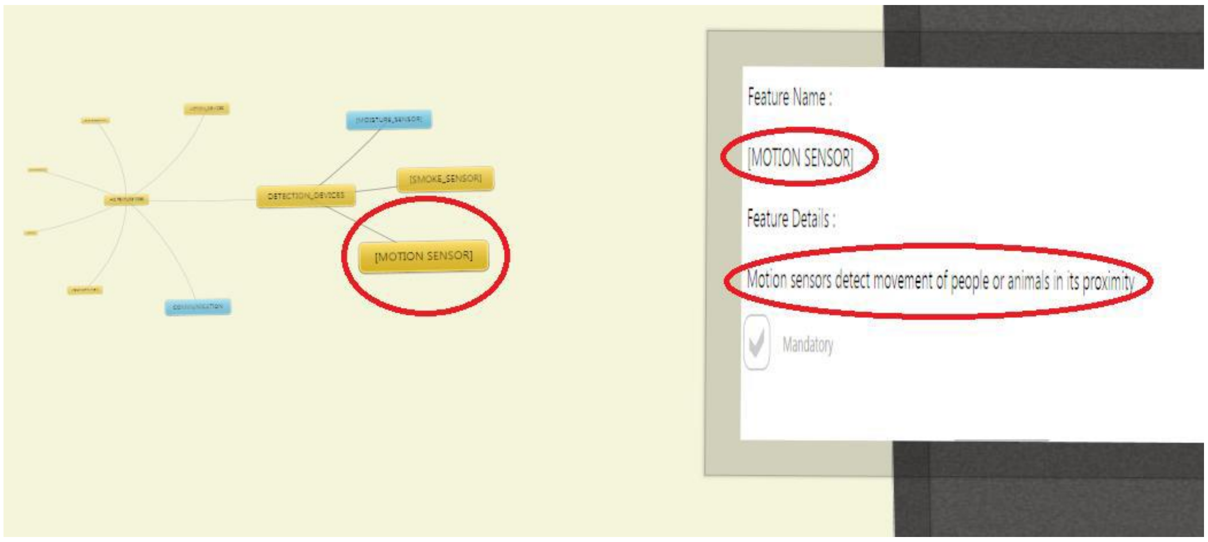


Figure 5.5: Options menu to load a tree



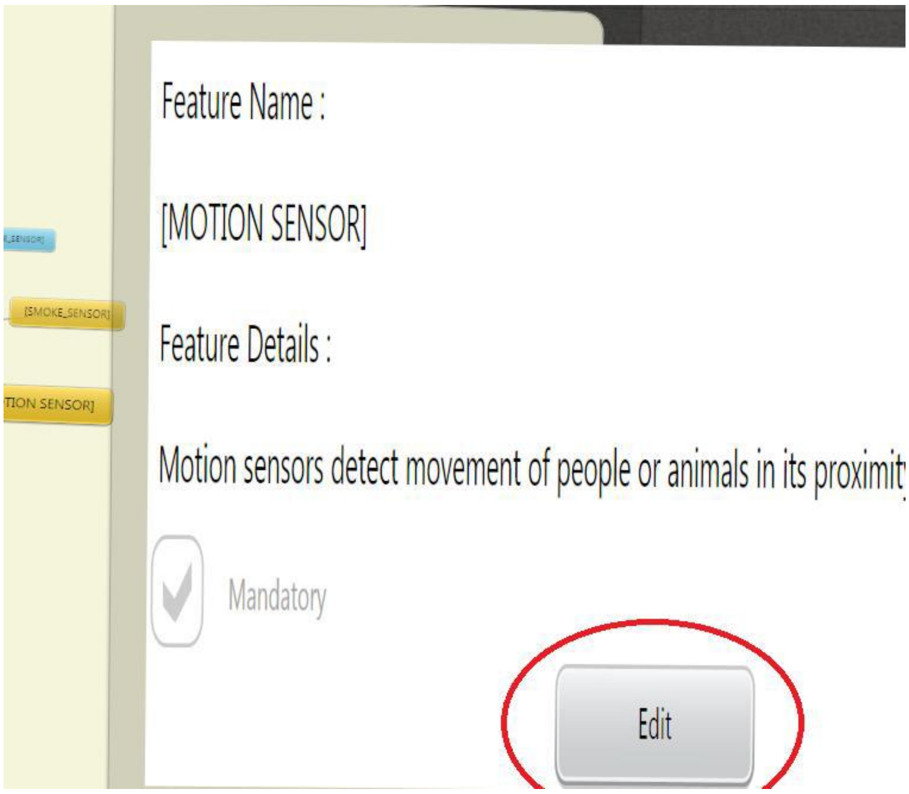
Feature Name :

[MOTION SENSOR]

Feature Details :

Motion sensors detect movement of people or animals in its proximity

Figure 5.6: Viewing details of a selected feature


Feature Name :

[MOTION SENSOR]

Feature Details :

Motion sensors detect movement of people or animals in its proximit

Figure 5.7: Click Edit button to start editing

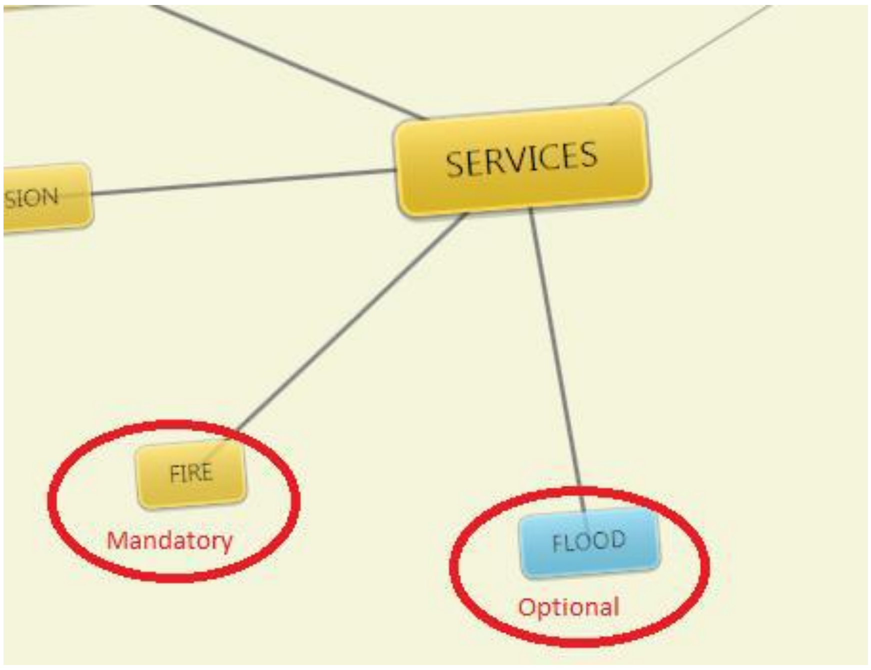


Figure 5.8: Mandatory and optional feature distinction

5.8 Summary

This chapter describes the early version of MUSA tool and framework, implemented as a proof-of-concept on two different (the Microsoft Surface and Windows 7 touch) platforms. This implementation was based on the four view model (4VM), a successful work on multiple-perspective-based variability management, which provides a modelling framework while using the separation-of-concerns approach to alleviate the challenge of information overloading. MUSA uses a mind-mapping modelling technique (hyperbolic tree) in the implementation of this theory, over the state-of-the-art in Human Computer Interaction (HCI).

Chapter 6

Musa Version 2

6.1 Introduction

The previous chapter presents and describes the theoretical foundation on the basis of which the first version of the MUSA tool and framework was developed, along with its implementation and the limitations that motivated the redesign of the framework. In this chapter, a new version of the MUSA tool, which exhibits a number of features that enable it to deal with large-scale systems, is presented. MUSA adopts the separation-of-concerns design principle by providing multiple perspectives to the model, each of which conveys a distinct set of information. The tool was demonstrated on an industrial case study consisting of more than 1,000 features. The demonstration was conducted to show the Structural View, which is displayed using a mind-mapping visualisation technique (hyperbolic trees), and the Dependency View, which is graphically represented using Logic gates.

In this study, we still recognize the use of the mind mapping approach using a hyperbolic tree as the best-known technique in making better use of a screen by representing large amounts of data without the problem of graphical overloading. It is better than any other approach, like traditional tree browsing interfaces, a space tree, file tree-like structures, and so on, which can be cumbersome to use as soon as the number of variants reach about a hundred.

6.2 The Musa Tool

The new version of the MUSA tool is implemented in Java and uses XML files to input/output data. It provides two different collaborative interfaces (i.e., views) for managing variability models, and their consistencies are maintained with the help of a centralised database (see Figure 6.1). The Development/Browser View is the default view when the application is initially launched. The main functionalities covered by this view include: (1) Representation of product line variability models using a hyperbolic browser; (2) creation of new feature trees for managing variability; and (3) editing existing feature models (e.g., changing a feature's name, its properties, and description; adding and deleting features, etc.).

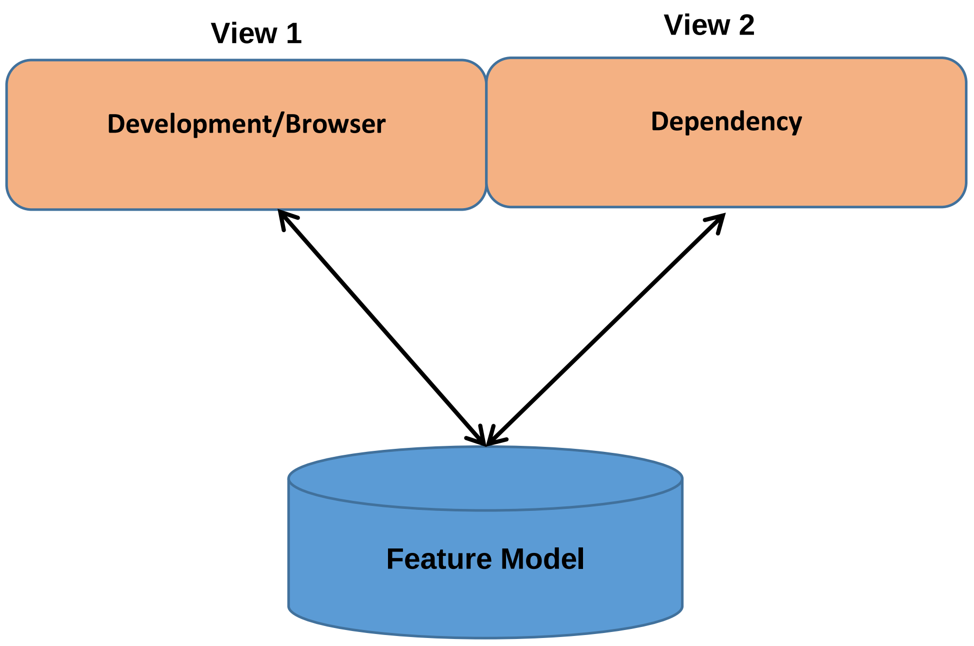


Figure 6.1: Description of MUSA's architecture

The hyper-tree browser uses hyperbolic geometry to place nodes around the root and provides smooth and continuous animation of the tree so that users can bring other nodes into focus by clicking, tapping on or dragging them. The advantage of using hyperbolic trees is reducing visual clutter compared to standard trees when the number of child nodes grows exponentially. The former employs hyperbolic space, which provides more room than Euclidean space. Using hyperbolic trees gives this MUSA tool an important advantage in scalability. The tool can display models with a large number of features; counting more than 1000+ features are in the relevant case study of this research (see Section 6.3).

6.3 Functionality of MUSA Using Case Studies

In this section, the main features of the new MUSA tool are presented by using real-life product line case studies. This is accomplished by showing how a new feature can be created from scratch, and then a variability model consisting of 100 features is shown. The full functionalities are then described using a large case study that consists of more than 1,000 features. The aim is to show how effective the approach is when applied to product lines of different sizes (i.e., it is capable of managing large or small-size variability without any overhead or extra effort) in terms of managing and visualising variability models. The use of these case studies enables the determination and assessment of the extent to which MUSA satisfies design needs, as compared to other tools available today. A video demo of the new version of MUSA tool in action can be found in: https://youtu.be/Oq18Wv8czUI.

6.3.1 Creating a New Feature from Scratch

A new feature can be created from scratch by tapping or clicking on the file menu and then selecting 'New Root'. The new root feature will be placed at the centre of the window view, as shown in Figure 6.2.

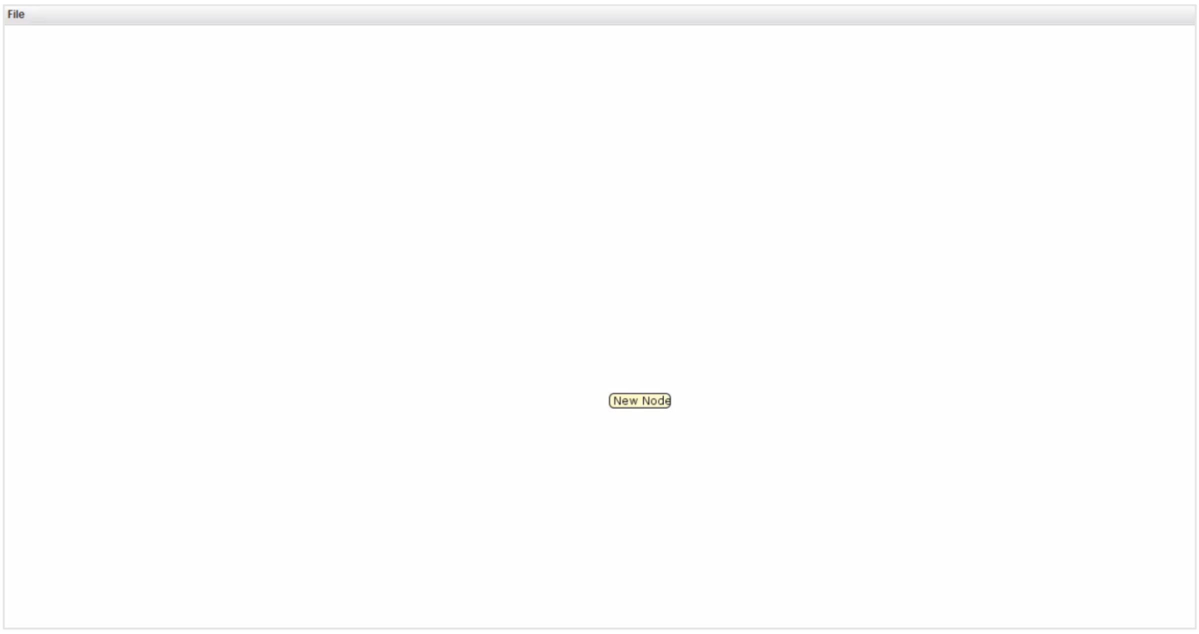


Figure 6.2: A new feature from scratch

To change the name of the new node, touch-hold it -> using the pop-up window shown in Figure 6.3, a new name can be typed in the text box, along with the description of the feature. As an example, the feature node was named as a ‘Test’ feature. Finally, select ‘OK’ to validate or ‘Cancel’ to end the process.

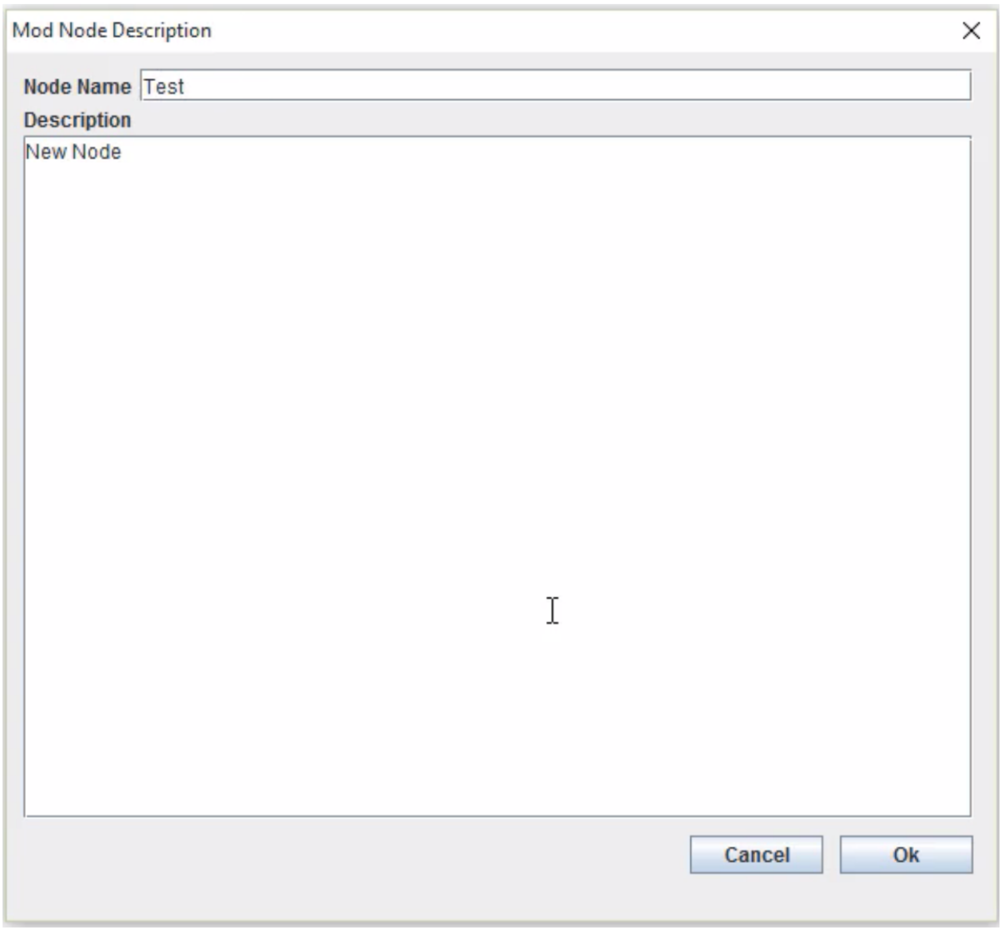


Figure 6.3: Adding a name for the new feature

Now, to add sub-features to the original root (Test) feature, double tap it -> select the ‘Add Button’ from the options that appear (see Figure 6.4), type a name for the new sub-feature, and then move down a bit and select its type as either mandatory, optional or alternative. A description associated with the feature can also be added, as shown in Figure 6.5. Finally, select ‘OK’ to validate or ‘Cancel’ to terminate the process. As an example, TF1-TF9 has been added (see Figure 6.6).

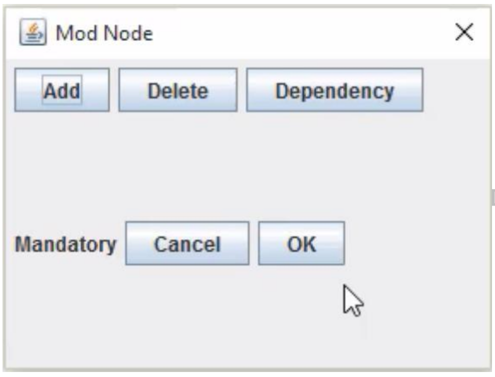
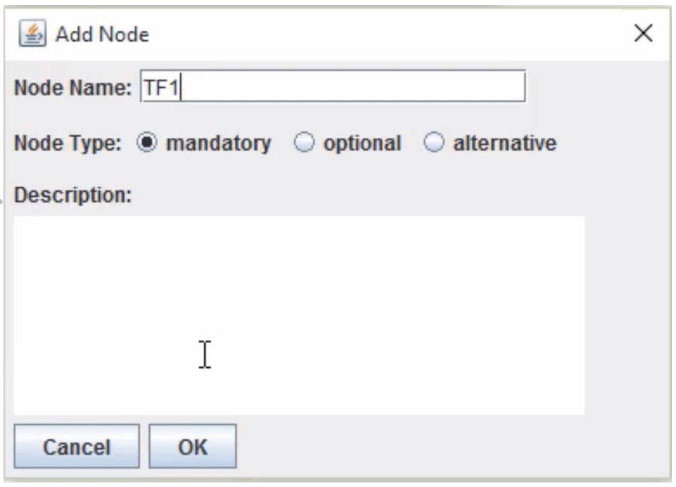


Figure 6.4: Select the Add button



Node Name: TF1

Node Type: [✓ mandatory] optional alternative

Description:

Figure 6.5: Type a name and select its type

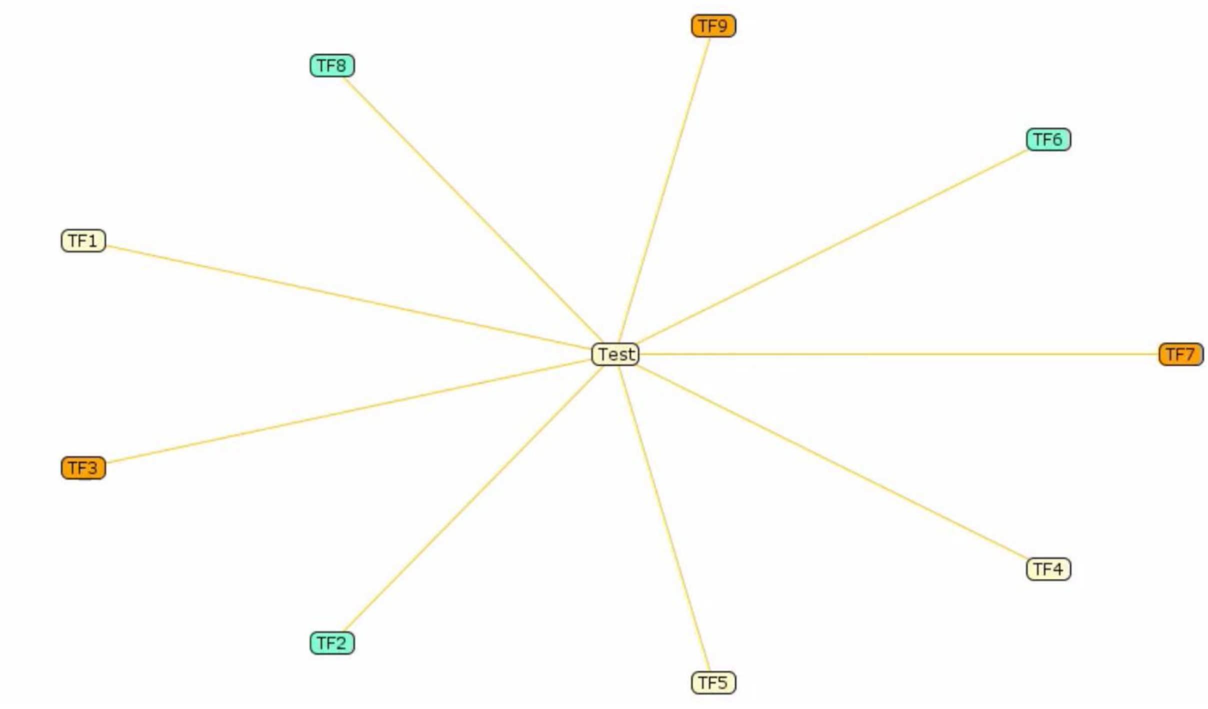


Figure 6.6: Sub-features of Test Feature- TF1-TF9

Note that the different colours associated with these features are for mandatory, optional and alternative: the light yellow is for mandatory and the green is for optional, while the burnt orange colour is for alternative. From here, more features can be added as required.

6.3.2 Medium and Large Scale-Size Models

Moving to one of the case studies used to evaluate the capability of MUSA, one can look at the top left corner of Figure 6.7 and see that this case study consists of 101 features. In fact, handling a variability of around 100 features is one of the limitations of most current variability management tools. Looking at the model, there are only five features attached directly to the root feature, leaving a wide gap between them. Therefore, to show that more features can be added without any overhead, another real-life case study with more than 1000 features has been used (see Figure 6.8).

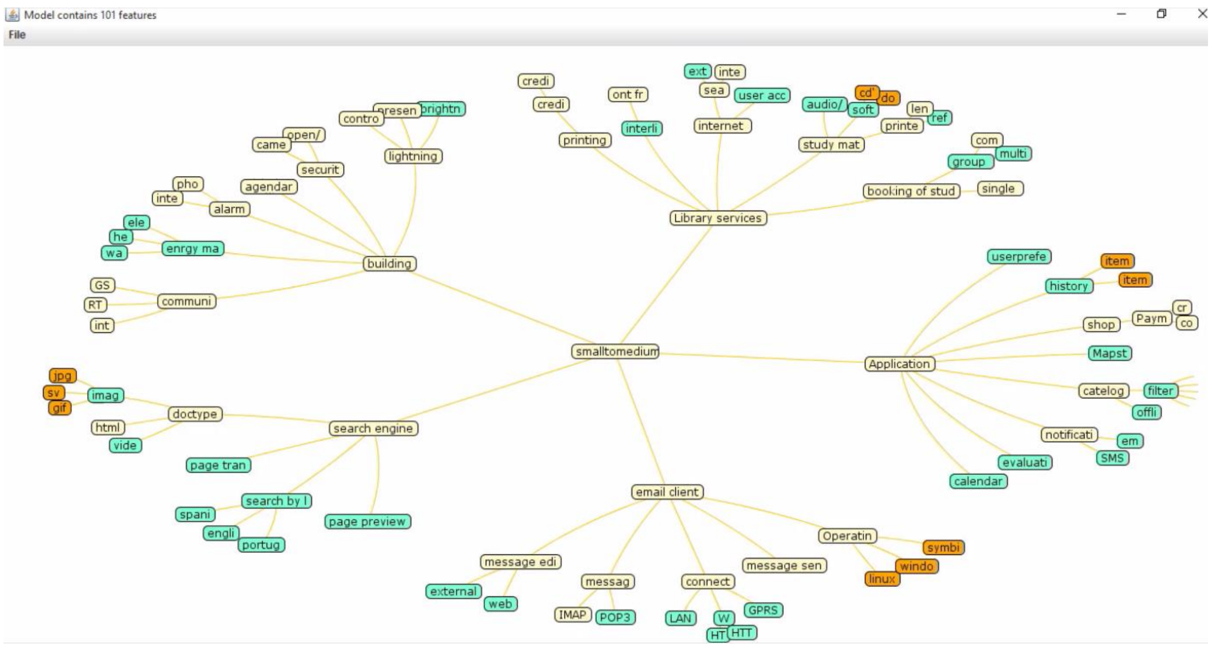


Figure 6.7: Medium scale-size model

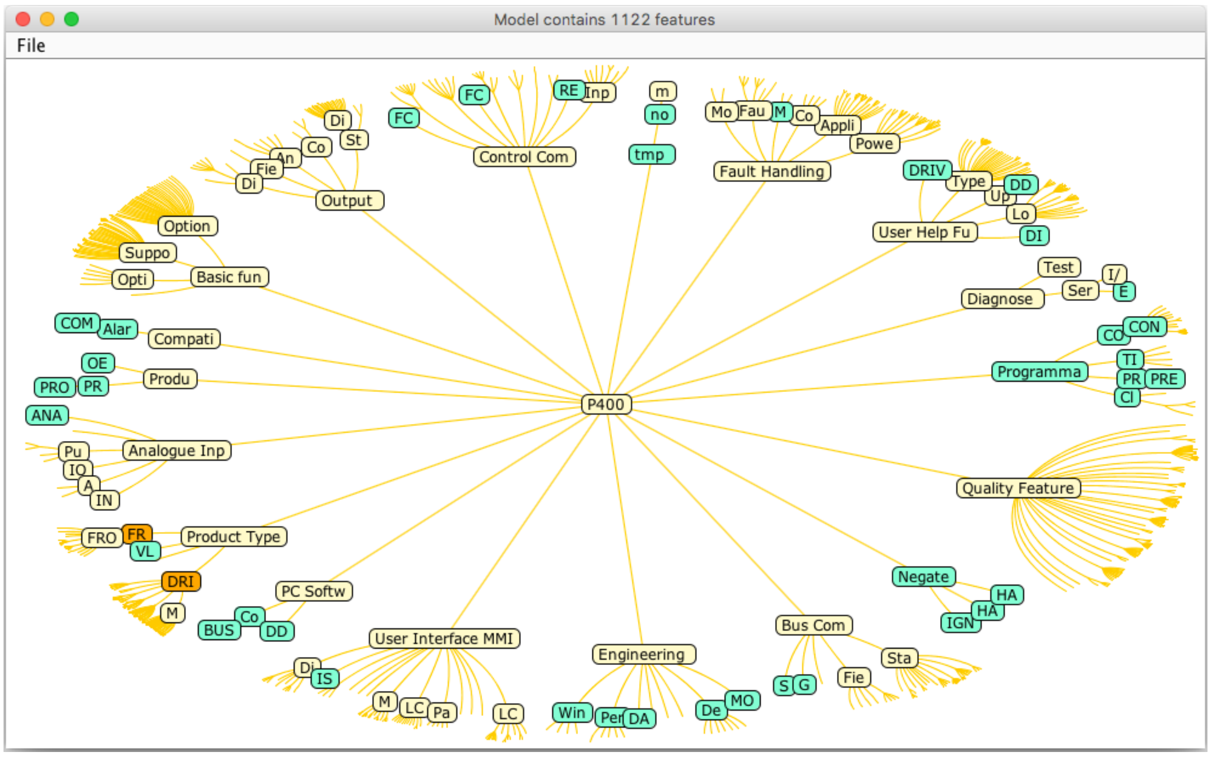


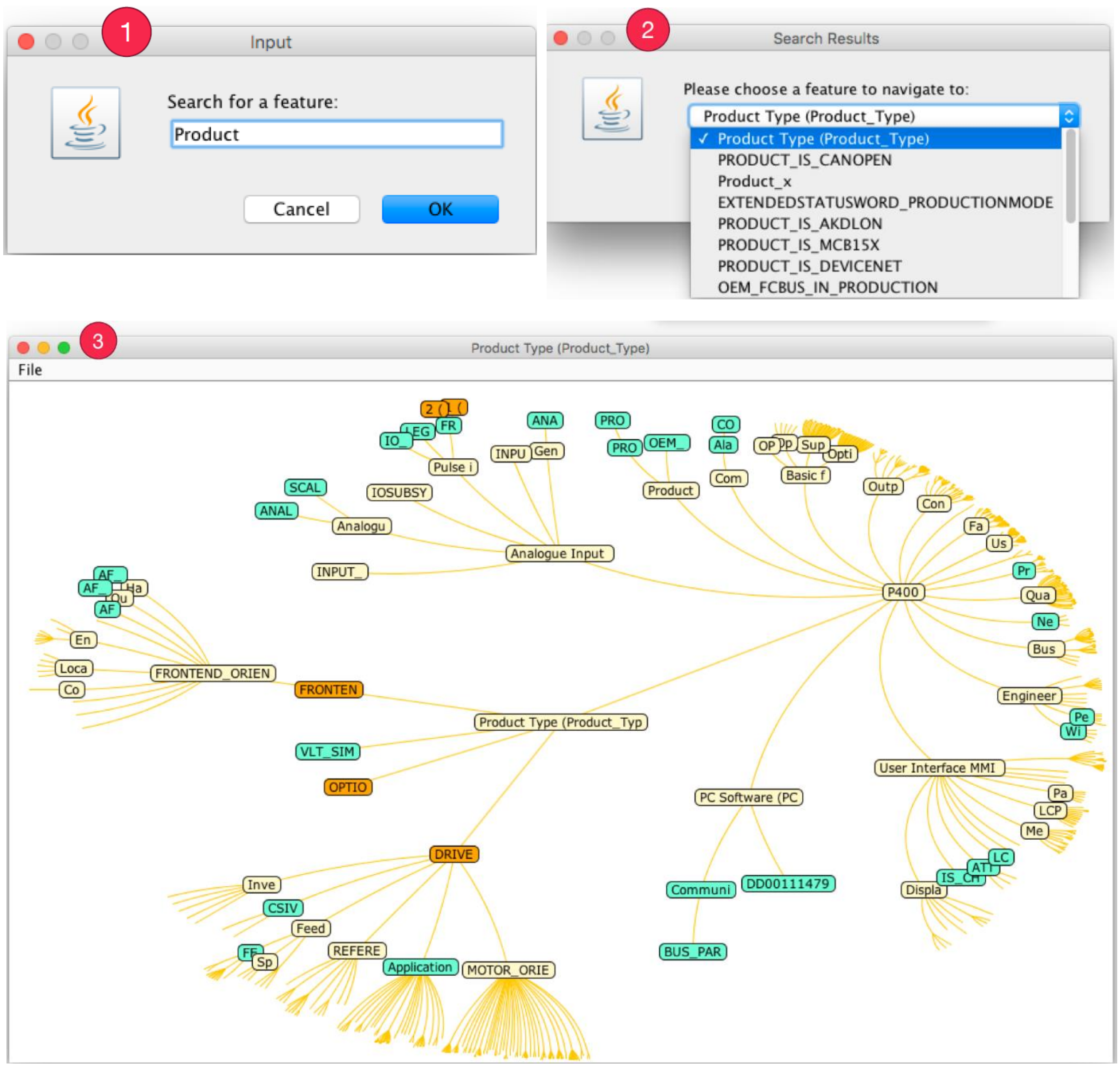
Figure 6.8: MUSA's main browser View

With reference to Figure 6.8, MUSA's browser view shows all the features of the model in the case study in a hyperbolic tree. By default, the root of the tree is centred, while further leaf names are hidden (but their connections remain in order to provide visual feedback for the user). The user can cycle through the features by swiping in any direction with a mouse or directly on a touchscreen. Selecting a feature will centre the screen over it, zooming if necessary, and displaying more connections to related features. Double-clicking anywhere on the background will centre the view back to the root of the model. When focusing on a particular node, MUSA places it at the centre of the screen with all its children, while out of focus nodes will reduce in size and be displayed towards the edge of the view.

However, upon double-tapping a feature node, the option menu with a number of possible options will pop up; this can be used to add a new feature to the existing tree, delete a feature from the tree, or view the dependency relationships that exist among the features (see Figure 6.4). Users can also use different gestures, such as pinching (for expanding nodes), panning (by moving two fingers on the screen to shift the feature model), or tapping with three fingers to centre the model to its root node.

Search in MUSA is straightforward by ‘touching and holding’ on any space (or right clicking), which brings up the search box. In the popup box, users can type the desired search keyword and a list of potential features will be displayed. Touching or clicking any result will centre the view on that particular feature. Figure 6.9 illustrates the search process. Adding or removing a feature in the model can be achieved by double-tapping or clicking on a feature node. A menu will appear with options to add or remove features. If, for instance, the Add button is selected, the user will be prompted with a window where she can type the name of a feature, such as TestFeature, and select its type as mandatory, optional or alternative (see Figure 6.5). The same menu displays an option to view dependencies in a different view. Upon tapping or clicking on the dependency option, the Dependency View will open, showing the selected feature with all its associated relationships. From this view, different kinds of dependency relationships can be created, edited or modified using Logic visualisation.

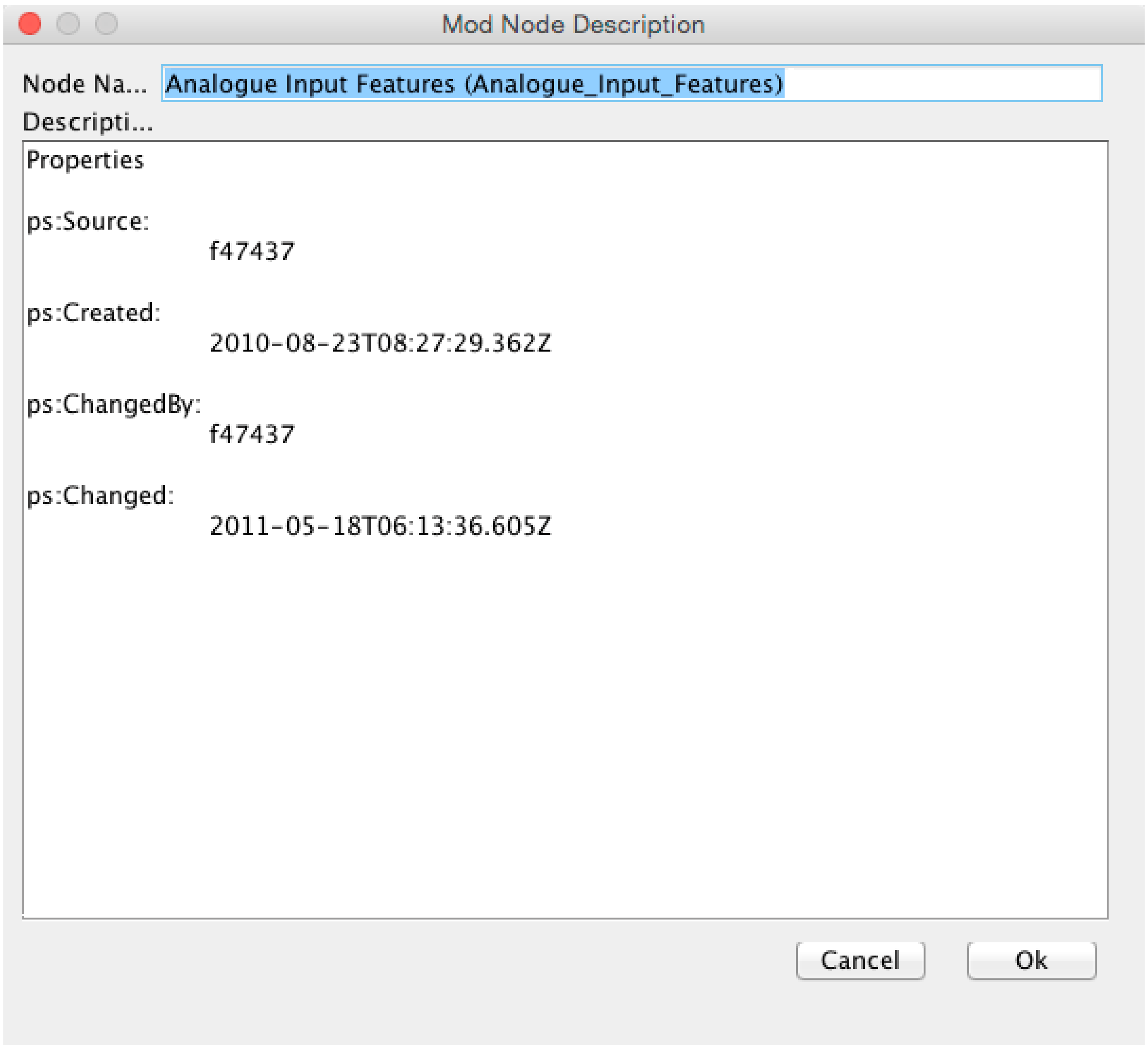
However, viewing the properties and the descriptions of a feature can be achieved by touch-holding or right-clicking it, and a window will then appear containing the details of the selected feature. Figure 6.10 is a properties display window of a feature called *Analogue Input Features (Analogue_Input_Features)*.


Search for a feature: Product

Please choose a feature to navigate to: Product Type (Product_Type)

✓ Product Type (Product_Type)
PRODUCT_IS_CANOPEN
Product_x
EXTENDEDSTATUSWORD_PRODUCTIONMODE
PRODUCT_IS_AKDLON
PRODUCT_IS_MCB15X
PRODUCT_IS_DEVICENET
OEM_FCBUS_IN_PRODUCTION

Figure 6.9: The search process in MUSA


Mod Node Description

Node Na... Analogue Input Features (Analogue_Input_Features)

Descripti...

Properties

ps:Source:
f47437

ps:Created:
2010-08-23T08:27:29.362Z

ps:ChangedBy:
f47437

ps:Changed:
2011-05-18T06:13:36.605Z

Figure 6.10: The feature properties window

6.3.3 Managing Feature Dependencies using Logic Circuits

From the dependency perspective, a separate view is proposed within the MUSA tool by using Logic Design to capture and model the dependency relationships. Once the user makes his/her selection of features from the browser view, the dependency model will take the user-selected feature set as an input and verify it against the model, pointing out any dependency relationships associated with that feature. At the same time, if no relationship for that selection exists, then a new window in the dependency view opens to create new dependencies, if needed. This provides simplicity in managing dependency relationships within large and complex variability models. This study used three basic Logic gate (AND, OR and NOT) symbols from which a user, such as an architect, can generate and resolve any relationship (from simple to complex dependency).

The dependency diagram in Figure 6.11 shows that the *Generic_Product_Code_Parameter* feature requires three other features to fulfil its tasks: the *Parameters 8-19*, the *Production Mode* and the *Product Type*. Figure 6.12 illustrates this. It also shows that one of the required features, the *Product Type,* is mutually dependent on the two other features *(Quality Features and Production Mode)*. Therefore, any selection of this feature will inclusively imply their selection. However, the diagram shows that a conflict exists between the *Product Type* and the *Operation Mode*; therefore, they cannot be chosen for the same product configuration, that is, they are mutually exclusive to each other (see Figure 6.13 for a breakdown). Hence, a bi-directional exclusive relationship exists between the two features.

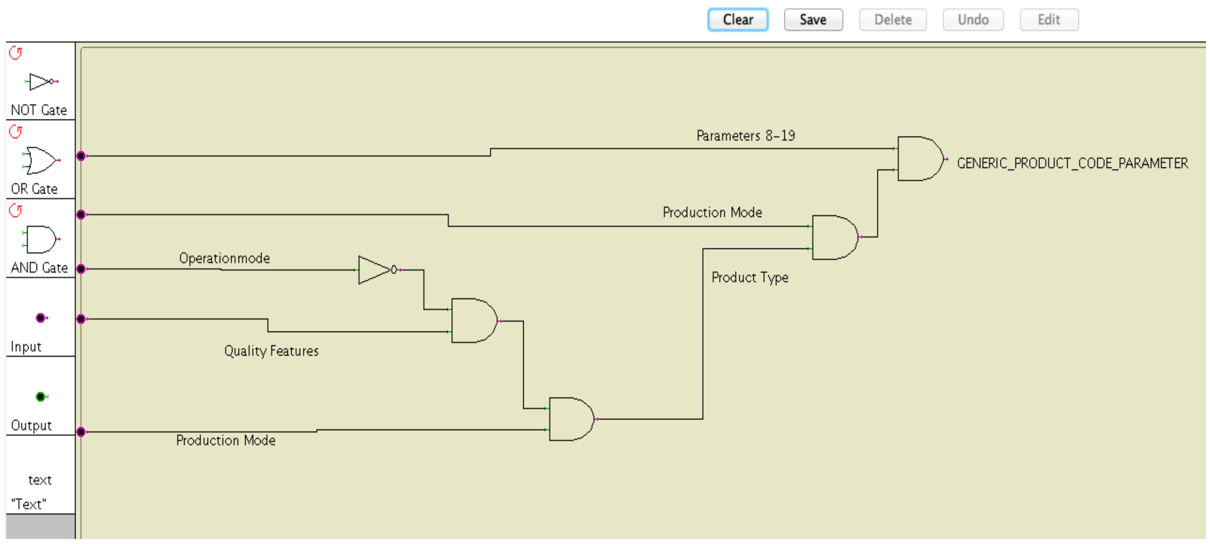


Figure 6.11: MUSA's dependency view

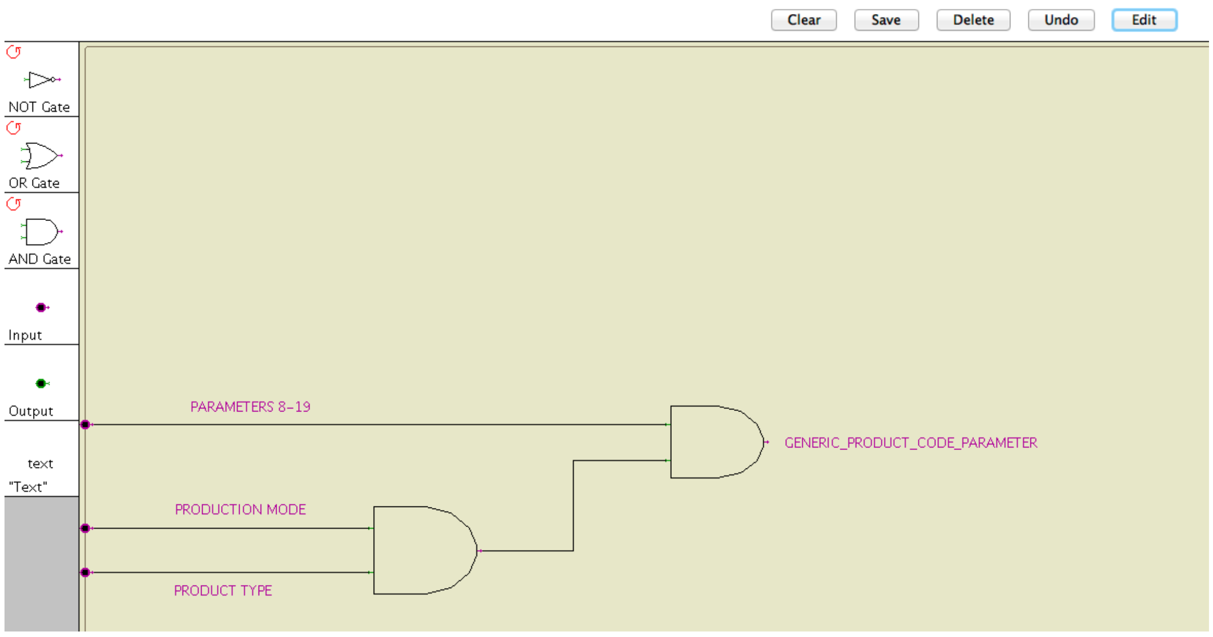


Figure 6.12: Generic_Product_Code_Parameter feature is mutually dependent on Parameters 8-19, Production Mode and Product Type features

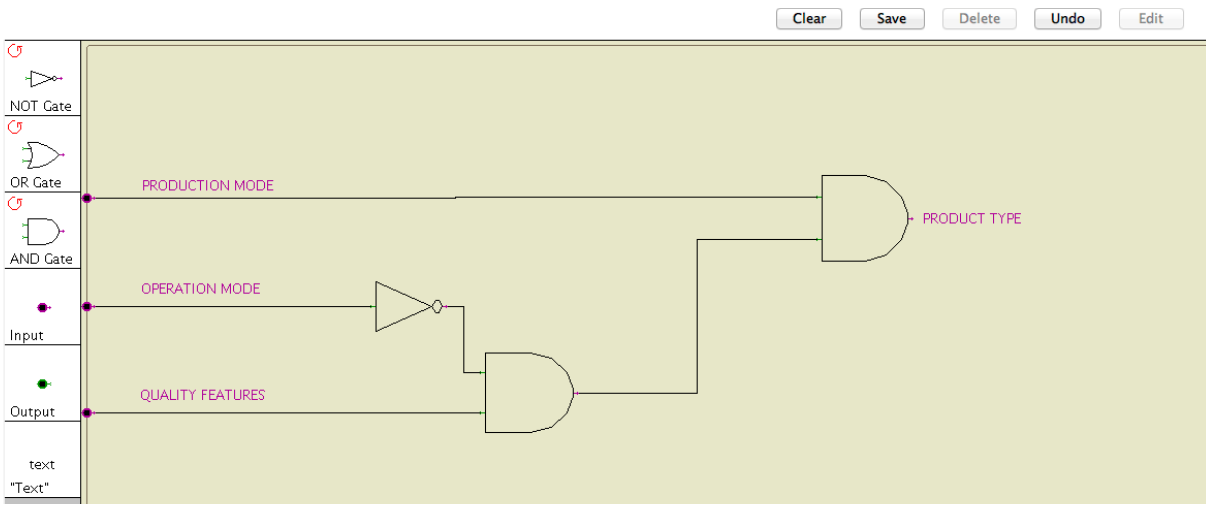


Figure 6.13: Mutually exclusive relationships between features

6.4 New Version of MUSA as Compared to Earlier Version

This section takes a look at some of the improvements of the new version of the MUSA tool suite over its predecessor.

The new MUSA system, as compared to its earlier version, is now independent of any specific technological platform, as it can be directly run on any hardware platform (PC, Mac, SunSparc, etc.) or software platform (MacOS, Unix, Windows, Linux, etc.). However, in addition to the inclusion of all functionalities of the previous version, a separate view has been introduced to the new MUSA to manage dependencies. This has alleviated the problem of graphical overloading when viewing and managing large variability models along with their dependencies, all from one view.

This version has also introduced a new mechanism for identifying alternative features (i.e., when exactly one feature in a group must be selected; if the parent feature is selected), which was lacking in the previous version. This is in addition to various improvements (such as innovative visualisation technique), that have been shown earlier in this chapter. On the other hand, the MUSA system was initially implemented based on Microsoft surface technology and Windows 7, with a touch pack platform. It used hyperbolic trees and supporting gesture-based interaction (Multitouch interaction) for representing and visualising variability models. This makes it a successful solution for creating and managing large-scale product lines.

However, due to some limitations with the surface platform, such as hardware issues inherited from surface technology, as well as software issues, such as platform dependency, MUSA was not adopted by many practitioners, leaving it as a prototype system. Although Microsoft has recently rolled out cheaper and portable versions of Surface, the earlier version of Surface was a bulky piece of hardware that came along with a table for it to be mounted upon. This made it very heavy and non-portable as a piece of hardware. In addition, the Surface was also too expensive when it initially hit the market.

Table 6.1: Comparisons between MUSA1 and MUSA 2

	MUSA 1	MUSA 2
Multi-Platform support	No	Yes
Innovative visualisation technique	Yes	Yes
Dependency management	No	Yes
Feature interaction	Yes	Yes

Multiple views	No	Yes
Modelling and management of variability	Yes	Yes
Identifying alternative features	No	Yes
Support for multitouch	Yes	Yes

6.5 Summary

This chapter introduces and describes the new version of MUSA that has been redesigned to better represent, visualise, and manage the variability of software product line models. This new version adopts the separation-of-concerns design principle and uses a mind-mapping approach (hyperbolic trees) to represent variability, as well as logic circuits to graphically represent the dependency and constraint relationships separately. This chapter also presents the different views showing the visualisation specifications and various functionalities of MUSA when populated with real data from case studies of different sizes.

PART III: VALIDATION

Chapter 7

Variability Management Evaluation Benchmark

7.1 Introduction

In this chapter, we present a benchmark for the evaluation of software quality attributes, as well as the quality attributes found to be important for software product line practice. These quality attributes are as follows: usability, performance, scalability, and integration. The purpose is to determine and gain a detailed understanding of where and how the quality of variability management tools could be improved. The study identified and selected 10 product line variability management tools which were based on their availability and support for feature models, and these were to be evaluated using the benchmark, in order to identify whether and to what extent these tools provided support for the identified quality attributes.

7.2 Methodology

In this section, the methodology used to collect data and underpin the entire study is presented.

In order to carry out this study, we applied a research methodology that combined both the features of qualitative and quantitative research methodologies. In the first step, a benchmark was developed, to be used consistently as a guideline in the evaluation process. As a crucial stage in the benchmarking design, we explored product line industries in order to know precisely what matters for the practitioners. We, therefore, used the outcome of an interview-based survey that involved a number of software product line practitioners, in which they were asked to list five quality attributes they deemed important for practical use of SPLs Variability Management (VM) tools. The identified quality attributes (usability, scalability, performance, and integration) were then used as key criteria to assess (i.e., how well the tools addressed them) the capability of SPLs-VM tools in the evaluation phase. Details of these quality attributes are given in section 7.4.

In the second step, the study focused on measuring the identified quality attributes, so as to ascertain their meanings and position. Hence, a further exploration into a number of internationally recognised standards and some respected reference models were carried out; these included ISO/IEC 9126 (ISO/IEC, 2001, ISO/IEC, 2003) (International Standard for Evaluation of Software Quality) and IEEE Standard 610.12 (IEEE Standard Glossary of Software Engineering Terminology). Among the other is the well-known Software Quality Metrics book (Fenton and Pfleeger, 1998), as well as An Effort-Based Framework for Evaluating Software Usability (Tamir et al., 2013). Having completed the survey and investigations on the identified quality attributes, and in the third step, the results of a study (presented in Chapter 2 and 4) were used. This study reported on a survey in which 37 existing product line-variability management tools were identified and analysed using a systematic literature review, from which 8 tools were selected, based on their availability and support for the graphical notations. However, 2 more publicly available tools were added using a separate search, making a total of 10 tools used in the evaluation process. The details of the identified tools and the criteria used when selecting a tool are given in section 7.5.

Finally, in the fourth step, an experimental evaluation was conducted (see Chapter 8), using 4 sample case studies of different sizes, and this was achieved by steadily applying the benchmark. The purpose was to assess how well the identified tools addressed the four quality attributes. This was followed by an opinion-based evaluation method that uses a questionnaire to obtain more insight into the user's opinion of the experience using the system. This was to know the extent to which the system is attractive.

7.3 Related Works

Many works have been reported by various authors within the SPL community in order to analyse, compare, or evaluate some of the existing variability management methods, tools, and techniques. However, to the best of our knowledge, no one has specifically evaluated these quality characteristics important for practical use of tools that support variability in SPLs.

For example, in (El Dammagh and De Troyer, 2011), a quality evaluation of nine feature modelling tools was conducted with the specific focus on quality criteria of usability, safety, and functional usability features. The main aim of the investigation was how to improve the quality in feature modelling tools, in general.

Study (Djebbi et al., 2007) evaluated four product line tools against certain criteria defined based on three perspectives; 1) criteria relating to product line engineering (2) criteria relating to tools capabilities and (3) criteria concerning project management. This is to determine their ability to satisfy industry expectations. In study (Simmonds et al., 2011), eight tools and techniques for variability modelling in software product line (SPL) or business process management (BPM) were evaluated based on various formalisms used in specifying software process variability.

The study analysed the tools in order to investigate their suitability for modelling variability in the software process. However, in order to assist engineers in selection of a suitable tool that best fits their needs, the authors in (Pereira et al., 2013) conducted an exploratory study that compares and analyses two feature modelling tools, based on data collected from 56 participants who experimentally used the tools. The study focused on evaluating the four common functionalities provided by feature modelling tools. These are: feature model editor, automated analysis of feature model, product configuration and tool notation.

7.4 Benchmark

This section presents the four quality attributes measured, sub-characteristics of each quality attribute and their detailed definitions. The section also gives in detail, how the measurement was carried out.

7.4.1 Quality Attributes

The four quality attributes this study measured are: usability, scalability, performance, and integration. These attributes were gathered from a study that used an interview based survey involving a number of software product line practitioners, in which they were asked to list five most important quality attributes for practical use of SPL tools. Figure 7.1 depicts the four quality attributes with their sub-characteristics.

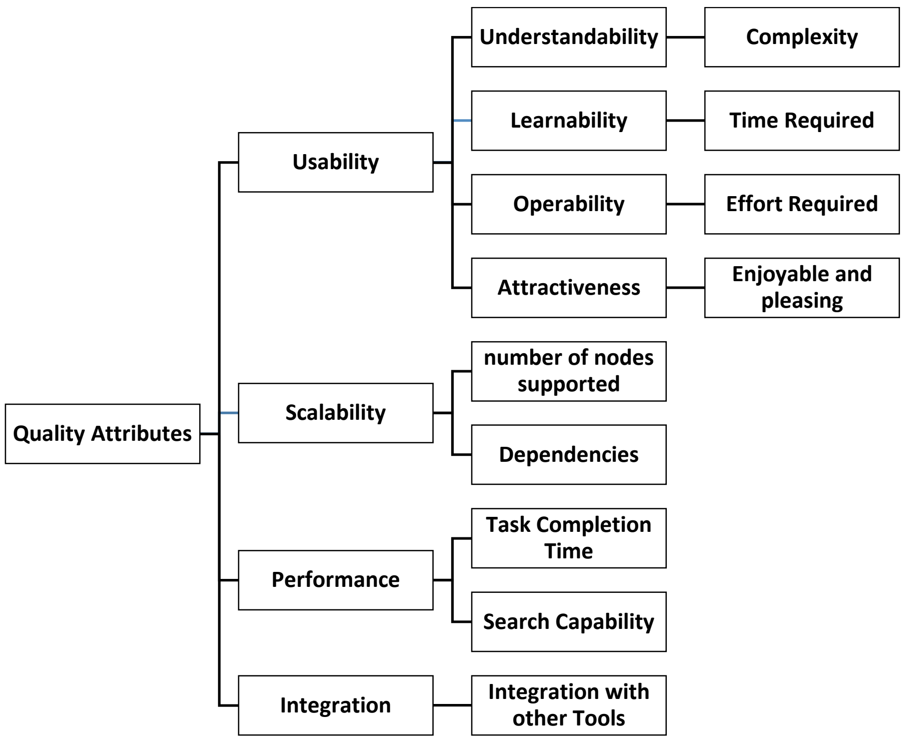


Figure 7.1: The quality attributes used

7.4.1.1 Usability Measure

Basics of sub-quality attributes under usability

i. Understandability: Complexity in using the software

ii. Learnability: Time required to fulfil a specified task

iii. Operability: Effort required to carry out a basic task

iv. Attractiveness: Is the software attractive to the target audience?

In order to determine and understand the main aspects that influence usability, this study based the measurement on the ISO 9126 (ISO/IEC, 2001, ISO/IEC, 2003) on software quality and measurement, which defined usability as ‘the capability of the software to be understood, learned, used and liked by the user, when used under specified conditions’. The standard identifies four to five key components of usability of a software product. Below are the detailed breakdown and the definitions of these sub-quality characteristics of usability:

i. Understandability

Can the software be understood easily? That is, the ability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and given the conditions of use. Understandability helps determine how easily the user can comprehend and use the software. We based the measurement of Understandability on study (Fenton and Pfleeger, 1998) where an ordinal scale was used as our measurement scale type (see Table 7.1) to measure the complexity of using the software. The ordinal scale provides a list of ordered alternatives from which respondents can select an option.

Table 7.1: Ordinal scale type

Value	Meaning
1	Trivial: commonly encountered (no exceptional effort needed)

2	Simple: Easy to manage and uncomplicated
3	Moderate: Being within average limit
4	Complex: Not easy to manage of being intricate
5	Incomprehensible: Impossible to manage of being not clear

ii. Learnability

Can the software be learnt easily? That is, the ability of the software product to enable the user to learn its application. Learnability is measured as the time that is required to fulfil a specified task. The specified task for this study is the need to add, delete, and edit a feature. This is in addition to the modelling of its dependency.

Learnability = Total Time required to **Add**, **Delete** or **Edit** a feature + **Dependency** Management

iii. Operability

Can the software be operated with minimal effort? That is, the capacity of the software product to allow the user to operate and control it. Operability was measured based on the efforts needed to accomplish the specified tasks (in this case) of adding, deleting, and editing a feature, together with the modelling dependency. Consequently, this effort equals the number of mouse clicks or screen touch (mc/st) + number of keyboard hits (kh). This measurement method is based on (Tamir et al., 2013).

Operability = **Efforts needed** to **Add**, **Delete** or **Edit** a feature + **Dependency** Management

Efforts = Number of mouse click or equivalent + Number of Keyboard strikes

iv. ***Attractiveness***

Is the interface of the software engaging? That is, the capability of the software product to be liked by the user. To measure attractiveness, this study based on (Fenton and Pfleeger, 1998) where a 5-point Likert scale is used to rank the software attractiveness, given a user a statement with which the user agrees or disagrees. The statement used for this study is:

The software is attractive (i.e. Enjoyable and pleasing).

1- Strongly Agree 2- Agree 3- Neither agree nor disagree 4-Disagree 5- Strongly Disagree

v. ***Compliance***

Does the software meet existing usability standards?

From the above definitions, usability can be measured by the degree to which a software product can satisfy the individual aspects of the definitions, i.e. to learn, understand, operate, and be attractive, while at the same time the software is compliant with and meets the existing usability standards. This is to be achieved under specified conditions in which a user or group of users carry out certain practical tasks.

7.4.1.2 Scalability Measure

Scalability, as it has been defined by (Berg et al., 2005), is the ability of the modelling approach to continue to meet its throughput objectives despite increasing or decreasing the amount of assets and elements that make up the models. A scalable variability modelling approach is the one that is useful when applied to a product line of any size (i.e. It should be capable of managing large or small size variability without any overhead or extra effort). Therefore, an approach will not be regarded as scalable if scaling only in one direction (i.e. downwards or upwards). However, a survey study on scalability aspects in (Chen and Babar, 2009) pointed out that, dependency relationships (such as variants to variants, variants to variation points or variation points to variation points) within variability models are the most discussed aspects in tackling scalability by modelling approaches. Hence, based on these studies, we used sample case studies of various sizes to serve as our basis for the experimental process of measuring scalability.

These cases were then classified into three different categories, which were then used to validate the selected tools with respect to this quality aspect. Section 8.2 of chapter 8 provides more details about the case studies.

The sample models are: (1) Small size, when a tool supports the development and management of 10-50 features before it starts to freeze or slow down. (2) Medium size, when the ability of variability management tool is to offer support for the development and management of 10-100 features when used, and (3) Large size, when it supports the development and management of variability models between 100-1000. At each level of testing of these various sample models, there was a practical investigation to see if the tools provide good support for dependency management and how it works. The scalability measure has been achieved experimentally, in order to gain a clear understanding of how and to what level the selected tools offer quality support for this attribute during the modelling process. Please note that it is not our purpose to measure the visualisation techniques deployed by these tools, but rather focus on the number of nodes they support.

7.4.1.3 Performance Measure

Performance evaluation according to (Ferrari, 1983) and(Kleinrock, 1976) includes externally observable system performance characteristics, such as response times and completion rates. However, IEEE standard 610.12 defined performance as the degree at which a system or a component completes designated tasks within given limits, such as speed, accuracy, or memory usage (IEEE, 1990). In this study, Performance is measured in relation to the scalability as the time it takes for each tool to validate the sample feature models assigned to it. That is, performance is measured as task completion time plus the search capability provided by the tools. Due to a large growth in size of the model, it becomes mandatory to investigate whether a tool can allow its user to search for a particular element of interest given several features.

7.4.1.4 Integration Measure:

The ability of a software tool to provide the means to either fully or partially integrate with other tools so that both tools can operate on the same set of data.

In this context, we will be using characteristics as follows:

Y = Yes, when a tool provides means to be fully integrated with other tools, and therefore operate on same set of data.

P = Partial, when a tool provides only half the features required for integration.

N = No, when a tool provides no means of integration.

7.5 Tools identification

This section provides a brief description of each tool used in the study. However, to make it easy for reading, the account is made in tabular form as shown in Table 7.2 where there are seven columns, in which the first column gives the name of each tool. The second column provides a brief description of each tool. While the third column presents the environment or platform based on which the tools were implemented, as well as whether the tool is run as a standalone application, plugin, or web services, together with the technology used to develop the tool. We also investigate the type of operation the tool supported i.e. the type of graphical notations used, as shown in the fourth column. The file format used by each tool is in column five. We further consider whether the tool is solely for commercial purpose, academic or both, shown in column six. Conversely, we inspect whether the tool is free and open source software or its evaluation copy could be obtained in column seven.

Table 7.2: Tools description

Tools	**Description**	**Environment/ Platform**	**Graphical Notation**	**File Format**	**Comme rcial/Ac**	**Open Source/**

			Types		**ademic**	**Evaluation copy**
FeatureIDE	FeatureIDE: an Eclipse plug-in tool that support all phases of Feature-Oriented software development for SPL development. The tool provides a configuration editor for creating and editing of configurations and provides support for valid product derivations. This is in addition to the detection and highlighting of dead features (Kästner et al., 2009).	Runs on Windows and Linux. Implemented as an eclipse plug-in developed using Java technology.	Graphical and text based Notations	Feature model file in a supported format (default: xml)	C/A	Free under L-GPL license v3
MUSA Case Tool	MUSA Case Tool: is a multi-touch variability modelling solution for software product line. It is a tool and framework that supports gesture based interaction for creating, visualizing, and maintaining large scale software product lines. MUSA was developed to address the scalability issue when (graphical overloading) visualizing large scale models (Bashroush, 2010).	Runs on Windows, Linux and Mac as stand-alone application. Developed using Java technology.	Graphical using Hyperbolic Tree notations	XML	A	Neither free nor evaluation copy
S2T2	S2T2 Configurator: A tool for interactive visual configuration of feature models with a formal reasoning engine that supports interactive functionality, such as calculating the consequences of user decisions based on the formal semantics of the feature modelling language (Pleuss and Botterweck, 2012).	Windows and Mac. Implemented in Java.	Graphical Notations	Conjunctive Normal Form (CNF)	A	Free
CVM Tool	CVM tool: A (Compositional Variability Management) for feature modelling and configuration, implemented as an experimental variability management tool for the evaluation of research approaches developed with close industry cooperation (Abele et al., 2010)	Runs on Windows and Mac. It is based on Java technology.	Graphical Notations based on Graphical Editing Framework (GEF)	XMI import and export.	C/A	Free

Familiar	Familiar: a fully integrated modelling environment that supports the development, manipulating and reasoning about feature models. Familiar provides different solutions including a standalone application, standalone console mode, and as a plugin for Eclipse platform (Acher et al., 2013).	Runs on Windows, Linux and Mac. Developed in Java language using XText.	Both Textual and Graphical Notations.	.treeml, other input/export are XML, fmprimitives, .tvl and .m	A	Free
CaptainFeature	CaptainFeature: a feature modelling tool with an integrated configurator for selecting features from the feature model (*CaptainFeature*, 2005).	Runs on Windows, Linux and Mac. The tool was implemented as a standalone application Developed in Java	Graphical through metamodelling notations.	XML	A	Free
Odyssey	Odyssey: A reuse Environment that contains various tools to construct a reuse infrastructure based on Product Lines, Domain Models and Component based Development. It provides support for domain engineers, domain specialists and software engineers who are responsible for the development of application within that domain (Braga et al., 1999).	Run on Windows and Linux. Used as a Standalone application. Developed using Java Technology.	Graphical using UML notation	XMI Import/Export	No	Free under GNU License.
XFEATURE	XFeature Tool: an Eclipse plug-in tool supports the modelling of SPL and the applications instantiated from them. The tool is used to build model of a set of configurable software assets by permitting the user to define their own feature meta-model (Pasetti and Rohlik, 2005).	Runs on Windows and Mac OS's. Implemented using Eclipse and XML Technology.	Graphical Notation	XML and XMI	A	Free Under GPL (General Public License).
PLUM	PLUM (Product Line Unified Modeller) a tool suit that follows a model Model-Driven Software Development approach. It is intended to provide support for the design, implementation and management of software product line. PLUM allows the product variability to be captured in what is so called a decision model, which implies analysing domain variability in terms of decisions and establishing dependencies among them (Aldazabal and Erofeev,	Eclipse Plug-in	Graphical using UML notation		C	Free

	2008).					
Pure::Variants	Pure::Variant: a tool support for software product line development and realization. It supports the creation and management of diverse variants of such product line. The tools are used to support various models from description of the problem domain of the PL, to the description of the implementation and for the selection of a specific product (Beuche, 2008) .	Runs on Windows, Mac or Linux. Standalone or also used as Web services.	Graphical Notations	XML-based exchange format.	C	Evaluation

7.6 Setting up of the Evaluation

The evaluation was achieved in two phases, both of which were carried out experimentally; in the first phase, an experiment was conducted using very small-scale models. This involves five PhD students, three of whom were from the domain of software engineering and the other two from the field of computing and technology possessing good modelling skills. During the experimentation, they were asked to create a very small feature diagram containing 8 feature nodes, two feature groups, and one feature constraint. Each one was provided with two tools.

A prior training session on how to use the tools was conducted to familiarize the users with the tools. The experimental evaluation in this stage was mainly to test the usability as a quality criterion, in order to gain a better understanding of how these tools could offer and support this quality attribute.

However, the use of a very small-scale model helps determine the readiness of these tools when used for larger case models.

While in the second phase of the experiment, unlike the first stage, the focus was not only on usability but also on scalability and performance. As stated in sections 7.4.1.2 and 7.4.1.3, scalability and performance were measured by dividing the experimental activities into three sub-divisions. Under each division, the following were examined: (1) the maximum number of features that a tool can accommodate. That is, in which of the three sub-divisions it falls: is it small, medium, or large. (2) What time it takes for a tool to accomplish the specific task assigned in such division and (3) what is the usability of a tool when accomplishing the task. For each of these sub-tasks, scalability is measured as the maximum number of features at which a tool starts to suffer a graphical overloading or slow down.

Furthermore, performance was measured as the time it takes for a tool to complete the specified task in the division. Finally, the usability of a tool while accomplishing the task was assessed as understandability, operability, and attractiveness. At this point, learnability was not taken into consideration because the performance of a tool can be more accurately measured when dealing with large scale models. In order to measure performance, learnability, and operability, a screen activity recorder software called Steps Recorder(*Steps Recorder*) was used to record the time taken, images, and the step by step activities of the experimental process.

7.7 Summary

This chapter describes a benchmark that was used persistently as the guideline to evaluate the MUSA tool, in comparison with other tools. The aim of this benchmark is to measure the four quality attributes (usability, scalability, performance, and integration), which were identified from an interview-based survey that involved a number of variability management (VM) practitioners. The chapter also presents and describes the 10 selected VM tools which are to be used in the experiment, as well as the criteria followed when choosing a tool. Finally, it explains how the evaluation was set to be carried out experimentally.

Chapter 8

Case Studies and Experimental Evaluation of the Tools

8.1 Introduction

This chapter applies the benchmark presented in the previous chapter, which served as our guideline. It first describes the four case studies of different sizes and data elements which were used in the experimentation; this includes how they were acquired (e.g. from industry) or formulated from the various sources. These sample cases were used to assess how well the identified feature modelling tools satisfied the four different quality attributes, as compared to our MUSA approach. The chapter presented and described the results of the evaluation. Finally, the lessons that were learned and a set of recommendations were described.

8.2 Case Studies

To illustrate how well and to what extent the selected tools satisfy the four quality attributes identified, we used multiple case studies out of which the largest scale case was acquired from Danfoss Power electronics. The case study is for Frequency Power Drives Product Line consisting of (1,300 variation points). The aim of this product line is to design and develop power drive to support any automation application and provide major energy savings and capability to control torque, acceleration, synchronization, position, and the overall performance (*IBM Rational DOORS*).

The remaining case studies were gathered from the results of careful examination of a large body of research work in the area of software product lines, from which feature models of various sizes were formed and used in the experimental process. The formulated models were therefore gathered from various sources and involved the formation of small, medium and large scale models. Among them is a case study (Thörn and Sandkuhl, 2009) for Library Services product line demonstrating the variability modelling of a wide range of services offered by a library to provide smooth and effective services to customers. This case study consists of 24 features.

In addition, a case of a house automation system product line that provides basic security, alarm, lighting, communication, and agenda services was also considered. This system is designed to serve as a middleware capable of interacting with other in-house physical devices such as heating equipment, lamps, and sensors in order to manage their functionality. This case study contained 20 features (Istoan et al., 2009). Another case study used was an email client adapted from (Akram and Abbas, 2009), used for sending and receiving e-mail. The product line model represents variations between different components of an email such as message editor, type of connection, operating system running, user practices and policies, communication protocols, and several other services. This case study had 18 features. However, in study (Mendonca, 2009), a simple feature model for a web search engine product line has been used. This case study represents various services offered by a search engine; these include page translation, doc-type, page preview, and ability for a user to search by language. This model had 14 features.

Finally, a mobile application product line that provides the ability to a user to search and buy products from an online catalogue in (Parra, 2011) was also used. This mobile application product line model among others contained its different expected services; among which are items catalogue, notification, authentication, history of items bought, shopping cat etc. The model had 23 features.

On the other hand, in order to build a small-scale variability model case study, we joined together three different cases and formed a model that consisted of more than 50 features based on which the experiment for the small case study was conducted. The three different models used are: email client, library services, and search engine-PL. However, we formulated a medium scale study by combining the five different cases, which gives us a single case study with 100 features serving as the medium scale model used in the experiment.

8.3 Results

This section presents and describes the results of the evaluation process of various quality attributes that have been assessed during experimentation, starting with the usability measurement under which its four sub-components were experimentally tested using the various sizes of sample models. That is, very small size, small size, medium and, large size, followed by scalability, Performance, and Integration respectively.

8.3.1 Usability

As stated above, usability was measured with respect to its four individual sub-components: understandability, learnability, operability, and attractiveness. For each of these components, two separate experiments took place: (1) usability was measured when a tool was applied to a very small-scale variability model (a model with only 8 features), and (2) usability when different sizes of variability models were applied (small, medium, and large scale sizes).

8.3.1.1 Understandability

By using an ordinal scale type, our dependent variables (comprehension and usage) were measured by asking the participants to specify their feeling about how easily they comprehend and use the tools or vice versa. The items in this scale are ordered with series of options or ranking order, with 1 being the most easy to manage and comprehend and 5 being the most difficult. Therefore, with respect to this quality, all the four sub-processes, Add, Delete, Edit, and Dependency, were carefully checked; therefore, when applied to very small-scale model, three tools (FeatureIDE, Familiar, and Pure::variants) score better in understandability with ranking 2 (simple), as depicted in column 1 of Table 8.1. Other four tools (MUSA, S2T2, CVM and CaptainFeature) scores 3, which is moderate. The poorest results obtained are for the Odyssey, XFeature, and PLUM with scores of 4 being complex. Consequently, none of the tools were given the maximum rating by each of the participants. Figure 8.1 summarizes the distribution of scores in a bar chart.

Table 8.1: Usability of very small-scale size model

Tools	**Usability**												
	Understandability= Comprehending and Usage				Learnability= Task Completion Time				Operability= Effort (mouse click or equivalent + Number of Keyboard strikes + dragging etc.)				Attractiveness
	Add	Delete	Edit	Dependency	Add	Delete	Edit	Dependency	Add	Delete	Edit	Dependency	
FeatureIDE	2				4:31				80				3
MUSA	3				6:55				103				2
S2T2	3				4:00				105				2
CVM Tool	3				8:31				184				2
Familiar	2				5:57				48				2
CaptainFeature	3				5:27				180				3
Odyssey	4				10:00				294				4
Pure::Variants	2				4:52				91				2
XFEATURE	4				12:38				217				4
PLUM	4				10:20				291				3

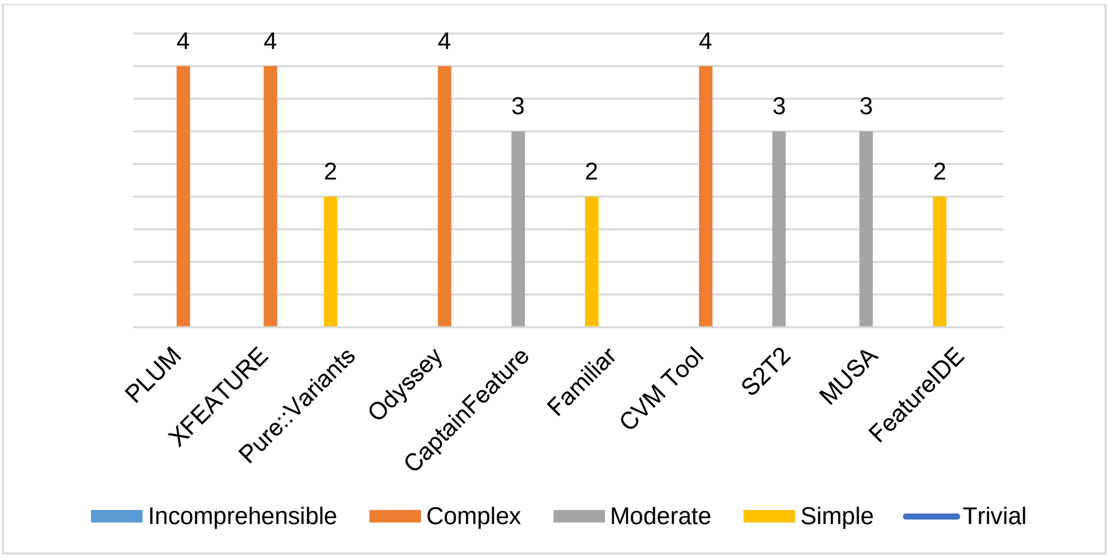


Figure 8.1: Understandability of very small-scale size model

Conversely, usability was measured when applied to variability models of different size (small, medium, and large), in that, only MUSA scores well in understandability (see Table 8.2 column 1) with ranking 2 while offering support for each of the four sub-processes. Following it were FeatureIDE, Familiar, CaptainFeature, Pure::Variants, and PLUM with ranking of 3 representing a moderate level of understanding; the worst results acquired were for S2T2, CVM, Odyssey, and XFeature with rank of 4, which is complex to understand. See Figure 8.2.

Table 8.2: Usability of different scale size

Tools	Understandability= Comprehending and Usage	Operability= Effort (mouse click or equivalent + Number of Keyboard strikes + dragging etc.)			Attractiveness
		10-50	**10-100**	**100-1000**	
FeatureIDE	3	214	433	No	3
MUSA	2	249	500	Yes	2
S2T2	4	341	572	No	3
CVM Tool	4	327	685	No	3
Familiar	3	376	697	No	2
CaptainFeature	3	601	1,311	No	3
Odyssey	4	338	No	No	4

Pure::Variants	3	253	439	No	3
XFEATURE	4	430	777	No	3
PLUM	3	220	No	No	4

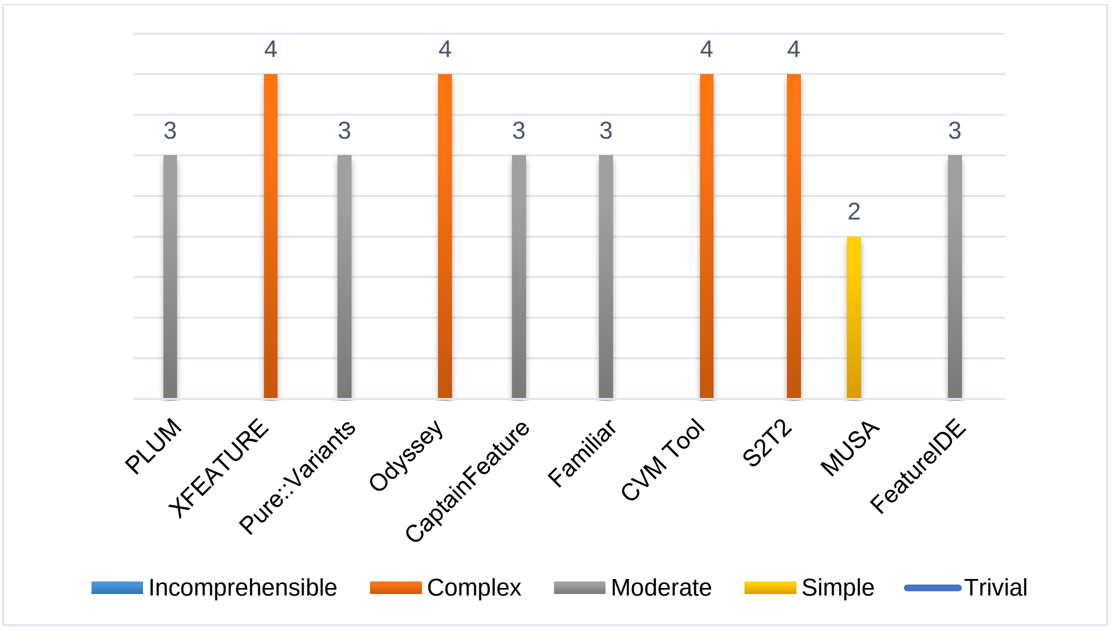


Figure 8.2: Understandability of different scale size

8.3.1.2 Learnability

Task completion time given in column 3 of Table 8.1 shows the results of the investigation on the time it takes for each tool to complete the task of adding, deleting, editing, and modelling dependency when applied to very small-scale model. In this, S2T2 was observed with least time to model variability of this size, followed by FeatureIDE, Pure::Variants, CaptainFeature, Familiar, MUSA, CVM, Odyssey and PLUM respectively, while XFeature consumed more time to accomplish the specified task. See Figure 8.3.

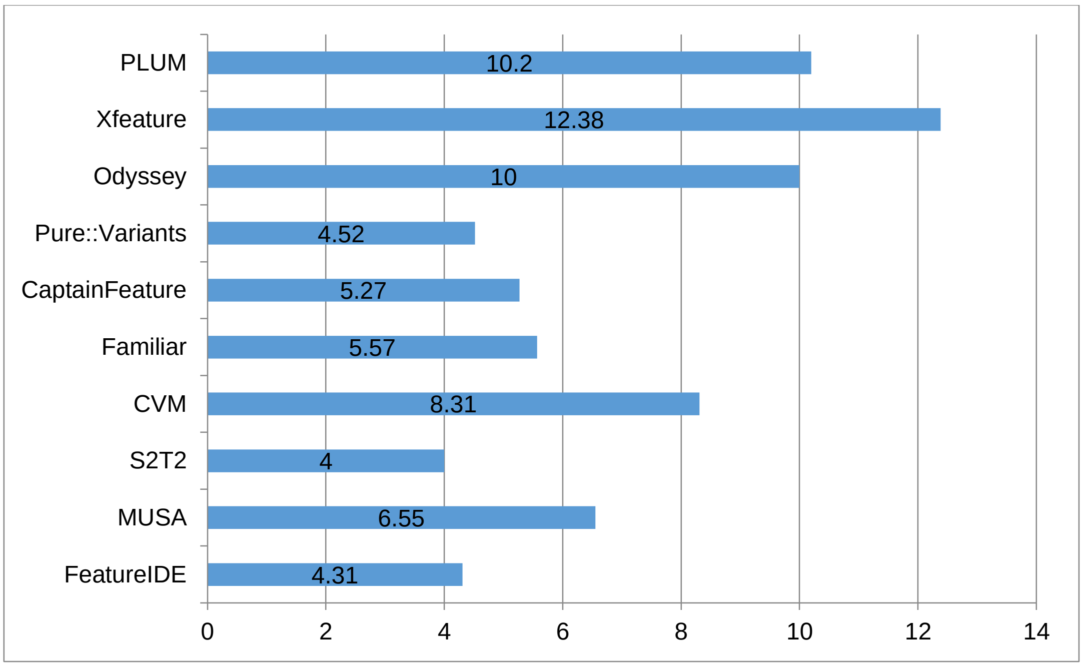


Figure 8.3: Learnability of very small-scale size model

8.3.1.3 Operability

The effort needed which includes mouse click, gesture based interaction, number of keyboard hits and dragging etc. shown in column 4 of Table 8.1. The operability of tools when applied to a very small sample size model: Familiar scores best with list number of efforts. Following it were FeatureIDE, Pure::Variants, MUSA, S2T2, CaptainFeature, CVM, XFeature, and PLUM respectively. Subsequently, Odyssey required more efforts to accomplish the task. See Figure 8.4.

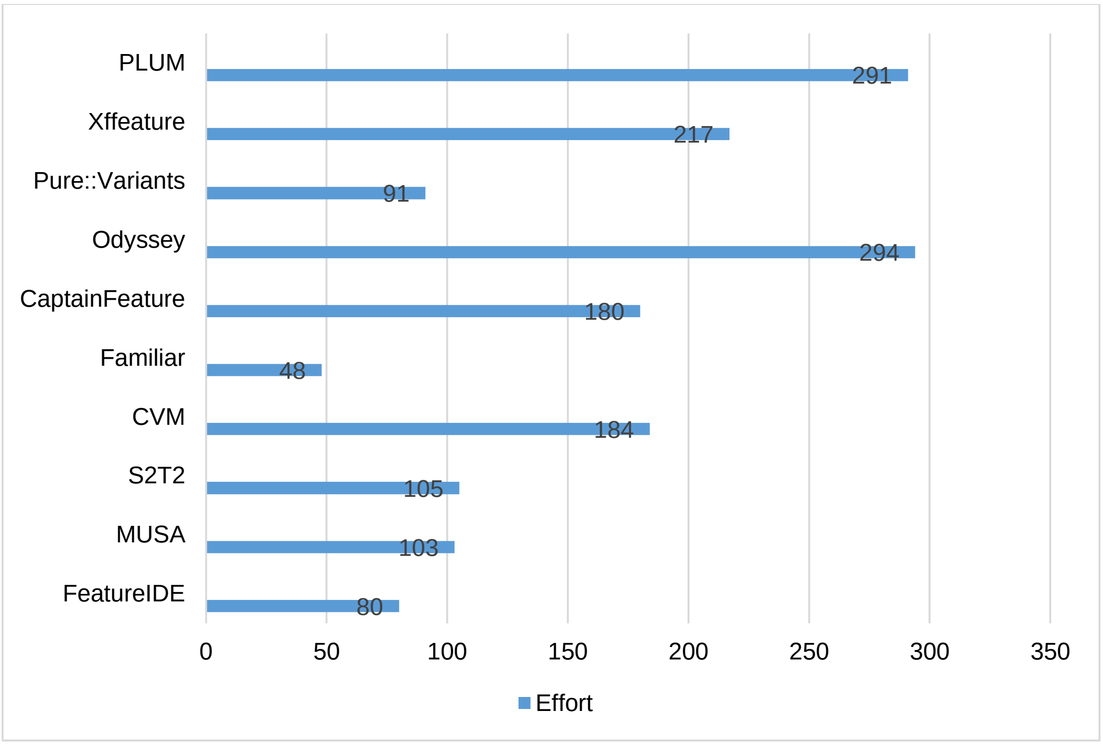


Figure 8.4: Operability of very small-scale size models

On the other hand, when applied to different scale variability models, (see Table 8.2 column 3, sub-column 1) the observations were: first, on applying to models that contain 10-50 features, FeatureIDE scores best, while PLUM, MUSA, Pure::Variants, CVM, Odyssey, S2T2, Familiar, and XFeature followed it respectively. With the CaptainFeature turned out with the worst results by requiring more efforts to carry out the task.

Secondly, when applied to sample models of size 10-100, FeatureIDE is still the best with less number of required operations, followed by Pure::Variants, MUSA, S2T2, CVM and Familiar respectively; the CaptainFeature required more operational efforts, as depicted in Table 8.2 column 3, sub-column 2. Unfortunately, tools like PLUM and Odyssey provide no mechanism to support a model with such number of features. Finally, when applied to variability models of size 100-1000, MUSA was the only tool capable of accommodating such large scale models (see Table 8.2 column 3, sub-column 3).

8.3.1.4 Attractiveness

The 5-point Likert scale (stated in 4) of Chapter 7) from which participants chose to rank the software attractiveness indicates that with very small size models, five tools were ranked with 2, i.e. ‘Agree’ by the participants. These include MUSA, S2T2, CVM, Familiar and Pure::Variants. While FeatureIDE, CaptainFeature and PLUM follow them with rank of 3, i.e. ‘Neither agree nor disagree’. The worst results were obtained for Odyssey and XFeature tools which were ranked with 4, i.e. ‘Disagree’ as can be seen in column 5 of Table 8.1. On the perspective of different size models, MUSA and Familiar tools score best with ‘Agree’, while FeatureIDE, S2T2, CVM, CaptainFeature, Pure::Variants and XFeature are following them with ‘Neither Agree nor-Disagree’. Odyssey and PLUM are the worst with ‘Disagree’ in this perspective. However, none of the tools scores ‘strongly Agree’ which is the highest score. See column 4 of Table 8.2.

8.3.2 Scalability

As stated in section 7.4.3 of Chapter 7, scalability was measured with respect to the number of feature nodes that a tool can accommodate, that is, in both upward and downward directions without any overhead. As indicated in Table 8.3 column 2, eight tools out of ten (FeatureIDE, MUSA, S2T2, CVM, Familiar, CaptainFeature, and Pure::Variants) were able to make it when applied to a small-scale variability model that contains 10-50 features. While unfortunately, three tools (Odyssey, XFeature, and PLUM) were only partially able to accommodate 50 features, with only Odyssey slightly accommodating more than 40 features; XFeature and PLUM were the only products to accommodate nearly 40 features. See Figure 8.5 for details.

Conversely, scalability when applied to a medium scale sample size containing 10-100 features showed that 7 of the ten tools (FeatureIDE, MUSA, S2T2, CVM, Familiar, CaptainFeature and Pure::Variants) successfully supported the sample of this size as well as managed the dependencies that existed among them. Three tools (Odyssey, XFeature and PLUM) turned out total failures in accommodating the sample size of this kind. See Table 8.3 column 3.

However, when applied to a large scale sample model that contained 100-1000 features, only MUSA tool was found to be capable of accommodating such features as depicted in Table 8.3 column 4. The overall summary results of scalability measure of the various sample sizes used in the experimentation are presented in Figure 8.5. The summary results of usability measures when applied to a very small-scale model are also presented in the figure.

Table 8.3: Scalability measure

Tools	Small size 10-50	Medium size 50-100	Large size 100-1000
FeatureIDE	Yes	Yes	No
MUSA	Yes	Yes	Yes
S2T2	Yes	Yes	No

CVM Tool	Yes	Yes	No
Familiar	Yes	Yes	No
CaptainFeature	Yes	Yes	No
Odyssey	Partially	No	No
Pure::Variants	Yes	Yes	No
XFEATURE	Partially	No	No
PLUM	Partially	No	No

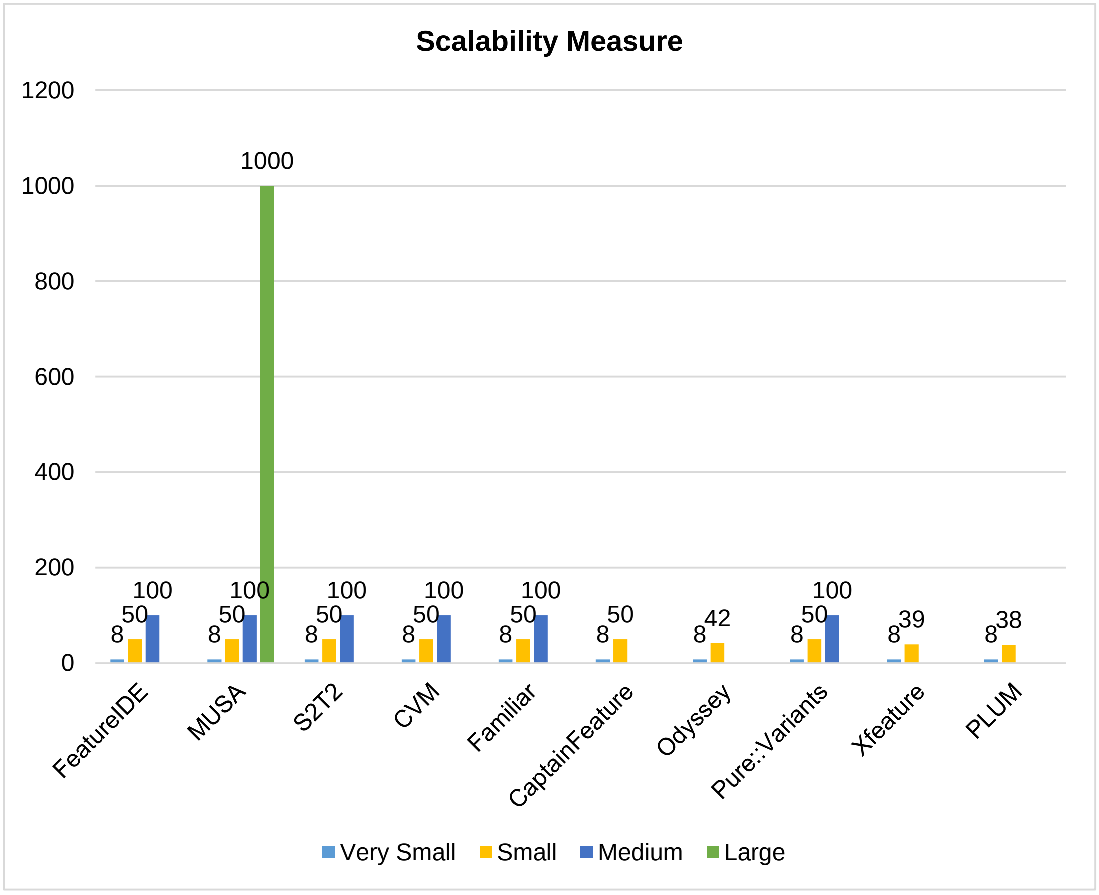


Figure 8.5: Scalability measure of various sample sizes

8.3.3 Performance

Performance was measured hand in hand with scalability as the time it took each tool to validate the assigned sample size plus the investigation if a tool provided a search mechanism for finding a particular feature of interest. Please refer to Table 8.4 for the summary of the overall results of performance measurement. In this, only MUSA and Pure::Variants provide search mechanism.

Table 8.4: Performance measure

Tools	Task Completion Time			Search Capability
	10-50	**50-100**	**100-1000**	
FeatureIDE	16:35	18:57	No	No
MUSA	18:33	16:38	Yes	Yes
S2T2	22:49	19:01	No	No
CVM Tool	16:09	17:22	No	No
Familiar	21:27	22:14	No	No
CaptainFeature	31:22	33:58	No	No
Odyssey	19:28	No	No	No
Pure::Variants	18:17	21:27	No	Yes
XFEATURE	24:41	No	No	No
PLUM	22:30	No	No	No

8.3.4 Integration

As stated in section 7.4.1.5 of Chapter 7, Integration was measured as the ability of a tool to provide the means to either fully or partially integrate with other tools so that both tools can operate on the same set of data. Tools integration has not been found as an issue for majority of the tools (FeatureIDE, S2T2, CVM, Familiar, XFeature, PLUM and Pure::Variants) are all found to be integrated either as Eclipse plugins or integrated with other in house developed tools. We found only three tools out of ten (MUSA, CaptainFeature and Odyssey) not integrated with any other one.

See Table 8.5 for the distribution of measurement.

Table 8.5: Integration measurement

Tools	Y = Yes	P = Partial	N = No	Integration with
FeatureIDE	Y			Eclipse plug-in
MUSA			N	
S2T2	Y			Eclipse plug-in

CVM	Y			Eclipse plugin
Familiar	Y			Plugin for Eclipse and Integrated with FeatureIDE
CaptainFeature			N	
Odyssey			N	
XFEATURE	Y			plug-in for the Eclipse platform
PLUM	Y			plug-in for the Eclipse platform
Pure::Variants	Y			Integrated into IDE's such as Eclipse or Rational Software Architect.

8.4 Discussion, Lessons Learned and Recommendations

This section presents the discussion of the results, the lesson learned, and the recommendations from the participant's comments.

The finding from this study as can be seen in section 8.3 has revealed that, with respect to the usability (see Table 8.1), majority of the tools were able to make it with either simple or moderate level in understandability when a very small-scale model is used. Likewise, many of them were able to fulfil their allocated task in less than 10 minutes. Similarly, only three of those tools required more than 200 operations to accomplish their given tasks. However, in regards to attractiveness, only 2 tools were rated with 'not- agree' on the basis of whether they are enjoyable and appealing when used.

On the other hand, this study also revealed that majority of these tools start to decline except (Bashroush, 2010) as soon as the number of features grow larger making them less usable. Likewise, they cannot scale well with the current situation where variability models are becoming very large and difficult to manage, especially in the real world industrial product line. Therefore, the study discovered that majority of these tools suffer scalability issues when the models start to reach around hundreds of attributes. In this regard, six tools (FeatureIDE, MUSA, CVM, Pure::Variants, Familiar and CaptainFeature) accommodated up to 100 features, which is the order of medium scale variability models. While three (Odyssey, XFeature and PLUM) tools failed to effectively support the small-scale model that contained 50 features.

Finally, with respect to the large scale model, which is the peak point for determining scalability in this study, only one tool MUSA was capable of accommodating more than a 1000 features. It is therefore the only tool that scaled in both directions; that is in upward and downward directions.

Following are the comments from the participants who used the tools during the experimental process using a very small-scale model:

On FeatureIDE: the user commented that the tool supports all the functionality expected from it with only slightly difficult to manage feature constraints. MUSA was found very effective specifically when creating and managing large variability features, but it is a bit difficult to model dependency using logic circuit. S2T2 Configurator was easy to use but managing dependency is a bit daunting as you need to click outside and then inside again before you can add a text for the dependency. CVM is workable, but managing constraints could only be achieved using a straight line arrow, which causes interruptions with other nodes in a large model. Familiar provides efficient ways of managing and handling growth of features by allowing zooming in and out of the entire feature tree. Unlike other tools, where the user has to scroll up or down to view the hidden features, Familiar enables moving around the entire feature tree from one place to another, making it easy to view the hidden features; however, managing constraints among features is a bit difficult to tackle. For CaptainFeature, users found its constraint dependency not simple and felt it would be difficult to use in large scale scenarios.

Odyssey seems not easy to understand as it is difficult to identify the alternative option representation. It also seems not supporting the ‘OR’ feature grouping. The tool is more towards UML based rather than feature modelling approach. Pure::Variants is an easy to use tool and has rich functionality such as different layouts (vertical and horizontal) zoom in and out, collapsing and expanding of elements or even to hide elements if one wishes to. It is easy to create and manage features. Editing is very easy in Pure::Variants and advance search mechanism is provided and constraints are managed in a separate view.

The tool, XFeature is complex to use; specifically, using the constraints between features is an issue and to identify the alternative feature grouping is difficult. PLUM seems easy to use, but to some extent creating and managing variability models is not straightforward. For example, connecting features is difficult to accomplish. Likewise, maintaining feature groups (i.e. OR and alternative) are not seen as easy to implement.

Based on these findings, it is of significant importance that the product line tool designers should realise and focus towards the increasing demand for making variability management tools more scalable to handle the complexity of the real world industrial product lines, as was also recommended in study (Chen and Babar, 2009). However, it is also recommended that the future tool deigns should be capable of managing both the small and large size variability models without any difficulty or extra labour. That is, there is need to make future tool designs more generic, instead of only focusing on ad-hoc designs that solve only a particular problem. This is of paramount importance from both scalability and usability aspects.

In addition, there is need to make future tool designs more flexible and straightforward when used, rather than requiring a user going into technical details in order to perform a simple operation. This has strong effects on usability, especially with respect to understandability and the tools' performance. Finally, providing support for integration with other tools could ease their adoption in practice by reducing the possibility of requiring changing the entire current practice. Please see the Appendix C for the screen shots of the tools' experimentation.

Table 8.6 summarizes the overall results of the evaluation. This indicates that the MUSA tool satisfied the four quality attributes, as compared to other tools, with the exception of one attribute, i.e. integration, which is part of our future work. Note that the alphabets U, O, and A under the usability column are for understandability, operability and attractiveness, respectively, while T and S under the performance column represent task and search.

Table 8.9: Summary results of the evaluation

Tools	Usability			Scalability			Performance		Integration
	U	**O**	**A**	**10-50**	**10-100**	**100-1000**	**T**	**S**	
FeatureIDE		X		X	X				X
MUSA	X	X	X	X	X	X	X	X	
S2T2				X	X				X
CVM Tool				X	X				X
Familiar				X	X				X
CaptainFeature				X	X				
Odyssey									
Pure::Variants		X		X	X			X	X
XFEATURE									X
PLUM									X

8.5 Summary

This chapter complements Chapter 7 by using the benchmark it presented, to conduct an experimental evaluation of MUSA tool as coppered to other 9 existing tools. It used four different case studies including: (1) for Frequency Power Drives Product Line consisting of (1,300 variation points). (2) A Library Services product line demonstrating the variability modelling of a wide range of services which were offered by a library with 24 features. (3) A house automation system product line, that provided basic security and contained 20 features, and (4) a mobile application product line that provided the ability to the user to search and buy products from an online catalogue with 23 features, together with the description of how they were gathered. The use of these case samples was to assess how the identified feature modelling tools satisfied the four different quality attributes. The chapter described the results of evaluation, the lessons learned, and the recommendations for future development.

PART IV:

CONCLUSION AND FUTURE WORK

Chapter 9

Conclusions, Contributions, and Future Work

9.1 Conclusion

The work proposed in this thesis investigated the reasons behind the lack of scalability with regards to the current variability management tools and techniques, something that will help us and other researchers accurately focus on the future efforts. The study not only provides a good overview of the existing variability management tools and the challenges for variability management tool support, but it also establishes the criteria and concepts for the comparison of such items. Based on the conclusions drawn from the survey, we proposed a new framework and a support tool for variability management that would address the scalability issue, which was the key challenge identified for variability management. The research also showed that managing dependency relationships separately from the actual representation of the variability models can significantly improve the scalability of a model's visualization, by reducing the complexity of viewing and managing them all from one view. Support for this was implemented by redesigning and creating a new version of the MUSA tool suite (a proof-of-concept variability management tool and framework was developed within our research group) that could address these challenges, and thus, lend itself to industrial large-scale applications. Such a development would ultimately allow for the creation and management of larger and more complex product lines. The new MUSA system was evaluated using a large-scale, multifaceted case study.

In part I of this document, we discussed about a systematic literature review survey which was conducted to have a better understanding of the main reasons behind the lack of scalability in the current variability management tools and techniques. We described the method used to identify, collect, and review the relevant papers in Chapter 2. We studied and presented an overview of 37 tools for managing the variability in software product lines, along with a number of commercial tools and the tools that have been adopted in the industry, with a focus on tool functionality, the technology used for development and implementation, and the type of notations supported in Chapter 3. We notably found that the key challenge of variability management tools is the scalability of models, that is, how to support the development of variability models that are still useful despite their size and complexity (Chapter 4).

We have also redesigned the MUSA (A Multitouch Variability Modelling Solution for Software Product Lines) framework and practical tool support, to allow the creation and management of larger and more complex product families and increase the productivity and the time-to-market of products. The variability model itself is implemented using a mind-mapping approach based on hyperbolic trees, providing an unprecedented potential for scalability. We described how effective the approach is when applied to the product lines of different sizes.

In part II, we described the earlier version of the MUSA tool in Chapter 5, as compared to the newer version in Chapter 6. A few of the key changes and improvements in the new MUSA over its predecessor are: (1) Applying the idea of separation-of-concerns design principle, and the use of logic circuit design, which is an additional view (Dependency view) for capturing and managing the dependency relationships that exist within the model that has been introduced. (2) Seamless support for cross-platform without any interruptions in the software. (3) Ability to recognise the presence of 'Alternative' feature set. (4) More innovative visualisation techniques that provide support for larger product line, etc.

In part III, we show how our approach met the design requirements, by evaluating our approach in comparison with other similar approaches. In Chapter 7, we described a benchmark that has been used steadily as a guideline during the evaluation process. The main focus while designing this benchmark was to know precisely what matters for the practitioners. For that reason, we used the outcome of an interview-based survey that involved a number of software product line practitioners, in which, they were asked to list five quality/attributes that they deemed important for the practical use of product lines variability management (VM) tools.

From this, four quality attributes were identified, which were usability, scalability, performance, and integration. In Chapter 8, we described the results of the experimental evaluation conducted, using multiple case studies, together with the lessons learned from the study, and closed the chapter with a set of recommendations.

In part IV, we concluded the thesis by summarising the contributions and discussing future work.

In summary, our evaluation results indicate that the MUSA tool and framework have significantly addressed the scalability challenges when dealing with large and complex product line models, as well as the shortcomings of usability and performance identified in the existing tools and techniques. This was achieved by ensuring that it was designed in line with industry requirements, thus maximizing the chances of it being adopted by industry, a major impact achievement for any researcher in this area. The findings show that MUSA can now be brought to industry for practical implementation.

9.2 Review of Contributions

This thesis contributes to the ongoing research on variability management in the field of software product line engineering by (1) identifying the key limiting factors affecting the scalability of the current variability management tools and techniques, through a close examination of the current literature in the field. (2) Identifying the barriers to industrial adoption of the current variability management tools. (3) Based on the findings of 1 and 2, we designed a tool and framework that addressed the identified shortcomings. (4) We finally implemented a working prototype of the system. We summarized the main contributions of this thesis below:

C1 *A systematic investigation and understanding of the state of the art tools that can be utilised in contemporary software product line development:* This study is a contribution to knowledge, as it conducts a systematic review of Variability Management tools according to the chronological order of development, and provides a conclusive evaluation of them. The results are intended to assist the practitioners in selecting the best available tools, which is based on their suitability towards a particular industrial task. The analysis also identifies the gaps in the field that should be addressed through further research of product line tools. Moreover, the analysis identifies the gaps in the research that should be addressed in more studies. Based on these results, we have collected the data and the necessary requirements for the development of our new MUSA tool.

C2 *Redesign of MUSA framework to improve the scalability of visualizing and representing variability models*: Although scalability was the main motivation for developing the early version of MUSA, redesigning and enhancing its capability to add more innovative visualisation techniques will increase its productivity, time-to-market, and allow for the creation and management of larger and more complex product families; hence, improving its scalability is required.

C3 *An additional view for capturing and managing dependency relationships that exist within the model separately:* Using the principle of separation-of-concerns, we have proposed a separate view called, "dependency view", to capture and manage dependency interaction independent from the actual representation of the models. This was achieved using a logic circuit. The main idea is to reduce the complexities, such as graphical overloading, when viewing and managing the dependencies of large variability points, all from one view.

C4 *A complete working prototype system of this has been implemented as a new MUSA:* Support for managing dependency relationships among the variability models has been implemented by redesigning and extending the current version of the MUSA tool suite (a proof-of-concept variability management tool and framework). This will allow for the creation and management of larger and more complex product families.

C5 *The new version of MUSA will be available as a multi-platform application:* To make it more generic and maximise its functionality, the new version of MUSA has been ported from Windows Presentation Foundation-WPF to Java technology. This has solved the main problem of platform dependency which is suffered by the existing version of MUSA.

C6 A benchmark for evaluating our approach: In order to evaluate the MUSA tool in comparison with other tools, we developed a benchmark for evaluating the quality attributes that are important for the practical use of SPL engineering tools, which has been applied in the evaluation process. The benchmark focused on measuring the four quality attributes are as follows: Usability, Performance, Scalability, and Integration. In addition, an evaluation study was conducted experimentally, and involved 10 feature-modelling tools. In order to know and get an insight on how well, and to what extent these tools satisfy these quality attributes, four case studies of different sizes were used as the basis for the experiment.

C7 Literature review process (Chapter 2): This process contributes to knowledge by providing empirical step-by-step guidelines to identify, collect, and review papers with: 1) a scope of the review clearly identified in advance; 2) a comprehensive search conducted to find all relevant studies; 3) the use of explicit criteria to include or exclude studies; 4) the establishment of standards to critically appraise study quality; and 5) the provision of explicit methods for extracting and synthesizing study findings. This process will benefit both new and experienced researchers by helping them avoid what is regarded as author's bias in research, while also providing a reliable basis for making decisions.

C8 Benchmarking process: The results of this will contribute to knowledge, as it will assist both practitioners and researchers alike by providing a standard and empirical approach to evaluating product line tools in the future. It also helps to identify and recommend areas that require attention in future tool design.

C9 The Context of Research: The distribution of the research context presented in Figure 4.8 of Chapter 4 indicates that there is a need to bridge the gaps between research in academia and industry through collaborative efforts. The figure shows that most studies (68%) have been conducted in an academic context, whereas only 16% of the studies are joint industrial academic endeavours. In 16% of the studies, no information was provided on the research context. Table 4.11 presents a list of all the studies with their research context. Please refer to Chapter 4 for details on this contribution.

9.3 Future Work

This thesis contributes fundamentally on how to overcome the scalability challenges when dealing with a large-scale product line model. However, looking at how important the usability aspect is in the software development perspective, we therefore, recognised it's significant throughout this research. In fact, it was one of the qualities we tested during the evaluation process. But, we still we need to consider improving the usability of the new dependency view in our future work. Likewise, integrating MUSA with other tools is also of paramount important. We summarised the key areas of focus in our feature work, which are as follows:

- **Usability of models when using the Logic Circuit**

  Dependencies are generally the 'rules' that have to be observed (or conditions that need to be satisfied). The use of logic circuit can significantly simplify the expressing of those rules, but, from the results of our evaluation in Chapter 8, there is a need to simplify the usage and make the view more interactive. However, there is also a need to consider more complex dependencies, such as "motor A requires that at least 3 motors B are selected", or "motor A requires motor B with a power above 200".

- **Integration with other tools or IDEs**

  To allow easy adoption of our MUSA tool, we need to consider integrating it with other tools, such as IBM Rational DOORs, or Integrated Development Environments (IDEs), such as Eclipse Modelling Framework (EMF), to allow either full or partial interoperability, or to allow third parties to integrate their automated reasoning techniques into a workspace where some basic services can be provided by default.

Bibliography

Abele, A., Papadopoulos, Y., Servat, D., Törngren, M. and Weber, M. (2010) 'The CVM Framework - A Prototype Tool for Compositional Variability Management'. *Proceedings Fourth International Workshop on Variability Modelling of Software-Intensive Systems VaMoS 2010*, Linz, Austria, 101-105.

Acher, M., Alférez, M., Galindo, J. A., Romenteau, P. and Baudry, B. (2014) 'ViViD: a variability-based tool for synthesizing video sequences'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM, 143-147.

Acher, M., Collet, P., Lahire, P. and France, R. B. (2011) 'Managing Feature Models with FAMILIAR: a Demonstration of the Language and its Tool Support'. *Proceedings of the 5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)* Belgium: ACM, 91-96.

Acher, M., Collet, P., Lahire, P. and France, R. B. (2013) 'FAMILIAR: A domain-specific language for large scale management of feature models', *Science of Computer Programming (SCP),* 78(6), pp. 657-681.

Akram, A. and Abbas, Q. (2009) *Comparison of Variability Modeling Techniques.* Tekniska Högskolan i Jönköping.

Aldazabal, A. and Erofeev, S. (2008) 'PLUM (Product Line Unified Modeller). Eclipse based Variability Management Tool'. *Proceedings of the Eclipse Summit Europe*, Ludwigsburg, Germany.

Antkiewicz, M. and Czarnecki, K. (2004) 'FeaturePlugin: Feature Modeling Plug-In for Eclipse'. *proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (ETX 2004)*, Vancouver, British Columbia, Canada: ACM Press, 67-72.

Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Peach, B., Wust, J. and Zettel, J. (2002) *Component-based product line engineering with UML.* London: Addison-Wesley.

Bachmann, F. and Clements, P. (2005) *Variability in Software Product Lines*, Pittsburgh, USA,: Software Engineering Institute (Technical Report CMU/SEI-2005-TR-012.

Bashroush, R. (2010) 'A NUI Based Multiple Perspective Variability Modeling CASE Tool'. *Proceedings of the 4th European Conference on Software Architecture (ECSA '10)*, Copenhagen, Denmark: LNCS, 523-526.

Bashroush, R., Al-Nemrat, A., Bachrouch, R. and Jahankhani, H. (2011) 'Visualizing Variability Models Using Hyperbolic Tree'. *Proceedings of the 23rd International Conference on Advanced Information Systems Engineering Forum (CAiSE)*, London, 113-120.

Bashroush, R., Spence, I., Kilpatrick, P., Brown, J. and Gillan, C. (2008) 'A Multiple Views Model for Variability Management in Software Product Lines'. *Proceedings of the Second International Workshop on Variability Modelling of Software-intensive Systems (VaMoS2008)*, Duisburg-Essen, Germany.

Bassett, P. (1997) *Framing Software Reuse: Lessons from the Real World.* Prentice-Hall.

Batory, D. (2004) 'Feature-Oriented Programming and the AHEAD Tool Suite'. *Proceedings of the 26th International Conference on Software Engineering (ICSE’04)* IEEE Computer Society, 702-703.

Batory, D., Sarvela, J. N. and Rauschmayer, A. (2004) 'Scaling step-wise refinement', *The IEEE Transactions on Software Engineering,* 30(6), pp. 355-371.

Bécan, G., Nasr, S. B., Acher, M. and Baudry, B. (2014) 'WebFML: synthesizing feature models everywhere'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM, 12-116.

'Bellsouth, IBM', (1993) *Mobile Phone News: Bellsouth and IBM Unveil Personal Communicator Phone*.

Berg, K., Bishop, J. and Muthig, D. (2005) 'Tracing software product line variability: from problem to solution space'. *Proceedings of the 2005 annual*

research conference of the South African institute of computer scientists and information technologists on IT research in developing countries, South Africa, 182 - 191.
Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K. and Wąsowski, A. (2013) 'A survey of variability modeling in industrial practice'. *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems*, Pisa, Italy: ACM, 7.

Berger, T., She, S., Lotufo, R., Wasowski, A. and Czarnecki, K. (2010) 'Variability Modeling in the Real: A Perspective from the Operating Systems Domain'. *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium: IEEE/ACM, 73-82.

Beuche, D. (2008) 'Modeling and Building Software Product Lines with pure::variants'. *Proceedings of the 12th International Software Product Lines Conference (SPLC 2008)*, Limerick, Ireland, 8-12 Sept. 2008: IEEE Computer Society, 358.

Beuche, D., Birk, A., Dreier, H., Fleischmann, A., Galle, H., Heller, G., Janzen, D., John, I., Kolagari., R. T., Maßen, T. v. d. and Wolfram, A. (2007) 'Using Requirements Management Tools in Software Product Line Engineering: The State of the Practice'. *Software Product Line Conference, 2007. SPLC 2007. 11th International*: IEEE, 84-96.

Beuche, D. and Spinczyk, O. (2003) 'Variant management for embedded software product lines with pure::consul and AspectC++'. *Proceedings of Companion of the 18th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA 2003)*, Anaheim, CA, USA: ACM, 108-109.

BigLever, S. I. *Product Line Engineering Solutions for Systems and Software*. Available at: http://www.biglever.com/solution/solution.html (Accessed: August 22 2012).

Botterweck, G., Janota, M. and Schneeweiss, D. 'A Design of a Configurable Feature Model Configurator'. *Proceedings of the 3rd International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 09)*, Seville, Spain: Universität Duisburg-Essen, 165-168.

Botterweck, G., Thiel, S., Nestor, D., Abid, S. b. and Cawley, C. (2008) 'Visual Tool Support for Configuring and Understanding Software Product Lines'. *12th International Software Product Line Conference*: IEEE, 77-86.
Braga, R. M. M., Werner, C. M. L. and Mattoso, M. (1999) 'Odyssey: a reuse environment based on domain models'. *Proceedings of the IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET'99)*, Richardson, TX, USA: IEEE, 50-57.

Brereton, P., Kitchenham, B. A., Budgen, D., Turner, M. and Khalil, M. (2007) 'Lessons from applying the systematic literature review process within the software engineering domain', *The Journal of Systems and Software,* 80, pp. 571–583.

Campbell, G. *MTP TOOL- The MetaProgramming Text Processor*: Prosperity Heights Software. Available at: http://www.domain-specific.com/index.html.

Capilla, R., Sánchez, A. and Dueñas, J. C. (2007) 'An Analysis of Variability Modeling and Management Tools for Product Line Development'. *Proceedings of the Software and Services Variability Management Workshop Concepts, Model and Tools*, Helsinki, Finland: Helsinki University of Technology Software Business and Business Institute, 32-47.

Capilla, R., Sierra, A. and Serrano, J. M. *VMWT: Variability Modeling Web Tool*. University Juan Carlos of Madrid. Available at: http://triana.escet.urjc.es/VMWT.

CaptainFeature (2005). Available at: http://captainfeature.sourceforge.net/ 2014).

Cechticky, V., Pasetti, A., Rohlik, O. and Schaufelberger, W. (2004) '*XML-Based Feature Modelling*'. *Proceedings of the International Conference on Software Reuse (ICSR 2004)*, Madrid, Spain: Springer-Verlag, 101-114.

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

Chen, L., Ali Babar, M. and Ali, N. (2009) 'Variability Management in Software Product Lines: A Systematic Review'. *Proceedings of the 13th International Software Product Line Conference (SPLC)*, San Francisco, CA, USA, 81-90.

Chen, L. and Babar, M. (2010) 'Variability Management in Software Product Lines: An Investigation of Contemporary Industrial Challenges', in Bosch, J. & Lee, J. (eds.) *Software Product Lines: Going Beyond Lecture Notes in Computer Science*: Springer Berlin Heidelberg, pp. 166-180.

Chen, L. and Babar, M. A. (2009) 'A Survey of Scalability Aspects of Variability Modeling Approaches'. *Proceedings of the 13th International Software Product Lines Conference*, San Francisco, USA.

Clements, P. and Northrop, L. (2002) *Software Product Lines: Practices and Patterns. SEI Series in Software Engineering* Massachusetts: Addison-Wesley.

Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K. and Wąsowski, A. 'Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches'. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, 01/25/2012: ACM, 173-182.

Daizhong, L. and Shanhui, D. (2009) 'Feature Dependency Modeling for Software Product Line'. *Computer Engineering and Technology, ICCET '09. International Conference on*, 256-260.

Damithc, l., Manasgupta, Jarzabek S. (2008) *XML-based Variant Configuration Language*. Available at: http://sourceforge.net/projects/fxvcl/files/.

Dhungana, D., Grünbacher, P. and Rabiser, R. (2010) 'The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study', *Automated Software Engineering,* 18(1), pp. 77–114.

Dhungana, D., Grünbacher, P. and Rabiser, R. (2011) 'The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study', *Automated Software Engineering,* 18(1), pp. 77–114.

Dietz, P. H. and Eidelson, B. D. (2009) 'SurfaceWare: Dynamic Tagging for Microsoft Surface'. *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction (TEI'09)*, Cambridge, UK, Feb 16-18 2009: ACM, 249-254.

Djebbi, O. and Salinesi, C. (2006) 'Criteria for Comparing Requirements Variability Modeling Notations for Product Lines'. *Fourth International Workshop on Comparative Evaluation in Requirements Engineering CERE '06.*: IEEE, 20-35.
Djebbi, O., Salinesi, C. and Fanmuy, G. (2007) 'Industry Survey of Product Lines Management Tools: Requirements, Qualities and Open Issues'. *Proceedings of the 15th IEEE International Requirements Engineering Conference*, New Delhi, India: IEEE, 301-306.

DOORS-TREK. Available at: http://www-01.ibm.com/support/docview.wss?uid=swg24032035 (Accessed: July 16 2014).

Dos Santos, R. F. and Frakes, W. B. (2009) 'DAREonline:A Web-Based Domain Engineering Tool'. *Proceedings of the 11th International Conference on Software Reuse: Formal Foundations of Reuse and Domain Engineering (ICSR 2009)*, Virginia Tech's Northern Virginia Center in Falls Church, VA, USA: Lecture Notes in Computer Science, 246-257.

Eclipse *Graphical Modeling Framework (GMF, Eclipse Modeling subproject)*. Available at: http://www.eclipse.org/gmf/ (Accessed: September 16 2015).

Eichelberger, H., El-Sharkawy, S., Kröher, C. and Schmid, K. (2014) 'EASy-producer: product line development for variant-rich ecosystems'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*: ACM, 133-137.

El Dammagh, M. and De Troyer, O. (2011) 'Feature Modeling Tools: Evaluation and Lessons Learned'. *Proceedings of the 30th International Conference on Advances in Conceptual Modeling: Recent Developments and New Directions (ER'11), Variability Workshop on Software Variability Management (Variability@ER11)*: Springer-Verlag Berlin Heidelberg, 120-129.

Eysholdt, M. and Behrens, H. (2010) 'Xtext: Implement Your Language Faster than the Quick and Dirty Way'. *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion*: ACM, 307-309.

Felfernig, A., Friedrich, G., Jannach, D. and Zanker, M. (2001) 'Intelligent Support for Interactive Configuration of Mass Customized Products.' *14th Int'l Conf. on Industrial & engineering applications of artificial intelligence and expert systems*: Springer, 746–756.

Fenton, N. E. and Pfleeger, S. L. (1998) *Software Metrics: A Rigorous and Practical Approach.* 2 edn. Boston, USA: PWS Publishing Company.

Ferber, S., Haag, J. and Savolainen, J. (2002) 'Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line'. *Proceedings of the Second International Conference on Software Product Lines, SPLC 2*, San Diego, CA, USA, August 19–22, 2002: Springer-Verlag, 37-60.

Fernández, R., Laguna, M. A., Requejo, J. and Serrano, N. (2009) *Development of a Feature Modeling Tool using Microsoft DSL Tools* [Technical Report], University of Valladolid. Available at: http://www.giro.infor.uva.es/fmt.pdf.

Ferrari, D. (1983) *Measurement and Tuning of Computer Systems.* New York: Prentice Hall.

Frakes, W., Prieto-Diaz, R. and Fox, C. (1997) 'DARE-COTS: A domain analysis support tool'. *Proceedings of the 17th International Conference of the Chilean Computer Science Society (SCCC '97)*, Santiago, Chile, 10-15 Nov 1997: IEEE Computer Society, 73-77.

Frakes, W., Prieto-Diaz, R. and Fox, C. (1998) 'DARE: Domain Analysis and Reuse Environment', *Annals of Software Engineering,* 5, pp. 125-141.

Gauthier, C., Classen, A., Storey, M.-A. and Mendonca, M. (2010) 'XToF: A Tool for Tag-based Product Line Implementation'. *Proceedings of the 4th International Workshop on Variability Modeling of Software Intensive Systems (VaMoS 2010)*, Linz, Austria: University of Duisburg-Essen, 163-166.

Gomaa, H. (2005) *Designing software product lines with UML: from use cases to pattern-based software architectures. The Addison-Wesley object technology series* Boston: Addison-Wesley.

Gomaa, H. and Shin, M. E. (2004) 'A Multiple-View Meta-modeling Approach for Variability Management in Software Product Lines'. *Proceedings of the 8th International Conference on Software Reuse: Methods, Techniques and Tools (ICSR)*, Madrid, Spain: LNCS, 274-285.

Gomaa, H. and Shin, M. E. (2004) 'Tool Support for Software Variability Management and Product Derivation in Software Product Lines'. *Proceedings Workshop on Software Variability Management for Product Derivation, Software Product Line Conference*, Boston, USA.

Gomaa, H. and Shin, M. E. (2007) 'Automated Software Product Line Engineering and Product Derivation'. *Proceedings of the 40th Hawaii International Conference on System Sciences*, Waikoloa, Big Island, HI, USA: IEEE Computer Society, 1530-1605.

Gómez, M., Mansanet, I., Fons, J. and Pelechano, V. (2012) 'Moskitt4SPL: Tool Support for Developing Self-Adaptive Systems'. *Proceedings of the 17th Conference on Software Engineering and Databases (JISBD'12), Jornadas SISTEDES*, Almeria, Spain.

Griss, M. L., Favaro, J. and d'Alessandro, M. (1998) 'Integrating Feature Modeling with the RSEB'. *Proceedings of the 5th International Conference on Software Reuse*, Vancouver, BC, Canada, 76-85.

Halmans, G. and Pohl, K. (2003) 'Communicating the variability of a software-product family to customers', *Software and Systems Modeling,* 2(1), pp. 15-36.

Heer, J., Card, S. K. and Landay, J. A. 'Prefuse: a toolkit for interactive information visualization'. *Proceedings of the SIGCHI conference on Human factors in computing systems*: ACM, 421-430.

Heidenreich, F. (2009) 'Towards Systematic Ensuring Well-Formedness of Software Product Lines'. *Proceedings of the 1st International Workshop on Feature-Oriented Software Development (FOSD'09)*, Denver, Colorado, USA: ACM, 69-74.

Heidenreich, F., avga, I. S. and Wende, C. (2008) 'On Controlled Visualisations in Software Product Line Engineering'. *2nd International Workshop on Visualisation in Software Product Line Engineering (ViSPLE*

2008) collocated with the 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland, September 8 - 12, 2008, 388.

Heidenreich, F., Jan, K. and Christian, W. (2008) 'FeatureMapper: Mapping Features to Models'. *Proceedings of the 30th International Conference on Software Engineering (ICSE 2008)*, Leipzig, Germany: ACM Digital Library, 943-944.

Heidenreich, F., Johannes, J., Karol, S., Seifert, M. and Wende, C. 'Derivation and refinement of textual syntax for models'. *Model Driven Architecture-Foundations and Applications*: Springer Berlin Heidelberg, 114-129.

Hervieu, A., Baudry, B. and Gotlieb, A. (2011) 'PACOGEN: Automatic Generation of Pairwise Test Configurations from Feature Models'. *Proceedings of the 22nd Annual International Symposium on Software Reliability Engineering (ISSRE 2011)*, Hiroshima, japan, Nov. 29 2011-Dec. 2 2011, 120-129.

Heuer, A., Lauenroth, K., Müller, M. and Scheele, J.-N. (2010) 'Towards Effective Visual Modeling of Complex Software Product Lines'. *Proceedings of the 14th International Conference on Software Product Line, {SPLC} 2010*, Jeju Island, South Korea, 229-238.

Holten, D., Isenberg, P., van Wijk, J. J. and Fekete, J.-D. (2011) 'An extended evaluation of the readability of tapered, animated, and textured directed-edge representations in node-link graphs'. *IEEE Pacific Visualization Symposium (PacificVis 2011)*: IEEE, 195-202.

IBM Rational DOORS. Available at: www.ibm.com (Accessed: July 23 2014).

IEEE (1990)*: IEEE Standard Glossary of Software Engineering Terminology*.

ISO (2001)*: ISO/IEC 9126-1: Software Engineering-Product Quality-Part 1: Quality Model*. Geneva Switzerland: International Standards Organization.

ISO (2003)*: ISO/IEC 9126-1: Software Engineering-Product Quality, Part-2, External Metrics*. Geneva, Switzerland: International Organization for Standardization.

Istoan, P., Nain, G., Perrouin, G. and Jézéquel, J.-M. (2009) 'Dynamic Software Product Lines for Service-Based Systems'. *Ninth IEEE International Conference on Computer and Information Technology, CIT '09*, 11-14 Oct. 2009: IEEE, 193-198.

Jacobson, I., Griss, M. and Jonsson, P. (1997) *Software Reuse: Architecture, Process and Organization for Business Success.* New York: Addison-Wesley-Longman.

Jaring, M. and Bosch, J. (2002) 'Representing Variability in Software Product Lines: A Case Study'. *Software Product Lines SPLC 2*, Springer Berlin, 219-245.

Jarzabek, S., Bassett, P., Zhang, H. and Zhang, W. (2003) 'XVCL: XML-based variant configuration language'. *Proceedings of the 25th International Conference on Software Engineering (ICSE '03)*, Portland, Oregon, USA, 3-10 May 2003: IEEE Computer Society, 810-811.

Johansen, M. F., Haugen, Ø. and Fleurey, F. (2012) 'An Algorithm for Generating t-wise Covering Arrays from Large Feature Models'. *Proceedings of the 16th International Software Product Line Conference (SPLC 2012)*, Salvador, Brazil: ACM.

Kadir, W. M. N. W. and Mohammad, R. (2008) *Advances in Software Engineering Research and Practice.*

Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E. and Peterson, A. S. (1990) *Feature Oriented Domain Analysis (FODA) feasibility study*, Software Engineering Institute, Carnegie Mellon University CMU/SEI-90-TR-21.

Kang, K. C., Kim, S., Lee, J., Kim, K., Kim, G. J., Shin, E. and Huh, M. (1998) 'FORM: A feature-oriented reuse method with domain-specific reference architectures', *Annals of Software Engineering,* 5(1), pp. 143-168.

Kang, K. C., Lee, J. and Donohoe, P. (2002) 'Feature-Oriented Product Line Engineering', *IEEE Software,* 19, pp. 58-65.

Kästner, C., Thüm, T., Saake, G., Feigenspan, J., Leich, T., Wielgorz, F. and Apel, S. (2009) 'FeatureIDE: Tool Framework for Feature-Oriented Software

Development.'. *Proceedings of the 31th International Conference on Software Engineering (ICSE)*, Vancouver, Canada: IEEE Computer Society, 611–614.

Kim, K., Kim, H., Ahn, M., Seo, M., Chang, Y. and Kang, K. C. (2006) 'ASADAL: A Tool System for Co-Development of Software and Test Environment based on Product Line Engineering'. *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, Shanghai, China: ACM, 783-786.

Kitchenham, B. and Charters, C. (2007) *Guidelines for Performing Systematic Literature Reviews in Software Engineering*, Keele University, UK EBSE-2007-1.

Klaus, P., Böckle, G. and van der Linden, F. (2005) *VARMOD Tool [Internet]*: Software Systems Engineering. Available at: http://www.sse.uni-essen.de/swpl/SEGOS-VM-Tool/index.html 18/05/2012).

Kleinrock, L. (1976) *Queueing Systems, Volume I: Theory, and Volume 2: Computer Applications.* New York: Wiley.

Krueger, C. and Clements, P. (2014) 'Systems and software product line engineering with gears from BigLever software'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM.

Krueger, C. W. (2006) 'Introduction to the Emerging Practice Software Product Line Development', *Methods & Tools,* (no. 3).

Krueger, C. W. (2007) 'The 3-Tiered Methodology: Pragmatic Insights from New Generation Software Product Lines'. *Proceeding of the 11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan: IEEE, 97-106.

Lamping, J., Rao, R. and Pirolli, P. (1995) 'A focus+context technique based on hyperbolic geometry for visualizing large hierarchies'. *In Proceedings of the Conference on Human Factors in Computing Systems*: ACM, 401-408.

Lettner, D., Petruzelka, M., Rabiser, R., Angerer, F., Prähofer, H. and Grünbacher, P. (2013) 'Custom-Developed vs. Model-based Configuration Tools: Experiences from an Industrial Automation Ecosystem'.

MAPLE/SCALE 2013, Workshop at the 17th International Software Product Line Conference (SPLC 2013), Tokyo, Japan: ACM, 52-58.

Levent V, O. (1998) 'Storage and retrieval of database constraints', *Information Systems,* 23(6), pp. 401-421.

Linden, A. and Fenn, J. (2003) *Understanding Gartner's hype cycles*: Strategic Analysis Report No R-20-1971. Gartner, Inc.

Lisboa, L. B. (2008) *ToolDAy - A Tool for Domain Analysis.* Masters, Federal University of Pernambuco, Recife, Brazil.

Lisboa, L. B., Garcia, V. C., de Almeida, E. S. and de Lemos Meira, S. R. (2011) 'ToolDAy: a tool for domain analysis', *International Journal on Software Tools for Technology Transfer (STTT) archive,* 13(4), pp. 337-353.

Martinez, J., Lopez, C., Ulacia, E. and del Hierro, M. (2009) 'Towards a Model-Driven Product Line for Web systems'. *Proceedings of the 5th Model-Driven Web Engineering Workshop MDWE in conjuction with the International Conference in Web Engineering (ICWE 2009)*, San Sebastian, Spain, 1-15.

Mehta, N. (1982) *A Flexible Machine Interface.* Masters Thesis, University of Toronto.

Meinicke, J., Thüm, T., Schröter, R., Krieter, S., Benduhn, F., Saake, G. and Leich, T. (2016) 'FeatureIDE: Taming the Preprocessor Wilderness'. *Proceeding of the International Conference on Software Engineering*, Austin, TX: ACM and IEEE CS.

Mendonca, M. (2009) *Efficient Reasoning Techniques for Large Scale Feature Models.* Doctor of Philosophy PhD, University of Waterloo.

Mendonca, M., Branco, M. and Cowan, D. (2009) 'S.P.L.O.T.: software product lines online tools'. *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*, Orlando, FL, USA, 10/25/2009: ACM, 761-762.

Microsoft (2008) *NUI*. Available at: http://research.microsoft.com/en-us/collaboration/focus/nui/ (Accessed: December 10 2015).

Modeling, H. F. Available at: http://caosd.lcc.uma.es/spl/hydra/index.htm (Accessed: July 11 2012).

Moon, M., Yeom, K. and Chae, H. (2005) ' An Approach to Developing Domain Requirements as a Core Asset Based on Commonality and Variability Analysis in Product Line', *IEEE Transactions on Software Engineering* 31(7), pp. 551-569.

Myllärniemi, V., Asikainen, T., Männistö, T. and Soininen, T. (2005) 'Kumbang configurator—A configuration tool for software product families'. *Proceedings of the IJCAI-05 Workshop on Configuration In conjunction with the 19th International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland.

Myllärniemi, V., Raatikainen, M. and Männistö, T. (2007) 'Kumbang tools'. *Proceedings of the 11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan: IEEE Computer Society, 135--136.

Nestor, D., O'Malley, L., Quigley, A., Sikora, E. and Thiel, S. (2007) 'Visualisation of Variability in Software Product Line Engineering'. *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*: ACM.

Nuseibeh, B., Kramer, J. and Finkelstein, A. (1994) 'A framework for expressing the relationships between multiple views in requirements specification', *IEEE Transactions on Software Engineering,* 20(10), pp. 760–773.

Park, K., Ryu, D. and Baik, J. (2012) 'An Integrated Software Management Tool for Adopting Software Product Lines'. *Proceedings of the 11th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2012)*, Shanghai, China, May 30 2012-June 1 2012, 553-558.

Parra, C. (2011) *Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations* Doctor of Philosophy, Lille 1 University - Science and Technology.

Pasetti, A. and Rohlik, O. (2005) *Technical Note on a CONCEPT FOR THE XFEATURE TOOL*, P&P Software GmbH / ETH Zurich (PP-TN-XFT-0001. Available at: www.pnp-software.com.

Pereira, J. A., Souza, C., Figueiredo, E., Abilio, R., Vale, G., Heitor and Costa, A. X. 'Software Variability Management- An Exploratory Study with Two Feature Modeling Tools'. *VII Brazilian Symposium on Software Components, Architectures and Reuse*: IEEE.

Pleuss, A. and Botterweck, G. (2012) 'Visualization of variability and configuration options', *International Journal on Software Tools for Technology Transfer (STTT),* 14(5), pp. 497-510.

Pohl, K., Böckle, G. and van der Linden, F. (2005) *Software Product Line Engineering: Foundations, Principles and Techniques.* Germany: Springer-Verlag Heidelberg.

Rabiser, R., Dhungana, D., Heider, W. and Grünbacher, P. (2009) 'Flexibility and End-User Support in Model-Based Product Line Tools'. *Proceedings of the 35th Euromicro Conference on Software Engineering and Advanced Applications (SEAA '09)*, Patras, Greece, 27-29 Aug. 2009: IEEE Computer Society, 508-511.

Reiser, M.-O. and Weber, M. 'Managing Highly Complex Product Families with Multi-Level Feature Trees'. *Proceeding of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, Minneapolis/St. Paul, MN: IEEE Computer Society, 149-158.

Russell, G., Burns, F. and Yakovlev, A. (2012) 'VARMA—VARiability modelling and analysis tool'. *Proceedings of the IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS 2012)*, Tallinn, Estonia, 18-20 April 2012, 378-383.

Samih, H. and Bogusch, R. (2014) 'MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing]'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM, 138-142.

SAP Configurator. Available at: www.sap.com (Accessed: July 16 2014).

SCALE'09 'Workshop on Scalable Modeling Techniques for Software Product Lines (SCALE'09)'. San Francisco, USA.

Schmid, K. and Schank, M. (2000) 'PuLSE-BEAT — A Decision Support Tool for Scoping Product Lines'. *Proceedings of the International Workshop Software Architectures for Product Families IW-SAPF-3*, Las Palmas de Gran Canaria, Spain: Lecture Notes in Computer Science, 65-75.

Seaman, C. B. (1999) 'Qualitative Methods in Empirical Studies of Software Engineering', *IEEE Transactions on Software Engineering,* 25(4), pp. 557-572.

Segura, S., Galindo, J. A., Benavides, D., Parejo, J. A. and Ruiz-Cortés, A. (2012) 'BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models'. *Proceedings of the 6th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*, Leipzig, Germany: ACM, 63–71.

Shakari, P. and Møller-Pedersen, B. (2006) 'On the Implementation of a Tool for Feature Modeling with a Base Model Twist'. *Proceedings of the Norwegian Informatics Conference (NIK 2006)*, University of Oslo, Norway, Nov., 81-93.

She, S. *Linux Variability Analysis Tools (LVAT)*. Available at: http://code.google.com/p/linux-variability-analysis-tools/ (Accessed: July 16 2012).

Shneiderman, B. and Plaisant, C. (2004) *Treemap 4.1.1*: University of Maryland. Available at: http://www.cs.umd.edu/hcil/treemap-history/ (Accessed: August 12 2015).

Simmonds, J., Bastarrica, M. C., Silvestre, L. and Quispe, A. (2011) *Analysing Methodologies and Tools for Specifying Variability in Software Processes*: Universidad de Chile, Santiago, Chile.

Sincero, J. and Schroder-Preikschat, W. (2008) 'The linux kernel configurator as a feature modeling tool'. *Proceedings of the 12th International Conference on Software Product Lines SPLC (2)*, Limerick, Ireland: Lero Int. Science Centre, University of Limerick, Ireland, 257-260.

Sinnema, M. and Deelstra, S. (2007) 'Classifying Variability Modeling Techniques', *Information and Software Technology,* 49(7), pp. 717–739.

Sinnema, M., Deelstra, S., Nijhuis, J. and Bosch, J. (2004) 'COVAMOF: A Framework for Modeling Variability in Software Product Families'. *Proceedings of the 3rd International Conference on Software Product Lines - SPLC*, Boston, MA, USA: Springer-Verlag Berlin Heidelberg, 197-213.

Sinnema, M., Deelstra, S., Nijhuis, J. and Bosch, J. (2006) 'Modeling dependencies in product families with COVAMOF'. *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, (ECBS 2006)*, 307.

SparxSystems Enterprise Architect. Available at: www.sparxsystems.com (Accessed: July 16 2014).

Stengel, M., Frisch, M., Apel, S., Feigenspan, J., Kästner, C. and Dachselt, R. (2011) 'View infinity: a zoomable interface for feature-oriented software development'. *Proceedings of the 33rd International Conference on Software Engineering (ICSE 2011)*, Waikiki, Honolulu, Hawaii, USA, 21-28 May 2011, 1031-1033.

Steps Recorder. Available at: http://pcsupport.about.com/od/termsp/p/problem-steps-recorder.htm.

Succi, G., Eberlein, A., Yip, J., Luc, K., Nguy, M. and Tan, Y. (1999) 'The design of Holmes: a tool for domain analysis and engineering'. *Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, Canada, 365-368.

Succi, G., Pedrycz, W., Yip, J. and Kaytazov, I. (2001) 'Intelligent design of product lines in Holmes'. *Proceedings of the Canadian Conference on Electrical and Computer Engineering*, 75-80.

Succi, G., Yip, J. and Liu, E. (2000) 'Analysis of the Essential Requirements for a Domain Analysis Tool'. *Proceedings of the ICSE Workshop on Software Product Lines: Economics, Architectures and Implications*, Limerick, Ireland: ACM.

Succi, G., Yip, J., Liu, E. and Pedrycz, W. (2000) 'Holmes: a system to support software product lines'. *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, Limerick Ireland, 786.

Tamir, D., Mueller, C., J and Komogortsev, O., V (2013) 'An Effort-Based Framework for Evaluating Software Usability Design', *ARPN Journal of Systems and Software,* 3(4), pp. 65-77.

Ter Beek, M. H. and Mazzanti, F. (2014) 'VMC: Recent Advances and Challenges Ahead'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM, 70-77.

Thiel, S. and Hein, A. (2002) 'Systematic Integration of Variability into Product Line Architecture Design'. *Proceedings of the 2nd International Conference on Software Product Lines*: Springer Berlin Heidelberg, 130-153.

Thörn, C. and Sandkuhl, K. (2009) *Feature Modeling: Managing Variability in Complex Systems. Complex Systems in Knowledge-based Environments: Theory, Models and Applications*: Springer Berlin Heidelberg, p. 129-162.

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G. and Leich, T. (2014) 'FeatureIDE: An Extensible Framework for Feature-Oriented Software Development', *Science of Computer Programming,* 79, pp. 70-85.

Tolvanen, J.-P. and Kelly, S. (2009) 'MetaEdit+: defining and using integrated domain-specific modeling languages'. *In Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications*: ACM, 819-820.

Trinidad, P., Benavides, D., Ruiz-Cort´es, A. and Segura, S. (2008) 'FAMA Framework'. *Proceedings of the 12th International Software Product Line Conference (SPLC '08)*, Limerick, Ireland, 8-12 Sept. 2008, 359-359.

Van der Linden, F. J., Schmid, K. and Rommes, E. (2007) *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering.* Berlin: Springer-Verlag.

Van Gurp, J., Bosch, J. and Svahnberg, M. (2001) 'On the Notion of Variability in Software Product Lines'. *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA)*, Amsterdam, Netherlands, 45-55.

Van Ommering, R., Van Der Linden, F., Kramer, J. and Magee, J. (2000) 'The Koala Component Model for Consumer Electronics Software', *Computer,* 33(3), pp. 78-85.

von der Maßen, T. and Lichter, H. (2004) 'RequiLine: A Requirements Engineering Tool for Software Product Lines'. *Proceedings of the 5th International Workshop on Product Family Engineering (PFE)*, Siena, Italy: Springer LNCS 168-180.

White, J., Schmidt, D. C., Wuchner, E. and Nechypurenko, A. (2007) 'Automating Product-Line Variant Selection for Mobile Devices'. *Proceedings of the 11th International Software Product Line Conference (SPLC 2007)*, Kyoto, Japan, 10-14 Sept. 2007: IEEE Computer Society, 129-140.

Yamany, A. E. E., Shaheen, M. and Sayyad, A. S. (2014) 'OPTI-SELECT: an interactive tool for user-in-the-loop feature selection in software product lines'. *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*, Florence, Italy: ACM, 126-129.

Zhang, B. and Becker, M. (2014) 'Variability code analysis using the VITAL tool'. *Proceedings of the 6th International Workshop on Feature-Oriented Software Development*, Florence, Italy: ACM, 17-22.

Zhang, H. and Jarzabek, S. (2001) 'XML-based method and tool for handling variant requirements in domain models'. *Proceedings of the 5th IEEE International Symposium on Requirements Engineering, 2001*, Toronto, Ont.: IEEE, 166 - 173.

Zhang, H. and Jarzabek, S. (2004) 'XVCL: a mechanism for handling variants in software product lines', *Science of Computer Programming,* 53, pp. 381–407.

APPENDES

Appendix A

DARE-COTS: DARE (Domain Analysis and Reuse Environment) is a tool implemented using commercial-off-the-shelf (COTS) and freeware that provides automated support for domain engineering activities, which includes identifying the common and variable features within a family of systems. DARE assists domain experts in different ways, such as carrying out analysis by providing useful high level guidance, recording and extracting domain information from documents and code using a stored section of the domain book (Frakes et al., 1997, Frakes et al., 1998). The tool was implemented as a standalone and was also used in-house as a web service developed at the College of Engineering, Department of Computer Science, VaginiaTech. DARE supports textual notation using MS Word and using a form-base component. The tool has been developed using various technologies as prototypes of different versions. These include the first version, which was developed in 1994 in C-language on a UNIX workstation. In 1995, the second prototype version was based on Visual Basic 3 on a PC running Windows. Another prototype version was created using commercial off-the-shelf tools (COTS) and freeware to provide automated support for domain analysis (Dos Santos and Frakes, 2009). DARE is not available as free and open source, nor could an evaluation copy be obtained.

Odyssey: A reuse environment that has been conceived as a framework consisting of various tools to construct a reuse infrastructure based on Product Lines, Domain Models and Component-Based Development. It provides support for conceptual models (e.g., use cases, feature models and other object oriented models using UML notation), architectural models (e.g., architectural and design pattern system in combining object-oriented models), and implementation models (e.g., reuse components set). Odyssey assists three different categories of users – domain engineers, domain specialists and software engineers – who are responsible for the development of applications within that domain (Braga et al., 1999). The Odyssey environment was developed as a stand-alone application using java technology at the Computer Science Department, Federal University of Rio de Janeiro. It provides support for development using graphical UML notations. With respect to Odyssey, no information has been provided to enable categorisation of the tool as either academic or commercial. The Odyssey reuse environment is a free and open source tool that is available under GNU license.

PuLSE-BEAT (Product Line Software Engineering Basic Eco Assistance Tool): An automated decision support tool for the analysis of data used by PuLSE-Eco to determine the scope of a product line during product line development activities. The tool was developed at the Fraunhofer Institute for Experimental Software Engineering (IESE) and provides support for both domain expert and method experts by delivering three different scopes: possible candidates, likely candidates, and strongly recommended candidates. These are in relation to thresholds of 0.5, 0.6 and 0.75, respectively. PuLSE-BEAT was developed based on Excel worksheets using Visual Basic for Applications and supports textual notations implemented based on a tabular form. Support for traceability and consistency management between these worksheets has been provided by the tool. PuLSE belongs to the category of academic tools. It is however, not available as free and open source, nor could an evaluation copy be obtained (Schmid and Schank, 2000).

Holmes: A collection of tools that supported product line (SPL) domain analysis and engineering, designed to endorse Sherlock (a DA&E methodology aimed at the development of object-oriented frameworks) SPL methodology by giving an additional functional interface and making better use of existing technologies and standards. Holmes aimed at providing varied support for SPL development, including market analysis and strategy, modelling, design, and development of the resulting product. This is achieved through the use of a critiquing system that provides semantic support for Holmes users by analysing the products and domain models (Succi et al., 1999, Succi et al., 2000b, Succi et al., 2000a, Succi et al., 2001). The tool was developed using Java language and implemented as a web service using Java Space technology and XML at the Department of Electrical and Computer Engineering, University of Alberta, Canada. The tool supports textual notations based on a matrix with rows and columns for the projections of the number of variants for variation points. It is recognised as an academic tool, rather than a commercial one. It is also not available as free and open source, and no evaluation copy could be obtained.

COVAMOF (ConIPF Variability Modelling Framework): A software variability modelling tool that represents variation points and dependencies as first-class citizens and provides ways to model the relationships among the complex and simple dependencies at all levels of abstraction. The tool supports software engineers during both the domain engineering and application engineering processes (Sinnema et al., 2004). In COVAMOF, the product family is divided into three abstraction layers – the features, the architecture and the component implementations. The variability occurs in all the abstraction layers, and is modelled by the COVAMOF Variability View (CVV). The CVV encompasses the variability in terms of variation points and dependencies. COVAMOF and all its functionality has been developed in Java and implemented as a stand-alone application. It has also been partly implemented as an extension to the Eclipse platform (Sinnema et al., 2006). It supports graphical notation for representing the variation points, variants and dependency. The tool was developed at the Department of Mathematics and Computing Science, University of Groningen, Netherlands, as part of the research sponsored by Configuration in Industrial Product Families (ConIPF), which was aimed at defining and validating methodologies for product derivation that are applicable in industrial applications. Neither a free version nor an evaluation copy of COVAMOF could be obtained.

Feature Modelling Plug-In (fmp): an Eclipse based tool supporting feature modelling editing and configuration purposes. It models variability based on feature diagrams, and uses cardinality-based feature modelling, specialisation and configuration. The tool was implemented as either stand-alone in Eclipse, or together with an fm2rsm plug-in in Rational Software Modeller (RSM), or Rational Software Architecture. Fmp2rsm integrates fmp with RSM and enables product line modelling in UML (Antkiewicz and Czarnecki, 2004). The fmp tool supports software product line developers in checking the consistency of the model, generating valid configurations, and checking partial configurations. The tool was developed at the University of Waterloo, Canada, based on the Java language and supports both graphical and textual notations using UML and XML, respectively. Although the project has been completed and the tool is no longer maintained, fmp is an open-source project hosted on SourceForge.

PLUSEE (Product Line UML Based Software Engineering Environment): an automated product line engineering tool in which a multiple-view model of the product line architecture and components are developed and stored in a product line repository. The multiple-view model describes the different features of a product family, including the common and variable characteristics of member products. The multiple-view model is represented using UML modelling notations. PLUSEE addresses the product line life cycle in three different phases: 1) the Product Line Requirements Modelling, consisting of the use case model view, which tackles the functional requirements of a product line in terms of use cases and actors; 2) The Product Line Analysis Modelling, comprising four different views – the Static Model View, the Collaborative Model View, the State Chart View and the Feature Model View. In the Static Model View, the fixed structural aspects of a software product line are addressed using classes and relationships among them, whereas the Collaborative Model View and the State Chart View work collaboratively. The dynamic aspects of a software product line are handled in the Collaborative Model View, while the State Chart View determines the state and state transitions for each of the dependent kernel, optional, and variant classes. The Feature Model View captures and represents the variability of a product line together with their dependency relationships. Finally, 3) the Product Line Design Modelling, which is the phase where the architecture of a product line is developed (Gomaa and Shin, 2004b, Gomaa and Shin, 2007, Gomaa and Shin, 2004a). PLUSEE was developed at the Department of Information and Software Engineering, George Mason University, using Rational Rose and Rational Rose RT graphical editors to support the multiple views. The tool supports graphical notation using UML and has been categorised as an academic tool. PLUSEE is not available for free or as an evaluation copy.

XML-Based Feature Model: An XML-based feature modelling tool that provides support for defining feature models and the enforcement of the relationships' instantiation between models and their meta-models. It offers a means to decompose a large feature diagram into extensible and self-contained modules. The tool, however, provides explicit support for both the modelling of the system family and of the applications instantiated from it. In addition, it outlines and provides support based on an XML approach, which is a way to the develop and express feature modelling tool at a low-cost (Cechticky et al., 2004). The tool was developed at the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland, in collaboration with P & P Software, which is a research spin-out of the Institute. It has been developed based on XML technology and therefore allows for the automatic derivation of

XML schema from the family model. It also supports textual notations and has been identified as an academic tool. The tool is not available for free, and no evaluation copy could be obtained.

AHEAD (Algebraic Hierarchical Equations for Application Design): a collection of Java-based tools that support product lines by way of compositional programming, in which features are the building blocks of the system. AHEAD is a toolset that supports multiple programs and multiple non-code representations written in different languages. The main tool of AHEAD is called the composer, which receives an equation as an input and then interprets the equation into its nested collective equivalents. After this, a composite feature directory will be created with the name of the input equation. Other tools comprised by AHEAD are for the implementation of the composition for non-code artefacts, such as HTML files, make files, design rule files, XML files and BNF-grammar files, among others (Batory et al., 2004, Batory, 2004). AHEAD tools were developed at the Department of Computer Sciences, University of Texas at Austin. The tools were not based on pure Java, but were developed using Java extended embedded domain specific languages (DSLs) for refinements, state machines, and metaprogramming. Therefore, it supports different Java dialects. The tools support textual notation based on expressions. AHEAD tools are available as free and open source for download.

XVCL (XML-based Variant Configuration Language): a variability management mechanism that comprises a method and a supported tool that can be applied to configure variants of various kinds of software product line assets, including architecture, code generation, UML modelling, test cases and documentation in the SPL approach. It supports product line developers to easily perform domain analysis in order to capture the common and variable requirements in a feature diagram. It then enables the building of reusable product line assets, comprising the domain model, product line architecture and generic components (Zhang and Jarzabek, 2004, Jarzabek et al., 2003). XVCL was developed at the National University of Singapore, to serve as a modern and versatile version of its predecessor, Bassett's frames (Bassett, 1997), a technology that has been successfully applied in industry for synthesising large COBOL and Java business applications. The tool was implemented using Java and XML technology, where frame-programming concepts were expressed as a mark-up language similar to XML, separating each frame called x-frame from an XML file. It supports textual notation designed as XML tags. XVCL is an open-source software available at SourceForge (damithc, 2008).

KUMBANG: a toolset for managing software product line variability that comprises two different, but mutually dependent, aspects – Kumbang Modeller and Kumbang Configurator. The Modeller allows for the creation and modification of models to capture product line variability from feature and architecture points of view. With Modeller, a user can specify the features of the system family and the architectural elements, as well as constraints and dependency relationships. On the other hand, the Configurator allows for deriving the configuration of individual product members through binding variability in the Modeller (Myllärniemi et al., 2005, Myllärniemi et al., 2007).

The Configurator assists users (such as architects) during the configuration process by providing a graphical interface to allow for entering requirements for individual products, in addition to checking to ensure that configuration is consistent and complete. Kumbang was developed at the Software Business and Engineering Institute, Helsinki University of Technology. The tool enables creating and editing configuration models graphically using UML-like notations. Kumbang is an open-source tool that can be downloaded online.

BVR (Base Variability Resolution): a prototype feature modelling tool that is characterised by establishing and maintaining the relationships between models: 1) Base model, which can be in a given language; 2) Variation model for variability specification, which usually contains variation elements and each element will be referenced to a base model, and is subject to variation, whereas those elements that are not subject to variation will be figured out; 3) Resolution model, for defining variability, as well as binding its specification that can be used to drive one or more new products in a product family. The BVR tool has been developed based on Java Development Technology (JDT) and Java programs for representing the Base model and the resolution of variation, respectively. In addition, the implementation was based on the Eclipse modelling framework (EMF). It supports graphical notation using basic right-click functionality. BVR is an open-source software available as an Eclipse plugin (Shakari and Møller-Pedersen, 2006).

ASADAL (A system Analysis and Design Aid tooL): a based-on FORM (Feature-Oriented Reuse Method) method that supports the entire product line development lifecycle, including domain analysis, architecture and component design, software code generation and verification and validation. ASADAL assists its users by providing various editors, including a feature analysis editor for feature selection and a feature-binding analysis modelling editor. Other design modelling editors include: i) a conceptual architecture model for stating abstract high-level functional elements of a product line, and control flow between them, ii) a process architecture model for demonstrating executable elements and relationships among them, iii) a component architecture model for expressing reusable concrete components to be used in system development, and iv) a design object model for component implementation. It generates executable code, such as Java, based on the information gathered from combined feature selection, processing macros embedded in various design models and components (Kim et al., 2006). The ASADAL tool was developed at the Computer Science and Engineering Department, Pohang University of Science and Technology, South Korea. It supports graphical notation using FODA (Feature-Oriented Domain Analysis) format. ASADAL has not been available in either a free open source or evaluation copy.

Scatter: a tool that supports architecture design for mobile devices by automating variant selection for mobile devices whose inputs are: i) the requirements of product line architecture and ii) the resources available on discovered mobile devices. The expected production is an optimal variant that can be used in the construction of mobile devices. Scatter helps developers visually model the product line architecture components, manage the dependencies and constraints among them, and handle the non-functional requirements associated with each component (White et al., 2007). Scatter was developed at Vanderbilt University Nashville, TN, USA, in collaboration with Siemen AG, Corporate Technology (SE 2) Munich, Germany. It was, however, implemented based on the Eclipse Generative Modelling Technology (GMT) project, using its open-source Generic Eclipse Modelling System (GEMS). The tool supports graphical notation that defines domain specific modelling language using a visio-like environment. There exists a compiler that converts the graphical models produced by the Scatter into a prolog knowledge base and constraints satisfaction problem (CSP) and, which is operated using a prolog constraint solver. The Scatter tool is characterised as an academic tool and could not be found either as an evaluation copy or as an open source tool.

VMWT (Variability Modelling Web Tool): a web-based variability management tool that allows for creating and storing product line projects. Adding variants and their associated variation points to a particular code component can allow the products configuration. Afterwards, a range of numeric values or an enumerated list will be specified. Once all the variants are added, all the variation points will then be added to the code components. VMWT supports management of the dependency and constraints that might exist within the added variation points and variants using Boolean relationships, such as AND, OR, XOR and NONE, or even the use of requires and exclude for the case of a complex decency. However, it enables checking for the completeness of the computed dependency and constraints. VMWT has been conceived as a prototype research at the University Rey Juan Carlos of

Madrid. The first version of the work was implemented as a web-based application, built with PHP and Ajax, running over Apache 2.0 (Capilla et al.). Although, the tool's link is still available in the published literature, it is no longer in existence anywhere. VMWT is an open-source tool.

LKC (Linux Kernel Configurator): a feature modelling tool delivered within the Linux Kernel that enables feature selection in a product line model. LKC comprises two main components that are used as its backbone. These include a parser and a dependency checker. LKC allows configuration options (feature selection) to be defined in a configuration database organised in a tree structure. The configuration database consists of a collection of configuration options built as a set of entries, where each entry comes with its own dependencies to be used in determining its visibility based on only one condition, that is, if its entry is also visible (Sincero and Schroder-Preikschat, 2008). The LKC tool was developed at the Department of Computer Science, Friedrich-Alexander University Erlangen-Nuremberg, firstly, as a prototype in 2002, whereas the current version is 1.3. The tool provides support for multiple notations, such as graphical based on basic right-click functionality, textual using strictly an LKC syntax, and command-line, among others. It belongs to the group of academic tools and was released as an open-source tool under (GNU) General Public License.

FeatureMapper: an Eclipse plugin tool used to map features from feature model to subjective modelling artefacts expressed by means of Ecore-based languages, such as Unified Modelling Language (UML2), Domain Specific Languages (DSLs) that could be defined based on the Eclipse Modelling

Framework (EMF), and text-based languages defined using EMFText. The tool provides support for both manual and automatic mapping and offers various views for mapping features. These include: 1) the realisation view, that specifies the features that need to be mapped to specific features, as well as the model elements that do not participate in the realisation; 2) the variant view, which point outs all the model elements of a specific product line variant, that is, all elements that are common to all products in a product line that are not yet mapped; 3) the context view, for colouring of features and model elements in a feature model. The colours are used for easy realisation of features; and 4) the properties changes-view, which allows for the changing of model elements, such as feature cardinality, names of elements or values initialisation. In this view, properties affected by the changes will be highlighted (Heidenreich et al., 2008b, Heidenreich, 2009, Heidenreich et al., 2008a). FeatureMapper was developed at Technische Universidad, Dresden, and the implementation was based on Eclipse Graphical Editing Framework (GEM). It supports different means of visualisation (both graphical and tree-based editors). Although there was not enough information available, FeatureMapper belongs to the category of academic tools and is available for free from the tool's website.

PLUM (Product Line Unified Modeller): a tool suite for Product Line Engineering that follows a Model-Driven Software Development approach to capture and analyse the product line variability in terms of decisions and establishes the dependencies among them. The tool allows product line developers to design, implement and manage product lines through a guided variability resolution process of the decision models. The variability resolution process consists of assigning values to the decisions in order to define a product of interest. This gives rise to the application model to represent concrete products of the product line. The PLUM tool suite has been developed at the European Software Institute (ESI) based on the concepts from the results of past International Research Projects (such as FAMILIES, ESAPS or CAFÉ and FLEXI), for representing product lines and its variability into model. The tool was implemented as an Eclipse plug-in using a wide range of eclipse framework technologies, such as EMF (Eclipse Modelling Framework) for providing basic building blocks for the models and their editors. GMF (Graphical Modelling Framework) for graphical assets editors. OCL (Object Constraint Language) for decision model's dependency engine and BIRT (Business Intelligence and Reporting Tools) for representing valuable SPL's metrics (Aldazabal and Erofeev, 2008, Martinez et al., 2009). It supports graphical notations using UML, and belongs to the category of commercial tools. PLUM is an open-source tool suite available on the ESI website.

XToF: an integrated tool support for product line development that lets users create feature diagrams, tagged blocks of code, classes and packages in object-oriented programs, feature configurations and the automatic generation of products by running the source code. The tool allows programmers to define, maintain, visualise and exploit traceability links between a feature diagram and a code base. It uses the capabilities of an open-source Eclipse plug-in called TagSEA, which provides mechanisms to filter tags, as well as define and list waypoints. XToF uses these mechanisms to link the TagSEA waypoints to features and blocks of code. Feature diagrams are to be loaded in the SXFM format of an XML file, the XToF then displays it and enables tags to be added, navigated and configured. The loaded feature diagram is then saved into the project folder, giving its path as property of the project. This allows all the project contributors to work in parallel by getting access to the project folder (Gauthier et al., 2010). XToF was developed as a result of collaborative efforts between three different institutions: the University of Victoria, the University of Waterloo and the University of Namur. It was implemented as a plug-in on the Eclipse platform and supports tag-based implementation in Java and C languages. XToF has been deployed in the industry and is therefore categorised as a commercial tool. It is available as an open-source tool.

ToolDay: a domain analysis tool aimed at supporting domain analysts to achieve an effective and systematic reuse of software artefacts through a semi-automation of the entire project. It is architecturally a three-layer-based view consisting of a graphical user interface layer (GUI), which allows a friendly environment for the users, a business layer that holds each ToolDay's group components, i.e., planning, domain modelling, domain validation and product validation, and a data persistency layer for saving information (Lisboa et al., 2011, Lisboa, 2008). ToolDay was developed at the Federal University of Pernambuco, Brazil and implemented using Eclipse's Graphical Modelling Framework (GMF). However, using the Eclipse standard, the tool saves information as XML files, where each file has its proper XML document, making it easy for other tools to import information from ToolDay. It supports graphical notation using the Eclipse platform through its Rich Client Platform (CRP). ToolDay is listed in the category of academic tools. Consequently, the tool is not available as either free open source software or as an evaluation copy.

View Infinity: a software product line (SPL) tool that supports visualisation and zooming functionality of different abstraction layers of SPL content, including feature model, file structure and source code. It offers seamless and semantic zooming of the feature model and source code that can be edited in other product line tools. This allows for a better, step-by-step visualisation of project data, while zooming the details of the presented information. The View infinity interface consists of three different views, including the feature model view, which allows for exploration of SPL's feature tree at its most abstract level. In this view, the feature model is viewed as a graph consisting of hierarchically-connected feature nodes. The user can disable and enable features to create a specific variant. However, from this view, a subsequent zooming of active features can take place, allowing one to deeply explore the actual implementation level of these features; first, it allows for zooming into the file view, where the details of a file structure are revealed. A more subsequent zooming would allow viewing into the detail implementation of individual code fragments of that particular feature. That is the code view (Stengel et al., 2011). View infinity was developed at the University of Magdeburg and implemented based on Java technology. The tool supports graphical notation based on FODA and textual notation using color-coded source code. The tool is open-source software available from the tool's website.

FAMILIAR (FeaAture Model script Language for manipulation and Automatic Reasoning): a language and tool environment for the management of variability feature models developed based on domain-specific language (DSL). It provides support for various operations related to variability management tasks, such as importing, exporting editing, composing, decomposing, configuring and reasoning about feature models. FAMILIAR was initially developed at I3S laboratory and is currently jointly and openly managed by the Triskell team (INRIA/IRISA/University of Rennes 1), the MODALIS team (I3S laboratory, University of Nice Sophia Antipolis) and the Colorado State University, USA. It was developed in Java language using XText, a framework for DSLs development. The tool was, however, implemented in different solutions for use. These includes: 1) FAMILIAR Tool that is fully integrated as a standalone modelling app executed as a JAR file; 2) A plugin for the Eclipse platform, integrated with XText and FeatureIDE, and 3) as a standalone version that supports console mode only (Acher et al., 2011, Acher et al., 2013). The tool supports both textual script for performing a series of operations on feature models and graphical notations for the visualisation and editing of feature models. FAMILIAR is available as a free open-source tool at GitHub pages.

DOPLER: (Decision-Oriented Product Line Engineering for effective Reuse): a flexible and extensible tool suit for variability modelling that allows for the creation of a meta-model to define the asset types (such as features, architectural elements, resources or properties), attributes and dependencies. The tool allows users to make decisions based on the models and are capable of determining the required assets of a product (Dhungana et al., 2010). Since 2006, Dopler has been under continuous development at Christian Dopler Laboratory for Automated Software Engineering, Johannes Kepler University, Linz, in partnership with Siemens VAI and Siemens CT. The tool was developed based on Java Technology and supports both textual and graphical notation. It has been a proprietary of Siemen VAI and therefore neither its evaluation copy nor a free version is available. Service-oriented software development, Eclipse-based development and enterprise resource planning, among others, are the other domains in which Dopler can be used (Dhungana et al., 2011, Rabiser et al., 2009).

FeatureIDE: an Eclipse-based tool and framework aimed at supporting entire development process of Feature-Oriented Software Development (FOSD) for the development of SPLs. Among others: 1) it allows variabilities and commonalities of a software system to be captured for the purpose of domain analysis; 2) it eases the implementation of all software systems of the domain, while at the same time, mapping code assets to features; 3) it provides the means to map requirements to features within the domain and feature configurations for a customised software system; and 4) allows the automatic generation of the software system. Since 2005, when it was first developed at the University of Magdeburg, FeatureIDE has been under constant development, resulting in various improvements; among others are full integration in Eclipse, support for both textual and graphical notations, in which features models can be edited (categorised using generalisations or specialisations or none of these), the highlighting of dead and false-optional features along with their corresponding constraints, a configuration editor for creating and editing configurations, a view for displaying statistics about the software product line, and so on (Thüm et al., 2014, Meinicke et al., 2016, Kästner et al., 2009). The implementation was based on Java technology, but provides support for other implementations, such as AspectJ extension and FeatureC++ extension. FeatureIDE is an academic tool and can be downloaded either directly from its website or in the Eclipse MarketPlace.

MOSKitt4SPL: a tool for modelling software product lines and the application of model-driven development distributed as a platform-independent plugin to be installed on any Eclipse modelling tools (EMT). M4SPL is based on the Eclipse Modelling Framework (EMF), Graphical Modelling Framework (GMF) and Atlas Transformation Language (ATL). The tool is designed to help software developers in analysing, designing and developing adaptive software systems. However, various editors have been provided by the tool to simplify the specification of self-adaptive systems, including: (1) Feature Models Editor for representing and describing the variability of a system in terms of features and from which possible configurations for different systems can evolve; (2) Feature Model Configuration Editor, in which variations of different systems can be defined; and (3) Resolution Model Editor, which provides declarative support among different system configurations (Gómez et al., 2012). M4SPL was developed at The Technical University of Valencia, Spain. It supports graphical notations based on various editors. The MOSKitt tool is a free open-source tool that can be downloaded along with the installation guides via a link by filling and submitting a contact form available from the tool's website. It can be used either as a standalone plug-in on Eclipse or integrated in the Model Driven Everything (MDE) MOSKitt environment.

ISMT4SPL (Integrated Software Management Tool for Adopting Software Product Lines): a based-on orthogonal variability model (OVM) approach to provide a method to reduce the complexity of variability management and allows traceability within the artefacts between domain and application engineering. It also supports dependency management among variants and their variation points by allowing for the automatic generation of variability models. Architecturally, ISMT4SPL comprises three abstraction layers: (1) System layer to ensure all fundamental functions, such as Legacy Requirements Management System function, Legacy Design Management System function and Legacy Configuration management System function; (2) Product line Layer for traceability management, variability management and Product line adapter that plays a role of intermediate function between the system layer and user interface layer; (3) User Interface layer that comprises various views, including the requirement view, design view and configuration view. All the views display a variability model window and project explorer window (Park et al., 2012). The tool was developed at the Korea Advanced Institute of Science (KAIST) and supports both graphical and textual notation. However, it belongs to the category of academic tools and neither an evaluation copy nor a free open-source version can be obtained.

BeTTy: an extensible and configurable tool and framework supporting benchmarking and testing on the analysis of feature models by examining their set of products sufficiently. Among others, BeTTy supports the automated detection of faults in feature models. Also, it supports the automated generation of test data for a number of operations performed on feature models based on the input data. It also includes an algorithm for feature model generation, which maximises user-defined optimisation criteria. When a feature model is generated in BeTTy, a more complex and extended feature models can be generated that reveals the performance of tools in pessimistic cases. On the framework end, BeTTy allows for the generation of several components for simplifying the performance evaluation of feature model analysis tools. The tool was developed at the University of Seville, Spain. It was implemented in Java and distributed as a jar file, which gives it the ability to be integrated into external projects, as well as through a web interface that facilitates the generation of customised random feature models. This can be used to assess the average performance of other tools during analysis process. BeTTy belongs to the category of academic tools, and it has also been freely distributed under GPL v3 license, available from the BeTTy website.

S2T2 Configurator: a feature configuration tool that supports the generation of a visual collaborative representation of the feature model and offers a proper explanation of the effects of the user’s actions using its reasoning engine. The reasoning engine depends on SAT solver and therefore requires the Conjunction Normal Form (CNF). Also, using its architectural design, it allows for mapping among visual components and their matching formal representations. The S2T2 configurator can validate complex Boolean constraints and textually represent them in a separate window. The tool's behaviour of focusing on the task of feature model configuration allows other forms of feature models, developed from other tools like SPLOT and AHEAD, to be imported. The tool was developed at the University of Limerick and implemented in Java, specified as a plug-in in Eclipse IDE. It supports both graphical notation using vertical node-link layout, where each feature group is represented by node link and dependencies and constraints are represented using FODA-like notation (Botterweck et al., 2009, Pleuss and Botterweck, 2012). It also allows the tree layout to be collapsed and expanded, similar to file explorer. From a textual perspective, it uses the notation to denote whether a feature's state has been set by the user. The S2T2 Configurator belongs to the category of academic tools and is a free open-source tool.

EASy-Producer: a tool supporting product line engineering by facilitating the development of variant-rich ecosystems. The tool reduces the complexity that is likely to result in losing all control during product configuration when variability models of various product lines from various organisations are composed in an ecosystem. EASy-Producer uses a technique called product line specialisation, which binds variability in an ecosystem through partial instantiation. Among others, using a specific table-based editor, the tool supports product configurations based on the variables filtered by the user. The configuration can also be validated through a reasoner that recognises conflicting values within the model. The tool supports two textual languages for product instantiation; these are the Variability Instantiation Language (VIL) and the Variability Template Language (VTL). The VIL assists engineers in specifying the artefacts' instantiation and during the process of product line production in a rule-based style. EASy-Producer was developed at the University of Hildesheim and implemented as both standalone application and plug-in for Eclipse. It supports both graphical (intended for non-expert users) and textual notation (largely intended for experts) (Eichelberger et al., 2014). The most recent version of the EASy-Producer's source can be found on the project website.

Opti-Select: an interactive multi-objective product line configuration tool designed based on the idea of UIL (User-In-the-loop) for the analysis and optimisation of features. It uses optimisation techniques and algorithms to merge the experience of both the analysts and stakeholders. The interactive feature of the tool allows it to integrate set techniques that provide a step-by-step modelling of features and their configuration, dependency and constraint management, solution optimisation and user exploration capability for the better satisfaction of stakeholders. Opti-Select was developed at the College of Computing and Information Technology, Arab Academy for Science, Technology, and Maritime Transport, Cairo, Egypt. Using a file tree format, the tool graphically allows the loading of the Simple XML Feature Model (SXFM) format. Configuration attributes can either be loaded or saved to a separate file, which can then be linked with relevant features. Once the SXFM file is loaded to the application, the complemented attributes will be displayed so that a user can alter the values of each attribute associated with every feature in order to make the necessary changes according to system requirements. In this, a selected feature or its attribute values can be changed (Yamany et al., 2014). Opti-Select is an academic tool. It is also a free open-source tool and an evaluation copy could be obtained.

MPLM-MaTeLo-product line manager: a product line management tool designed based on the Eclipse Rich Client Platform (RCP) as an extension of the industrial MaTeLo tool chain, to offer a model-based testing formalism from which variants can be generated for MaTeLo to allow the derivation of specific test cases for product line variants. Thus, it gives the ability to generate test cases for each variant. It allows formal communications among features and requirements by relating product line model usage with a variability model. In summary, the tool assists the product line development by defining the product to be developed, configuring features, establishing a link between requirements and features, building a link between products and features, creating new products, generating test plans and creating test models for a product. The MPLM model variability uses the Orthogonal Variability Modelling approach and therefore associates the product line features with that of OVM model. Product configuration is achieved by selecting the desired features (Samih and Bogusch, 2014). The tool was developed at ALL4TEC/INRIA Rennes and supports graphical notation using file tree-like notations. MPLM-MaTeLo has been applied in an industrial setting, and therefore belongs to the category of industry tools.

***VITAL* (Variability ImprovemenT Analysis)**: a variability code analysis toolset that can automatically extract variability models from variability code and allows code visualisation and measurement. Parsing variability code can be achieved using the conditional compilation method. Once the potential variability code is passed, further analyses are conducted at the semantic level to extract reflexion models, with their various elements as well as dependencies. The variability code extraction is implemented as a macro constant, while a variation point is implemented as an 'ifdef' block using Vars in its 'ifdef' statement. The tool was developed at the University of Kaiserslautern, in collaboration with the Fraunhofer Institute of Experimental Software Engineering (IESE), both in Germany. Although different techniques may be used in the identification of and parsing of variability elements, currently, the C-Preprocessor (CPP) code parser has been used in implementation. This enables industries to filter and get the analysis of variability code. The tool supports textual notation using CPP code (Zhang and Becker, 2014). It belongs to both academic and industry tools. The VITAL tool is not available to download as free open-source software, and likewise, no evaluation copy can be obtained.

ViViD: a variability management tool for synthesising variants for video sequences for realisations of videos with different characteristics, such as distinctions of luminance and the calculation of vehicles and people to cover a range of testing scenarios. Among others, the tool supports variability modelling language and an environment to enable the modelling of variations within a video sequence. It also allows for the generation of testing configurations for the variants of video sequences corresponding to those configurations. In addition, based on the valid number of configurations in a variability model, the tool supports prioritisation for pair-wise configurations while maintaining the maximum and minimum ad-hoc objective functions (Acher et al., 2014). ViViD was developed at the University of Rennes in close collaboration with MOTIV industrial partners. It supports textual notations created based on the open-source Xtext language workbench. The tool has been applied in industry, and therefore belongs to both academic and commercial tool categories. Although the ViViD tool is not available as an open-source tool, all the tool's components can be downloaded from its website.

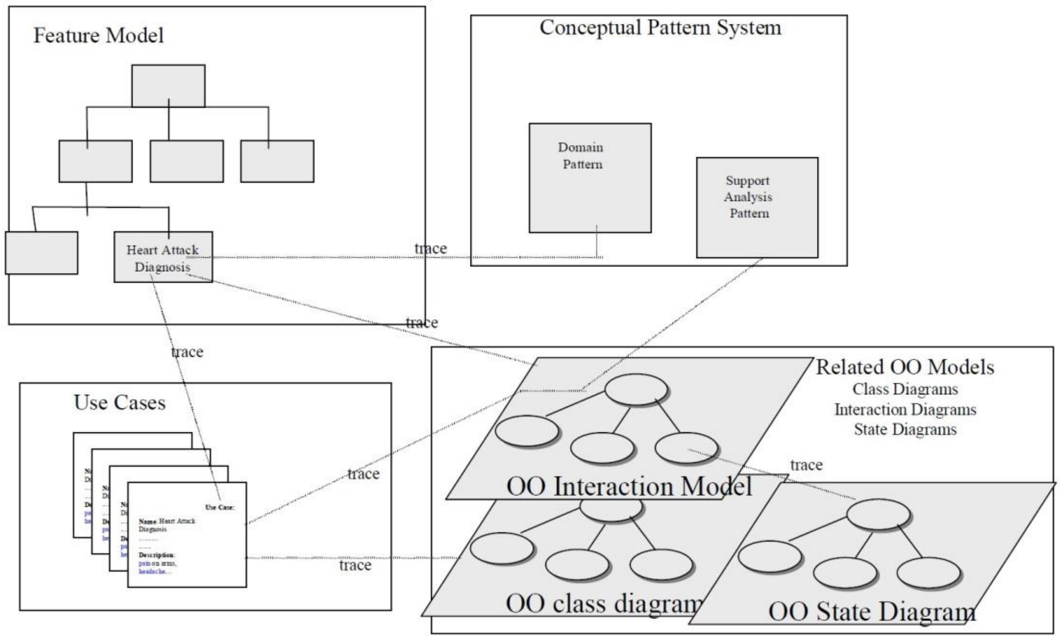
VMC (Variability Model Checker): a software product line tool supporting modelling and analysis concepts that can be specified in a value-passing process algebra, supplemented with a set of optional variability constraints. As inputs, it takes a product family model, together with variability constraints, and then using its variability-aware version of action (v-ACTL), it offers a logical analysis on the behavioural variability of a product family to its valid products. VMC advances SPL development processes by means of variability-aware logic interpreted over a Model Transition System (MTS) with some additional variability constraints. VMC was developed at the Institute of National Research Council of Italy, ISTI-CNR, and developed in Ada language to allow its compilation on various platforms, including Windows, Linux, Mac OS-X and Solaris. Its computational model was based on a combination of automata formed from a series of algebraic processes originated from the value-passing Calculus of Communicating Systems and Communicating Sequential Processes (CCS/CSP) like calculus. It supports textual notation constituted by command-line prompt. It also supports graphical notation using html-oriented GUI and when integrated with graph-drawing tools (Ter Beek and Mazzanti, 2014). The tool is not open-source software; likewise, an evaluation copy cannot be obtained.

WebFML: a product line online environment for synthesising feature models from different kinds of software artefacts, such as propositional formula and dependencies using graphs or matrices. It provides interactive support via logical heuristics (a technique used in problem-solving), clusters (contains the desired set of parents and children features), and ranking list (a list of parent candidates for every feature) to allow multiple choices of substantial and desired hierarchy. It also provides a mechanism for translating variability artefacts into feature models, in addition to speeding up and supervising the building process of feature models. This tool helps practitioners minimise their efforts while handling complex variability models. The tool supports partial integration with other tools, such as FAMILIAR, so that the scripts files computed on those tools can be managed with the help of an integrated console. The tool was developed at Inria/IRISA, University of Rennes 1, France, and implemented using JavaScript based on Dagre and D3 libraries (Bécan et al., 2014). Its web interface supports graphical notations using the tree explorer view. The tool demo is publicly available from the tool’s website.

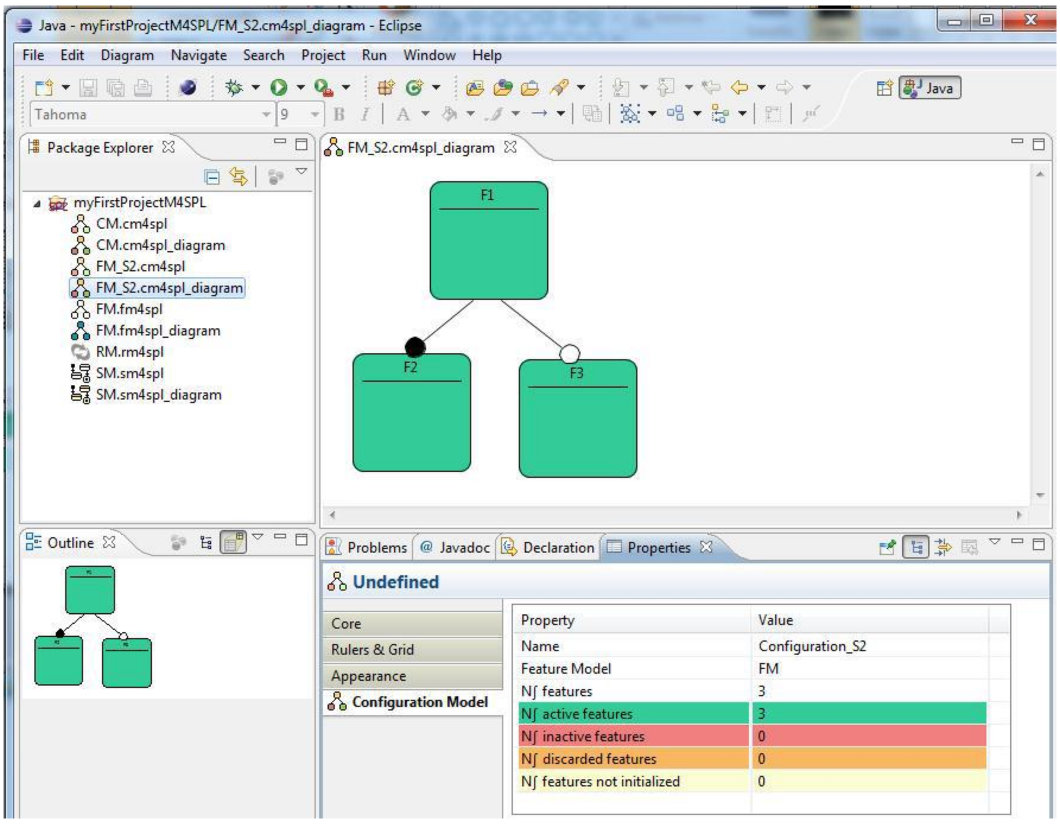
Appendix B

Table 4.2: Tools with FODA-like visual notations

[S10]

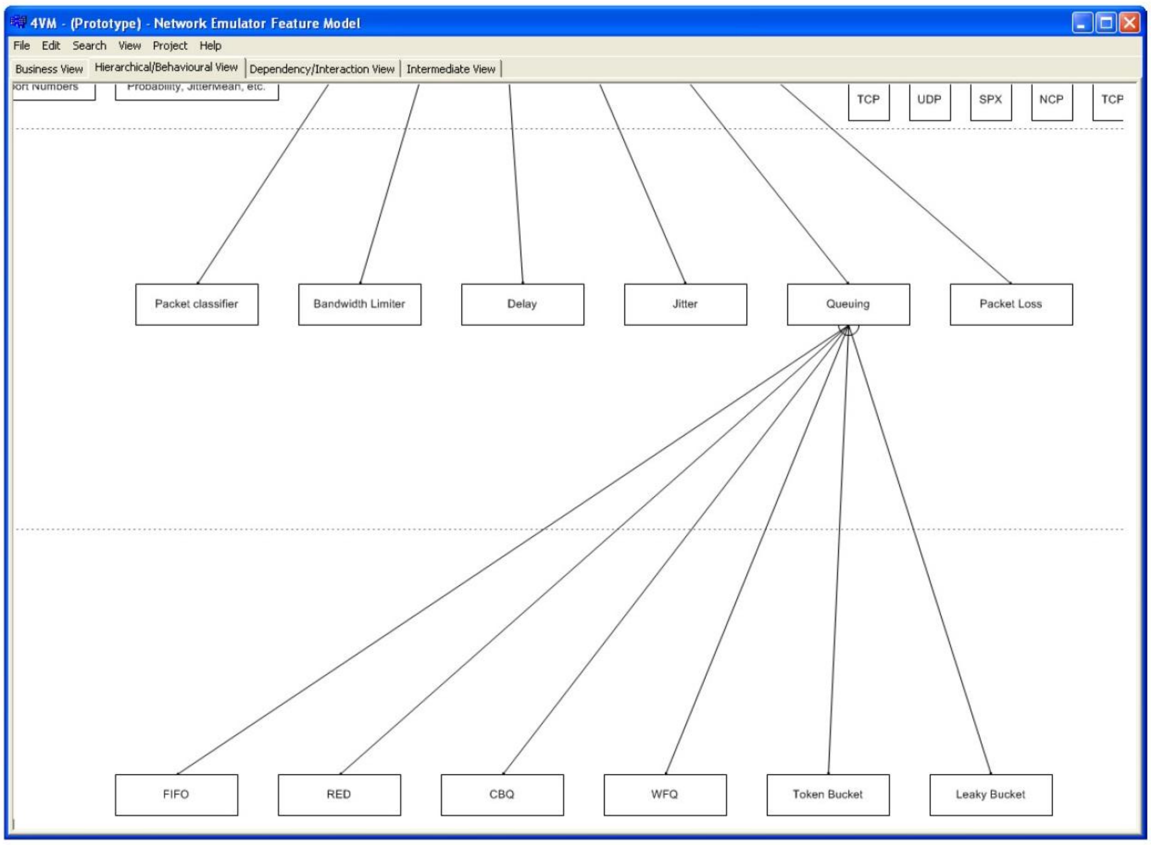


[S21]

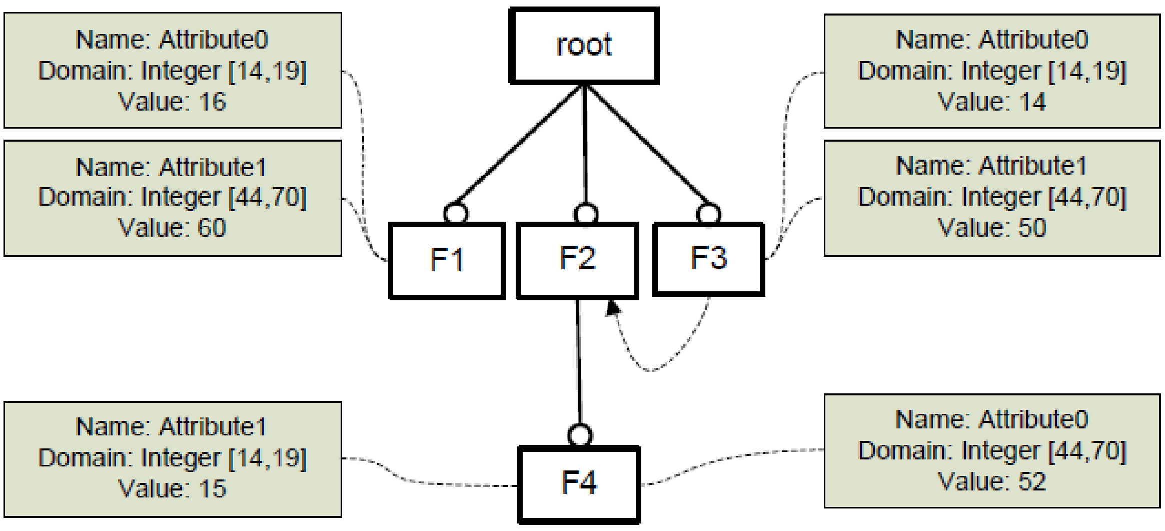


Property	Value
Name	Configuration_S2
Feature Model	FM
N∫ features	3
N∫ active features	3
N∫ inactive features	0
N∫ discarded features	0
N∫ features not initialized	0

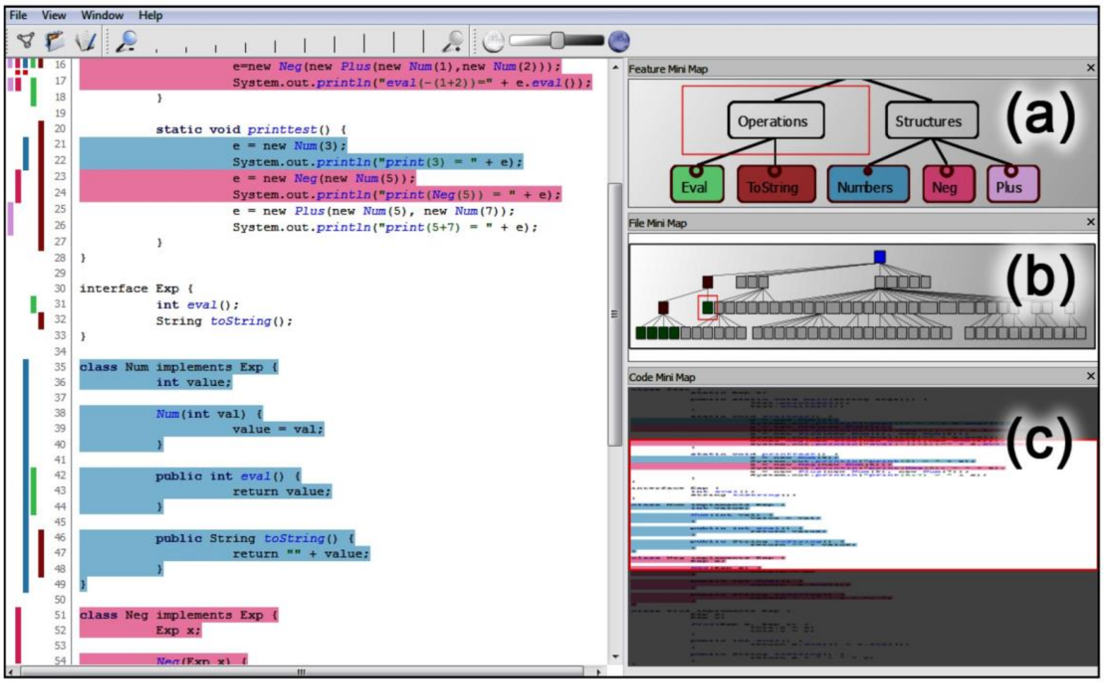
[S11]


4VM - (Prototype) - Network Emulator Feature Model
File Edit Search View Project Help
Business View
Hierarchical/Behavioural View
Dependency/Interaction View
Intermediate View
ort Numbers
Probability, JitterMean, etc.
TCP
UDP
SPX
NCP
TCP
Packet classifier
Bandwidth Limiter
Delay
Jitter
Queuing
Packet Loss
FIFO
RED
CBQ
WFQ
Token Bucket
Leaky Bucket

[S22]

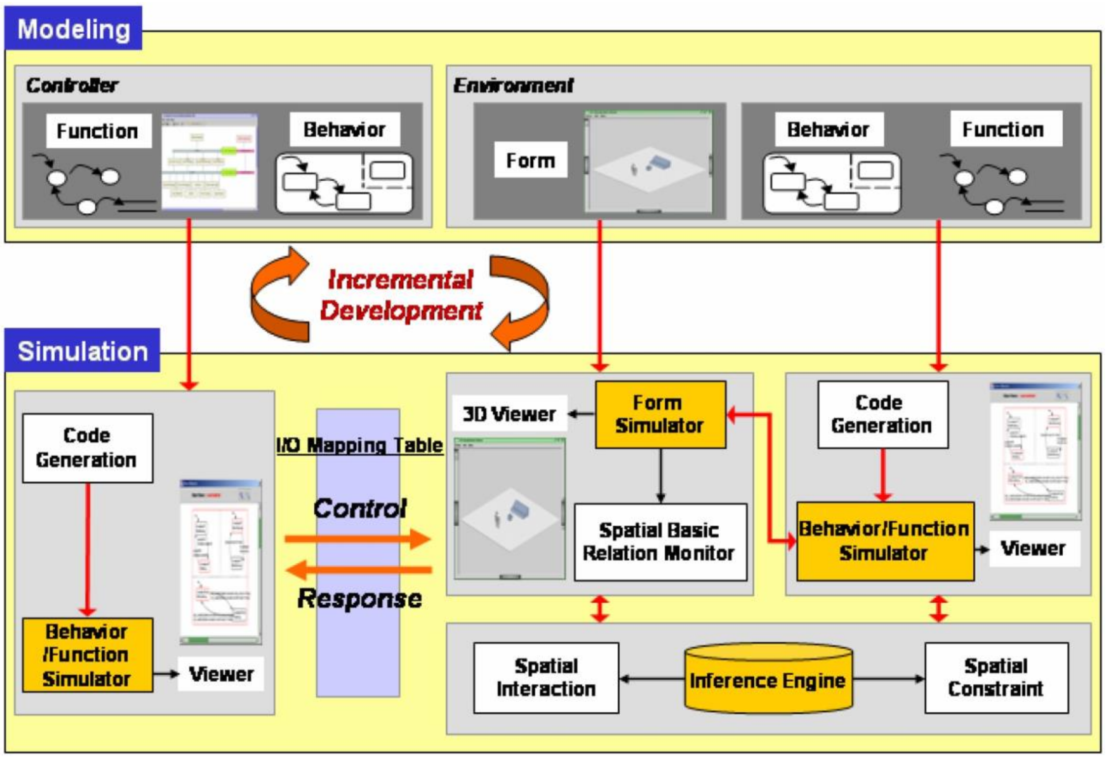

Name: Attribute0
Domain: Integer [14,19]
Value: 16
Name: Attribute1
Domain: Integer [44,70]
Value: 60
root
F1
F2
F3
F4
Name: Attribute0
Domain: Integer [14,19]
Value: 14
Name: Attribute1
Domain: Integer [44,70]
Value: 50
Name: Attribute1
Domain: Integer [14,19]
Value: 15
Name: Attribute0
Domain: Integer [44,70]
Value: 52

[S14]

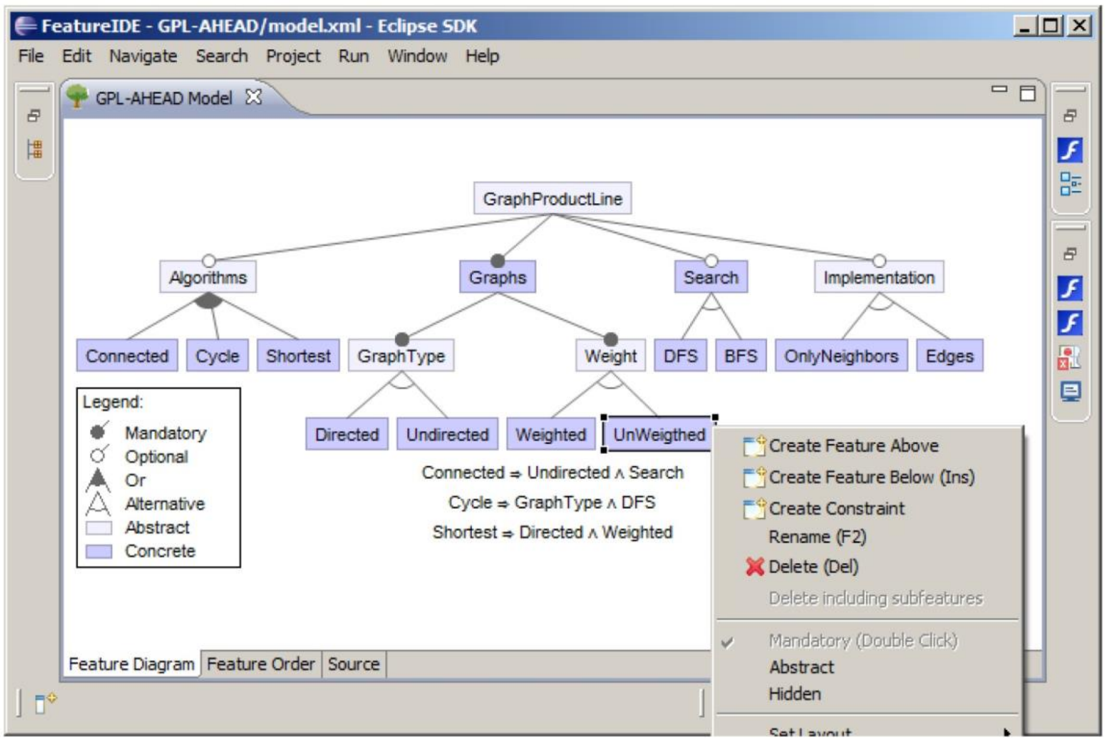


```
                e=new Neg(new Plus(new Num(1),new Num(2)));
                System.out.println("eval(-(1+2))=" + e.eval());
        }

        static void printtest() {
                e = new Num(3);
                System.out.println("print(3) = " + e);
                e = new Neg(new Num(5));
                System.out.println("print(Neg(5)) = " + e);
                e = new Plus(new Num(5), new Num(7));
                System.out.println("print(5+7) = " + e);
        }
}

interface Exp {
        int eval();
        String toString();
}

class Num implements Exp {
        int value;

        Num(int val) {
                value = val;
        }

        public int eval() {
                return value;
        }

        public String toString() {
                return "" + value;
        }
}

class Neg implements Exp {
        Exp x;

        Neg(Exp x) {
```

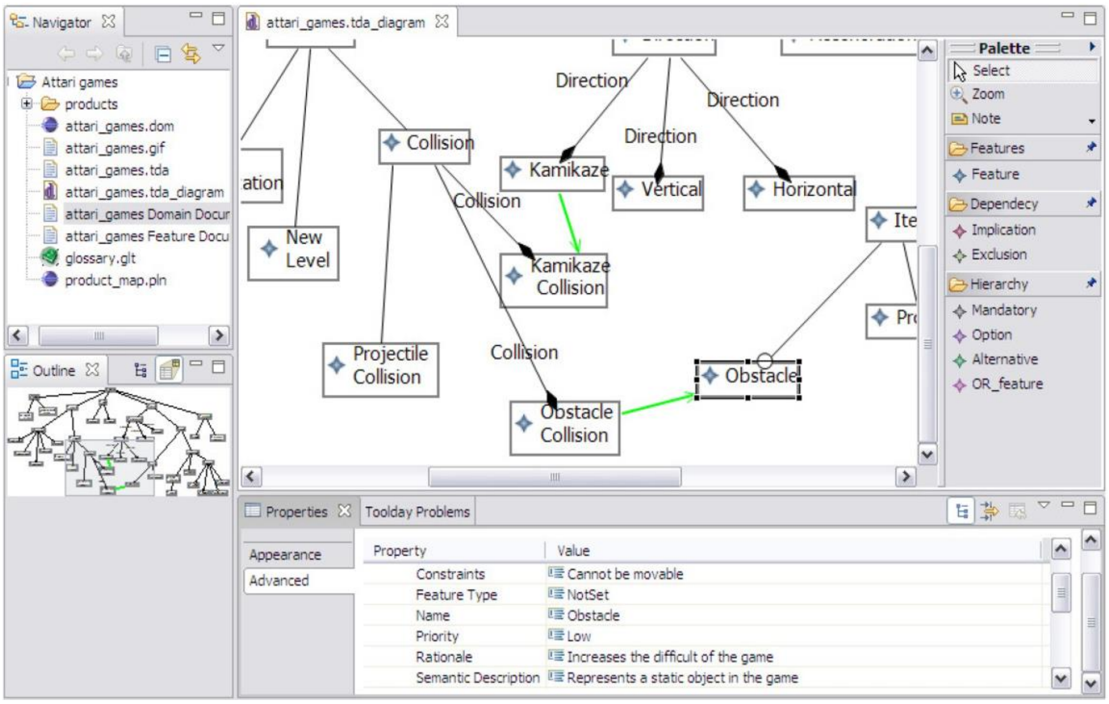
[S25]



[S15]

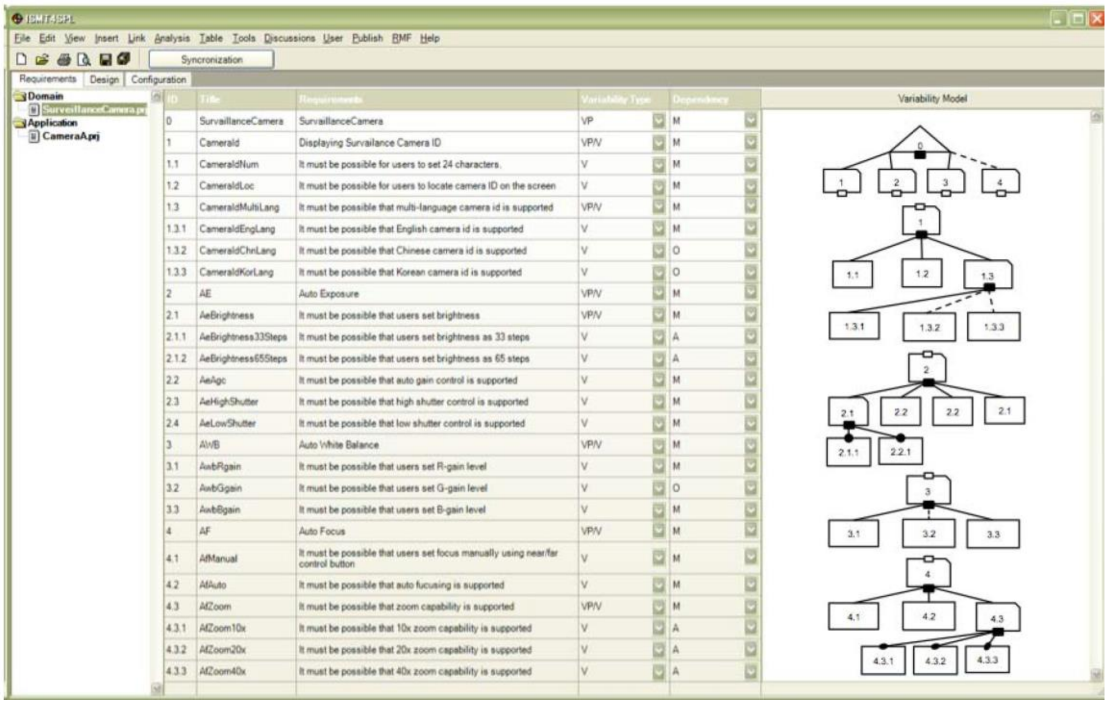


[S27]



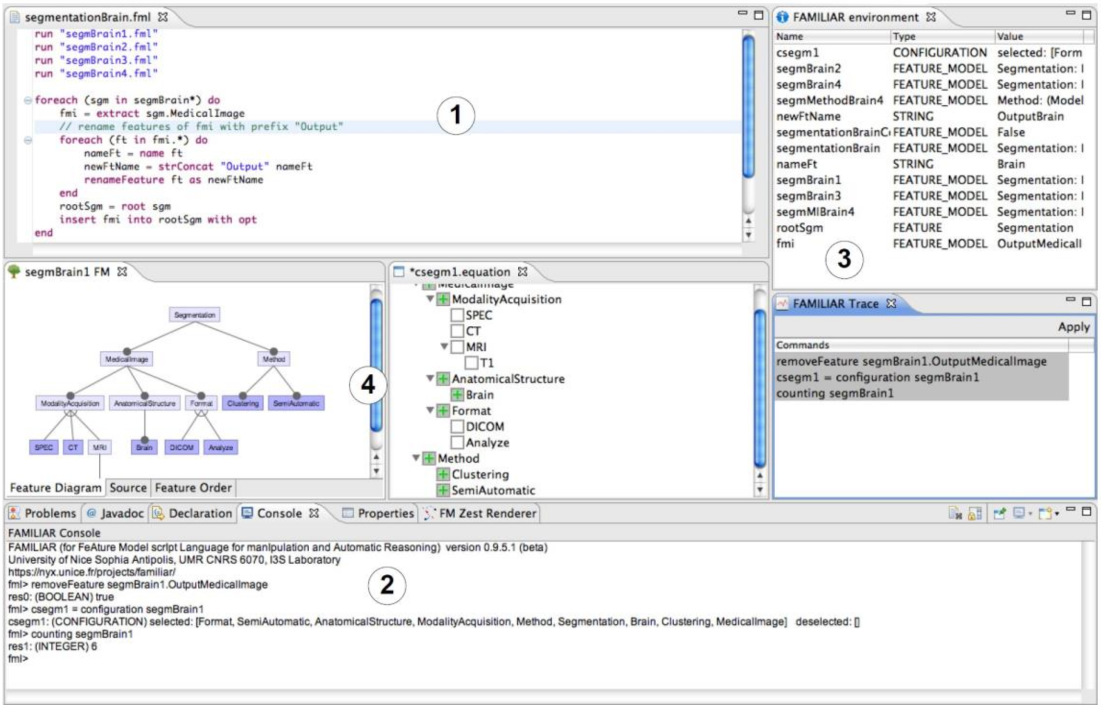
Property	Value
Constraints	Cannot be movable
Feature Type	NotSet
Name	Obstacle
Priority	Low
Rationale	Increases the difficult of the game
Semantic Description	Represents a static object in the game

[S17]



ID	Title	Requirements	Variability Type	Dependency
0	SurvaillanceCamera	SurvaillanceCamera	VP	M
1	CameraId	Displaying Survailance Camera ID	VP/V	M
1.1	CameraIdNum	It must be possible for users to set 24 characters.	V	M
1.2	CameraIdLoc	It must be possible for users to locate camera ID on the screen	V	M
1.3	CameraIdMultiLang	It must be possible that multi-language camera id is supported	VP/V	M
1.3.1	CameraIdEngLang	It must be possible that English camera id is supported	V	M
1.3.2	CameraIdChnLang	It must be possible that Chinese camera id is supported	V	O
1.3.3	CameraIdKorLang	It must be possible that Korean camera id is supported	V	O
2	AE	Auto Exposure	VP/V	M
2.1	AeBrightness	It must be possible that users set brightness	VP/V	M
2.1.1	AeBrightness33Steps	It must be possible that users set brightness as 33 steps	V	A
2.1.2	AeBrightness65Steps	It must be possible that users set brightness as 65 steps	V	A
2.2	AeAgc	It must be possible that auto gain control is supported	V	M
2.3	AeHighShutter	It must be possible that high shutter control is supported	V	M
2.4	AeLowShutter	It must be possible that low shutter control is supported	V	M
3	AWB	Auto White Balance	VP/V	M
3.1	AwbRgain	It must be possible that users set R-gain level	V	M
3.2	AwbGgain	It must be possible that users set G-gain level	V	O
3.3	AwbBgain	It must be possible that users set B-gain level	V	M
4	AF	Auto Focus	VP/V	M
4.1	AfManual	It must be possible that users set focus manually using near/far control button	V	M
4.2	AfAuto	It must be possible that auto fucusing is supported	V	M
4.3	AfZoom	It must be possible that zoom capability is supported	VP/V	M
4.3.1	AfZoom10x	It must be possible that 10x zoom capability is supported	V	A
4.3.2	AfZoom20x	It must be possible that 20x zoom capability is supported	V	A
4.3.3	AfZoom40x	It must be possible that 40x zoom capability is supported	V	A

[S30]



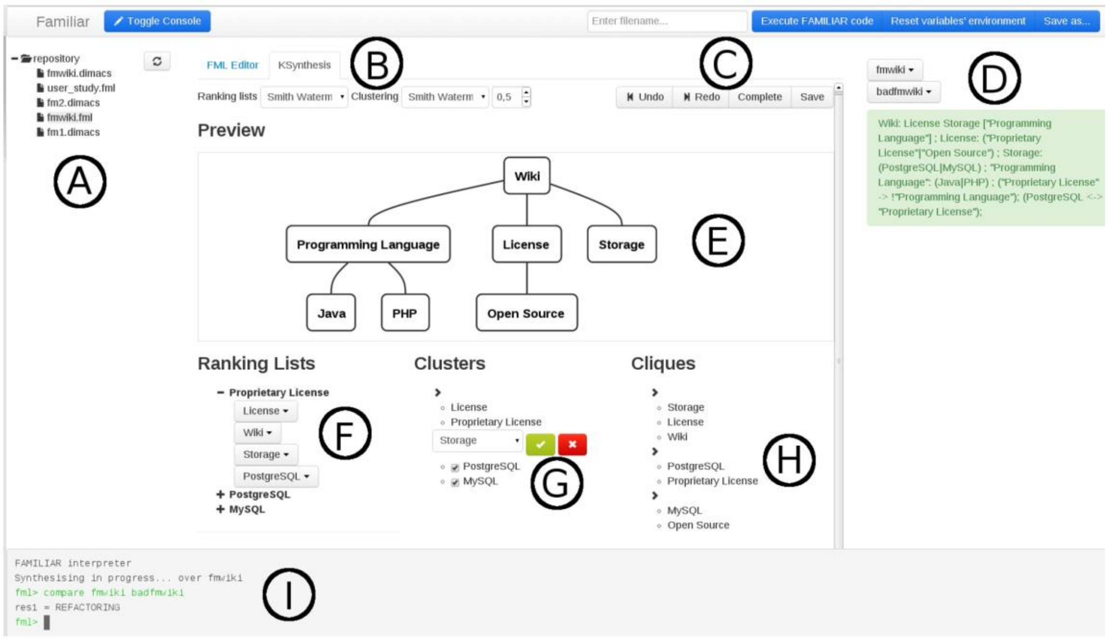
```
run "segmBrain1.fml"
run "segmBrain2.fml"
run "segmBrain3.fml"
run "segmBrain4.fml"

foreach (sgm in segmBrain*) do
    fmi = extract sgm.MedicalImage
    // rename features of fmi with prefix "Output"
    foreach (ft in fmi.*) do
        nameFt = name ft
        newFtName = strConcat "Output" nameFt
        renameFeature ft as newFtName
    end
    rootSgm = root sgm
    insert fmi into rootSgm with opt
end
```

Name	Type	Value
csegm1	CONFIGURATION	selected: [Form
segmBrain2	FEATURE_MODEL	Segmentation: I
segmBrain4	FEATURE_MODEL	Segmentation: I
segmMethodBrain4	FEATURE_MODEL	Method: (Model
newFtName	STRING	OutputBrain
segmentationBrainC	FEATURE_MODEL	False
segmentationBrain	FEATURE_MODEL	Segmentation: I
nameFt	STRING	Brain
segmBrain1	FEATURE_MODEL	Segmentation: I
segmBrain3	FEATURE_MODEL	Segmentation: I
segmMIBrain4	FEATURE_MODEL	Segmentation: I
rootSgm	FEATURE	Segmentation
fmi	FEATURE_MODEL	OutputMedicalI

```
Commands
removeFeature segmBrain1.OutputMedicalImage
csegm1 = configuration segmBrain1
counting segmBrain1
```

```
FAMILIAR Console
FAMILIAR (for FeAture Model script Language for manIpulation and Automatic Reasoning)  version 0.9.5.1 (beta)
University of Nice Sophia Antipolis, UMR CNRS 6070, I3S Laboratory
https://nyx.unice.fr/projects/familiar/
fml> removeFeature segmBrain1.OutputMedicalImage
res0: (BOOLEAN) true
fml> csegm1 = configuration segmBrain1
csegm1: (CONFIGURATION) selected: [Format, SemiAutomatic, AnatomicalStructure, ModalityAcquisition, Method, Segmentation, Brain, Clustering, MedicalImage]   deselected: []
fml> counting segmBrain1
res1: (INTEGER) 6
fml>
```

[S37]

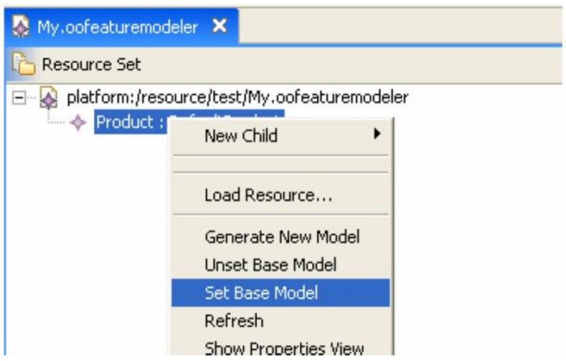


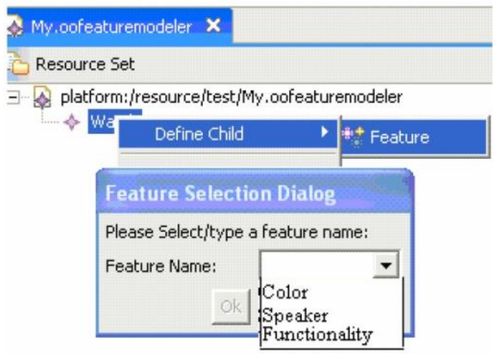
Wiki: License Storage ["Programming Language"] ; License: ("Proprietary License"|"Open Source") ; Storage: (PostgreSQL|MySQL) ; "Programming Language": (Java|PHP) ; ("Proprietary License" -> !"Programming Language"); (PostgreSQL <-> "Proprietary License");

```
FAMILIAR interpreter
Synthesising in progress... over fmwiki
fml> compare fmwiki badfmwiki
res1 = REFACTORING
fml> 
```

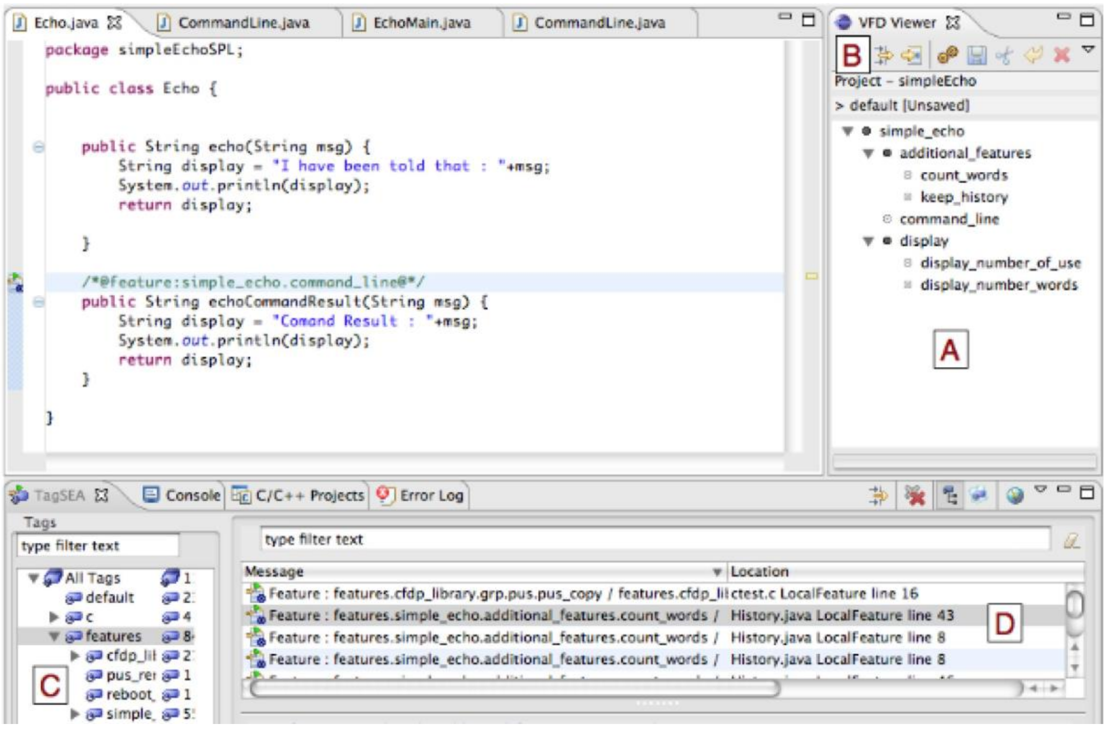
Table 4.3: Tools with file Tree-like visualisation

[S7]





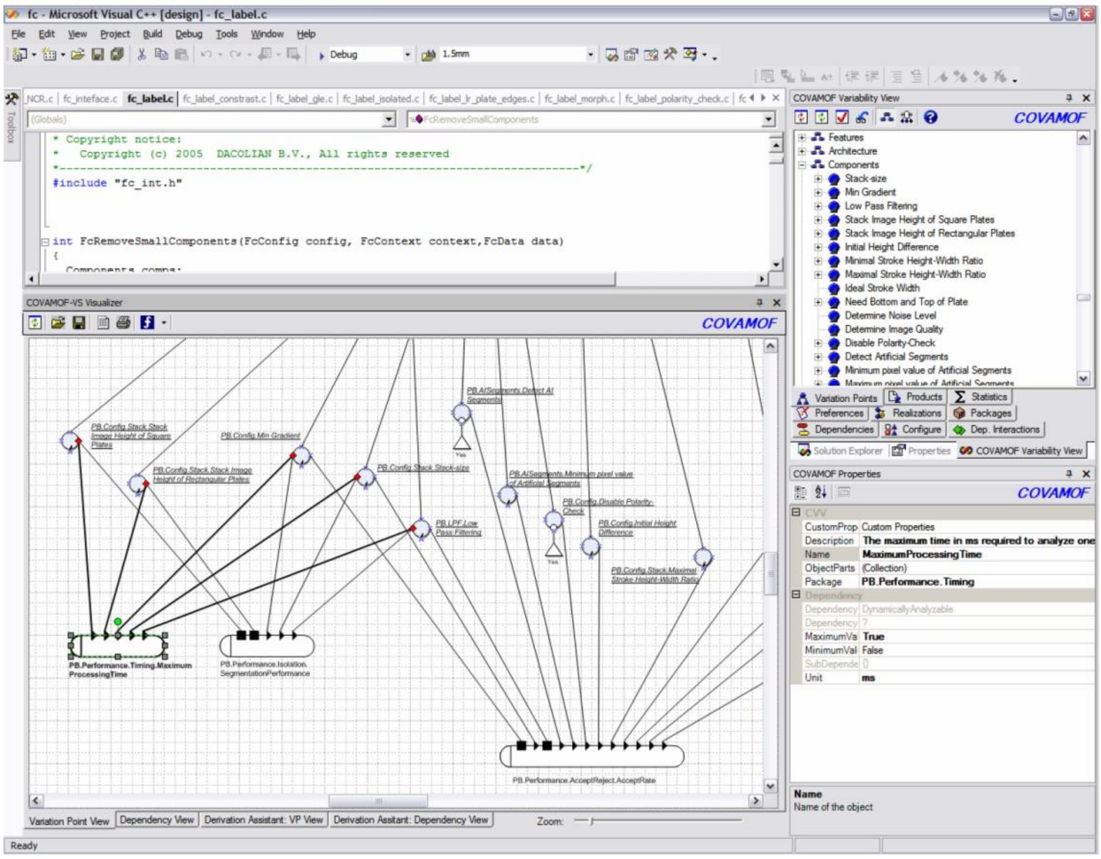
[S13]



```
package simpleEchoSPL;

public class Echo {

    public String echo(String msg) {
        String display = "I have been told that : "+msg;
        System.out.println(display);
        return display;

    }

    /*@feature:simple_echo.command_line@*/
    public String echoCommandResult(String msg) {
        String display = "Comand Result : "+msg;
        System.out.println(display);
        return display;
    }

}
```

Message	Location
Feature : features.cfdp_library.grp.pus.pus_copy / features.cfdp_lil	ctest.c LocalFeature line 16
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 43
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 8
Feature : features.simple_echo.additional_features.count_words /	History.java LocalFeature line 8

[S8]


fc - Microsoft Visual C++ [design] - fc_label.c
File Edit View Project Build Debug Tools Window Help
Debug
1.5mm
NCR.c | fc_inteface.c | fc_label.c | fc_label_constrast.c | fc_label_gle.c | fc_label_isolated.c | fc_label_lr_plate_edges.c | fc_label_morph.c | fc_label_polarity_check.c | fc
(Globals)
FcRemoveSmallComponents
COVAMOF-VS Visualizer
COVAMOF
PB.Performance.Timing.Maximum ProcessingTime
PB.Performance.Isolation. SegmentationPerformance
PB.Performance.AcceptReject.AcceptRate
Variation Point View | Dependency View | Derivation Assistant: VP View | Derivation Assistant: Dependency View
Zoom:
Ready
COVAMOF Variability View
Variation Points | Products | Statistics
Preferences | Realizations | Packages
Dependencies | Configure | Dep. Interactions
Solution Explorer | Properties | COVAMOF Variability View
COVAMOF Properties

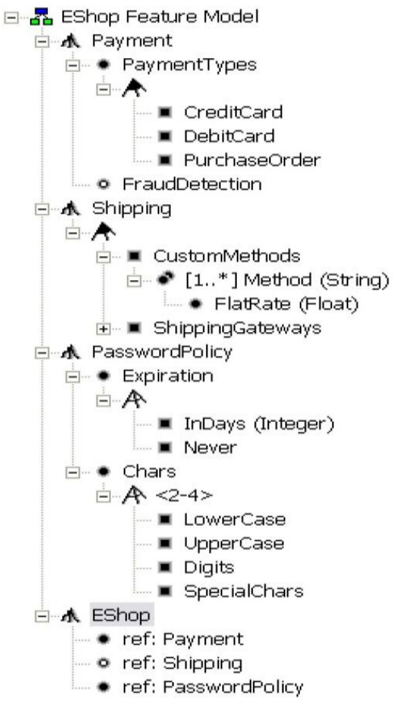
```
 * Copyright notice:
 *   Copyright (c) 2005  DACOLIAN B.V., All rights reserved
 *---------------------------------------------------------------------------*/
#include "fc_int.h"


int FcRemoveSmallComponents(FcConfig config, FcContext context,FcData data)
{
  Components comps;
```

```
Features
Architecture
Components
    Stack-size
    Min Gradient
    Low Pass Filtering
    Stack Image Height of Square Plates
    Stack Image Height of Rectangular Plates
    Initial Height Difference
    Minimal Stroke Height-Width Ratio
    Maximal Stroke Height-Width Ratio
    Ideal Stroke Width
    Need Bottom and Top of Plate
    Determine Noise Level
    Determine Image Quality
    Disable Polarity-Check
    Detect Artificial Segments
    Minimum pixel value of Artificial Segments
    Maximum pixel value of Artificial Segments
```

CVV	
CustomProp	Custom Properties
Description	The maximum time in ms required to analyze one
Name	MaximumProcessingTime
ObjectParts	(Collection)
Package	PB.Performance.Timing
Dependency	
Dependency	DynamicallyAnalyzable
Dependency	?
MaximumVa	True
MinimumVal	False
SubDepende	0
Unit	ms

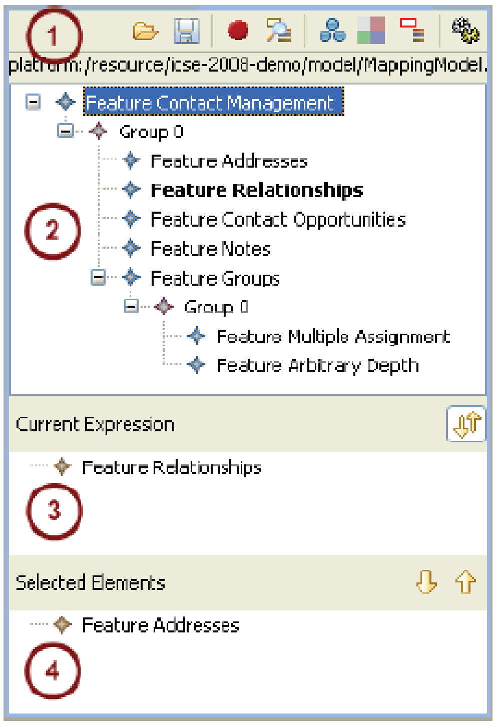
Name
Name of the object

[S16]

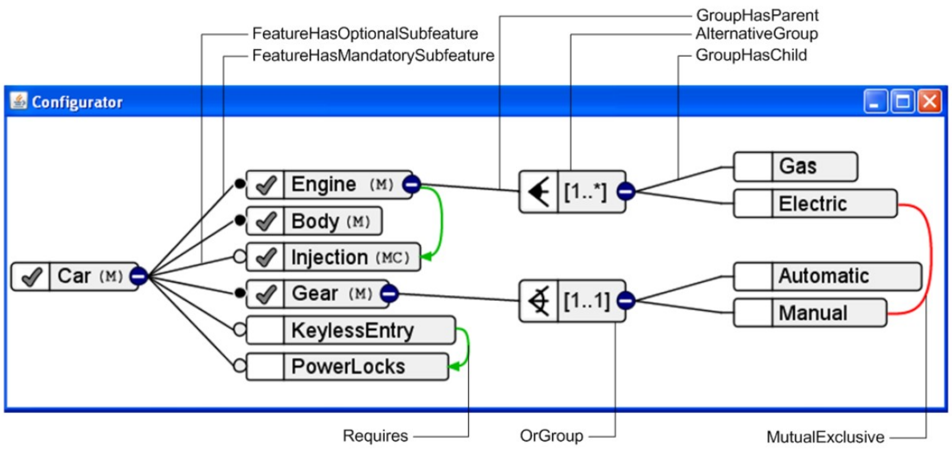


```
EShop Feature Model
  Payment
    PaymentTypes
      CreditCard
      DebitCard
      PurchaseOrder
    FraudDetection
  Shipping
    CustomMethods
      [1..*] Method (String)
        FlatRate (Float)
    ShippingGateways
  PasswordPolicy
    Expiration
      InDays (Integer)
      Never
    Chars
      <2-4>
        LowerCase
        UpperCase
        Digits
        SpecialChars
  EShop
    ref: Payment
    ref: Shipping
    ref: PasswordPolicy
```

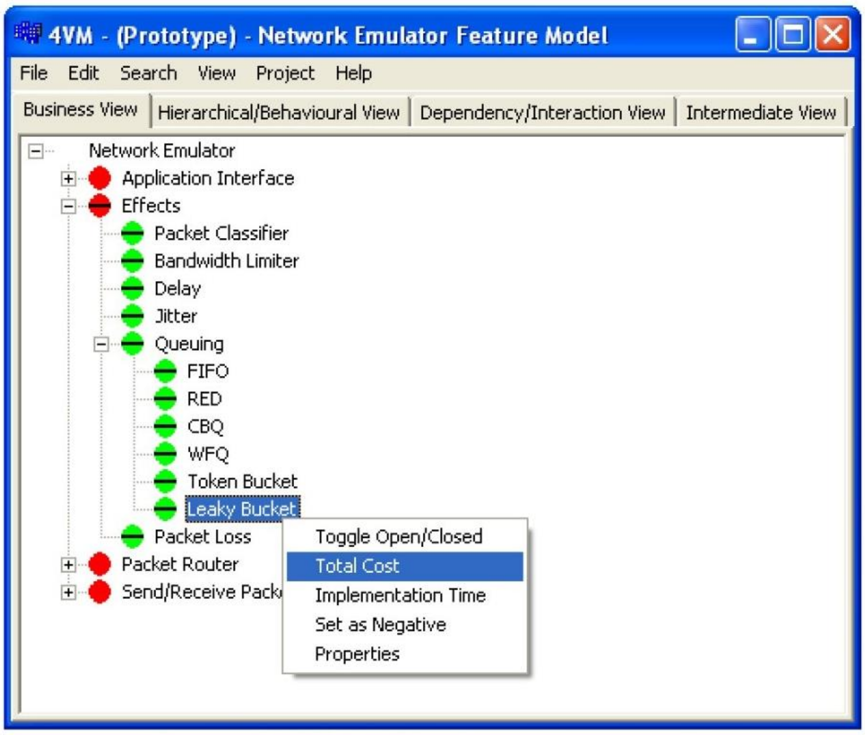
[S9]


1
platform:/resource/icse-2008-demo/model/MappingModel.
Feature Contact Management
Group 0
Feature Addresses
Feature Relationships
Feature Contact Opportunities
Feature Notes
Feature Groups
Group 0
Feature Multiple Assignment
Feature Arbitrary Depth
2
Current Expression
Feature Relationships
3
Selected Elements
Feature Addresses
4

[S24]


FeatureHasOptionalSubfeature
FeatureHasMandatorySubfeature
GroupHasParent
AlternativeGroup
GroupHasChild
Configurator
Car (M)
Engine (M)
Body (M)
Injection (MC)
Gear (M)
KeylessEntry
PowerLocks
[1..*]
[1..1]
Gas
Electric
Automatic
Manual
Requires
OrGroup
MutualExclusive

[S11]



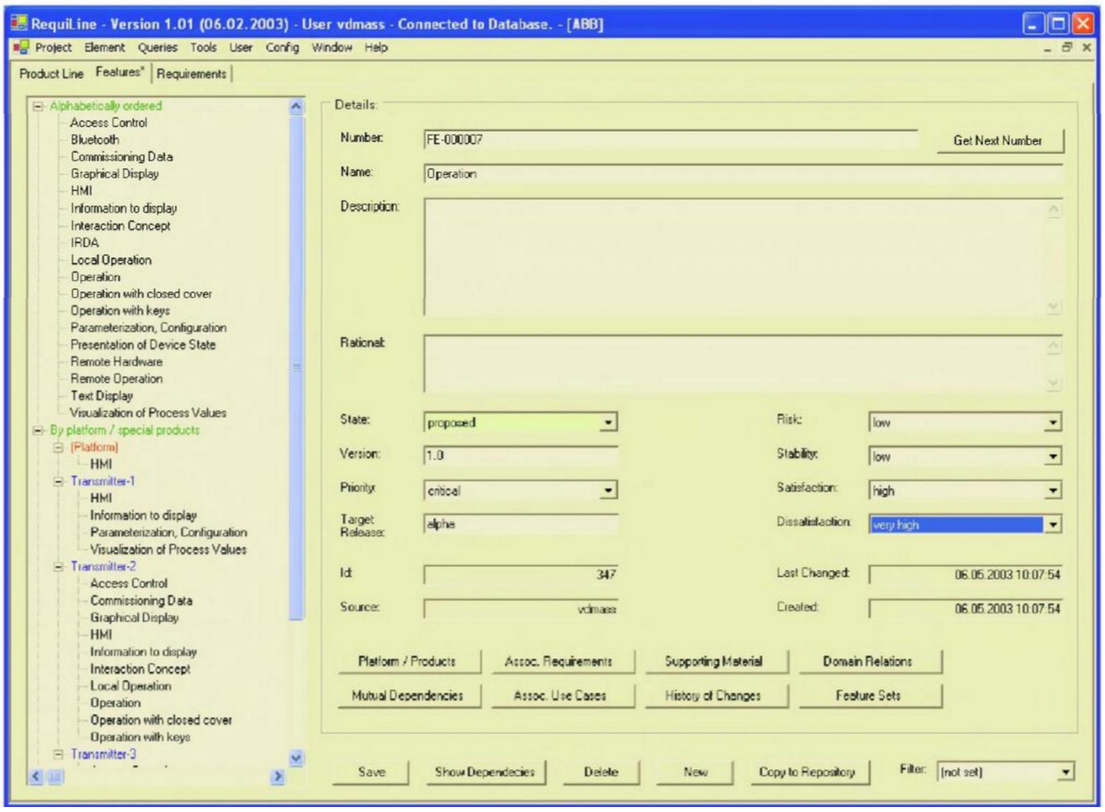
4VM - (Prototype) - Network Emulator Feature Model

File Edit Search View Project Help

Business View | Hierarchical/Behavioural View | Dependency/Interaction View | Intermediate View

- Network Emulator
  - Application Interface
  - Effects
    - Packet Classifier
    - Bandwidth Limiter
    - Delay
    - Jitter
    - Queuing
      - FIFO
      - RED
      - CBQ
      - WFQ
      - Token Bucket
      - Leaky Bucket
    - Packet Loss
  - Packet Router
  - Send/Receive Pack

[S26]



RequiLine - Version 1.01 (06.02.2003) - User vdmass - Connected to Database. - [ABB]

Project Element Queries Tools User Config Window Help

Product Line | Features* | Requirements

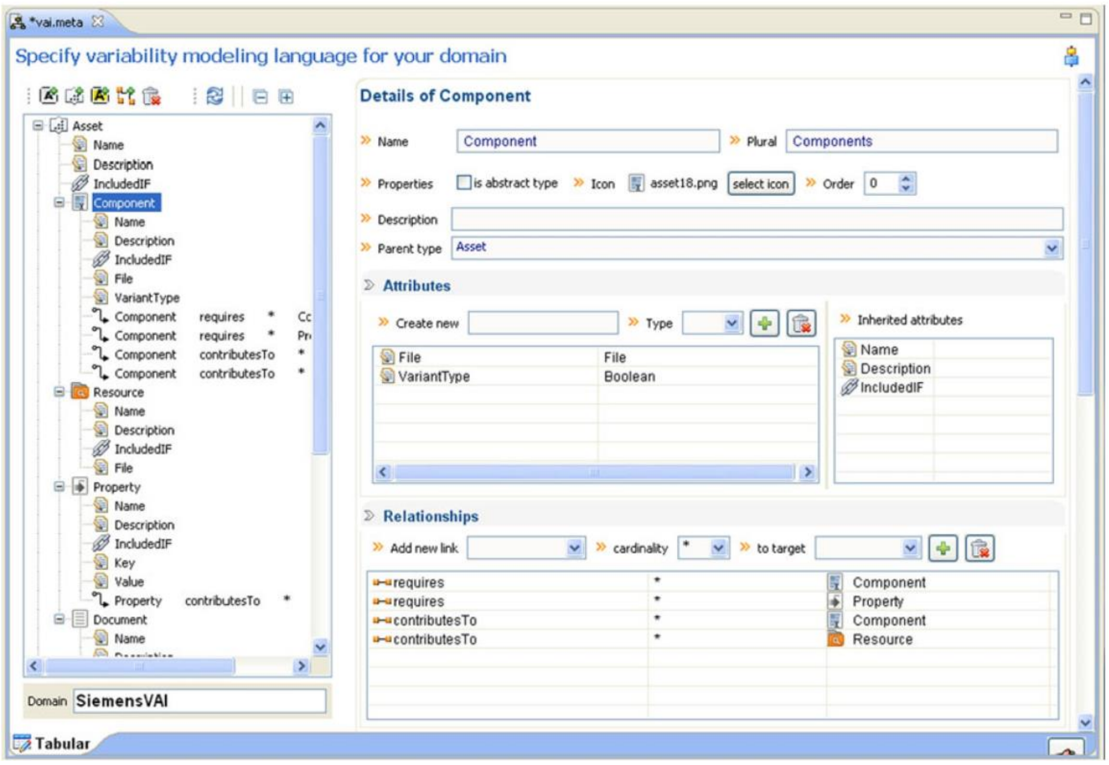
- Alphabetically ordered
  - Access Control
  - Bluetooth
  - Commissioning Data
  - Graphical Display
  - HMI
  - Information to display
  - Interaction Concept
  - IRDA
  - Local Operation
  - Operation
  - Operation with closed cover
  - Operation with keys
  - Parameterization, Configuration
  - Presentation of Device State
  - Remote Hardware
  - Remote Operation
  - Text Display
  - Visualization of Process Values
- By platform / special products
  - [Platform]
    - HMI
  - Transmitter-1
    - HMI
    - Information to display
    - Parameterization, Configuration
    - Visualization of Process Values
  - Transmitter-2
    - Access Control
    - Commissioning Data
    - Graphical Display
    - HMI
    - Information to display
    - Interaction Concept
    - Local Operation
    - Operation
    - Operation with closed cover
    - Operation with keys
  - Transmitter-3

Details:

Number:	FE-000007		
Name:	Operation		
Description:			
Rational:			
State:	proposed	Risk:	low
Version:	1.0	Stability:	low
Priority:	critical	Satisfaction:	high
Target Release:	alphe	Dissatisfaction:	very high
Id:	347	Last Changed:	06.05.2003 10:07:54
Source:	vdmass	Created:	06.05.2003 10:07:54

Filter: (not set)

[S12]


*vai.meta

Specify variability modeling language for your domain

```
Asset
    Name
    Description
    IncludedIF
    Component
        Name
        Description
        IncludedIF
        File
        VariantType
        Component    requires        *   Cc
        Component    requires        *   Pr
        Component    contributesTo   *
        Component    contributesTo   *
    Resource
        Name
        Description
        IncludedIF
        File
    Property
        Name
        Description
        IncludedIF
        Key
        Value
        Property    contributesTo   *
    Document
        Name
```

Domain: SiemensVAI

Details of Component

Name: Component Plural: Components

Properties: ☐ is abstract type Icon: asset18.png select icon Order: 0

Description:

Parent type: Asset

Attributes

Create new: Type:

File	File
VariantType	Boolean

Inherited attributes:

- Name
- Description
- IncludedIF

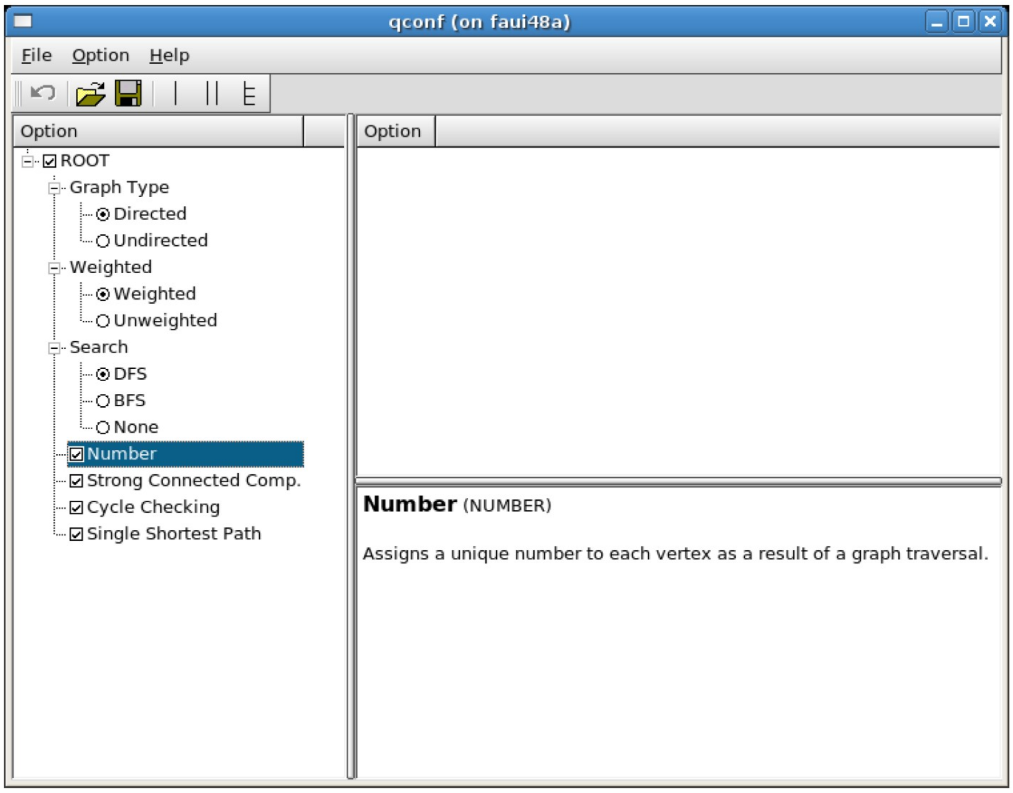
Relationships

Add new link: cardinality: * to target:

requires	*	Component
requires	*	Property
contributesTo	*	Component
contributesTo	*	Resource

Tabular

[S28]


qconf (on faui48a)

File Option Help

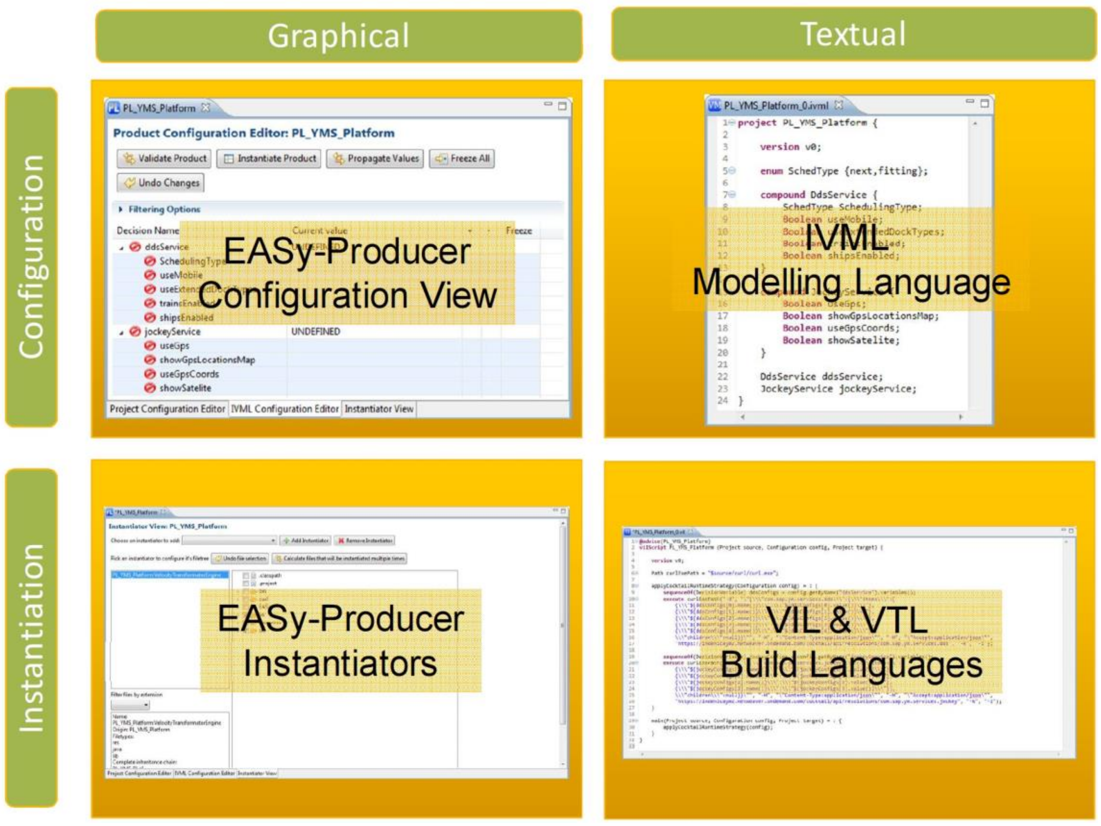
```
Option
[✓] ROOT
    Graph Type
        (●) Directed
        ( ) Undirected
    Weighted
        (●) Weighted
        ( ) Unweighted
    Search
        (●) DFS
        ( ) BFS
        ( ) None
    [✓] Number
    [✓] Strong Connected Comp.
    [✓] Cycle Checking
    [✓] Single Shortest Path
```

Option

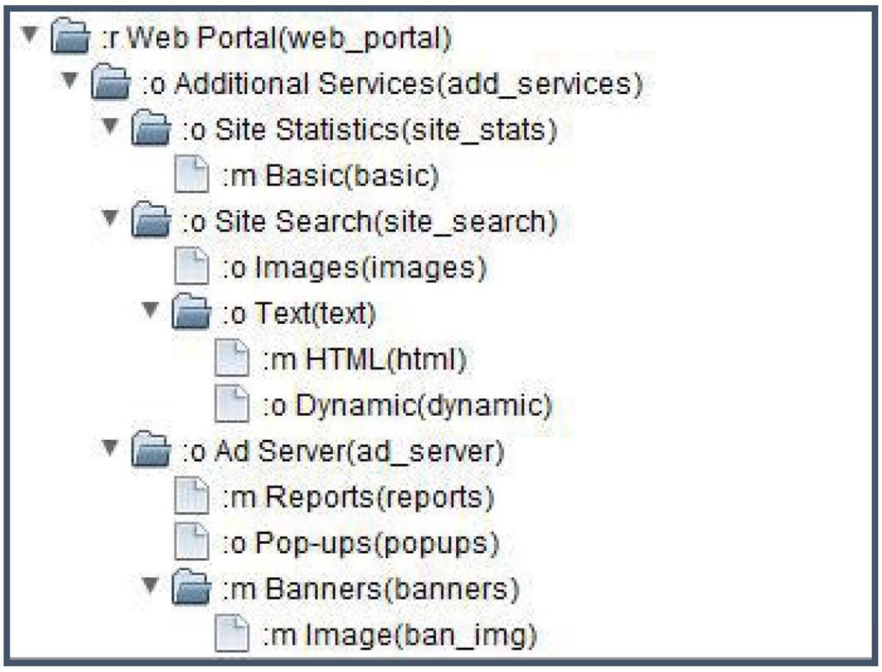
Number (NUMBER)

Assigns a unique number to each vertex as a result of a graph traversal.

[S31]

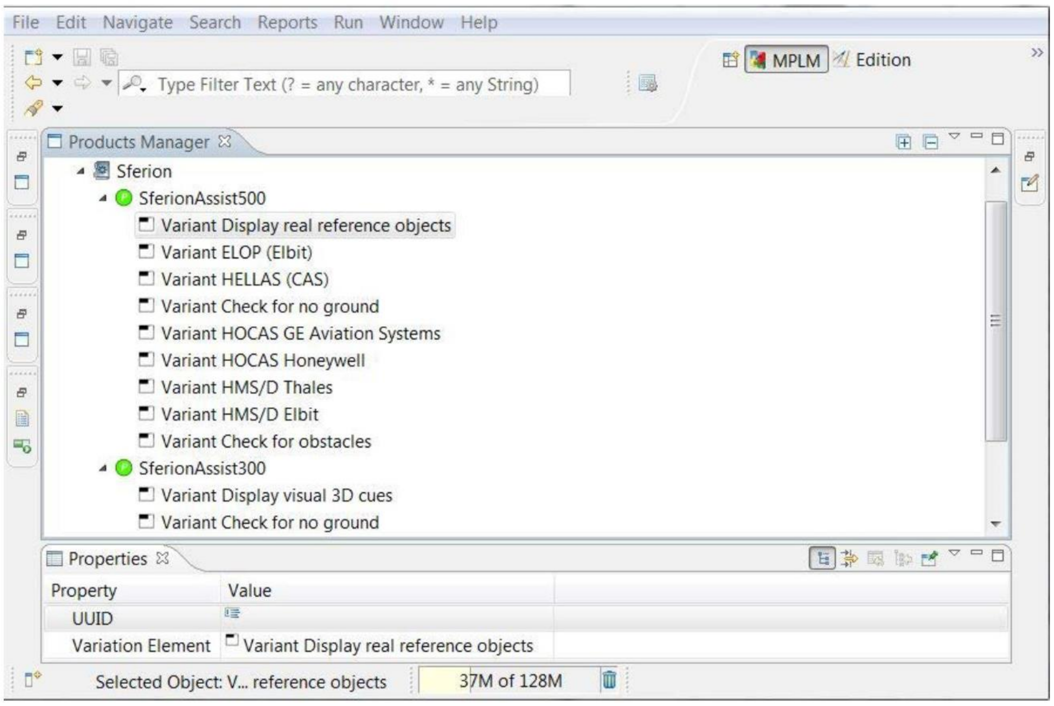

Graphical
Textual
Configuration
Instantiation
EASy-Producer Configuration View
IVML Modelling Language
EASy-Producer Instantiators
VIL & VTL Build Languages
Product Configuration Editor: PL_YMS_Platform
Validate Product
Instantiate Product
Propagate Values
Freeze All
Undo Changes
Project Configuration Editor
IVML Configuration Editor
Instantiator View

[S32]



- :r Web Portal(web_portal)
  - :o Additional Services(add_services)
    - :o Site Statistics(site_stats)
      - :m Basic(basic)
    - :o Site Search(site_search)
      - :o Images(images)
      - :o Text(text)
        - :m HTML(html)
        - :o Dynamic(dynamic)
    - :o Ad Server(ad_server)
      - :m Reports(reports)
      - :o Pop-ups(popups)
      - :m Banners(banners)
        - :m Image(ban_img)

[S33]


File Edit Navigate Search Reports Run Window Help
Type Filter Text (? = any character, * = any String)
MPLM
Edition
Products Manager
Properties

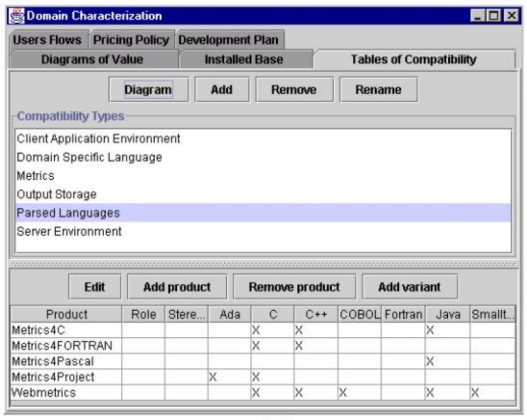
- Sferion
  - SferionAssist500
    - Variant Display real reference objects
    - Variant ELOP (Elbit)
    - Variant HELLAS (CAS)
    - Variant Check for no ground
    - Variant HOCAS GE Aviation Systems
    - Variant HOCAS Honeywell
    - Variant HMS/D Thales
    - Variant HMS/D Elbit
    - Variant Check for obstacles
  - SferionAssist300
    - Variant Display visual 3D cues
    - Variant Check for no ground

Property	Value
UUID	
Variation Element	Variant Display real reference objects

Selected Object: V... reference objects   37M of 128M

Table 4.4: Tools with Graph, Logic Diagrams, UML and Hyperbolic Tree visualisations

[S2]

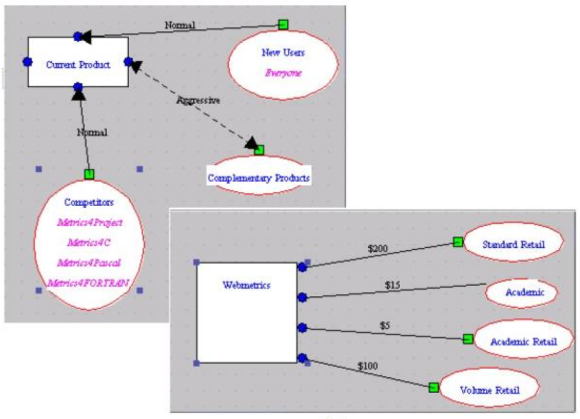


Compatibility Types

- Client Application Environment
- Domain Specific Language
- Metrics
- Output Storage
- Parsed Languages
- Server Environment

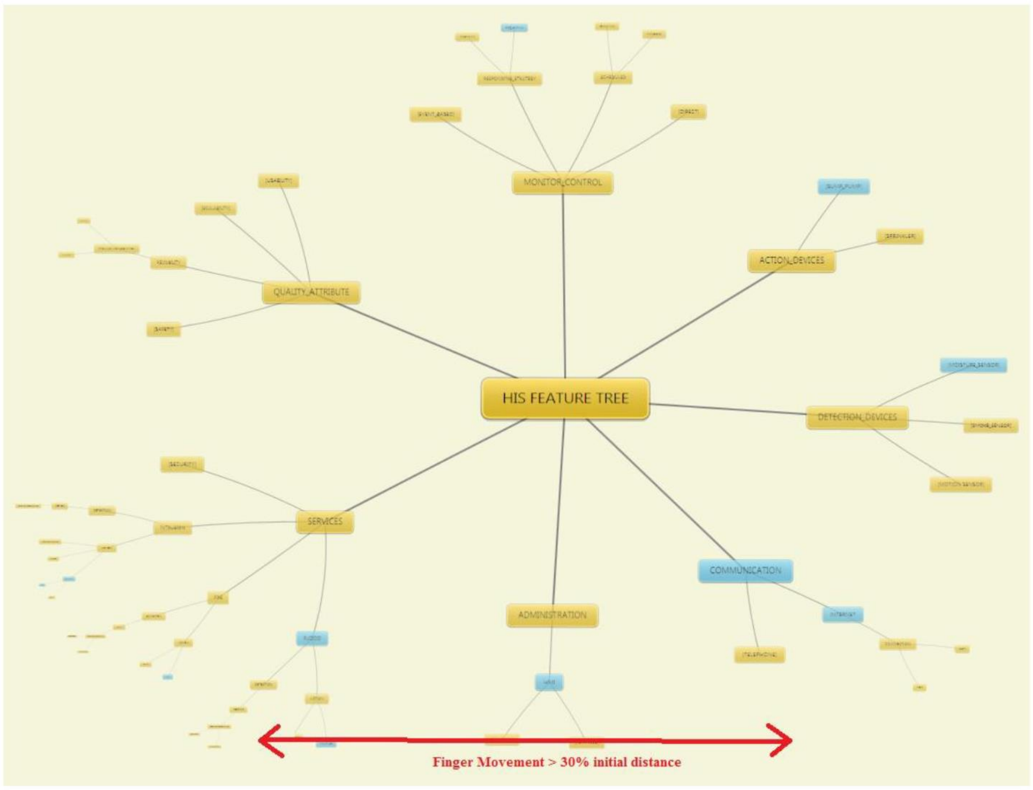
Product	Role	Stere...	Ada	C	C++	COBOL	Fortran	Java	Smallt...
Metrics4C				X	X			X	
Metrics4FORTRAN				X	X				
Metrics4Pascal								X	
Metrics4Project			X	X					
Webmetrics				X	X	X		X	X

(a)

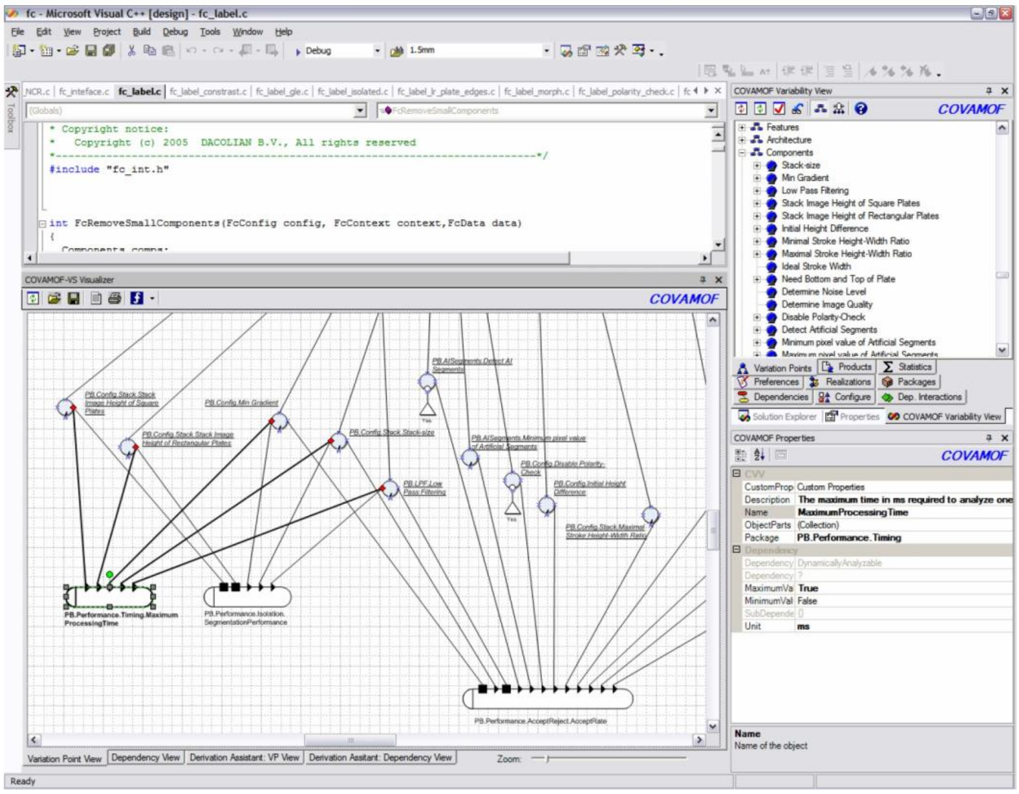


(b)

[S11]



[S8]


fc - Microsoft Visual C++ [design] - fc_label.c
File Edit View Project Build Debug Tools Window Help
Debug
1.5mm
NCR.c | fc_inteface.c | fc_label.c | fc_label_constrast.c | fc_label_gle.c | fc_label_isolated.c | fc_label_lr_plate_edges.c | fc_label_morph.c | fc_label_polarity_check.c | fc
(Globals)
FcRemoveSmallComponents
Toolbox

```
* Copyright notice:
*   Copyright (c) 2005  DACOLIAN B.V., All rights reserved
*--------------------------------------------------------------------------*/
#include "fc_int.h"


int FcRemoveSmallComponents(FcConfig config, FcContext context,FcData data)
{
  Components comps;
```

COVAMOF-VS Visualizer
COVAMOF
PB.Config.Stack.Stack Image Height of Square Plates
PB.Config.Min Gradient
PB.Config.Stack.Stack Image Height of Rectangular Plates
PB.Config.Stack.Stack-size
PB.AISegments.Detect AI Segments
PB.AISegments.Minimum pixel value of Artificial Segments
PB.Config.Disable Polarity-Check
PB.LPF.Low Pass Filtering
PB.Config.Initial Height Difference
PB.Config.Stack.Maximal Stroke Height-Width Ratio
PB.Performance.Timing.Maximum ProcessingTime
PB.Performance.Isolation. SegmentationPerformance
PB.Performance.AcceptReject.AcceptRate
Variation Point View | Dependency View | Derivation Assistant: VP View | Derivation Assitant: Dependency View
Zoom:
Ready
COVAMOF Variability View
COVAMOF

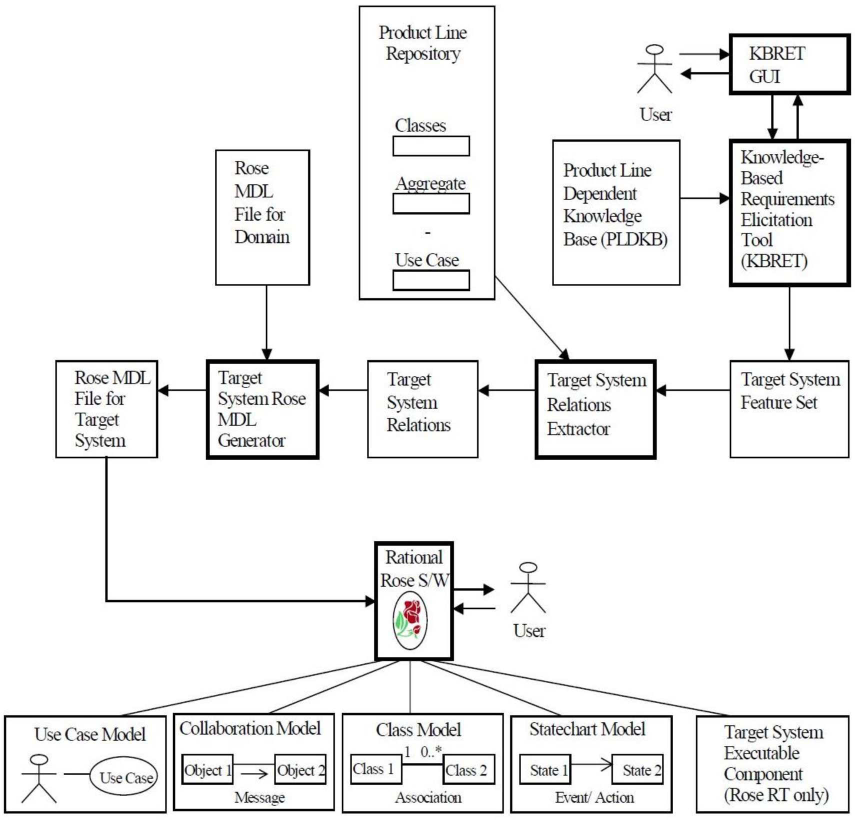
- Features
- Architecture
- Components
  - Stack-size
  - Min Gradient
  - Low Pass Filtering
  - Stack Image Height of Square Plates
  - Stack Image Height of Rectangular Plates
  - Initial Height Difference
  - Minimal Stroke Height-Width Ratio
  - Maximal Stroke Height-Width Ratio
  - Ideal Stroke Width
  - Need Bottom and Top of Plate
  - Determine Noise Level
  - Determine Image Quality
  - Disable Polarity-Check
  - Detect Artificial Segments
  - Minimum pixel value of Artificial Segments
  - Maximum pixel value of Artificial Segments

Variation Points | Products | Statistics
Preferences | Realizations | Packages
Dependencies | Configure | Dep. Interactions
Solution Explorer | Properties | COVAMOF Variability View
COVAMOF Properties
COVAMOF

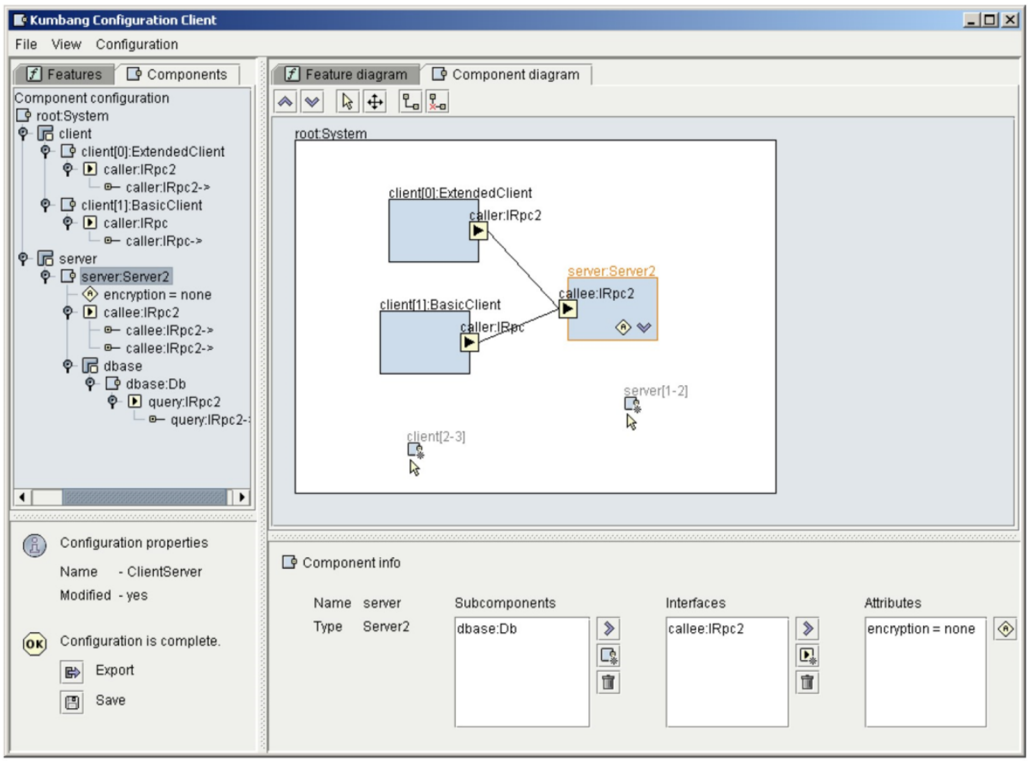
CVV	
CustomProp	Custom Properties
Description	The maximum time in ms required to analyze one
Name	MaximumProcessingTime
ObjectParts	(Collection)
Package	PB.Performance.Timing
Dependency	
Dependency	DynamicallyAnalyzable
Dependency	?
MaximumVa	True
MinimumVal	False
SubDepende	()
Unit	ms

Name
Name of the object

[S5]



[S18]



```
Component configuration
root:System
  client
    client[0]:ExtendedClient
      caller:IRpc2
        caller:IRpc2->
    client[1]:BasicClient
      caller:IRpc
        caller:IRpc->
  server
    server:Server2
      encryption = none
      callee:IRpc2
        callee:IRpc2->
        callee:IRpc2->
      dbase
        dbase:Db
          query:IRpc2
            query:IRpc2-
```

Configuration properties

Name - ClientServer

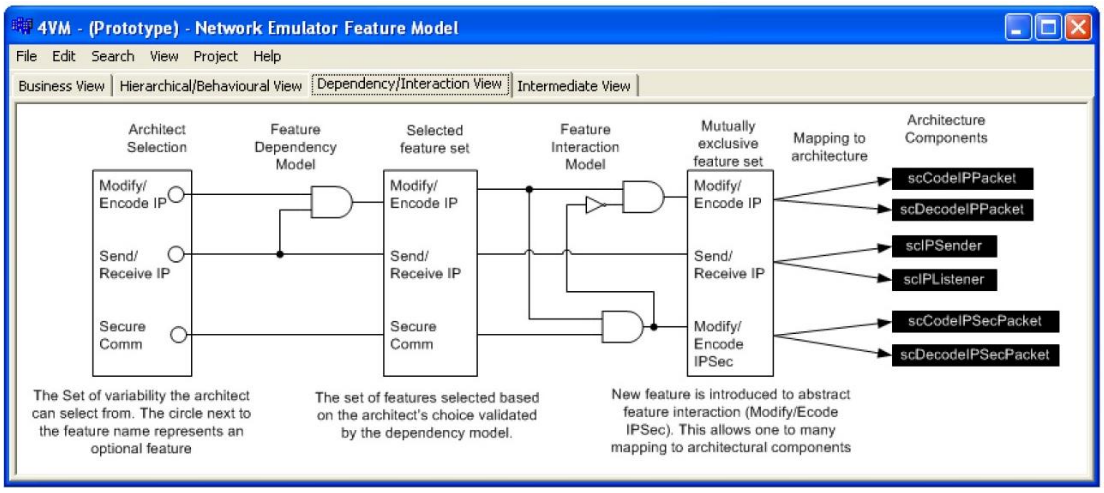
Modified - yes

Configuration is complete.

Component info

Name	server	Subcomponents	Interfaces	Attributes
Type	Server2	dbase:Db	callee:IRpc2	encryption = none

[S11]



The Set of variability the architect can select from. The circle next to the feature name represents an optional feature

The set of features selected based on the architect's choice validated by the dependency model.

New feature is introduced to abstract feature interaction (Modify/Ecode IPSec). This allows one to many mapping to architectural components

Appendix C

This appendix presents the screen-shots of an experimental implementation that was discussed in Chapter 8, using the four different case studies, and a number of variability management tools which are selected, based on certain criteria as described in Chapter 7. The experimentation mainly focused on determining how those tools addressed the four quality attributes: Usability, Performance, Scalability, and Integration.

The screen-shots are, therefore, exposed how these tools addressed the case studies of various sizes (small, medium, and large) and data elements. For instance, tools in (Figure A1, Figure A2, Figure A3, Figure A4, Figure A8, and Figure A9) accommodated up to 100 features, which is the medium scale size variability model. This indicates that majority of these tools suffer from scalability issues, when the models start to have around hundreds features. However, other tools, such Odyssey and PLUM (Figure A6 and Figure A7), failed to effectively support the small-scale model that contained 50 features. On the other hand, for the large scale model, which is the peak point for determining scalability in the experimentation, only one tool, MUSA, was capable of accommodating more than 1000 features.

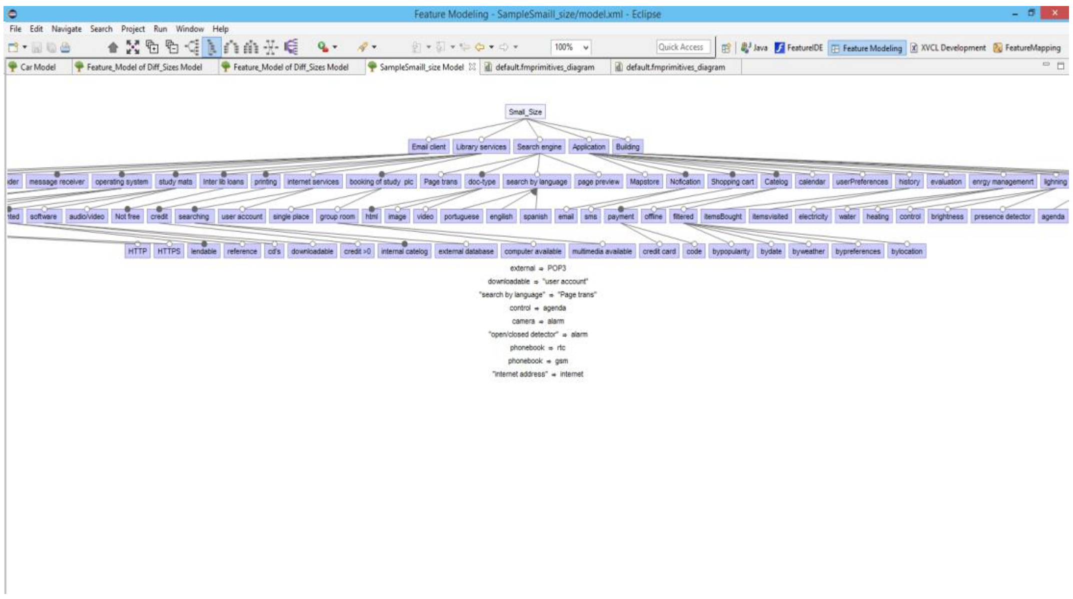

Feature Modeling - SampleSmaill_size/model.xml - Eclipse
Small_Size
Email client
Library services
Search engine
Application
Building
external ⇒ POP3
downloadable ⇒ "user account"
"search by language" ⇒ "Page trans"
control ⇒ agenda
camera ⇒ alarm
"open/closed detector" ⇒ alarm
phonebook ⇒ rtc
phonebook ⇒ gsm
"internet address" ⇒ internet

Figure A1: FeatureIDE

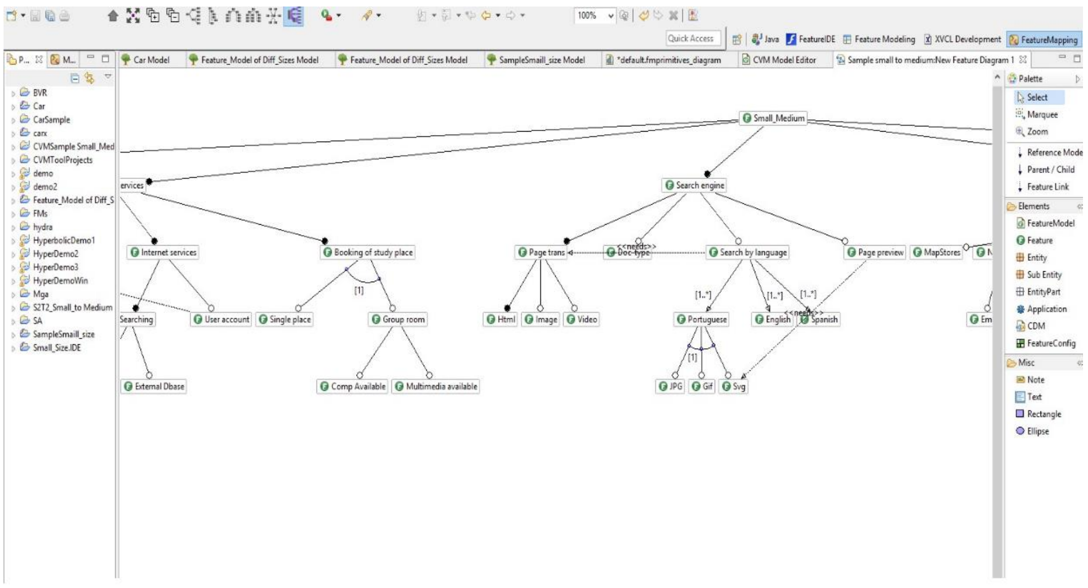


Figure A2: FAMILIAR

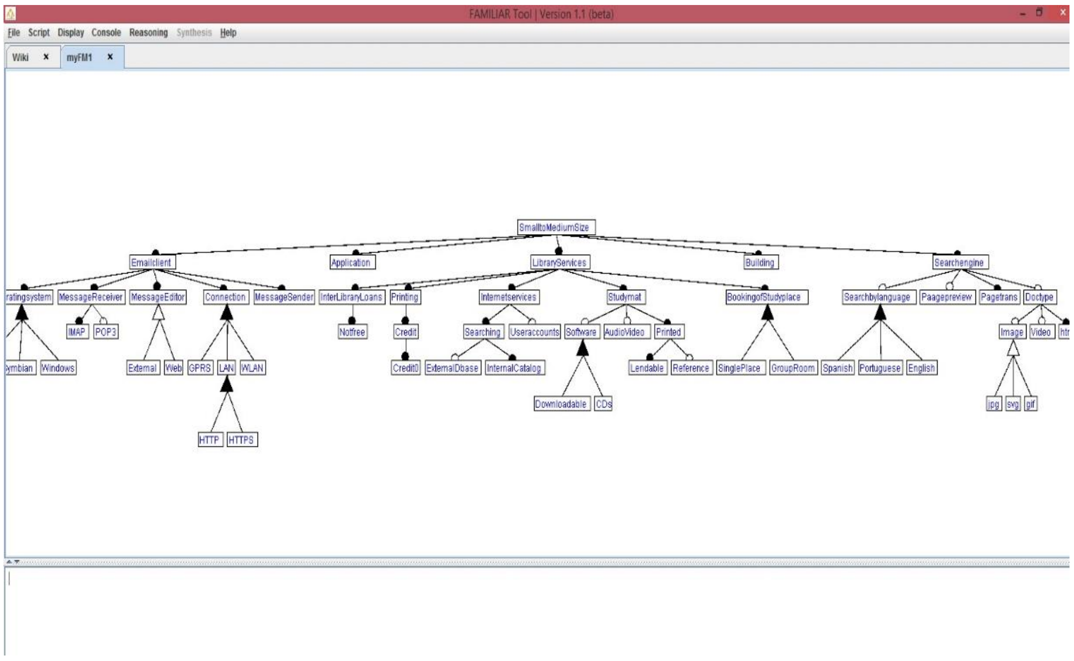


Figure A3: CVM Tool

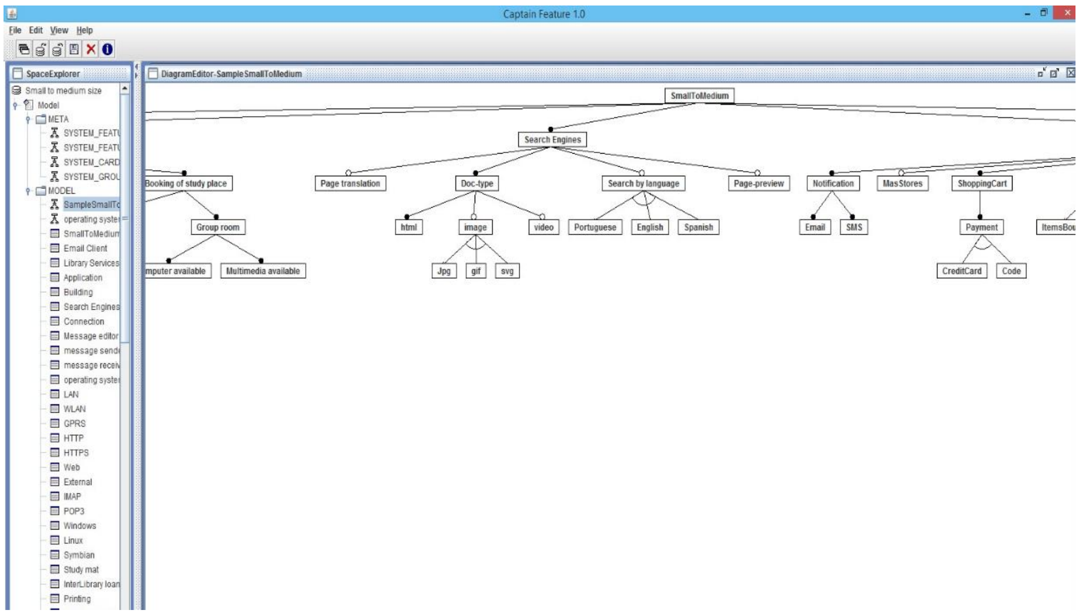


Figure A4: CaptainFeature

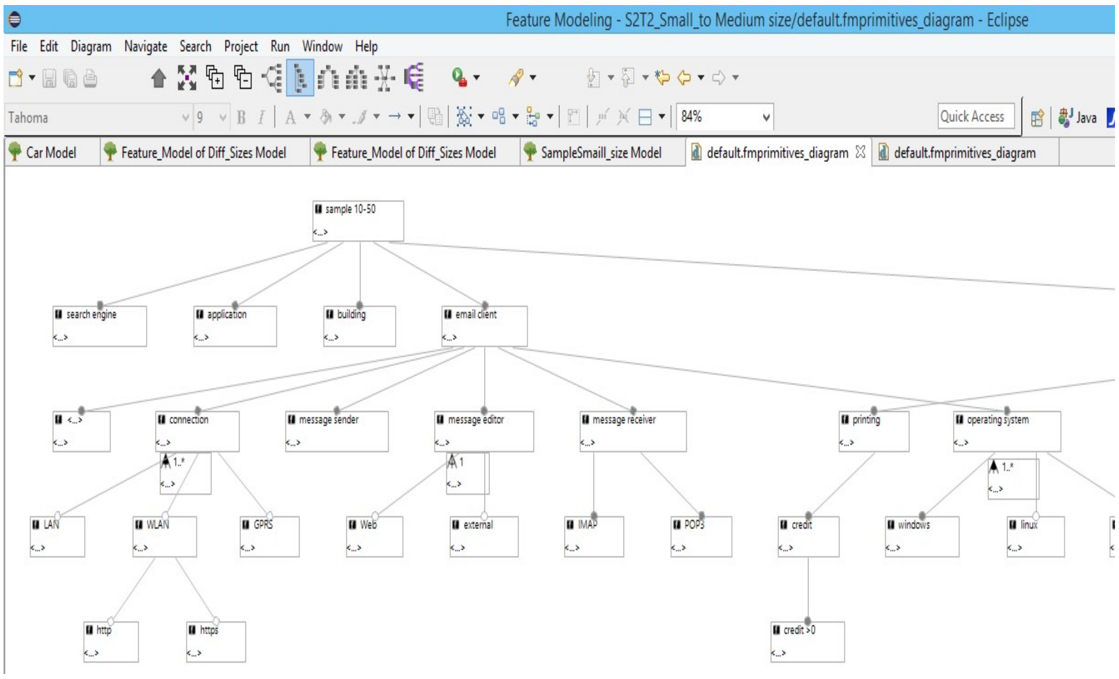


Figure A5: S2T2

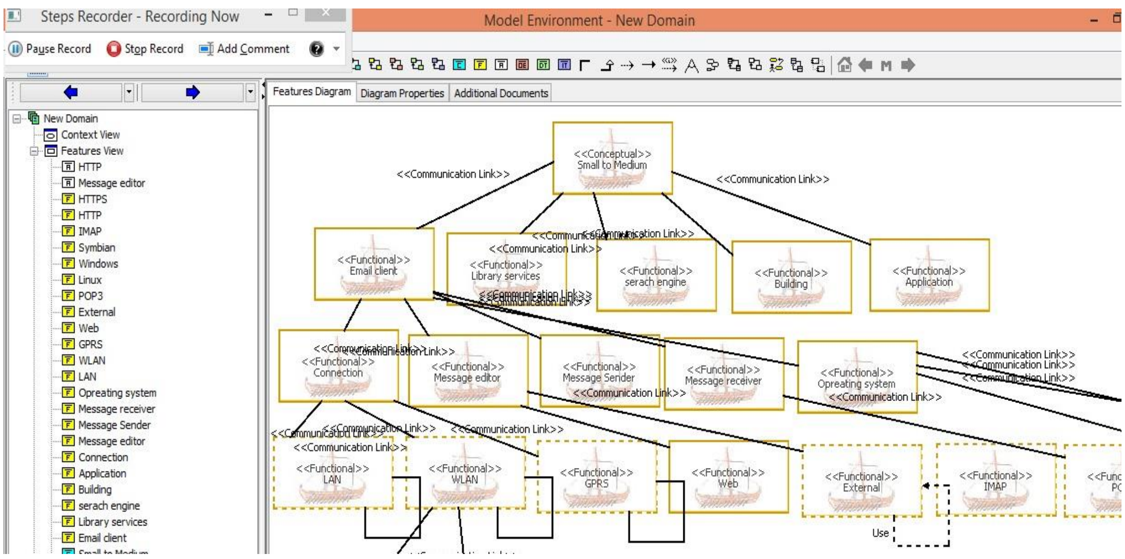


Figure A6: Odyssey

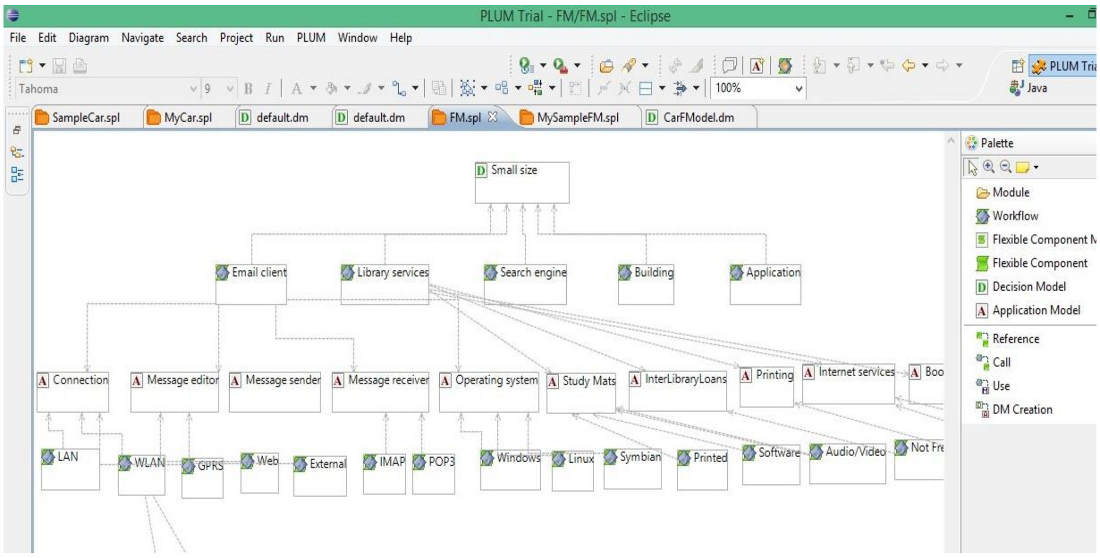


Figure A7: PLUM

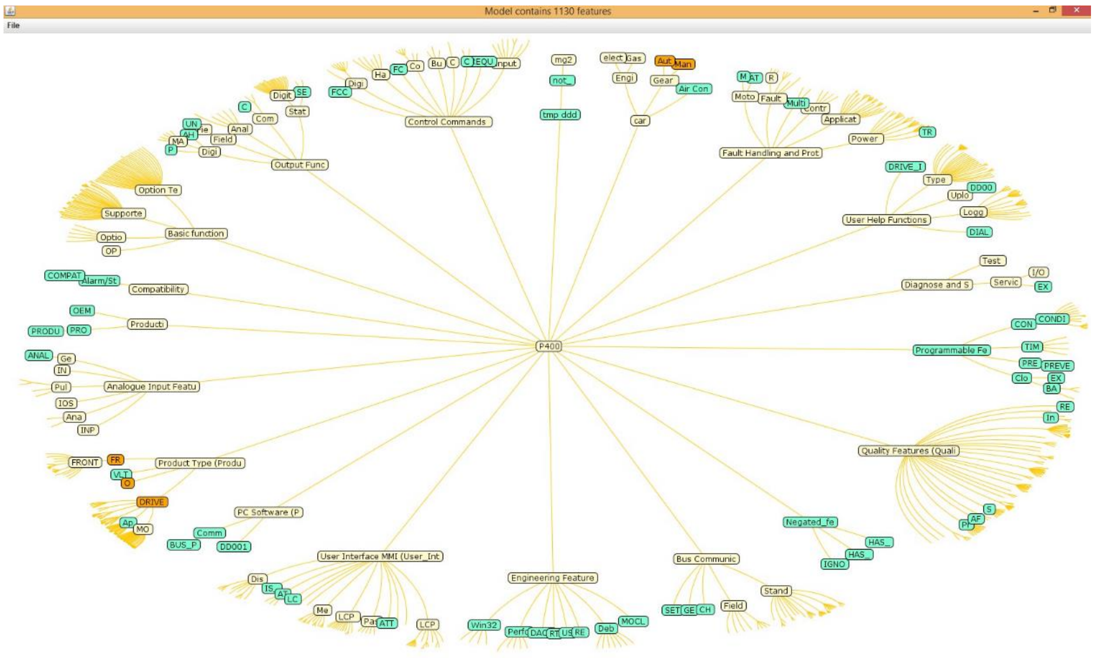
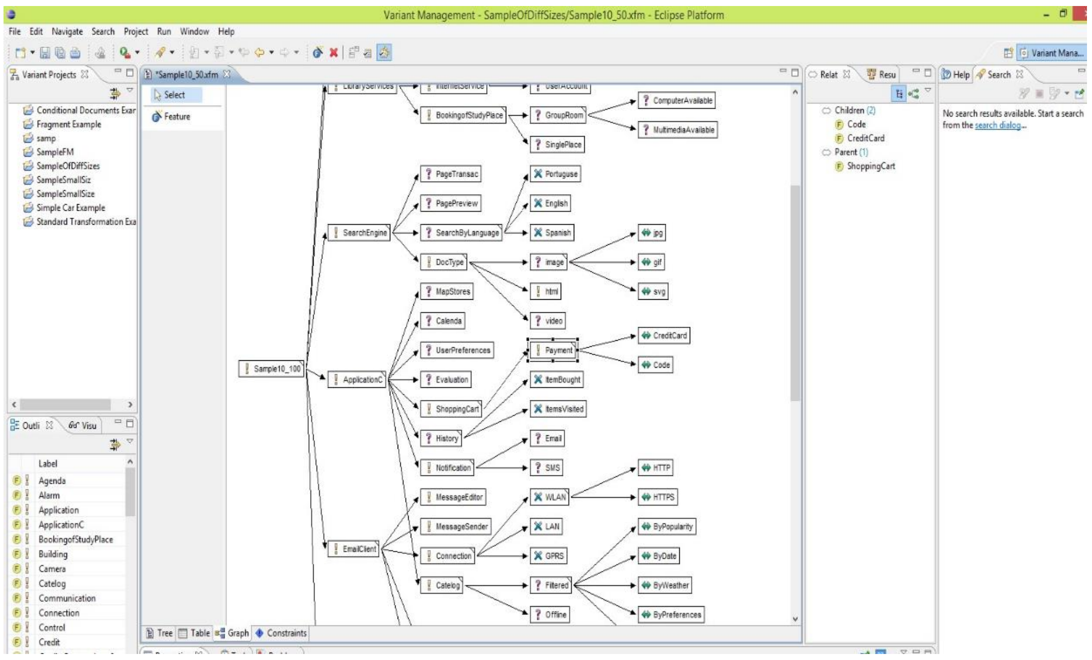


Figure A8: MUSA



No search results available. Start a search from the search dialog...

Figure A9: Pure::variants