

Automated Analysis of Software Product Lines with Orthogonal Variability Models

Extending the FaMa Ecosystem



Fabricia Carneiro Roos Frantz

Advisors:

Dr. David Benavides Cuevas

Dr. Antonio Ruiz Cortés



European Doctoral Dissertation

AUTOMATED ANALYSIS OF SOFTWARE PRODUCT LINES WITH ORTHOGONAL VARIABILITY MODELS



EXTENDING THE FAMA ECOSYSTEM

FABRICIA CARNEIRO ROOS FRANTZ

UNIVERSITY OF SEVILLE

EUROPEAN DOCTORAL DISSERTATION
SUPERVISED BY

DR. DAVID BENAVIDES CUEVAS

AND

DR. ANTONIO RUIZ CORTÉS



DECEMBER, 2011

First published in December 2011 by
The Department of Computer Languages and Systems
ETSI Informática
Avda. de la Reina Mercedes s/n
Sevilla, 41012. SPAIN

Copyright © MMXI Fabricia Carneiro Roos Frantz
<http://www.isa.us.es/fabricia.roos>
frfrantz@unijui.edu.br

In keeping with the traditional purpose of furthering science, education and research, it is the policy of the publisher, whenever possible, to permit non-commercial use and redistribution of the information contained in the documents whose copyright they own. You however are *not allowed* to take money for the distribution or use of these results except for a nominal charge for photocopying, sending copies, or whichever means you use redistribute them. The results in this document have been tested carefully, but they are not guaranteed for any particular purpose. The publisher or the holder of the copyright do not offer any warranties or representations, nor do they accept any liabilities with respect to them.

Classification (ACM 1998): D.2.13 [Software Engineering]: Reusable Software: Domain Engineering; D.2.4 [Software Engineering]: Software/Program Verification; D.2.9 [Software Engineering]: Management: Software quality assurance.

Support: PhD scholarship has been granted by the Evangelischer Entwicklungsdienst e.V. (EED). Additional support for research visit granted by the Andalusian Government, and for attending conferences by the European Commission (FEDER) and Spanish Government under CICYT projects Web-Factories (TIN2006-00472) and SETI (TIN2009-07366), by the Andalusian Government under projects ISABEL (TIC-2533) and THEOS (TIC-5906), and by Evangelischer Entwicklungsdienst e.V. (EED).

Don David Benavides Cuevas y Don Antonio Ruiz Cortés, profesores Titulares del Área de Lenguajes y Sistemas Informáticos de la Universidad de Sevilla,

HACEN CONSTAR

que Doña Fabricia Carneiro Roos Frantz, Mestre en Ciencias de la Computación por la Universidade Federal de Santa Catarina, ha realizado bajo nuestra supervisión el trabajo de investigación titulado

*Automated Analysis of Software Product
Lines with Orthogonal Variability Models.
Extending the FaMa Ecosystem*

Una vez revisado, autorizamos el comienzo de los trámites para su presentación como Tesis Doctoral al tribunal que ha de juzgarlo.

Fdo. Dr. David Benavides Cuevas y Dr. Antonio Ruiz Cortés
Área de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Sevilla, Diciembre de 2011

Yo, Fabricia Carneiro Roos Frantz, con NIE número X8406986-A,

DECLARO

Ser la autora del trabajo que se presenta en la memoria de esta tesis doctoral que tiene por título:

*Automated Analysis of Software Product
Lines with Orthogonal Variability Models.
Extending the FaMa Ecosystem*

Lo cual firmo en Sevilla, Diciembre de 2011.

Fdo. Fabricia Carneiro Roos Frantz

In addition to the committee in charge of evaluating this dissertation and the two supervisors of the thesis, it has been reviewed by the following researchers:

- Dr. Benoit Baudry (INRIA, France)
- Dr. Rick Rabiser (Johannes Kepler University, Austria)
- Dr. Maurice H. ter Beek (Institute of Information Science and Technologies of the Italian National Research Council, Italy)
- Dr. Jaejoon Lee (Lancaster University, UK)

University of Seville

The committee in charge of evaluating the dissertation presented by Fabricia Carneiro Roos Frantz in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Software Engineering, hereby recommends _____ of this dissertation and awards the author the grade _____.

D. Miguel Toro Bonilla

Catedrático de Universidad

Univ. de Sevilla

Dña. Coral Calero

Catedrática de Universidad

Univ. de Castilla-La Mancha

Dña. Lidia Fuentes

Catedrática de Universidad

Univ. de Málaga

D. Juan Garbajosa

Catedrático de Escuela Universitaria

Univ. Politécnica de Madrid

D. Roberto Lopez Herrejon

Senior Researcher

Johannes Kepler University

To put record where necessary, we sign minutes in _____,
_____.

*To Rafa.
To my family.*

Contents

Acknowledgements	xi
Abstract	xiii
Resumen	xv
Resumo	xvii
Abstrakt	xix

I Preface

1 Introduction	3
1.1 Research context	4
1.1.1 Software product lines	4
1.1.2 Variability models	6
1.1.3 Automated analysis of variability models	8
1.1.4 Automated analysis of attribute-aware variability models ..	9
1.2 Contributions	10
1.2.1 Summary of contributions	12
1.2.2 Developed tool	14
1.2.3 Potential benefits	15
1.3 Research visits and collaborations	17
1.4 Structure of this dissertation	18

II Background Information

2	Variability Models	23
2.1	Introduction	24
2.2	Feature models	24
2.2.1	Basic feature models	25
2.2.2	Cardinality-based feature models	26
2.2.3	Extended feature models	28
2.3	Orthogonal variability models	29
2.4	Summary	33
3	Automated Analysis of Feature Models	35
3.1	Introduction	36
3.2	Analysis operations on feature models	37
3.2.1	Input and output parameters	38
3.2.2	Operations overview	39
3.3	Automated support for the analysis of feature models	42
3.3.1	Constraint programming	43
3.3.2	Propositional logic	45
3.4	Automated analysis of feature models with abstract features	48
3.5	Summary	49

III Our Contribution

4	Motivation	53
4.1	Introduction	54
4.2	Problems	56
4.3	Analysis of current solutions	58
4.3.1	Modelling Concepts	58
4.3.2	Automated analysis of OVMs	60
4.3.3	Interoperability between OVM and feature model tools	62
4.4	Discussion	63
4.5	Summary	66
5	Automated Analysis of OVMs	67

5.1	Introduction	68
5.2	Dealing with abstract elements	68
5.3	Mapping OVM into Constraint Satisfaction Problem (CSP)	71
5.3.1	Full mapping	71
5.3.2	Selective mapping	73
5.4	Analysis operations for the full mapping	79
5.4.1	Variations	82
5.4.2	Number of variations	83
5.4.3	Filter	83
5.4.4	Void OVM	84
5.4.5	Valid configuration	85
5.4.6	Valid variation	86
5.4.7	Dead elements	87
5.4.8	False optional elements	87
5.4.9	Commonality degree	88
5.4.10	Refactoring	89
5.5	Analysis operations for the selective mapping	90
5.5.1	Number of variations and all variations	90
5.5.2	Void OVM	91
5.5.3	Dead and false optional	92
5.5.4	Commonality degree	93
5.5.5	Refactoring	95
5.6	Summary	96
6	Automated Analysis of Attribute-aware OVMs	97
6.1	Introduction	98
6.2	Attribute-aware OVM	98
6.3	Attribute-based Model	102
6.3.1	Attributes	103
6.3.2	Domain constraints	106
6.4	The automated analysis process	108
6.5	Mapping Attribute-aware OVM into CSP	110
6.6	Analysis operations on Attribute-aware OVMs	113
6.6.1	Operations for detecting anomalies	113
6.6.2	Valid attribute condition	114
6.6.3	Optimal variation	117

6.7	Summary	119
7	Evaluating the approach with FaMa-OVM	121
7.1	Introduction	122
7.2	Radio Frequency Warner (RFW) product line: a study of a case	122
7.2.1	System overview	122
7.2.2	System components	124
7.3	Specifying the RFW product line using OVM	125
7.4	Expressing attributes for the RFW product line	127
7.4.1	Attributes	127
7.4.2	Domain constraints	134
7.5	Automating the analysis using FaMa-OVM	136
7.5.1	The FaMa-OVM tool	136
7.5.2	The textual format for the RFW product line	138
7.5.3	Analysis results	140
7.6	Summary	144

IV Final Remarks

8	Conclusions and Future Work	147
8.1	Conclusions	147
8.2	Discussion, limitations and extensions	149
8.3	Other future work	152

V Appendices

A	Interoperability Between OVM and FM Tools	157
B	Selective Mapping Rules	193
C	RFW Product Line Specification	201
D	Acronyms	207
	Bibliography	209

List of Figures

1.1	Illustration of mass customisation in the sunglasses industry	5
1.2	Summary of variability model notations	9
1.3	Summary of research problems	11
1.4	The FaMa ecosystem	15
1.5	A multi product line using heterogeneous variability models	16
2.1	Example of a basic feature model	26
2.2	Set relationship with group cardinality	27
2.3	Sample of feature cardinality	28
2.4	Sample of an extended feature model	28
2.5	Orthogonality of OVM (based on [90])	29
2.6	Graphical notation for OVMs	30
2.7	OVM metamodel (from [102])	31
2.8	Example of an OVM	32
3.1	Process for the automated analysis of feature models	37
3.2	A simple feature model for the mobile phone product line	38
3.3	Sample of void feature model	39
3.4	Typical cases of dead features	41
3.5	Typical cases of false optional features	41
3.6	Classification of feature model edits (from Thüm et al. [125])	43
3.7	Feature model with abstract features	48
4.1	Different views of variability of a software product line	56
5.1	Process for the automated analysis of OVMs	68
5.2	Product line component diagram with variability	69

5.3	Product component diagrams	70
5.4	OVM for a mobile phone example	73
5.5	Two different approaches for the selective mapping	75
5.6	Variation points from the mobile phone product line	76
5.7	Evolution of the mobile phone product line	76
5.8	Excerpt of the OVM for the mobile phone product line	82
5.9	Example of an OVM becoming void	85
5.10	Typical cases of dead elements in OVM	87
5.11	Typical cases of false optional elements in OVM	88
5.12	Refactoring between two OVMs	89
5.13	Sample OVM with two variation points	90
5.14	OVM with a dead element when applying the full mapping	93
5.15	OVM with dead and false optional elements	93
6.1	Excerpt of a feature model extended with attributes	99
6.2	OVM documenting variability of base models	100
6.3	(a) Attributes as a base model, (b) Attributes of base model elements	101
6.4	Attribute-aware OVM	102
6.5	Attribute-based metamodel	103
6.6	Example of basic and derived attributes	104
6.7	Example of global attributes	106
6.8	Domain constraints on attributes in the Attribute-aware OVM	107
6.9	Analysis of Attribute-aware OVM	108
6.10	Process for the automated analysis of AOVMs	109
6.11	AOVM for a mobile phone example	112
6.12	Example of void AOVM	114
6.13	Example of a dead variant in the AOVM	115
6.14	Example of a false optional variant in the AOVM	115
7.1	Functionality of the RFW	123
7.2	Overview of the RFW system	124
7.3	Excerpt of the RFW OVM	126
7.4	Attributes of Positioning system and Antenna	128
7.5	Domain constraints on RFW AOVM	135
7.6	FaMa-OVM web Site	137
7.7	FaMa-OVM extending FaMa-FW	138
7.8	RFW in FaMa-OVM textual format	139

List of Figures

vii

A.1	The feature model metamodel	158
A.2	The OVM metamodel	158
C.1	RFW OVM without excludes and requires dependencies	204

List of Tables

3.1	Mapping from a feature model to CSP	45
3.2	Mapping from a feature model to propositional logic	47
3.3	Multiple products may result in the same program variant	48
4.1	Summary of related works	64
5.1	Full mapping from an OVM to CSP	72
5.2	Full mapping from a mobile phone OVM to CSP	74
5.3	Mapping variability dependencies into CSP	78
5.4	Mapping requires constraints into CSP	79
5.5	Mapping excludes constraints into CSP	80
5.6	Selective mapping from a mobile phone OVM to CSP	81
5.7	Different set of variations depending on the approach	91
5.8	The set of variations represented by an OVM example	92
5.9	Dead elements	94
5.10	Dead and false optional elements	95
5.11	Results of commonality operation	95
5.12	Models representing two different sets of variations	96
6.1	Mapping AOVM into a CSP	111
7.1	Attributes in the RFW product line	130
7.2	Values of basic attributes when associated with variants	131
7.3	Values of derived attributes when associated with variation points	133
7.4	Equations for the values of global attributes	134
7.5	Analysis operations results	141
B.1	Mapping rules for variability dependencies	194

B.2	VP requires VP mapping rules for mandatory VPs	195
B.3	VP requires VP mapping rules for optional VPs	196
B.4	V requires VP mapping rules	197
B.5	V requires V mapping rules	197
B.6	VP excludes VP mapping rules for mandatory VPs	198
B.7	VP excludes VP mapping rules for optional VPs	199
B.8	V excludes VP mapping rules	200
B.9	V excludes V mapping rules	200
C.2	RFW domain constraints	203
C.1	RFW excludes and requires dependencies	205

Acknowledgements

I am very pleased to be able to thank all the people who have collaborated, in one way or another, to the realization of this work. I am aware that it is impossible to name all of them here, since during these years of hard work I have received many comments, support, and advices from several colleagues, professors, friends and from my family.

First of all, I want to deeply thank my supervisors, Dr. David Benavides and Dr. Antonio Ruiz Cortés, whose help, advice and supervision was invaluable. I am very grateful to them for their confidence and words of encouragement, which are very important in such a long-term research work.

During the research period, I have also had the pleasure to be part of the Applied Software Engineering (ISA) research group. I wish to thank all the members of the ISA group for their warm welcome and great help along this time. I would especially like to thank José A. Galindo for his commitment and time dedicated to our fruitful discussions.

In the same way, I would like to thank Dr. Artur Boronat and Dr. Cristina Gacek, who kindly hosted me and dedicated their time to discussions during my research visits to their Universities. I am also very thankful for the PhD scholarship granted by the Evangelischer Entwicklungsdienst (EED) and all the other support received from them, without which this project would not have been possible.

Finally, and more importantly, I have to thank my family and my husband, Rafa. I really wish to thank my mother, sister, and brother for their help and understanding during all these years of distance. Thanks to all of you for waiting patiently for this day. All this years living so far from you were not easy, but your love and encouragement kept me strong and I carried on. The “*saudade*” is huge, but I am sure it was worth. I also would like to thank my lovely mother in law and father in law for their fondness and time for

long hours of virtual conversations. Thanks Rafa, for your love, patience, and endless motivation, without your help, I may perhaps have reached the end, but I am sure that it would have been harder and much less enjoyable.

Abstract

*You will have only one opportunity
to make a first impression.*

Popular saying

Software product line engineering is a software development paradigm that aims to build a family of software products by reusing a common set of core assets. In this paradigm, variability models are central artefacts, since they document the variability amongst products in a product line. Over the past twenty years, a number of variability modelling approaches have been proposed in order to document and manage variability, such as feature modelling, decision modelling, and orthogonal variability modelling. Amongst them, feature modelling is the most popular. In this approach, feature models are used to provide a compact representation of all the products of a product line in terms of features.

The automated analysis of variability models is defined as the computer-aided extraction of information from variability models. This is an active research topic that has received the attention of many researchers during the last twenty years. Most of this research has been focused on feature models, resulting in a set of analysis operations, techniques, and tools to automate the analysis of this kind of models. The existence of other variability models is naturally leading to the need for new techniques and tools to support their automated analysis as well. Furthermore, there is a need for extending variability with attributes, so that the analysis can take into account not only variability in terms of functional features, but also in terms of attributes.

Variability models usually contain elements that are used only to structure the variability of the product line, and therefore do not have any impact on the generated models, such as requirements, design, or implementation models.

We refer to these elements as abstract elements. Most of the variability modelling languages do not provide an explicit way to express abstract elements. Furthermore, the majority of the current approaches for the automated analysis of variability models can only reason about the combinations of all the elements in the variability model, but not about those that may be relevant for the user, *i.e.*, those that have some impact on other models of the product line. Therefore, abstract elements should be made explicit in the variability models, so that the analysis that only considers relevant elements can be performed.

The Orthogonal Variability Model is a modelling language to define the variability of a software product line. It is a known standard of the product line community that interrelates the variability in base models such as requirement models, design models, component models, and test models. In this dissertation, we provide a set of techniques and tools to support the automated analysis of Orthogonal Variability Models. An important strength of our contribution lies in the fact that we provide support for dealing with attributes and abstract elements. First, we make abstract elements explicit in the orthogonal variability models, and then we provide two techniques to automate the analysis of such models, one omitting abstract elements and other considering all the elements of the model. Second, we provide a technique to enrich orthogonal variability models with attributes and to automate their analysis.

Our contributions have been integrated into a tool that is built as part of the FaMa ecosystem, which is a framework for the analysis of variability models developed by our research group. In order to demonstrate the effectiveness of our techniques and analysis tool we present an evaluation using a product line in the automotive domain, which was created in a German project by a leading car company. Such evaluation allowed the detection of false optional and dead elements in the orthogonal variability model that represents the variability of such product line, and the verification of attribute conditions as well.

Resumen

*Sólo tendrás una oportunidad
de dar una primera impresión.*

Dicho popular

La ingeniería de líneas de producto software es un paradigma de desarrollo de software que permite la creación de una familia de productos software por medio de la reutilización de un conjunto común de activos software. En este paradigma, los modelos de variabilidad son artefactos centrales. Dichos modelos documentan la variabilidad entre los distintos productos de una línea de productos. En los últimos veinte años, un conjunto de técnicas para el modelado de la variabilidad se han propuesto con el fin de documentar y gestionar la variabilidad, tales como el modelado de características, el modelado de decisión y el modelado ortogonal de variabilidad. La más popular es la del modelado de características. En esta técnica, se usan modelos de características para representar de forma compacta todos los productos de una línea de productos en términos de características.

El análisis automático de modelos de variabilidad se define como la extracción de información de los modelos de variabilidad asistida por ordenador. Esa es un área de investigación activa que ha recibido la atención de los investigadores durante los últimos veinte años. Gran parte de esa investigación ha ido enfocada a los modelos de características, resultando en un conjunto de operaciones de análisis, de técnicas y de herramientas para el análisis automático de ese tipo de modelos. Con la aparición de otros modelos de variabilidad, se ha detectado la necesidad de proporcionar nuevas técnicas y herramientas para dar soporte al análisis automático de dichos modelos. Además, existe la necesidad de extender la variabilidad con atributos, de manera que el análisis no solamente lleve en cuenta la variabilidad en términos de las características funcionales, sino también en términos de atributos.

Los modelos de variabilidad por lo general contienen elementos que se utilizan sólo para estructurar la variabilidad de la línea de productos, y por lo tanto no tienen ningún impacto en los modelos que se generan, tales como los modelos de requisitos, diseño o implementación. Estos elementos se conocen como elementos abstractos. La mayoría de los lenguajes de modelado de variabilidad no proporcionan una forma explícita de expresar los elementos abstractos. Además, la mayoría de los enfoques actuales para el análisis automatizado de los modelos de la variabilidad sólo pueden razonar acerca de las combinaciones de todos los elementos en el modelo de variabilidad, pero no sobre los que pueden ser relevantes para el usuario, es decir, aquellos elementos que tienen algún impacto en otros modelos de la línea de productos. Por lo tanto, los elementos abstractos deben ser expresados explícitamente en los modelos de variabilidad, por lo que se pueda analizar modelos de variabilidad teniendo en cuenta únicamente los elementos pertinentes.

El modelo de variabilidad ortogonal es un lenguaje de modelado para definir la variabilidad de una línea de productos de software. Se trata de una notación usual en la comunidad de línea de productos que interrelaciona la variabilidad en los modelos base, tal como los modelos de requisitos, diseño, componentes y prueba. En esta tesis doctoral, se presenta un conjunto de técnicas y herramientas para dar soporte al análisis automático de los modelos de variabilidad ortogonales. Una importante ventaja de nuestra contribución se basa en el soporte a los atributos y a los elementos abstractos. En primer lugar, se hacen explícitos los elementos abstractos en los modelos de variabilidad ortogonal, y se proporcionan dos técnicas para automatizar el análisis de estos modelos, una en la que se omiten los elementos abstractos, y otra en la que se tienen en cuenta todos los elementos del modelo. En segundo lugar, se proporciona una técnica para enriquecer los modelos ortogonales de variabilidad con atributos y se automatiza su análisis.

Nuestras contribuciones han sido integradas en una herramienta que se ha construido como parte del ecosistema de FaMa, que es un marco para el análisis de los modelos de la variabilidad desarrollada por nuestro grupo de investigación. Con el fin de demostrar la eficacia de nuestras técnicas y de nuestra herramienta de análisis se presenta una evaluación usando un caso desarrollado en la industria alemana de automóviles. Dicha evaluación ha sido útil para detectar elementos opcionales falsos y elementos muertos en el modelo de variabilidad ortogonal de dicha línea de producto y también la verificación de restricciones sobre los atributos de este modelo.

Resumo

*Só terás uma oportunidade
de causar uma primeira impressão.*

Provérbio popular



engenharia de linhas de produtos de software é um paradigma de desenvolvimento de software que permite a criação de uma família de produtos de software através da reutilização de um conjunto comum de ativos principais. Neste paradigma, os modelos de variabilidade são artefatos centrais, uma vez que documentam a variabilidade entre diferentes produtos de uma linha de produtos. Nos últimos 20 anos, um conjunto de técnicas para a modelagem da variabilidade têm sido propostas a fim de documentar e gerenciar a variabilidade, tais como a modelagem de características, a modelagem de decisões e a modelagem ortogonal da variabilidade. A mais conhecida é a modelagem de características. Nesta técnica, os modelos de características representam de forma compacta todos os produtos de uma linha de produtos em termos de características.

A análise automática de modelos de variabilidade é definida como a extração de informações dos modelos de variabilidade assistida por computador. Essa é uma área de pesquisa ativa que tem recebido atenção de pesquisadores ao longo dos últimos 20 anos. Grande parte desta pesquisa tem sido centrada em modelos de características, o que resultou em um conjunto de operações de análise, técnicas e ferramentas para a análise automática de tais modelos. Com o surgimento de outros modelos de variabilidade, identificou-se a necessidade de proporcionar novas técnicas e ferramentas de apoio para análise automática de tais modelos. Além disso, há uma necessidade de estender a variabilidade com atributos, de modo que a análise não só tenha em conta a variabilidade em termos de características funcionais, mas também em termos de atributos.

Geralmente os modelos de variabilidade contêm elementos que são usados apenas para estruturar a variabilidade de uma linha de produtos, e que, portanto, não têm impacto sobre os modelos gerados, tais como modelos de requisitos, projeto ou implementação. Estes elementos são conhecidos como elementos abstratos. A maioria das linguagens de modelagem da variabilidade não proporcionam uma forma para expressar explicitamente elementos abstratos. Além disso, a maioria das abordagens atuais para análise automática de modelos de variabilidade só raciocinam sobre a combinação de todos os elementos do modelo, mas não sobre a combinação daqueles que podem ser relevantes para o usuário, ou seja, aqueles elementos que têm um impacto sobre outros modelos da linha de produtos. Portanto, os elementos abstratos nos modelos de variabilidade devem ser expressados explicitamente, para que de esta forma seja possível fazer a análise de modelos de variabilidade levando em conta apenas os elementos relevantes.

O modelo de variabilidade ortogonal é uma linguagem de modelagem para definir a variabilidade em linhas de produtos de software. Trata-se de uma notação conhecida na comunidade de linhas de produtos que interrelaciona a variabilidade dos modelos base, tais como modelos de requisitos, projeto, componentes e teste. Nesta tese, apresentamos um conjunto de técnicas e ferramentas para dar suporte a análise automática de modelos de variabilidade ortogonais. Uma vantagem de nossa contribuição é que a mesma dá suporte para elementos abstratos e atributos. Em primeiro lugar, tornamos explícitos os elementos abstratos nos modelos de variabilidade ortogonais, e, proporcionamos duas técnicas para automatizar a análise destes modelos. Um técnica omite os elementos abstratos e a outra leva em consideração todos os elementos do modelo. Logo, também proporcionamos uma técnica para enriquecer os modelos de variabilidade ortogonais com atributos e automatizamos sua análise.

Nossas contribuições foram integradas em uma ferramenta construída como parte do ecossistema FaMa, o qual é um *framework* para a análise de modelos de variabilidade, desenvolvida por nosso grupo de pesquisa. Para demonstrar a eficácia das nossas técnicas e da nossa ferramenta de análise, apresentamos uma avaliação utilizando um caso desenvolvido pela indústria automobilística alemã. Esta avaliação foi útil para detectar elementos opcionais falsos e elementos mortos no modelo de variabilidade ortogonal da linha de produtos em questão e também a verificação de restrições sobre os atributos deste modelo.

Abstrakt

*Man hat nur eine einzige Gelegenheit,
einen ersten Eindruck zu hinterlassen.*

Volksweisheit

Produktlinienentwicklung ist ein Softwareentwicklungsparadigma, das durch die Wiederverwendung eines gemeinsamen Satzes von Softwareaktiva auf eine Familie von Softwareprodukten abzielt. In diesem Paradigma sind Variabilitätsmodelle zentrale Artefakte, da sie die Variabilität innerhalb von Produkten einer Produktlinie dokumentieren. In den vergangenen zwanzig Jahren wurde eine Reihe von Ansätzen zur Variabilitätsmodellierung unterbreitet, um Variabilität wie etwa Featuremodellierung, Entscheidungsmodellierung und orthogonale Variabilitätsmodellierung zu dokumentieren und zu verwalten. Unter ihnen ist Featuremodellierung am beliebtesten. In diesem Ansatz werden Featuremodelle verwendet, um in Bezug auf die Funktionen eine kompakte Darstellung aller Produkte einer Produktlinie zu bieten.

Die automatisierte Analyse von Variabilitätsmodellen wird als computer-gestützte Informationsgewinnung aus Variabilitätsmodellen definiert. Dies ist ein dynamischer Forschungsgegenstand, der in den letzten 20 Jahren die Aufmerksamkeit vieler Forscher erregte. Der größte Teil dieser Forschung hat sich auf Featuremodelle konzentriert, was zu einer Reihe von Analyseoperationen, -techniken und -werkzeugen zur Automatisierung der Analyse dieser Art von Modellen führte. Die Existenz anderer Variabilitätsmodelle führt in natürlicher Weise zu einem Bedarf nach neuen Techniken und Werkzeugen zur Unterstützung automatisierter Analysen. Darüber hinaus besteht Bedarf für die Erweiterung der Variabilität mit Attributen, so dass die Analyse nicht nur die Variabilität in Bezug auf die funktionellen Eigenschaften, sondern auch in Bezug auf die Attribute berücksichtigen kann.

Variabilitätsmodelle enthalten in der Regel Elemente, die nur zur Strukturierung der Produktlinienvariabilität verwendet werden und somit keine Auswirkungen auf die erzeugten Modelle wie etwa auf Anforderungen, Design oder Implementierungsmodelle haben. Diese Elemente bezeichnen wir abstrakte Elemente. Die meisten der Variabilitätsmodellierungssprachen bieten keine ausdrückliche Möglichkeit, abstrakte Elemente zu bezeichnen. Darüber hinaus kann die Mehrheit der derzeitigen Ansätze für die automatisierte Analyse der Variabilitätsmodelle nur die Kombination aller Elemente im Variabilitätsmodell erörtern, nicht aber diejenigen, die für die Nutzer relevant sind, also diejenigen, die einen gewissen Einfluss auf andere Modelle der Produktlinie ausüben. Daher sollten abstrakte Elemente explizit in den Variabilitätsmodellen selber gemacht werden, so dass nur relevante Elemente beinhaltende Analysen durchgeführt werden können.

Das orthogonale Variabilitätsmodell ist eine Modellierungssprache zur Variabilitätsbestimmung einer Softwareproduktlinie. Dies ist ein bekannter Standard der Produktliniencommunity, der die Variabilität in Basismodellen wie etwa Anforderungsmodellen, Designmodellen, Komponentenmodellen und Testmodellen verknüpft. In dieser Dissertation stellen wir eine Reihe von Techniken und Werkzeugen zur Unterstützung der automatisierten Analyse von orthogonalen Variabilitätsmodellen zur Verfügung. Eine wichtige Stärke unseres Beitrags liegt darin, dass wir Unterstützung für den Umgang mit Attributen und abstrakten Elementen bieten. Zuerst erarbeiten wir abstrakte Elemente explizit in den orthogonalen Variabilitätsmodellen, daraufhin bieten wir zwei Techniken zur Automatisierung der Modellanalyse an – eines unter Verzicht auf abstrakte Elemente und ein weiteres unter Berücksichtigung aller Elemente des Modells. Zweitens bieten wir eine Technik, um orthogonale Variabilitätsmodelle mit Attributen zu bereichern und ihre Analyse zu automatisieren.

Unsere Beiträge wurden in einem Werkzeug integriert, das als Teil des Ökosystems FaMa fungiert, welches eine von unserer Arbeitsgruppe entwickelte Struktur zur Analyse von Variabilitätsmodellen ist. Um die Effektivität unserer Techniken und Analysewerkzeuge zu demonstrieren, präsentieren wir eine Auswertung der Produktlinie in der Automobildomäne, die in einem deutschen Projekt von einem führenden Automobilkonzern in Deutschland geschaffen wurde. Eine solche Auswertung erlaubte die Erkennung falscher optionaler und toter Elemente im orthogonalen Variabilitätsmodell – das die Variabilität dieser Produktlinie darstellt – sowie die Prüfung der Attributvoraussetzungen.

Part I

Preface

Chapter 1

Introduction

A journey of a thousand miles starts with a single step.

*Lao Zi, 6th–5th century B.C.
Chinese taoist Philosopher*

In this dissertation, we report on our contributions to develop a set of techniques and tools to support the automated analysis of Orthogonal Variability Models taking into account attributes. In this chapter, we first describe the topics that constitute the context of our research work in Section §1.1. In Section §1.2, we summarise our main contributions and research activities. In Section §1.3, we report on the research visits and collaborations carried out during the development of this dissertation. Finally, in Section §1.4, we present the structure of this dissertation.

1.1 Research context

In the next subsections, we concisely introduce the main concepts that will be used throughout the rest of this dissertation. In Section §1.1.1, we present software product lines. Section §1.1.2 introduces variability models. In Section §1.1.3, we introduce automated analysis of variability models, and finally, in Section §1.1.4, we introduce automated analysis of attribute-aware variability models.

1.1.1 Software product lines

In the last decade, the trend towards globalised business competition has pressured organisations to find ways to offer more diversified goods with reduced development cost and time-to-market. As it happened in earlier times, both research community and industry have believed in software reuse as a powerful means to improve productivity and quality in software development [69, 85, 98]. Software product line has emerged as one of the most promising ways of software reuse. Software product lines institutionalise systematic reuse throughout all software development phases. This software development paradigm has increased productivity of IT-related industries, reduced time-to-market, and allowed for developing more diversified goods [32, 77, 122].

The story of software product lines begins with the application of *mass customisation* to the production of software [8, 13]. In contrast to the *mass production*, where a large number of standardised products are built and then distributed to the customers, in the mass customisation paradigm products are made on-demand according to individual customer needs. Tseng and Jiao define mass customisation as “producing goods and services to meet individual customer needs with near mass production efficiency” [132]. In this definition, we notice that there is a concern to efficiently produce a large amount of products, and at the same time, meet the needs of each individual customer. Figure §1.1 depicts an example of how mass customisation is offered to customers in the sunglasses industry. Customers can choose the features that better meet their needs by selecting features in a web configurator.

In applying mass customisation to the production of software, a software development organisation that produces individual software products replaces its production of single products by the production of a family of

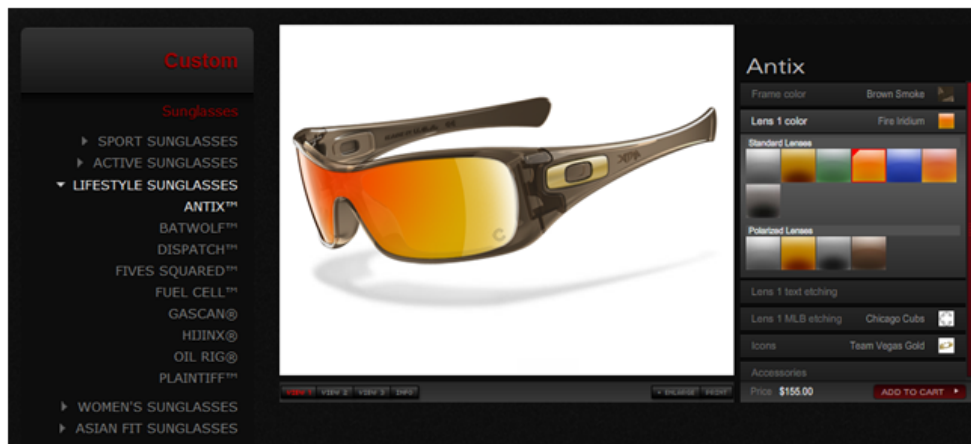


Figure 1.1: Illustration of mass customisation in the sunglasses industry.

similar products. In this way, organisations can repeatedly build similar software products, and then, customise these products according to individual customer needs by adding specific features. A software family is composed of a set of varied software products that are constructed from a set of core assets designed for a specific domain. As these products belong to the same application domain, they share more commonalities (*i.e.*, common features) than singularities. Consider, for instance, the commonalities amongst current mobile phone systems (*e.g.*, calls, ring tones, messaging, and alarm clock). One of the definitions for software product line, given by Clements and Northrop [32], is as follows:

“a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Software product line engineering usually consists of two development processes, namely: *domain engineering* and *application engineering* [102]. In domain engineering, common software artefacts are designed and developed for reuse. In application engineering, specific products are derived by reusing a set of the aforementioned domain artefacts.

Software product lines are traditionally designed and managed in an intra-organisational context. However, due to the success of its product line, a com-

pany can be interested in expanding its product line platform (*i.e.*, the core assets) outside its organisational boundaries. Once the company decides to make its platform available to the community, the company transitions from a software product line approach to a *software ecosystem* approach [24, 89]. The emerging trend of software ecosystems is motivated by multiple benefits, such as increasing the value of products for existing users and heightening attractiveness for the new ones, as well as, collaborations with partners to accelerate and share cost of innovation. However, the adoption of a software ecosystem brings new challenges, such as management of frequent platform releases, and coordination mechanisms to manage the complexity of the relationship amongst the number of parties involved (*i.e.*, external and internal developers, partner companies and independent solution vendors). Examples of software ecosystems can be found in different areas of the software industry like operating systems, web applications, and end-user programming (*e.g.*, Android, Amazon, and Microsoft PopFly). Although software ecosystems became popular with the web 2.0, they have been around for decades [24]. Some software ecosystems are derived from classical desktop applications, *e.g.* MS Office suite. This is an example of a successful application in the market place without the support of an ecosystem that is opened up to promote contributions from its user community.

1.1.2 Variability models

Developing software using a product line paradigm implies planning and building software assets for reuse, and subsequently reusing these assets. A key difference between software product line development and traditional software development is the need for modelling variability, which allows for mass customisation. Variability is a concern in all phases of the software product line development [25]. Modelling variability means the commonalities and variabilities amongst software products have to be described in order to express and understand the complexity and diversity of products in the domain of interest [115].

Variability models are central artefacts in all phases of software product line engineering. They explicitly and effectively document the commonalities and variabilities amongst software products, *i.e.*, the rules that constraint the possible combinations of reusable assets in a product. These assets include requirements, design models, test cases, etc. The derivation of an individual product is done by selecting options in the variability model.

Over the past twenty years, a number of variability modelling approaches have been proposed. Roughly speaking, they can be grouped into four main categories:

Feature modelling. It is one of the most popular and has gained the most attention of both research and industry. The first feature model was proposed by Kang et al. [71], in 1990, as part of the method Feature-Oriented Domain Analysis (FODA). Since then, several other feature-model-oriented approaches were proposed based on FODA, such as in [35, 37, 52, 64, 72, 73, 105], for a detailed description about feature model semantics we refer the reader to Schobbens et al. [116]. Feature modelling approaches usually focus on describing the product line domain. The key idea of this approach is to capture in a feature model the set of possible products of a product line.

Decision modelling. It focuses on decisions rather than domain description. Decisions were first introduced by Campbell et al. [28] as *“actions which can be taken by application engineers to resolve the variations for a work product of a system in the domain”*. Most of the existing decision modelling approaches have been influenced by the Synthesis method [33] which first introduced the idea of decision models. In this report, a decision model was defined as *“a set of requirements and engineering decisions that determine the variety of work products in the domain, and must be resolved by an application engineer to define and construct work products”*. Schmid and John [114], Forster et al. [55], Dhungana et al. [39, 42], amongst others, use decision models as variability modelling language.

Orthogonal modelling. It was introduced with the aim to explicitly document the variability of software product lines in a separate model. They provide a cross-sectional view of the variability of the product line across all software development artefacts. One-way references to the base model describe how the base model elements can vary. Bachmann et al. [2] proposed the use of orthogonal variability models to provide this separate view of the variability by documenting explicitly the variation points. The *Orthogonal Variability Model (OVM)*, proposed in [102], is the variability model we focus on throughout this dissertation. This approach proposes to document variability in a separate model, and interrelates variability on the base product line models. OVM is mainly characterised by considering variation points as first-class citizens. Another approach proposed to make variability models orthogonal to the product line models is the Common Variability Language (CVL) [54], which is a generic language to define variability on any other model defined by means of MetaObject Facility compliant metamodels, in-

cluding Unified Modelling Language (UML) and domain-specific languages. CVL is an attempt to standardise a language for variability modelling in the Object Management Group.

UML-based modelling. Variability can be modelled in traditional existing models, such as requirement models and design models. Some approaches, as those proposed by Gomaa [61], Gomaa and Shin [62], Gomaa [63], Ziadi et al. [156], propose including commonality and variability in the UML models. However, this strategy has some shortcomings, such as the mixing of variability and other modelling concepts [65, 90]; the variability is implicitly spread across the product line software artefacts.

There are other variability modelling proposals that we do not have included in those categories, as for instance COVAMOF by Sinnema et al. [120], or DebianVML by Galindo et al. [56], which provides a way to describe Debian Packages Repositories as software product line models. Figure §1.2 shows a summary of some variability model notations found in the literature.

1.1.3 Automated analysis of variability models

The configuration of a product is done by selecting desired and valid options in the variability model during application engineering. For example, in a mobile phone product line, an option could be the selection of the screen resolution, which can be basic or high. This option demands a constraint to ensure that the products support only one screen resolution simultaneously. Variability models can have thousands of options. Therefore, it was recognised that their manual treatment is a difficult and error-prone task [71]. Consequently, several contributions were done on the automated analysis of variability models, mostly focusing on feature models [13].

The *automated analysis of variability models* deals with the computer-aided extraction of information from variability models [13]. This information supports many technical, managerial, and marketing decisions throughout all phases of software product line development [8]. Typical operations of analysis allow knowing whether a variability model is void (*i.e.*, it does not represent any product), whether a given product is represented by a variability model, or whether a model contains errors. A recent survey has identified up to 30 different analysis operations on feature models and a number of analysis techniques and tools for their automation [13].

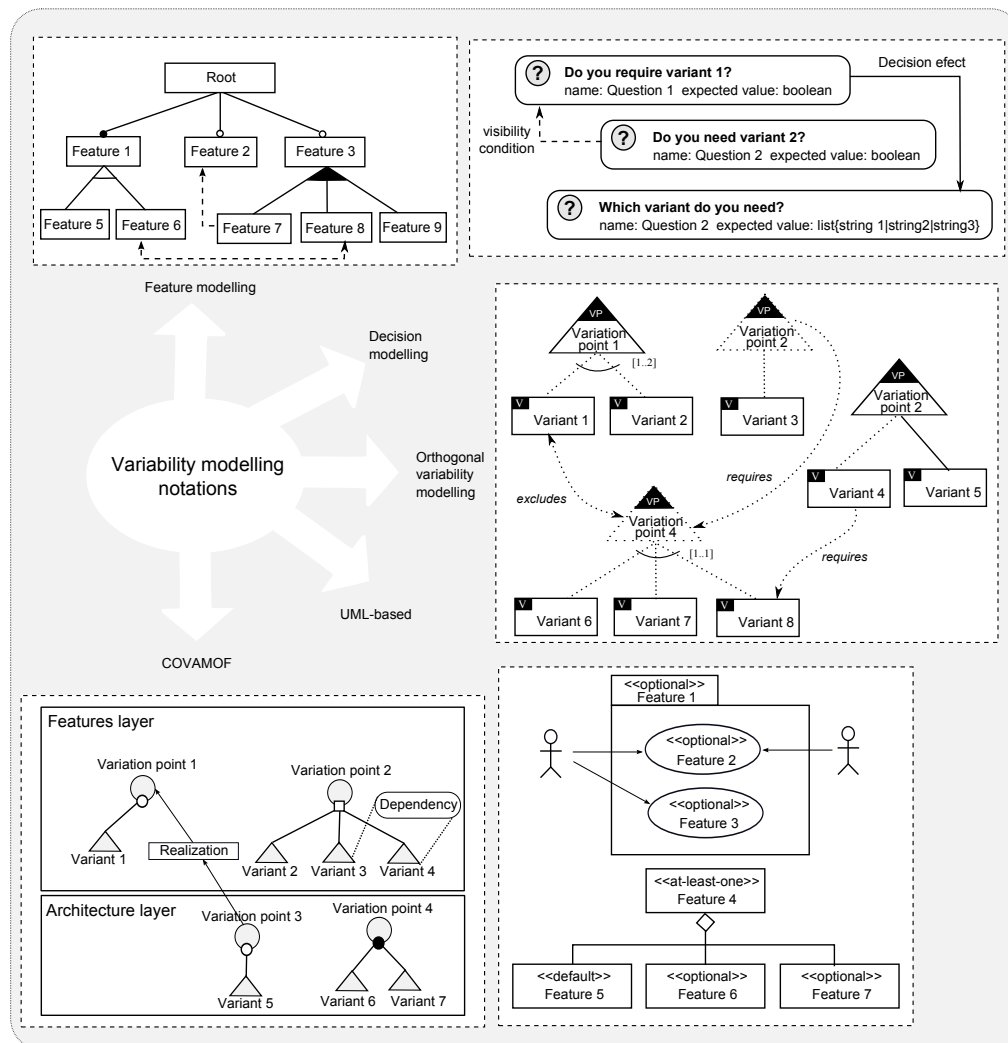


Figure 1.2: Summary of variability model notations.

1.1.4 Automated analysis of attribute-aware variability models

The specification of variability can be extended with measurable attributes (e.g., CPU and memory consumption) and constraints on these attributes (e.g., memory consumption should be in a range of values) in order to express some properties about different products [13]. For example, in cases in which there are limitations of resources such as memory capacity and CPU time, the

derivation of products that does not satisfy those conditions must be avoided. When attributes are captured and added to variability models, we refer to them as attribute-aware variability models.

The product line engineer may want to verify whether it is possible to build a product that satisfies required attributes or to verify whether there is any contradiction in the specification of attributes. This analysis is possible when having an attribute-aware variability model specification. In software product line engineering, this analysis is an essential activity to guarantee that the derived software products fulfil attribute constraints. We refer to *automated analysis of attribute-aware variability models* as the computer-aided extraction of valuable information from attribute-aware variability models.

1.2 Contributions

The main goal of this dissertation is to provide a set of techniques and tools to support the automated analysis of attribute-aware OVMs. Since we have previous experience with the analysis of feature models, we are interested in applying this knowledge to other domains by providing support for the automated analysis of other variability models, and thus, endowing the FaMa ecosystem with a new analysis component.

We have chosen OVM because it is a variability modelling language supported by an acknowledged research group of the product line community, and so far its automated analysis was hardly explored. In addition, OVM and feature model diagrams, though similar, are different in terms of their elements and structure. OVMs, unlike feature models that are composed of features and relationships amongst features, are composed of two different elements (*i.e.*, variation points and variants) and relationships amongst them; moreover, they are not hierarchically structured. OVM only documents information about the variability of product line artefacts, but not data information. Hence, relating OVM elements with attributes is currently a challenge, since these elements cannot be annotated with attributes as was done with feature models. Furthermore, given that we have a product line case from the automotive industry, in which OVM was used to document variability, by providing support for the automated analysis of OVM we would be able to apply our approach to a case, and therefore contribute with the product line community. In the pursuit of our goal we have identified some research problems to be addressed, which are summarised in Figure [§1.3](#).

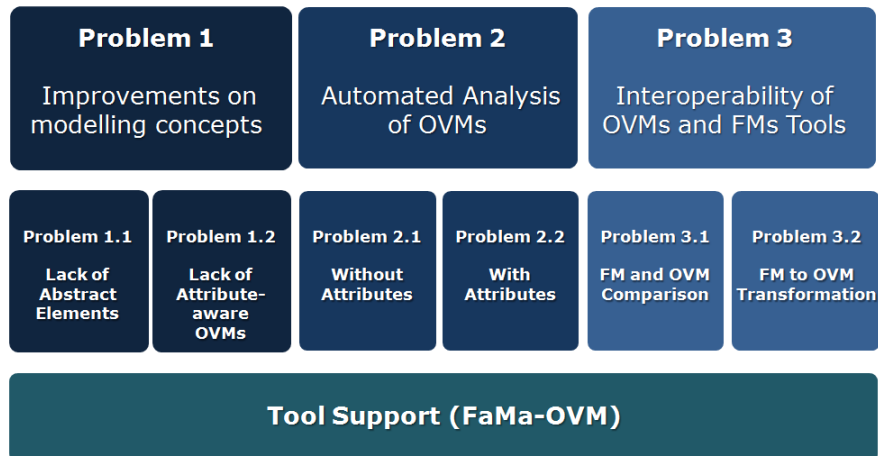


Figure 1.3: Summary of research problems.

The first problem concerns the improvements on some OVM modelling concepts that should be done. Currently, there is no way to explicitly denote abstract elements in OVM. In addition, when using OVM to represent variability in software product line development, there are no techniques to specify attributes and constraints on them.

The second problem regards the lack of a suitable support for the automated analysis of OVMS. It is practically impossible to carry out OVM analysis manually and besides, it is an error-prone task. The automated analysis of OVM has hardly been explored by the research community. In addition, the analysis of attribute-aware OVMS is an important activity to guarantee that the derived software products reach the desired attribute constraints. Due to the complexity of this analysis it is essential to rely on automated support.

The third problem refers to the lack of support for the interoperability between OVM and feature model tools. In order to understand the connection between both languages, a comparison study should be done, allowing to identify the weaknesses and strengths of these languages in a specific context or domain. In addition, in the current literature, there is no approach for the interoperability between the two languages. A first step would be to provide support for the transformation of a feature model into an OVM.

In Section §1.2.1 we summarise the main contributions we have made to solve each one of the research problems we have identified in the field of auto-

mated analysis of OVMs. In Section §1.2.2, we introduce FaMa-OVM, which is the tool we have developed to automate the analysis of attribute-aware OVMs. These contributions have been published in a relevant journal, a national conference, and national and international workshops.

1.2.1 Summary of contributions

In this dissertation, we have made the following contributions to solve the research problems presented in Section §1.2:

i. IMPROVEMENTS ON MODELLING CONCEPTS

Problem statement: Abstract elements should be made explicit in the OVM.

Contribution: We have made the abstract elements in the OVM explicit, so that we are able not only to analyse an OVM with all its elements, but also analyse it leaving out abstract elements. These contributions are presented in Sections §5.2, §5.3, and §5.5.

Problem statement: The lack of a technique to associate attributes to variability in the OVM.

Contribution: We have proposed a model to express attributes and constraints on these attributes, so that it can be related to variability in the OVM. We have identified what should be the main characteristics of this model, and defined its relationship with the variability in the OVM. These contributions are presented in Sections §6.2 and §6.3.

ii. AUTOMATED ANALYSIS OF OVMs

Problem statement: The lack of automated support for the analysis of OVMs.

Contribution: We have developed an approach and provided a tool for the automated analyses of OVMs. For this purpose, we have provided a number of analysis operations on OVMs, defined a mapping from an OVM to a constraint satisfaction problem, and used an off-the-shelf constraint solver to automate the analysis. These results are presented in Chapter §5.

Problem statement: Attribute-aware OVMS need an operational support to automate their analysis.

Contribution: We have provided support to automate the analysis of attribute-aware OVMS. For this purpose, we have proposed a mapping from an attribute-aware OVM to a constraint satisfaction problem, and provided a tool support where a specific constraint solver was used to automate the analysis. These results are presented in Chapter §6.

The main results of the aforementioned contributions were published in [110, 111], and preliminary results in [108].

iii. INTEROPERABILITY BETWEEN OVM AND FEATURE MODEL TOOLS

During our research period other research problems have arisen, for which we have contributed taking the first step towards a possible solution. We may remark that these results do not form part of the core of this dissertation. Some of them are presented in Appendix §A. We intend to continue this work as future work.

Problem statement: Understanding the differences between both languages requires a comparison study.

Contribution: We have informally compared both languages, identified the main differences between them, and discussed some issues that came to light when trying to apply the same process for the analysis of feature models to OVM. The main results of these contributions have been published in [106].

Problem statement: Currently, there are no approaches for the transformation of feature models to OVM.

Contribution: We have proposed an algorithm to transform a feature model into an OVM, and implemented it by using a model-driven development approach. The preliminary results of this contribution were published in [107, 109].

In addition to the set of techniques for the automated analysis of OVMS and attribute-aware OVMS, we have developed a tool to which we refer to as FaMa-OVM. In Subsection §1.2.2, we introduce FaMa-OVM and describe how it has endowed the FaMa ecosystem.

1.2.2 Developed tool

The results of this dissertation have been integrated into FaMa-OVM, a tool support for the automated analysis of OVMs. This tool has been released under open source license and is accessible from the FaMa-OVM web site [48]. We believe that FaMa-OVM would be useful for those researchers, such as Bencomo et al. [17], Elfaki et al. [45], Loughran et al. [79], Metzger et al. [91], Peña [100], Petersen et al. [101], and Mærsk-Møller and Jørgensen [80], who, amongst others, work with Orthogonal Variability Models.

This dissertation continues the work on the analysis of variability models developed in the last years by our research group, the Applied Software Engineering research group (ISA). The ISA group has focused on the development of a product line of tools for the analysis of variability models and a framework, referred to as FeAture Model Analyser (FaMa), to support it. Following the recent trend towards a transition from a software product line environment to ecosystems, the ISA group has decided to open up the FaMa product line to external partners in order to share the costs and benefits of innovation. Hence, we are currently in the process of moving from a product line strategy to a software ecosystem approach [24]. With FaMa-OVM we have endowed the FaMa ecosystem with a new analysis component. This work was a good experience in the context of ISA group since we were able to evidence the benefits of FaMa framework when developing a new tool for the analysis of a different variability model.

Figure §1.4 illustrates the FaMa ecosystem, in which the FaMa framework is the core (*i.e.*, the platform). Around it, a number of extensions are built, which we mainly organise in three different groups:

Testing and benchmarking. The FaMa Test Suite (FaMa TeS), and the framework, Benchmarking and TesTing on the analysis of feature models (BeTTy) are contributions added to the FaMa ecosystem to provide functional and performance testing of feature model analysis tools [19, 47].

Analysis components. At the time of writing this dissertation, there is one extension, FaMa-FM, in the field of analysis component tools that is already integrated into the FaMa framework. FaMa-FM integrates different solvers for automated analyses of feature models (JavaBDD [142], SAT [18], Choco [75], and JaCoP [70] solvers are implemented). Another extension that is being integrate into FaMa ecosystem is FaMa-DEB, which is a tool to support the automated analysis of Debian packages descriptions (DEB) models. The third

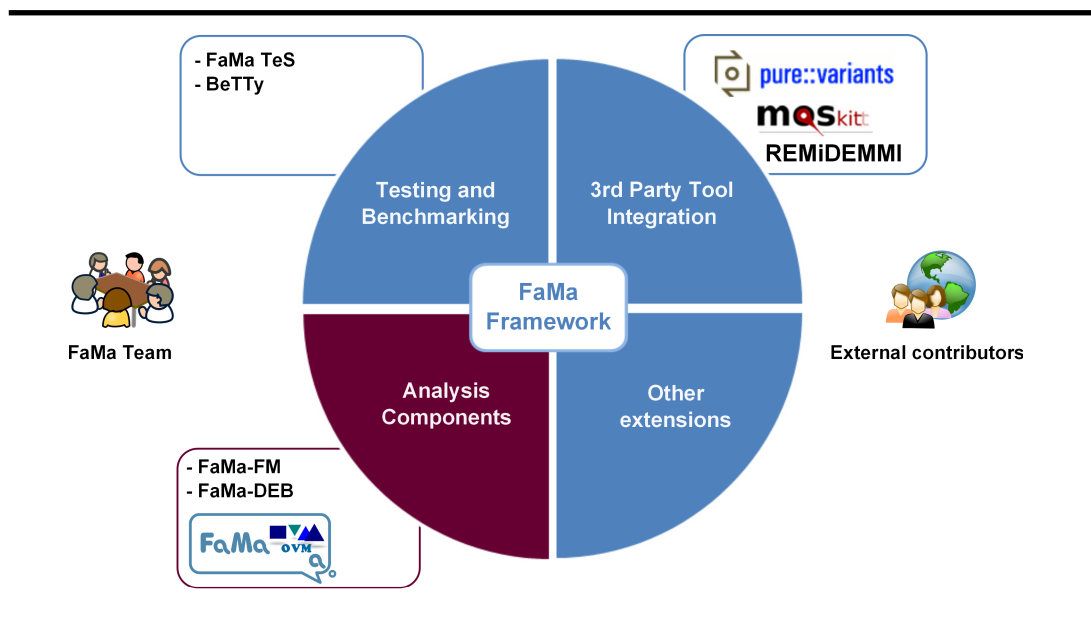


Figure 1.4: *The FaMa ecosystem.*

extension to be integrated into the ecosystem is FaMa-OVM tool, which is part of the results of this dissertation.

Third part tool integration. These extensions are carried out by external collaborators. At the time of writing this dissertation, FaMa is integrated into the visual editor MOSKitt feature modeller [94], it is being integrated into the commercial tool pure::variants [103]^{†1}, and being integrated into the OVM Editor of the case tool REMiDEMMI (Requirements Engineering and Management in Domain Engineering with Multi-Model Interaction) [67].

1.2.3 Potential benefits

OVM was devised to document variability of software product lines, however it can be applied to other areas, such as done by Mietzner et al. [92]. Thus, although our contribution targets the software product line community, other OVM users can benefit from it.

In a German project, a leading car company has used OVM to model their product line. The resulting product line represents a typical case in which

^{†1}In the context of the DiVA European project (<http://www.ict-diva.eu/>)

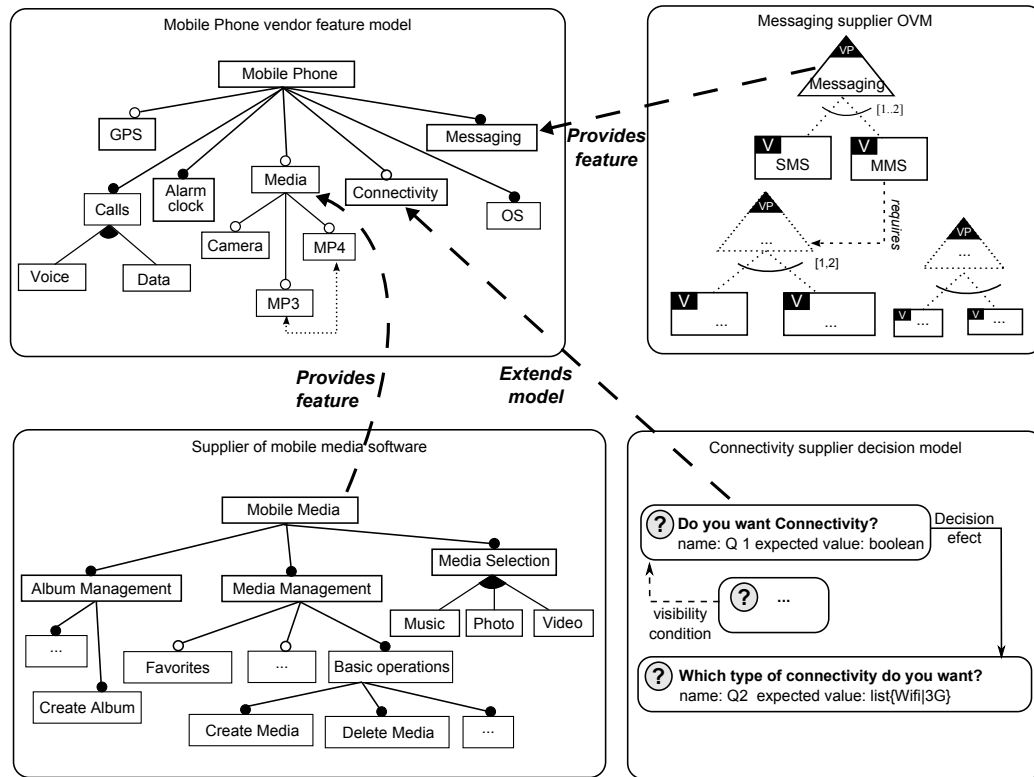


Figure 1.5: A multi product line using heterogeneous variability models.

we can apply our approach. We have studied this case and shown that the engineers of such product line can benefit from our approach to automatically extract information from their OVMs.

Another context to which our approach can be useful is the multi product lines. In this context, several organisations collaborate to build products. Vendor and suppliers usually maintain their own product lines, which often are developed and managed using heterogeneous techniques and tools [41]. In Figure 1.5, we present an example to illustrate a common scenario in which heterogeneous variability modelling approaches are used to document variability of a multi product line in the mobile phone domain. In this example, there is a main vendor of mobile phones that uses resources from several suppliers. The vendor uses a feature model to express the possible choices that the customer can select. The suppliers use different approaches to express variability of their product line, namely: feature modelling, OVM, and

decision modelling. In order to express how the features provided by suppliers are related to features in the mobile phone feature model, relationships amongst the variability models are defined. The mobile phone vendor defines in the feature model that a mobile phone always has messaging resource and can support media and connectivity. One supplier provides a mobile media software, which has several, detailed configuration choices. Another supplier uses an OVM to describe the variability of the messaging system, whereas the other supplier uses a decision model to express connectivity choices.

1.3 Research visits and collaborations

Throughout the development of these doctoral studies we carried out some research activities in collaboration with the following universities:

- *Newcastle University (United Kingdom)*. A research visit was paid to the School of Computing Science for the period from the 25th of February until the 3rd of April, 2009. The focus of this visit was on discussing our research problems and gathering feedback about variability modelling languages.
- *Leicester University (United Kingdom)*. A research visit was paid to the Department of Computer Science for the period from the 1st of October until the 29th of December, 2009. We worked closely with the creator of MOMENT2 [22, 23], a suite of tools that provides support for formal model-driven development. Amongst other activities, we implemented, using MOMENT2's, a first version of our algorithm for the feature model to OVM transformation presented in [107]. The refinement and application of this algorithm is one of the tasks we placed as future work. In addition to this, we took the opportunity to learn more about the use of Maude language [31] for giving semantics to variability models and specifying their verification, since researchers of this department have an expertise in the use of this language. MOMENT2 uses Maude as the underlying engine to model check invariants and lineal temporal logic properties of model transformations [22].
- *University of Duisburg-Essen (Germany)*. During the development of this dissertation we collaborated with members of the Software Systems Engineering research group at the Institute for Computer Science and Business Information Systems. Our collaboration aimed at applying our

results, particularly automated analysis of attribute-aware OVMs, to a case of study. Amongst other results, the collaboration resulted in a publication in the special issue on Quality Engineering for Software Product Lines of the Software Quality Journal [111].

- *Federal University of Rio Grande do Sul (Brazil)*. We have organised a workshop in Seville, Spain from the 11th until 15th of October, 2010 to present research results of both Brazilian and Spanish research groups aiming at preparing a collaboration project, and gathering feedback on our work.

1.4 Structure of this dissertation

This dissertation is organised as follows:

Part I: Preface. Comprises this introduction chapter, in which we present our research context and summarise our contributions.

Part II: Background Information. Provides the reader with information regarding to the research context in which our work has been developed. In Chapter §2, we survey the most common notations of feature models providing some examples and describe OVM. In Chapter §3, we present a summary of the most relevant analysis operations on feature models found in the literature and the proposed automated support for them. This chapter is based on an extensive literature review on the analysis of feature models presented by Benavides et al. [13].

Part III: Our Contribution. Reports on the core contributions we made with this dissertation and is organised in four chapters. In Chapter §4, we motivate our research work, present the problems addressed in this dissertation, analyse current solutions, and conclude that current works do not provide support for the problems we addressed. In Chapter §5, we present our approach to automate the analysis of regular OVMs, in which we describe the analysis process and the techniques used to automate this analysis. In Chapter §6, we introduce attribute-aware OVM, by defining a way of expressing and relating attributes to OVM, and report on our proposal to automate the analysis of attribute-aware OVMs. In Chapter §7, we present a proof of concepts implementation of our approach using concrete solvers. In this chapter we present a study of a

case from the automotive domain in which we have used our approach to demonstrate its feasibility.

Part IV: Final Remarks. Concludes this dissertation and highlights some future research directions in Chapter §8.

Part V: Appendices. In Appendix §A, we present a set of mapping rules for the Feature model to OVM transformation. A complete list of the selective mapping rules defined in our dissertation is presented in Appendix §B. In Appendix §C, we present the OVM specification for the case we have studied. Finally, in Appendix §D, we clarify the meaning of the acronyms used throughout this dissertation.

Part II

Background Information

Chapter 2

Variability Models

*Variability is the law of life, and as no two faces are the same,
so no two bodies are alike(...)*

*William Osler, 1849–1919
Canadian Physician*

Variability models are mainly used to document the variability of software product lines. Variability modelling is used to efficiently describe the commonalities and variabilities of a family of software products. One of the most popular variability modelling techniques is feature modelling. The first feature modelling approach was presented as part of the feature-oriented domain analysis (FODA). Since then, several extensions of FODA have been proposed. Although the majority of works address feature models, there are many approaches providing other alternatives to variability management. In this chapter, we focus on those approaches to which we will refer throughout this dissertation. The remainder of this chapter is structured as follows: Section §2.1 introduces the concept of variability modelling. A brief survey of the most common notations for feature modelling is presented in Section §2.2. Section §2.3 presents an overview of the orthogonal variability modelling approach. Finally, Section §2.4 summarises the chapter.

2.1 Introduction

Variability modelling was introduced in software product line engineering to allow for an efficient documentation of what is common and what is different in the products of a family of software products. Variability modelling techniques provide a mechanism to specify rules that constrain the combination of reusable assets. These constraints may come from technical restrictions or any domain decision throughout all development phases. For instance, requirement engineers have to define which requirements are mandatory (*i.e.*, common to all products) and which ones are optional (*i.e.*, may or may not be in a specific product). Similarly, design engineers have to know the dependencies and incompatibilities amongst software components. Therefore, variability models rely on describing a set of options that must be selected in order to derive a specific product line instance. The derivation process is done by selecting desired and valid options in the variability model during application engineering.

2.2 Feature models

Feature models have been proposed to represent in an abstract way the commonalities and variabilities amongst products in a software product line. They provide an abstract representation of all the *products* of the product line, which are determined by all the possible combinations of features.

A feature model is usually composed of two main elements: features and relationships between them. In this context, features are any domain abstraction relevant to stakeholders and are typically increments in program functionality [125]. Features are arranged in a tree-like structure. Constraints of the type *requires* and *excludes* between features can be added, leading to additional complexity, thus resulting in a directed acyclic graph [116].

The first feature modelling approach was proposed in 1990 by Kang et al. [71] as part of the Feature-Oriented Domain Analysis (FODA). Since then, several extensions of FODA have been proposed. In the next sections, we provide an overview of the classical notation of feature models, which we refer to as *Basic feature models*, and briefly describe *Cardinality-based* and *Extended feature models*. For a more complete description about feature model semantics we refer the reader to Schobbens et al. [116].

2.2.1 Basic feature models

We call the FODA [71] and Feature-RSEB [64] feature models basic since they use simple relationships between features. In the FODA approach, the relationships between features can be of two types:

- i. *Hierarchical relationship.* This relationship is defined between a parent feature and its child features. A child feature can only be part of those products in which the parent feature appears. FODA provides three types of hierarchical relationships:

Mandatory. It means that when the parent is part of a specific product, the child also must be part of it.

Optional. It means that when the parent is part of a specific product, the child may or may not be part of it.

Alternative. It means that when the parent is part of a specific product, one and only one of the child features in the set must be in the product.

- ii. *Cross-tree constraints.* They are of two types:

Requires. Feature X requires Y means that if the feature X is included in the product, then feature Y must be included as well, but not vice versa.

Excludes. Feature X excludes Y means that if the feature X is included in the product, then feature Y cannot be included, and vice versa.

In 1998, Griss et al. [64] proposed integrating the Feature-Oriented Domain Analysis method (FODA) into the so-called Reuse-Driven Software Engineering Business (RSEB). We refer to this notation as Feature-RSEB. They extend FODA feature models by adding a new hierarchical relationship between parent and child features, as follows:

Or-relationship. It means that when the parent is part of a specific product, one or more of the child features must be in the product.

Figure §2.1 depicts a basic feature model example inspired by the mobile phone industry. This feature model represents a product line in which every product contains eight features, namely, `MobilePhone`, `UtilityFunctions`,

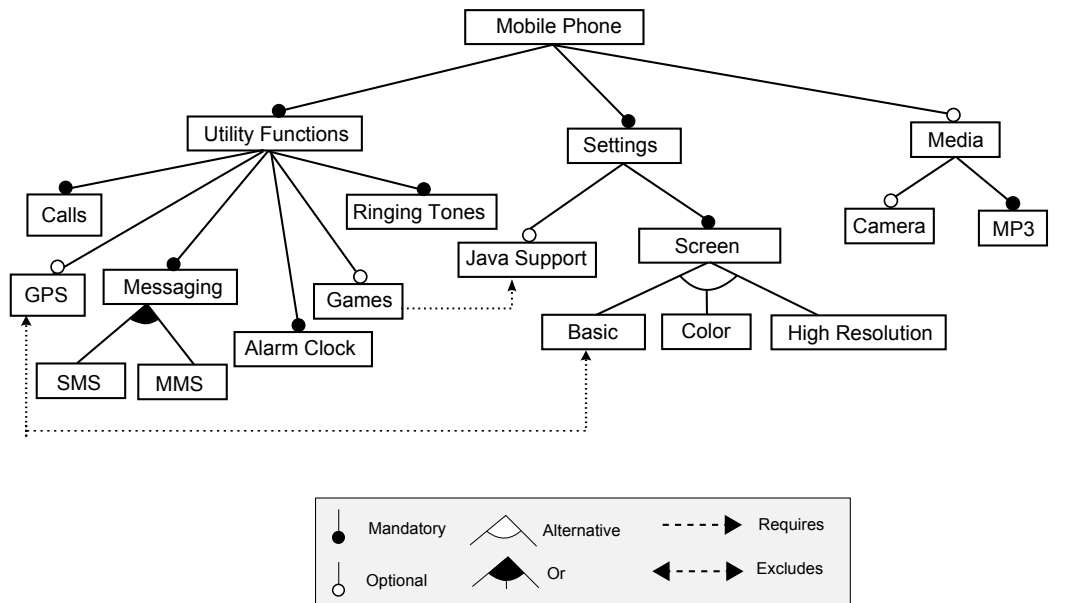


Figure 2.1: Example of a basic feature model.

Calls, Messaging, AlarmClock, RingingTones, Settings, and Screen. Furthermore, the product line may have: *i*) five features, namely, GPS, Games, JavaSupport, Media, and Camera, which can be selected or left out at will, however, if Media is selected, MP3 must be selected as well; *ii*) the grouped features Basic, Color and HighResolution that are possible choices of their parent feature (Screen), but one and only one of these grouped features can be selected, and *iii*) the grouped features SMS and MMS that are possible choices of their parent feature (Messaging) with the *Or* relationship defining that one or more features of the group must be selected. In addition, the constraints *requires* and *excludes* impose limitations on the possible combinations of features. In this case, when Games is selected, JavaSupport must be selected as well, and GPS and Basic features cannot be part of the same product.

2.2.2 Cardinality-based feature models

In [104, 105], Riebisch and others introduced the concept of *set of features* and proposed adding multiplicities (a.k.a cardinalities) to this set in order to avoid ambiguities. To this effect, the authors proposed keeping the same no-

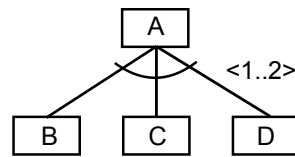


Figure 2.2: Set relationship with group cardinality.

tation for mandatory and optional features as in FODA, but replacing the *alternative* and *or-relationships* with a new relationship, as follows:

Set-relationship. It relates a parent feature with a set of child features. This set can include group cardinalities, which are represented by an interval, $\langle n..n' \rangle$. This cardinality limits the number of child features in the group that can be part of a product, at least n and at most n' . For instance, in the relationship depicted in Figure §2.2, the number of child features that can be part of a product is 1 or 2. The *alternative* relationship in FODA is replaced by cardinality $\langle 1 - 1 \rangle$, meaning that one and only one of the child features must be part of the product. Similarly, the *or-relationship* is replaced by cardinality $\langle 1 - n \rangle$, where n is the number of features in the set, which means that one or more (at most n) of the child features must be part of the product.

Later, Czarnecki et al. [36, 37] proposed another extension to the FODA feature models. In these works, the authors introduce a new hierarchical relationship, namely feature cardinality, maintaining the mandatory, optional, and set relationship previously described.

Feature cardinality. It limits the number of instances of the feature that can be part of a product. This cardinality is represented by a sequence of intervals of the form $[n..n']$ with n as lower bound and n' as upper bound. For instance, in Figure §2.3, the feature cardinality indicates that the number of instances of the feature B that must be part of a product is at least 2 and at most 4. This relationship can generalise the original mandatory and optional relationships. The mandatory relationship can be represented by the feature cardinality $[1..1]$ and the optional one by $[0..1]$.

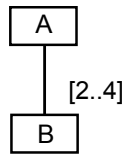


Figure 2.3: Sample of feature cardinality.

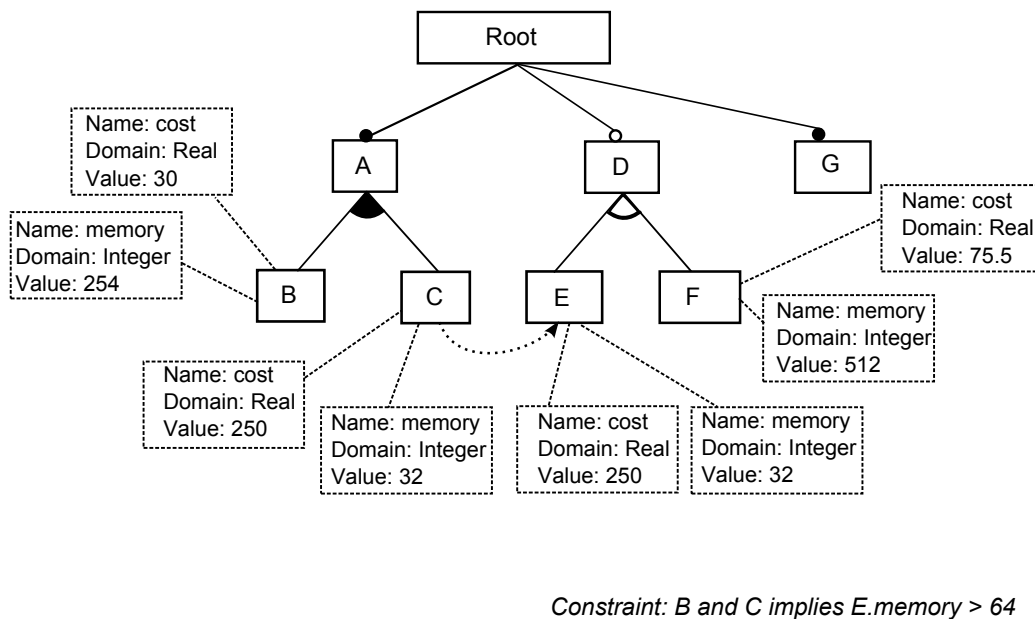


Figure 2.4: Sample of an extended feature model.

2.2.3 Extended feature models

Some authors have identified the need to extend feature models with attributes such as memory consumption, binary size and development cost [15, 37, 72]. The purpose of this extension is to add measurable information about the features, which is done by introducing *attributes* to features. As stated by [13], there is no consensus on a notation to define attributes. However, most proposals agree that an attribute should consist of a name, a domain and a value. Figure 2.4 shows an example of an extended feature model using the notation proposed by [15].

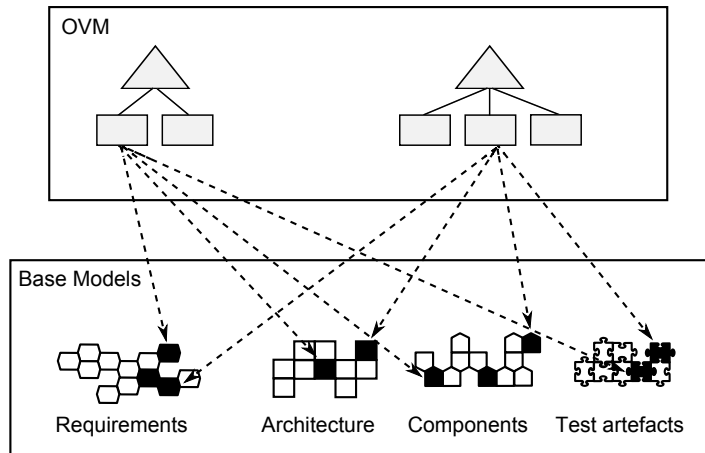


Figure 2.5: Orthogonality of OVM (based on [90]).

Note that, this extension enables the inclusion of more complex constraints amongst features and attributes. For example, it is possible to specify constraints like: “If feature *B* and feature *C* are selected, then memory of feature *E* must be higher than 64”.

2.3 Orthogonal variability models

The *Orthogonal Variability Model* (OVM) is a modelling language proposed by Pohl et al. [102] to define the variability of a software product line in an orthogonal way, *i.e.*, it provides a cross-sectional view of the variability across all product line artefacts. OVM interrelates the variability in base models such as requirement models, design models, component models, and test models (see Figure §2.5). The traceability between OVM and the different types of base models is established through artefact dependencies (dashed lines in Figure §2.5). Figure §2.6 shows an example of a graphical notation for OVM. On the right hand side, the meaning of each graphical element and their relationships is depicted.

An OVM is composed of two main elements: *variation points* and *variants*. We refer to these elements as variability elements, and they have the following meaning:

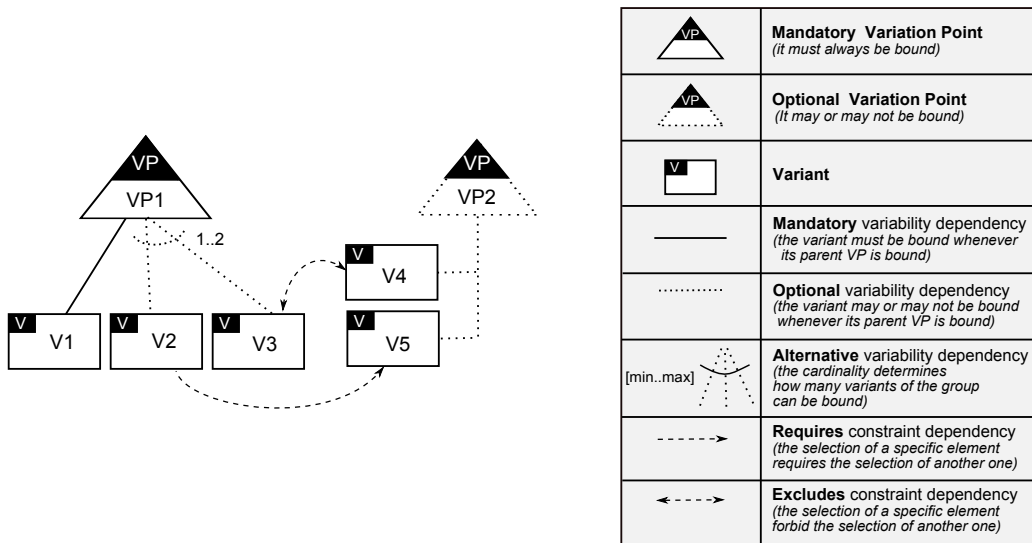


Figure 2.6: Graphical notation for OVMs.

Variation Point (VP). It documents what can vary within artefacts of the product line, *i.e.*, where differences exist in the final software product, and are chosen by the customer or engineer of the software product line. For instance, products may differ with respect to operating systems they support, with respect to whether they provide access to the internet or not, and so on. Variation points can be mandatory or optional.

Variant (V). It is related to a variation point and documents how such variation point can vary. For instance, products may come with two different operating systems.

The relationships between variability elements can be of two types:

Variability dependencies. They define the rules that constraint the possible choices (variants) to a variation point. Variability dependencies can be of three types, namely: *mandatory*, *optional*, and *alternative*, as shown in Figure §2.7. The cardinality in the alternative relationship determines how many variants can be chosen simultaneously. We refer to the variability dependency between a variation point and a variant as parent-child relationship.

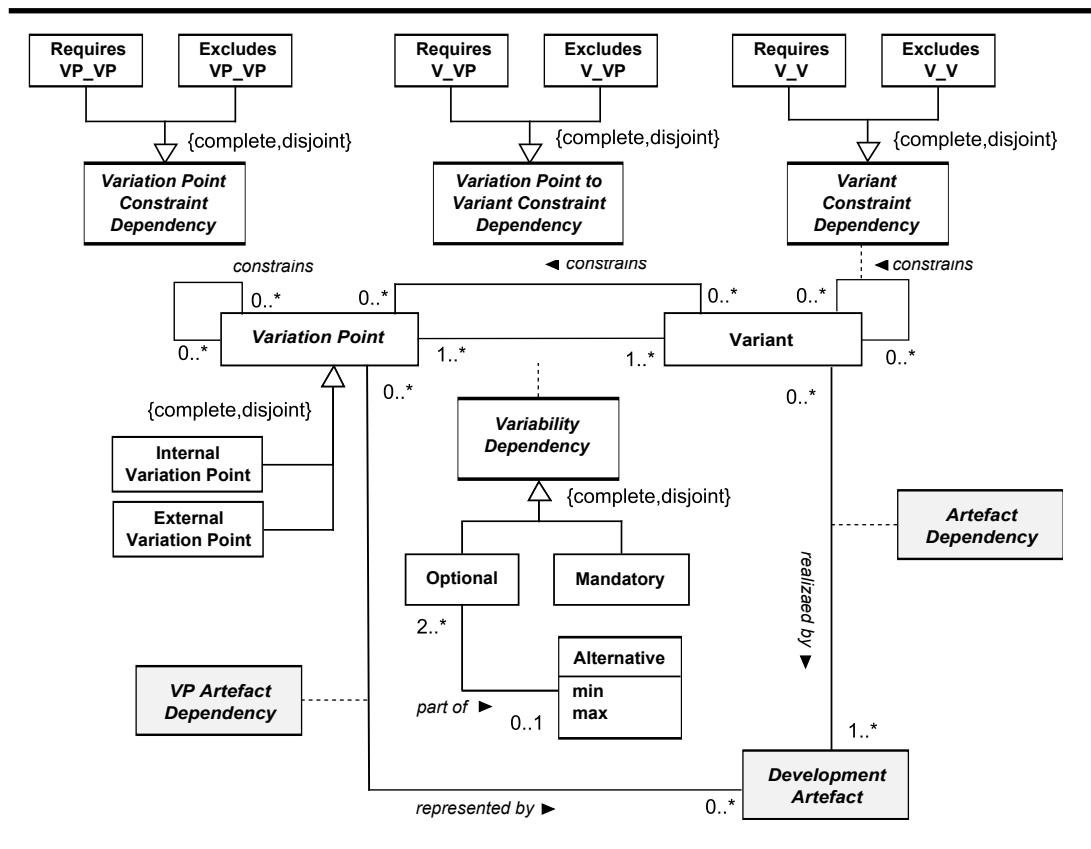


Figure 2.7: OVM metamodel (from [102]).

Constraint dependencies. They define possible dependencies and incompatibilities amongst variant selections. They are of two forms: *excludes* and *requires*, as shown in Figure §2.7.

The first OVM's abstract syntax was defined in [102] by means of a meta-model, which describes what is a well-formed OVM diagram (see Figure §2.7). In this meta-model, `InternalVariationPoint` and `ExternalVariationPoint` represent the type of variation point with regard to the visibility of the variability they document, *i.e.*, *internal variation point* has child variants that are only visible to developers but not to customers, and *external variation point* has child variants that are visible to developers and customers. These classification of variation points is not observed in the graphical notation. Furthermore, the grey elements depicted in the meta-model represent the relationship between variability and artefacts, where `DevelopmentArtefact` is an abstract class that represents any kind of software artefact, realised by association

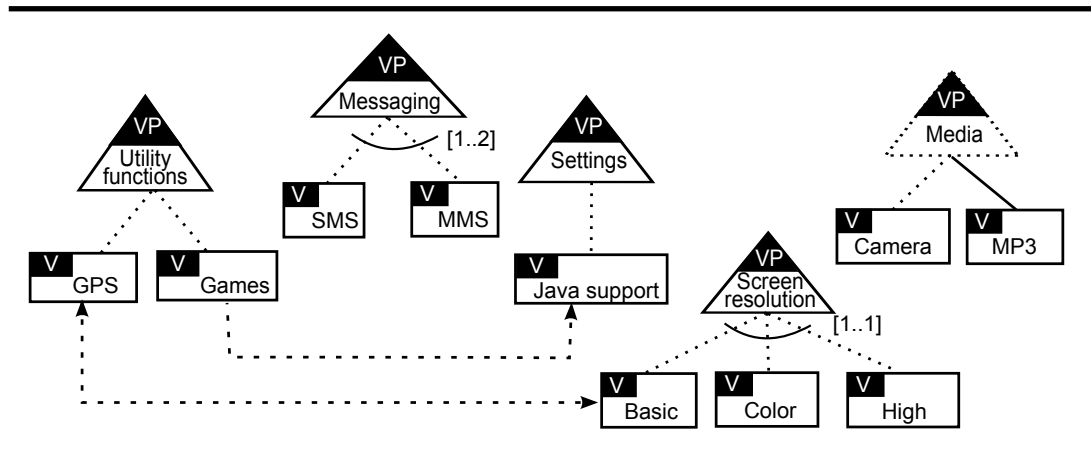


Figure 2.8: Example of an OVM.

relates Variant with DevelopmentArtefact, and represented by association relates VariationPoint with DevelopmentArtefact. These associations determine that *i*) a variant must be related to one or more artefacts, but an artefact can but does not have to be related to one or more variants, and *ii*) a variation point can be related to zero or more artefacts, and an artefact can be related to zero or more variation points.

Later, the previously described abstract syntax was defined in mathematical notation by Metzger et al. [91]. In their work, the authors introduced two small changes in the language, as follows:

- two types of variation points are distinguished, *Optional* and *Mandatory* (see VP1 and VP2 in Figure §2.6);
- the variation points have at least one child and each variant has at most one parent.

As an example, Figure §2.8 depicts a possible OVM for the mobile phone product line presented in Figure §2.1.

In summary, an OVM provides an abstract representation of all the *variations* of the product line, which are determined by all the possible combinations of variation points and variants in the OVM.

2.4 Summary

In this chapter, we have introduced the concept of variability models. In particular, we have presented an overview of feature models, the most popular variability model in the literature. We also have presented extended feature models since the concepts introduced in this kind of model are relevant to the context of this dissertation. Furthermore, we have introduced the orthogonal variability modelling language, which is the focus of our research and thus is used throughout the writing of this dissertation. For a more complete and rigorous definition of orthogonal variability modelling language we refer the reader to Metzger et al. [91].

Chapter 3

Automated Analysis of Feature Models

*Computers are useless.
They can only give you answers.*

*Pablo Picasso, 1881–1973
Spanish Cubist painter*

The automated analysis of feature models deals with the computer-aided extraction of information from feature models. In this chapter, we give an overview of the main contributions found in the literature in the context of automated analysis of feature models. In Section §3.1, we report on the importance of automated analysis of variability models, especially feature models, and describe the automated analysis process used by most of the approaches. Next, in Section §3.2, we summarise some of the analysis operations proposed by the research community. In Section §3.3, we present some of the approaches providing automated support for the analysis of feature models. In Section §3.4, we provide information about the automated analysis of feature models with abstract features. Finally, Section §3.5 summarises the chapter.

3.1 Introduction

In the software product line community, it is well-known that variability in product lines is increasing [6, 78, 121, 122, 143]; feature models may have thousands of features, and these features may have complex dependencies amongst them. Moreover, the number of possible products (*i.e.*, feature combinations) may grow exponentially with the number of features in the feature model. For instance, the number of possible different products that can be derived from the mobile phone product line is 45, as shown in Figure §2.1 (Section §2.2.1). However, if we increase the number of features in this product line from 19 to 30 features, and the new relationships between features increases the number of possible combinations, the number of possible products may reach 5040. Therefore, the management of such models is practically impossible without an automated tool support. In addition, analysis operations such as whether a product belongs to a product line or whether a feature model contains any errors, can become a very difficult and error-prone task to be done manually. As a consequence, automated analysis of feature models has increasingly become a relevant research topic for the software product line community.

The automated analysis of software product lines by means of variability models can be considered as the computer-aided extraction of information from variability models [13]. As reviewed by Benavides et al. [13], many authors have proposed approaches to automate the analysis of feature models. Currently, there are some contributions on the automated analysis of other variability models (*e.g.*, DOPLER decision models [84]), however they are practically based on the same concepts introduced in the analysis of feature models. Therefore, in this chapter, we focus our attention on describing concepts and analysis operations in the context of the analysis of feature models.

The purpose of automated analysis of feature models is to extract information from feature models using automated mechanisms [6]. This extraction is mainly performed in a two-step process depicted in Figure §3.1. The first step consists of translating an input data into a representation or paradigm such as propositional logic, description logic or any ad-hoc data structure. Then, in the second step, off-the-shelf solvers or specific algorithms are used to automatically analyse the intermediate representation generated from the given input, and then provide the obtained result. The input data is made up of one or more parameters, but it necessarily includes a feature model. The other parameters to be included in the input data are determined by the analysis operations to be carried out in the second step. This implies that analysis op-

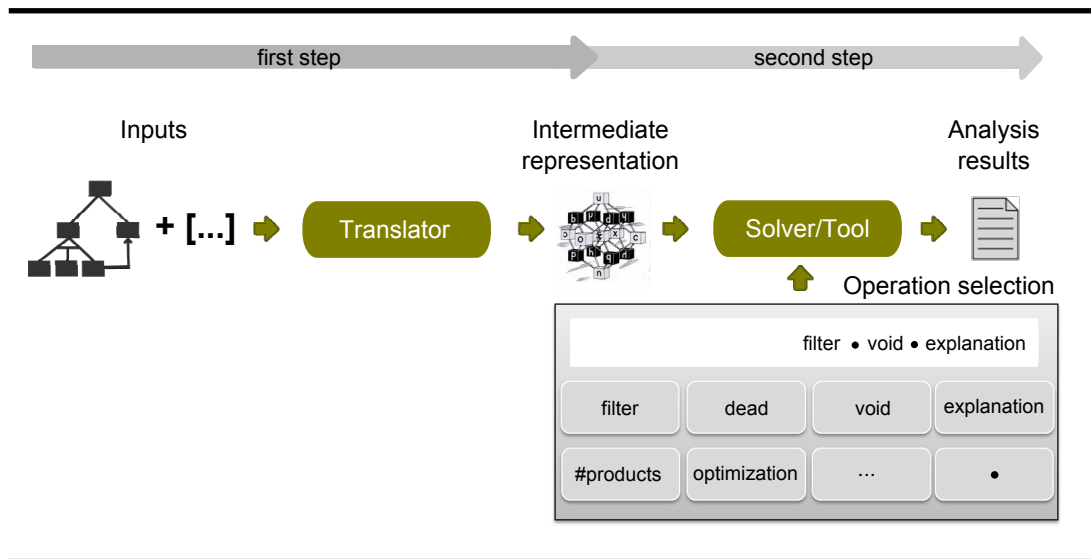


Figure 3.1: Process for the automated analysis of feature models.

erations are central and thus practically guide the analysis process.

In the rest of this chapter, we briefly report on the most relevant analysis operations found in the literature, and also on the different proposals providing automated support for the analysis of feature models. First, we explain the operations and give some examples. Then, we present and explain the approaches proposed to automate the analysis.

3.2 Analysis operations on feature models

An analysis operation takes a set of parameters as input and returns a result as output. From the results obtained, product line engineers, product managers or developers can improve their decision making [8, 99, 128]. In 2010, Benavides et al. [13] performed an exhaustive literature review to bring the software product line community up to date with current literature on the analysis of feature models. In this literature review, 30 different analysis operations that can be performed on feature models were identified.

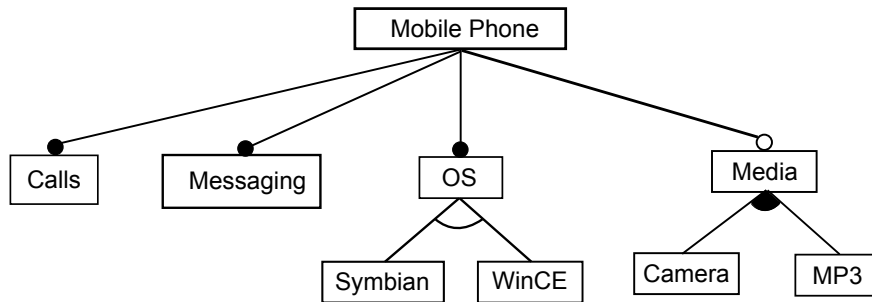


Figure 3.2: A simple feature model for the mobile phone product line.

3.2.1 Input and output parameters

Before we delve into the definition of each operation, it is important to clarify some of the terms that are necessary to understand the operation definitions, since they are usually ambiguously defined in the literature. According to Benavides et al. [13], in addition to feature models, typical input and output parameters for the analysis operations can be described as follows:

- *Configuration*. Given a feature model with a set of features F , a configuration is a 2-tuple of the form (S, R) such that $S, R \subseteq F$ being S the set of features to be selected and R the set of features to be removed such that $S \cap R = \emptyset$. There are two types of configurations, as follows:

- ◊ *Full configuration*. If $S \cup R = F$ the configuration is called *full configuration*. To give an example, we use a simpler feature model for the mobile phone presented in Figure §3.2. A possible full configuration (FC) for this model can be as follows:

$$FC = (\{\text{MobilePhone, Calls, Messaging, OS, Symbian}\}, \{\text{WinCE, Media, Camera, MP3}\})$$

- ◊ *Partial configuration*. If $S \cup R \subset F$ the configuration is called *partial configuration*. As an example, a possible partial configuration (PC) for the model in Figure §3.2 can be as follows:

$$PC = (\{\text{MobilePhone, Calls, Messaging, OS, Symbian}\}, \{\text{MP3}\})$$

- *Product*. A product is equivalent to a full configuration, but in this case, features that should be removed can be omitted. Thus, the product (P) equivalent to the full configuration described above is as follows:

$$P = \{\text{MobilePhone, Calls, Messaging, OS, Symbian}\}$$

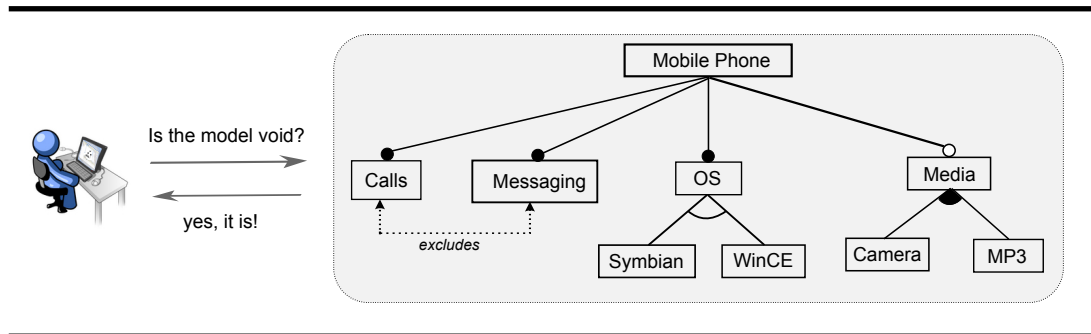


Figure 3.3: Sample of void feature model.

3.2.2 Operations overview

Some of the most common analysis operations on feature models are as follows:

Void feature models. A feature model is void if it does not represent any product. Figure §3.3 shows how the feature model previously presented in Figure §3.2 become void by the wrong use of a constraint. The *excludes* constraint makes the simultaneous selection of *Calls* and *Messaging* impossible, which leads the model to a contradiction since both features are mandatory and thus must be in all products. The automation of this operation may be helpful since the manual detection of errors is error-prone and time-consuming [5, 71, 127].

Valid product. A product is valid if it belongs to the set of products represented by the feature model. For instance, consider the feature model in Figure §3.2 and the products, P_1 and P_2 , described below.

$P_1 = \{\text{MobilePhone, Calls, Messaging, OS, Symbian}\}$
 $P_2 = \{\text{MobilePhone, Messaging, OS, Symbian, Media, MP3}\}$

Product P_1 is valid since it includes all the mandatory features, namely, *MobilePhone*, *Calls*, *Messaging*, and *OS* and besides, one and only one of the operating systems is selected, *Symbian*. On the other hand, product P_2 does not belong to the set of products represented by the model since it does not include all the mandatory features. This operation may be helpful for product line managers to verify whether a given product is available in the software product line.

All products. This operation returns the set of products represented by a given feature model. For instance, the set of all products of the feature model presented in Figure §3.2 is as follows:

```
P1 = {MobilePhone, Calls, Messaging, OS, Symbian}
P2 = {MobilePhone, Calls, Messaging, OS, Symbian, Media, MP3}
P3 = {MobilePhone, Calls, Messaging, OS, Symbian, Media, Camera}
P4 = {MobilePhone, Calls, Messaging, OS, Symbian, Media, Camera, MP3}
P5 = {MobilePhone, Calls, Messaging, OS, WinCE}
P6 = {MobilePhone, Calls, Messaging, OS, WinCE, Media, MP3}
P7 = {MobilePhone, Calls, Messaging, OS, WinCE, Media, Camera}
P8 = {MobilePhone, Calls, Messaging, OS, WinCE, Media, Camera, MP3}
```

This operation may be helpful to give an overview of the scope of the product line.

Number of products. This operation returns the number of products represented by a given feature model. As an example, the feature model depicted in Figure §3.2 represents 8 products, whereas the feature model in Figure §2.1 represents 45.

This operation provides information about flexibility and complexity of the software product line [15, 38, 137]. In the example in Figure §2.1, if we simply remove the requires from Games to JavaSupport, the number of products raises from 45 to 60. A big number of potential products may reveal a more flexible product line; however, this may increase its complexity.

Filter. Given an input configuration, it returns the set of products including such configuration that can be derived from a given feature model. As an example, the set of products of the feature model in Figure §3.2 applying the filter with the partial configuration ($\{\text{Symbian}\}, \{\text{MP3}\}$), where feature Symbian is selected and feature MP3 is removed, are as detailed below:

```
P1 = {MobilePhone, Calls, Messaging, OS, Symbian}
P2 = {MobilePhone, Calls, Messaging, OS, Symbian, Media, Camera}
```

Dead features. A feature is dead if it does not appear in any of the products represented by a given feature model. This is an undesirable situation since the model gives to the user a false view of the product line domain. Figure §3.4 shows some typical cases that generate dead features in feature models (features “D” are dead).

False optional features. A feature is false optional if despite it not being modelled as mandatory, it is included in all the products represented by the

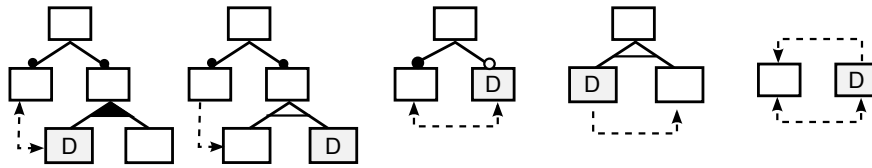


Figure 3.4: Typical cases of dead features.

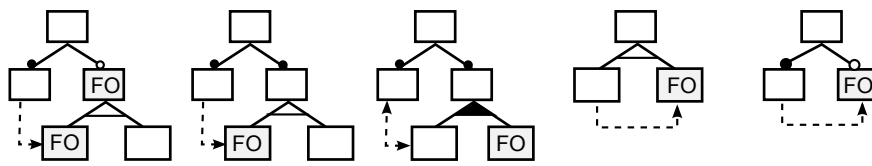


Figure 3.5: Typical cases of false optional features.

model. Just as with dead features, false optional features give a wrong idea of the product line domain. Figure §3.5 shows some typical cases that generate false optional features (features “FO” are false optional).

Commonality. This operation returns the percentage of valid products represented by a given feature model in which an input feature appears. For instance, the commonality of feature Camera in the model in Figure §3.2 is 0.5 since Camera is included in 4 out of the 8 products represented by the model. This means that feature Camera appears in 50% of the products of the product line. This operation can easily be generalised in order to receive as input a configuration instead of a single feature, as described in [13].

Optimisation. This operation returns the optimal product(s), *i.e.*, the solution(s) that optimise a problem associated with an objective function. This operation applies to extended feature models where features have attributes. Finding the optimal solution, as opposed to any possible solution, may be helpful for making attribute-aware decisions. As an example, consider the model in Figure §2.4, on page 28. This model has several attributes called memory, which are associated to some features. Each of these attributes has a value. So, if a product line engineer wants to know *which product of the set of products represented by this model consumes less memory*, the optimisation operation can be used. The ob-

jective function would be defined as the sum of values of the attributes memory, and the optimal product(s) would be one(s) that minimise this objective function. Consider that the set of all possible products is as follows:

$P_1 = \{\text{Root}, A, B, G\}$
 $P_2 = \{\text{Root}, A, B, D, E, G\}$
 $P_3 = \{\text{Root}, A, B, D, F, G\}$
 $P_4 = \{\text{Root}, A, B, C, D, E, G\}$
 $P_5 = \{\text{Root}, A, C, D, E, G\}$

Then, the optimal product is product P5 since it minimises the sum of values of attributes memory; features C and D consumes less memory than features B and F.

Edits. These operations give information about how two different models are related. These operations are useful for determining how a model has evolved over time. Editing a feature model produces a new feature model. In [125], the authors consider that modifications to evolve a feature model can be classified as: refactorings, specialisations, generalizations, or arbitrary edits, as shown in Figure §3.6. An edit is a refactoring when the original and the resulting feature models represent exactly the same set of products; it is a specialisation when existing products are removed, but no new products added; it is a generalisation when new products are added, but no existing products are removed; and, an edit is arbitrary when it cannot be classified into any of the previous edits.

For a more formal definition of the analysis operations on feature models we refer the reader to [8, 44].

3.3 Automated support for the analysis of feature models

As identified by Benavides et al. [13], a number of different techniques and algorithms proposed to automatically analyse some of the analysis operations on feature models were identified. These works can be divided into four categories according to the logical paradigm or technique used to automated the analysis: namely, *propositional logic*, *constraint programming*, *description logic*, and *ad-hoc*. Ad-hoc solutions are those that are not classified in the former groups. In the next sections, we give more details about approaches

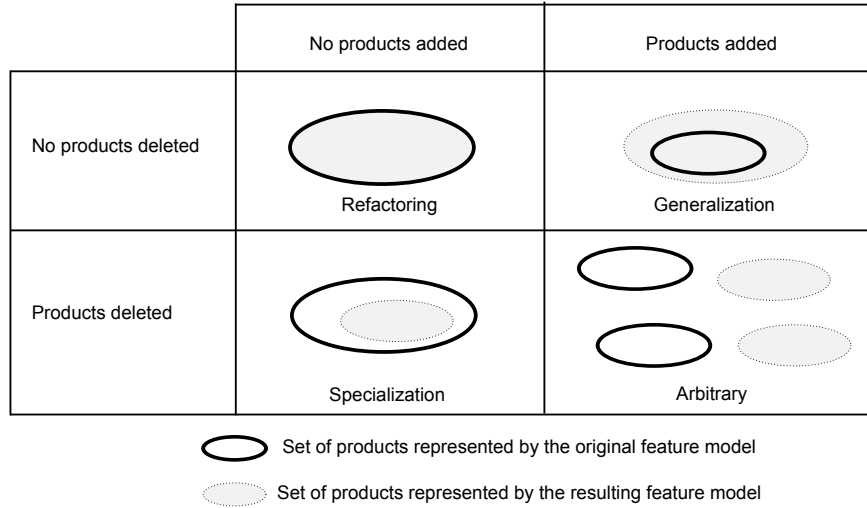


Figure 3.6: Classification of feature model edits (from Thüm et al. [125]).

that use constraints programming and propositional logic paradigm, which we consider more relevant in the context of our dissertation. For a more extensive survey of approaches providing automated support for the analysis of feature models we refer the reader to [13].

3.3.1 Constraint programming

Constraint Programming is a discipline that relies on a set of techniques and algorithms to deal with *constraint satisfaction problems* (CSPs) [1]. A CSP is defined as a set of variables, a set of domains for those variables, and a set of constraints restricting the values of those variables [131]. A CSP solver takes a problem modelled as a CSP and determines whether there is a solution for the problem. Particularly, it searches for a valid set of variable values that simultaneously satisfies all constraints. For example, suppose x_1, x_2, x_3 are variables of a CSP, all with domains in $[1, 2, 3]$, and $(x_1 = x_2), (x_2 < x_3)$ being the constraints. A solution to this CSP is an assignment to every variable of some value in its domain such that it does not violate any of the constraints. Therefore, a possible solution to this CSP is $((x_1 \mapsto 1), (x_2 \mapsto 1), (x_3 \mapsto 2))$. We usually denote the set of solutions of a CSP as $\text{sol}(\psi)$.

Unlike propositional formulas, constraint programming deals not only

with boolean variables but also offer the possibility to work with numerical variables, such as integer or intervals, which, for instance, allow dealing with attributes of features, enabling them to maximise or minimise values.

The mapping of a feature model into a CSP can change depending on the solver that is used later for the analysis. The language used to define CSPs, and therefore the type of constraints allowed in common CSP solvers can vary greatly. In order to provide a generic CSP language to specify feature models as CSPs, Benavides [8] has formally defined a general abstract language of CSPs for feature models, which is independent of the solver. Roughly speaking, this generic CSP allows defining constraints by using operators, such as *and*, *or*, *not*, *implies*, *biconditional*, *atMostOne*, and *AtLeastOne*.

In general, the mapping of a feature model into CSP follows the next steps:

- i. each feature of the feature model becomes a variable of the CSP with a domain in $\{0..1\}$ or $\{\text{true}, \text{false}\}$, depending on the solver to be used;
- ii. each relationship of the model becomes a constraint which depends on the type of relationship. Some auxiliary variables can be needed;
- iii. a constraint assigning true to the variable that represents the root is defined, *i.e.*, “ $\text{root} \iff \text{true}$ ” or “ $\text{root} == 1$ ”, depending on the variables’ domains;
- iv. the resulting CSP is the one defined by the variables of step *i* with the corresponding domains and a constraint that is the conjunction of all constraints defined in steps *ii* and *iii*.

As proposed by some authors, after the mapping of feature models into CSP, an off-the-shelf solver is used to automatically analyse the CSP and thus the feature model. Some of the solvers used for this analysis are: JaCoP [70], Choco [75], OPL studio [95], GNU Prolog [60], and SkyBlue [113].

In Table §3.1 we list the rules for translating each type of relationship in the feature model into a generic CSP, as provided by Benavides et al. [15]. In this table, we also present the rules for the mapping of the mobile phone feature model presented in Figure §2.1 into the JaCoP solver.

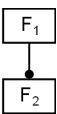
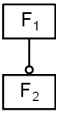
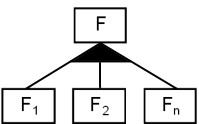
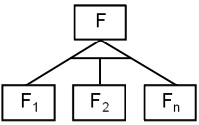
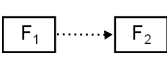

	Relationship	CSP mapping	JaCoP-like notation
MANDATORY		$biconditional(id(F_1), id(F_2))$	MobilePhone = UtilityFunctions MobilePhone = Settings UtilityFunctions = Calls UtilityFunctions = Messaging UtilityFunctions = AlarmClock UtilityFunctions = RingingTones Settings = Screen Media = MP3
OPTIONAL		$implies(id(F_2), id(F_1))$	if (MobilePhone = 0) Media = 0 if (UtilityFunctions = 0) GPS = 0 if (UtilityFunctions = 0) Games = 0 if (Settings = 0) JavaSupport = 0 if (Media = 0) Camera = 0
OR		$biconditional(id(F_1), id(F_2)),$ $atLeastOne(\{F_1, F_2, \dots, F_n\})$	if (Messaging > 0) sum (SMS, MMS) in {1..2} else SMS = 0, MMS = 0
ALTERNATIVE		$biconditional(id(F_1), id(F_2)),$ $atMostOne(\{F_1, F_2, \dots, F_n\})$	if (Screen > 0) sum (Basic, Color, highResolution) in {1..1} else Basic = 0, Color = 0, HighResolution = 0
REQUIRES		$implies(id(F_1), id(F_2))$	if (Games > 0) JavaSupport > 0
EXCLUDES		$implies(id(F_1), not(id(F_2)))$	if (GPS > 0) Basic = 0

Table 3.1: Mapping from a feature model to CSP.

3.3.2 Propositional logic

There are a number of approaches in the literature proposing the use of propositional formulas for the automated analysis of feature models. A propositional formula consists of a set of primitive symbols or variables, which are interpreted as either true or false, and a set of logical connectives (e.g., \wedge , \vee ,

\neg , \Rightarrow , \Leftrightarrow). If the values of all variables are given, the formula determines a unique truth value. It has been proved that every propositional formula can be converted into an equivalent *Conjunctive Normal Form* (CNF) [34]. A CNF is a standard form to represent propositional formulas where only \wedge , \vee , \neg connectives are allowed. We may remark that a propositional formula can be considered an special instance of CSP, where the variables are boolean and constraints are composed of logical connectives.

In general, the mapping of a feature model into a propositional formula follows four steps, namely:

- i. each feature of the feature model becomes a variable in the propositional formula;
- ii. each relationship of the model becomes one or more small formulas; the specific formula depends on the type of relationship. Some auxiliary variables can be needed;
- iii. a constraint assigning true to the variable that represents the root is defined, *i.e.*, " $\text{root} \Leftrightarrow \text{true}$ ";
- iv. the resulting formula is the conjunction of all the resulting formulas of step *ii* plus the additional constraint of step *iii*.

After the mapping of feature models into a propositional formula, an off-the-shelf solver is used to automatically analyse the formula and thus the feature model. The most commonly used solvers are SAT [18], and BDD [142].

Propositional satisfiability problem (SAT) is the problem of determining if there is a variable assignment that makes a given propositional formula (in CNF) evaluates to true. SAT is a well-known NP-complete problem [34]. However, the advances in SAT solving algorithms have significantly improved SAT solver tools allowing them to decide the satisfiability of industrial problems with tens of thousands of variables and millions of clauses [83].

Another way of determining if a given formula is satisfiable is by using a *Binary Decision Diagram* (BDD) solver. This is a software package that takes a propositional formula as input (not necessarily in CNF) and translates it into a BDD (*i.e.*, a data structure that is used to represent a propositional formula). The formula being represented and the ordering of the variables determine the size of the BDD. It is of crucial importance to care about variable ordering since in the worst case it increases exponentially the size of the structure. It

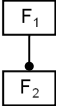
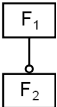
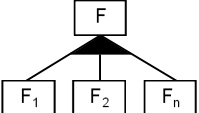
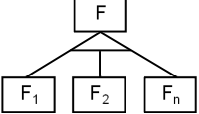
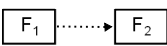
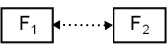
	Relationship	Propositional logic mapping	Mobile phone example
MANDATORY		$F_1 \leftrightarrow F_2$	MobilePhone \leftrightarrow UtilityFunctions MobilePhone \leftrightarrow Settings UtilityFunctions \leftrightarrow Calls UtilityFunctions \leftrightarrow Messaging UtilityFunctions \leftrightarrow AlarmClock UtilityFunctions \leftrightarrow RingingTones Settings \leftrightarrow Screen Media \leftrightarrow MP3
OPTIONAL		$F_2 \rightarrow F_1$	Media \rightarrow MobilePhone GPS \rightarrow UtilityFunctions Games \rightarrow UtilityFunctions JavaSupport \rightarrow Settings Camera \rightarrow Media
OR		$F \leftrightarrow (F_1 \vee F_2 \vee \dots \vee F_n)$	Messaging \leftrightarrow (SMS \vee MMS)
ALTERNATIVE		$(F_1 \leftrightarrow (\neg F_2 \wedge \dots \wedge \neg F_n \wedge F)) \wedge$ $(F_2 \leftrightarrow (\neg F_1 \wedge \dots \wedge \neg F_n \wedge F)) \wedge$ $(F_n \leftrightarrow (\neg F_1 \wedge \dots \wedge \neg F_{n-1} \wedge F))$	$(\text{Basic} \leftrightarrow (\neg \text{Color} \wedge \neg \text{HighResolution} \wedge \text{Screen})) \wedge$ $(\text{Color} \leftrightarrow (\neg \text{Basic} \wedge \neg \text{HighResolution} \wedge \text{Screen})) \wedge$ $(\text{HighResolution} \leftrightarrow (\neg \text{Basic} \wedge \neg \text{Color} \wedge \text{Screen}))$
REQUIRES		$F_1 \rightarrow F_2$	Games \rightarrow JavaSupport
EXCLUDES		$\neg(F_1 \wedge F_2)$	$\neg(\text{GPS} \wedge \text{Basic})$

Table 3.2: Mapping from a feature model to propositional logic.

is usually possible to find a good variable ordering that reduces this size, but finding the best variable ordering is an NP-complete problem [20]. On the other hand, BDD solvers are able to count the number of possible solutions very efficiently [26].

In Table §3.2, we list the concrete rules for translating each type of relationship in the feature model into a small formula, as provided by Benavides et al. [11]. In the same figure, as an example, we add the mapping of the mobile phone feature model presented in Figure §2.1.

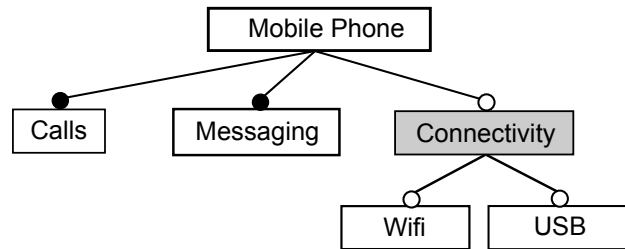


Figure 3.7: Feature model with abstract features.

Valid Products	Program Variants
{MobilePhone, Calls, Messaging}	{MobilePhone, Calls, Messaging}
{MobilePhone, Calls, Messaging, Connectivity}	
{MobilePhone, Calls, Messaging, Connectivity, Wifi}	{MobilePhone, Calls, Messaging, Wifi}
{MobilePhone, Calls, Messaging, Connectivity, USB}	{MobilePhone, Calls, Messaging, USB}
{MobilePhone, Calls, Messaging, Connectivity, Wifi, USB}	{MobilePhone, Calls, Messaging, Wifi, USB}

Connectivity is an abstract feature

Table 3.3: Multiple products may result in the same program variant.

3.4 Automated analysis of feature models with abstract features

Most of the approaches that provide automated support for the analysis of feature models aim at reasoning about the combination of all features (valid products) represented by the feature model. They do not take into account whether these features are relevant at implementation level or not. In order to reason about the combinations of features that are relevant at implementation level, which they refer to as program variants, Thüm et al. [126] have proposed to explicitly denote abstract features in feature modelling. Figure §3.7 depicts an example of a feature model, in which feature Connectivity is abstract.

Abstract features are features used usually to structure a feature model, but that do not have any impact at implementation level, *i.e.*, selecting or removing abstract features does not make any difference in the generated code. Therefore, the set of valid products and the set of program variants are not

equivalent. Usually, multiple products represented by a feature model result in the same program variant. For example, the model in Figure §3.7 represents five valid products, however, they result in only 4 different program variants, as shown in Table §3.3. In Section §5, we comment and discuss the possible impact of abstract elements on the analysis operations.

3.5 Summary

In this chapter, we have presented the main concepts in the context of the automated analysis of feature models. We have focused on the analysis operations and the different kinds of automated support found in the literature. In particular, we have considered those operations that are more relevant in the context of our dissertation, and those related works proposing support for the analysis of feature models, which are based on propositional logic or constraint programming.

Part III

Our Contribution

Chapter 4

Motivation

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them.

*William Bragg, 1862–1942
British physicist and Nobel Prize in Physics 1915*

Our goal in this chapter is to introduce the problems identified during our research and to motivate this dissertation. We analyse the current solutions and bring attention to the contributions we have done to solve the problems. In Section §4.1, we motivate our research. In Section §4.2, we present the main problems addressed in this dissertation. In Section §4.3, we review the current solutions found in the literature. In Section §4.4, we resume, analyse, and compare current solutions and contextualise our contributions. Finally, Section §4.5 summarises the chapter.

4.1 Introduction

Variability models have a key role in documenting variability of software product lines, allowing the variability management, especially during product derivation [119]. The automated analysis of variability models is an active research topic that has received attention of many researchers during the last twenty years [13]. Most of this research has been focused on feature models, resulting in a number of analysis operations, techniques and tools to automate the analysis of this kind of models.

The variability in variability models can be extended with measurable attributes (e.g., CPU and memory consumption) and constraints on these attributes, resulting in attribute-aware variability models. For example, in cases in which there are limitations of resources, such as memory capacity and CPU time, the derivation of products that do not satisfy those conditions must be avoided. The automated analysis of attribute-aware variability models is also important to guarantee that the derived software products reach the desired attribute constraints. The early analysis of this kind of models is particularly important, since any anomaly should be identified before the derivation of specific products. Furthermore, the wrong use of constraints on attributes may cause anomalies in the specification, leading to contradictory or to misleading information about the scope of the product line. Such anomalies should be detected and avoided to assure that desired products can be configured. If any anomaly is not detected early, all products that were developed based on the anomalous domain artefacts have to be corrected. This can lead to high cost and effort.

The increasing interest in variability modelling techniques has led to research papers and industrial experiences on the use, formalisation and application of different variability models, such as the works presented by Bühne et al. [27], Dhungana et al. [39, 42], Forster et al. [55], Lauenroth and Pohl [76], Loughran et al. [79], Mazo et al. [84], Metzger et al. [91], Petersen et al. [101], Schmid et al. [115], Sinnema et al. [120], Sun et al. [123], and Dhungana et al. [41]. Therefore, due to the existence of different variability modelling techniques, different organisation needs, and a multi-organisation software development environment, it is necessary to rely on a support for the automated analysis of variability models beyond feature models. Furthermore, this analysis should take into account attribute-aware variability models. For example, as we describe in §1.2.3, this necessity is identified in the context of multi product lines, as shown in Figure §1.5, on page 16.

In the literature we have found a number of approaches to perform the automated analysis of variability models; however, the vast majority of them are focused on feature models and hardly address attribute-aware variability models. Our main motivation in this dissertation is to provide support for the automated analysis of both OVMs and attribute-aware OVMs. There is so far only one approach dealing with the automated analysis of OVMs. Nevertheless, that proposal does not solve many of the problems we have addressed in this dissertation, which are as follows:

- i. Elements cannot be made explicit in the OVM language .
- ii. None of the approaches provide attribute-aware OVMs.
- iii. The automated analysis of OVMs has been little explored.
- iv. None of the approaches provide support for the automated analysis of attribute-aware OVMs.

We are aware that feature models are one of the most common approaches employed to represent variability of software product lines. However, there may be some cases where OVM can be considered as a more appropriate option [16, 27, 79]. We believe that it would be of practical importance to have interoperability between FaMa-FM and FaMa-OVM analysis tools (see Section §1.2.3). In addition, we believe that the interoperability between different variability modelling tools can become useful in a software ecosystems built specifically to support software product line development. Currently, there is no such support.

In the context of FaMa ecosystem we are already able to automatically analyse feature models, and we intend to provide support for OVMs. A tool support for transforming feature models into OVM and vice versa would support the interoperability between FaMa-FM and FaMa-OVM, allowing the engineers of a product line to work with different views of the product line variability. On the one hand, a hierarchical view of all features of the product line represented with feature models, and on the other hand, another view where only the variation points are visualised represented with OVM, see Figure §4.1. Currently, there is no tool support that provides interoperability between feature models and OVM.

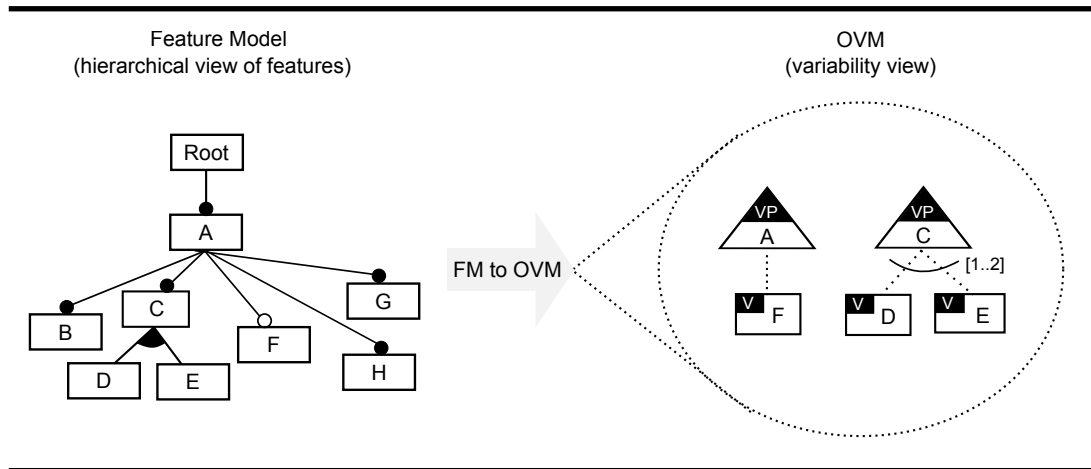


Figure 4.1: Different views of variability of a software product line.

4.2 Problems

The main contributions of this dissertation were motivated by the following problems:

i. IMPROVEMENTS ON MODELLING CONCEPTS

Abstract elements should be made explicit in the OVM. Currently, there is no way to explicitly denote abstract elements in an OVM. Abstract elements are used to structure the variability, but are not relevant for the analysis.

The lack of a technique to associate attributes with OVM. Currently, when using OVM to represent variability in software product line development, there are no techniques to specify attributes and constraints on them. When attributes are specified, and thus can be associated to variability in the OVM, the analysis of attribute-aware OVMs can be carried out.

ii. AUTOMATED ANALYSIS OF OVMs

The lack of automated support for the analysis of OVMs. In the software product line community, it is well-known that variability in prod-

uct lines is increasing. OVMs may have thousands of variants and variation points, and complex dependencies amongst these elements as well. It is practically impossible to manually carry out the analysis of OVMs, and besides, it is an error-prone task. The automated analysis of OVM has hardly been explored by the research community. Moreover, the analysis of OVMs considering abstract elements has not been addressed so far. In addition, we intend to endow the FaMa ecosystem by providing support for the automated analysis of OVMs.

The lack of automated support for the analysis of attribute-aware OVMs. In software product line engineering, analysis of attribute-aware variability models can be used to guarantee that the derived software products reach the desired attribute constraints. This analysis can *i)* detect anomalies that are introduced in the model when adding attributes specifications, and *ii)* support attribute-aware decisions. Due to the complexity of this analysis it is essential to rely on automated support.

iii. INTEROPERABILITY BETWEEN OVM AND FEATURE MODEL TOOLS

The lack of a comparison between feature model and OVM languages. The question of “*Which approach is the most suitable and in which context?*”, is not solved yet. A comparison study to identify the weaknesses and strengths of these languages, considering specific contexts, would be of interest for the software product line community.

Feature models to OVM transformation. Currently, there is no approach for the interoperability of feature models and OVM. It is likely that a software product line development team may need to work with different variability modelling techniques when working in collaboration with another team. It would be helpful for these teams to have an automatic transformation between both variability modelling languages.

4.3 Analysis of current solutions

In the next section, we present some related works found in the literature. We have grouped them according to the aforementioned problems: modelling concepts, automated analysis of OVMs, and interoperability between OVM and feature model tools.

4.3.1 Modelling Concepts

With regard to abstract elements, all the works we have found in the literature do not provide support for making explicit the abstract elements in OVM; all of them deal with feature models, as can be seen below.

Some authors have acknowledged that not all features are relevant at implementation level (*i.e.*, abstract features) [7, 143]; however, those authors only considered the implicit use of abstract features and did not provide a way of specifying them. Batory et al. [7] noted that some features are “empty” because no code needs to be written to implement that functionality. White et al. [143] commented that features that consume resources or affect implementation are most often the leaves of the feature model.

Schobbens et al. [116] distinguished two types of features, namely: primitive and compound. Primitive features are those that will influence in the final product, and compound features are just intermediate features used for decomposition. Metzger et al. [91] also distinguished two types of features, which they call primitive and non-primitive. The former are the set of features that the modeller considers relevant, whereas the latter are the ones considered irrelevant. In both approaches, the authors emphasised that primitive features and leaf features are different concepts, since non-leaf features can be deemed relevant by the modeller. For generality purposes, they leave it to the modeler to define which features in the feature diagram are relevant or not.

Thüm et al. [126] introduced the concept of abstract features, *i.e.*, features used to structure a feature model, but that do not have any impact on the implementation level. They argued that feature models should provide an explicit way of defining abstract features. Taking into account abstract features, analysis techniques can reason about the combination of all features as well as about the combinations of features relevant at implementation level. They provided a technique based on propositional formula to automate the analysis

of both approaches.

With regard to attribute-aware OVMs, we have not found any approach dealing with them. In the following, we report on those approaches that deal with other attribute-aware variability models.

Accordingly to Montagud and Abrahão [93], a number of research efforts have been made to capture attributes and their relationship with functional features. Most of them were focused on feature models.

Kang et al. [71] were the first to suggest the relationship between feature models and attributes. In their work, the authors contemplated the addition of feature attributes with quantified values. They also introduced the need to define relationships between features and attributes. Later, Kang et al. [72] made an explicit reference to non-functional features, which they defined as a kind of feature, referred to as *capability feature*. Other authors have also proposed the extension of feature models with so-called feature attributes [15, 37, 133]. Benavides et al. [15] coined the term *extended feature models* for feature models extended with attributes.

Zhang et al. [152] introduced Bayesian Belief Networks (BBN) for predicting the impact of feature selection on quality attributes. BBN represent domain experts' knowledge and experiences from the development of former similar projects. The author's approach can predict and assess the quality of a product line member by performing a quantitative analysis over BBN.

Etxeberria and Sagardui [46] presented a method for cost-effective quality evaluation of a product line taking into account variability on quality attributes. The authors used a quality feature tree to represent quality attributes, and provided a way to represent qualitative and quantitative impacts of functional features on quality attributes, as well.

Karataş et al. [74] proposed a language to express extended feature models. They addressed feature-attribute and attribute-attribute constraints.

Bagheri et al. [4] took into consideration the stakeholders' desired quality attributes (referred to as *soft constraints*) during a feature model configuration, and used a fuzzy propositional language for the analysis. They provided an interactive feature model configuration process, where they annotate features with high-level abstract objectives. By using fuzzy form, they express how features contribute to satisfy these objectives.

Zhang et al. [149] proposed a quantitative analysis method to identify and

measure the impact functional features can have on quality attributes, based on the judgement of domain experts. The authors used a pair-wise comparison method called Analytical Hierarchical Process to measure the relevant importance of functional features on quality attributes. Later, in [150], the authors developed an evaluation method to ensure the correctness of judgements of domain experts, and in [151], they developed a quality attribute knowledge base (QA_KB) to represent the inter-relationships amongst quality attributes. This approach can assist application engineers during the product configuration process; allowing validation of quality requirements and modification of configured product to satisfy quality requirements, and giving indicators for feature selection. For this purpose, the authors provided an ad-hoc algorithm.

Sinnema et al. [120] proposed COVAMOF, which is a variability modelling framework for modelling variabilities and constraints on attributes. In COVAMOF, attributes are expressed as dependencies related to variation points. These dependencies have, amongst other things, a function that determines their values, depending on the selected variants, and a constraint on these values. The value of the dependencies can represent formal or informal knowledge. The former is represented using algebraic expressions, and the latter can contain or refer to documented knowledge (e.g., HTML documents). They do not provide automated support for the analysis of COVAMOF.

4.3.2 Automated analysis of OVMs

In the literature we have found several proposals that provide support for the automated analysis of variability models, however, only Metzger et al. [91] have focused on OVMs. Next, we report on these proposals.

Metzger et al. [91] proposed to separate variability into two dimensions, namely: product line variability and software variability. The former refers to variability in the problem space, whereas the latter refers to variability in the solution space. These authors proposed to use OVM to represent product line variability and feature models to represent software variability. Furthermore, they provided an approach to automatically analyse the relationship between both models. As part of their work, they provide an indirect way to automatically analyse OVMs. First, they transform an OVM into a Varied Feature Diagram (VFD⁺), which is a formal "back-end" language used to define semantics and automating analysis, and in doing so, they reuse the semantics of analysis operations on VFD⁺. To carry out this transformation, they provide

an ad-hoc algorithm. Second, they map the VFD^+ to a propositional formula and then automatically analyse the OVM by means of SAT4j solver [18].

Accordingly to Benavides et al. [13], there are several proposals providing automated analysis of basic or cardinality-based feature models. Several analysis operations on feature models and, as well as automated support for them were proposed. These approaches can be classified in four different groups, according to the logic paradigm or method used to provide automated support, namely: *i)* constraint programming, *ii)* propositional logic, *iii)* description logic, *iv)* proposals integrating more than one paradigm and/or solver, and *v)* works that provide ad-hoc tools not categorised in the former groups.

Dhungana et al. [39] proposed the decision-oriented variability modelling language DOPLERVML as part of the DOPLER tool suite [40]. They were the first authors to address the formal definition of decision-oriented variability modelling approaches, by defining the formal semantics of DOPLERVML. In DOPLERVML, *decisions* define variability in the problem space while *assets* define the reusable artefacts and their dependencies in the solution space. The problem and the solution space are linked with *inclusion conditions*. Based on the formal semantics, the authors provided support to generate all possible configurations represented by the DOPLER variability model. They also developed a tool that, based on decisions taken by the user, can arrive at a set of included assets for a specific product configuration. Later, Mazo et al. [84] provided an approach to automate the analysis of DOPLERVML. They presented a mapping from DOPLERVML into CSP, and then used the constraint solver GNU Prolog [60] to automatically verify the variability model. The authors introduced four analysis operations applicable to DOPLERVML, namely: *void model*, *non-attainable validity conditions and domains value*, *dead decisions and assets*, and *redundant relationships*.

With regards to automated analysis of attribute-aware OVMs, we have not found any approach that provides support for it. Therefore, next, we report on those approaches that deal with the automated analysis of other attribute-aware variability models.

Benavides et al. [15] proposed the automated analysis of extended feature models by using constraint programming. In this work, the authors presented a simple and high-level example where they illustrate a possible mapping from feature attributes to CSP. In addition, they proposed a number of analysis operations on the extended feature models.

Djebbi et al. [43] provided a set of rules to translate feature models into

boolean constraints in order to extract information from feature models. Their analysis method is carry out by using queries. They described a tool under development based on constraint programming using GNU-Prolog solver, which provides support for the optimisation operation.

White et al. [143] proposed a method known as Filtered Cartesian Flattening to solve the problem of optimally selecting a set of features that simultaneously satisfy a number of resource constraints. They applied several existing algorithms to this problem, which performed much faster and offered an approximate solution.

Tun et al. [133] separated feature descriptions into three feature models relating to the requirements, the problem world context, and the specifications. These feature models may contain quantitative attributes and constraint on attributes. Once requirements are selected for a desired product, one or more products that satisfy the requirements and the quantitative constraints are generated. In addition, they provided a way to configure, from a feature model, solutions that satisfy quality requirements and quantitative constraints, as well.

Karataş et al. [74] proposed a language to build extended feature models and provided a mapping from these models to a CSP. They used the CLP(FD) constraint solver [30] to implement their approach, enabling the automated analysis of extended feature models involving more complex relationships.

4.3.3 Interoperability between OVM and feature model tools

Currently, there is no approach providing support for the interoperability between OVM and feature model tools. In addition, as far as we know, there is only one work comparing both feature model and OVM languages [68]. In that work, the authors propose a metamodel-based classification of variability modelling techniques. Although their goal was to study the fundamental characteristics of variability models in order to choose the most appropriate one for a particular application context, they provided only a first step in this direction. They did not provide any criteria for the comparison nor provided any example of applications of the studied variability models.

4.4 Discussion

A summary of the related works is shown in Table §4.1. Proposals are listed horizontally. The properties we use to compare the proposals are listed vertically, and are grouped into modelling concepts and automated analysis. They are as follows: *i) abstract elements*: the proposal provides explicit mechanism to denote abstract elements in the variability model; *ii) attributes*: the proposal provides explicit mechanism to relate attributes and variability models; *iii) without attributes*: the proposal provides automated support for the analysis of variability models without attributes; *iv) with attributes*: the proposal provides automated support for the analysis of attribute-aware variability models; and, *v) with abstract elements*: the proposal provides automated support for the analysis of variability models considering abstract elements.

The cells of the matrix indicate the information about a proposal in terms of property supported. Cells marked with '+' indicate that the proposal of the row provides explicit support for the property of the column. The last two columns on the right hand-side add information about the variability models addressed by each proposal, and group the proposals according to the paradigm they use for the analyses. The variability models approached are: feature model (F), decision model (D), COVAMOF (C), and OVM (O). This information is also reported in the final rows of Table §4.1. The paradigm used are: *i) Propositional Logic (PL)*; *ii) Constraint Programming (CP)*; *iii) (Multi)*, works that integrate more than one paradigm and/or solver; and *iv) (Others)*, proposals that use their own tools not categorised in the former groups. The cell marked with '-' indicate that the proposals in these rows do not provide automated support for the analysis of variability models.

With regarding to abstract elements, from the analysis of the related works, we have identified that Thüm et al. [125] and Schobbens et al. [116] were the only authors that proposed a way to differentiate abstract features from non abstract features in feature models. Furthermore, we have not found in the literature any approach dealing with abstract elements in OVM. In this dissertation, as part of our contributions, we provide a way to express abstract elements in OVM for analysis purpose, which is a novel approach.

Regarding attribute-aware variability models, we have realised that the addition of attributes to variability models has mostly been proposed for feature model approaches. In [15] the authors provided a way to add attributes to features, but they do not provide much detail about the values of the attributes, as acknowledged in [13], and besides, relationships between attributes are hier-

	Modelling Concepts		Automated Analysis				
	Abstract elements	Attributes	without attributes	with attributes	with abstract elements		
Proposals							
Gheyi et al. [58], Sun et al. [124]			+			F	PL
Mannion [81], Mannion and Camara [82]			+			F	
Mendonça et al. [87, 88]			+			F	
Thüm et al. [125], Batory [5]			+			F	
van der Storm [135, 136], Yan et al. [147]			+			F	
Zhang et al. [153, 154, 155]			+			F	
Czarnecki and Kim [38]		+	+			F	
Thüm et al. [126]	+		+		+	F	
Metzger et al. [91]			+			F and O	
Djebbi et al. [43], Benavides et al. [14]			+			F	CP
Trinidad et al. [127, 129], White et al. [144, 146]			+			F	
Benavides et al. [9, 10]		+	+			F	
Benavides et al. [15], Karataş et al. [74]		+	+	+		F	
Mazo et al. [84]			+			D	
Benavides et al. [11, 12]			+			F	Multi
Segura [117]			+			F	
Tun et al. [133]		+	+	+		F	
Zaid et al. [148], Wang et al. [140, 141]			+			F	Others
Fan and Zhang [49], Cao et al. [29]			+			F	
van Deursen and Klint [137], Bachmeyer and Delugach [3]			+			F	
von der Massen and Lichter [138, 139]			+			F	
Gheyi et al. [59], Hemakumar [66]			+			F	
Mendonça et al. [86], Fernandez-Amoros et al. [51]			+			F	
Osman et al. [96, 97], Salinesi et al. [112]			+			F	
van den Broek and Galvao [134]			+			F	
Kang et al. [71]		+	+			F	
White and Schmidt [145], White et al. [143]			+	+		F	
Zhang et al. [152], Bagheri et al. [4]		+	+	+		F	
Kang et al. [72], Czarnecki et al. [37]		+	+			F	
Dhungana et al. [39, 40]			+			D	
Zhang et al. [149, 150, 151]		+				F	
Etxeberría and Sagardui [46]		+				F	
Schobbens et al. [116]	+					F	
Sinnema et al. [120]		+				C	
OUR PROPOSAL	+	+	+	+	+	O	CP

F = Feature model

D = Decision model

C = COVAMOF

O = OVM

Table 4.1: Summary of related works.

archically organised in the feature tree, *i.e.*, the relationship between attributes exists only between a parent and a child. In this dissertation, we propose a more comprehensive approach to express attributes and constraints. In DOPLERVML approach, attributes were not explicitly defined for decision models, and in COVAMOF [120], attributes are only related to variation points, but not to variants, the direct impact a variant can have on another cannot be specified. Although Metzger et al. [91] have explored the automated analysis of OVMs, they do not provide support for attributes. In this dissertation, we propose a technique to relate attributes and constraints on attributes to variability in the OVM. We were inspired by the ideas presented in [74] to define a model for expressing attributes.

With regarding to the automated analysis of variability models without attributes, we have concluded that most of the current literature provide support for the automated analysis of feature models. Regarding the DOPLERVML, its formal semantics presented in [39] allowed devising algorithms for automating operations on decision-oriented variability models. The first work in this direction was presented by Mazo et al. [84], in which the authors provided automated support for the analysis of DOPLERVML. Furthermore, as far as we know, there is no automated support for the analysis of COVAMOF variability models. To the best of our knowledge, there is only one approach that deals with the automated analysis of OVMs [91] without attributes, which address few analysis operations and is done indirectly, by means of feature model semantics. In this dissertation, we provide support to automate the analysis of OVMs without attributes. For this purpose, we define a number of analysis operations on OVMs, provide a set of mappings rules to transform directly OVM to CSP, and provide tool support.

We also may remark that we did not find any related work considering analysis of attribute-aware OVMs. Etxeberria and Sagardui [46] focused on the evaluation process of the product line using feature models, and do not provide support for the automated analysis of attribute-aware feature models. Other authors have looked at more qualitative means of evaluating attribute constraints based on goal-oriented analysis [4, 133], both works are focused on feature models analysis. In this dissertation, we propose an approach to automate the analysis of attribute-aware OVMs. We consider that the works presented in [143] and [149] are complementary to ours. The former research results could be applied to our optimisation problem in order to improve scalability, and the latter could be used as a way of identifying and measuring values of attributes. In addition, we have realised that the key aspect to automate the analysis of attribute-aware variability models is the technique or

paradigm used for the analysis. Most of the approaches that deal with attributes propose the use of constraint programming to automate the analysis, since it allows to deal with non-boolean variables. In this dissertation, we propose to use constraint programming as the logical paradigm to automate the analysis of OVMs.

As far as we know, only Thüm et al. [125] provide support for the automated analysis of feature models considering abstract features. In this dissertation, we provide support for both the analysis of OVMs without abstract elements and the analysis of OVM with abstract elements.

With regarding to the interoperability between feature models and OVM, we may remark that we did not find any approach that provide support for the interoperability of feature model and OVM tools. We have identified only one work comparing both languages, which is still preliminary. As part of our contribution, we have informally compared both languages and identified some differences between them [106], however, our results are a first step towards a more rigorous comparison study. We intend to continue this work as future work. In addition, we have proposed an algorithm to transform a feature model into an OVM [107], and implemented it by using a model-driven development approach. This work also are going to be continued. This discussion is not summarised in Table §4.1.

4.5 Summary

In this chapter, we have presented the main problems that motivated our research. We have reviewed the related literature and compared our contribution with others regarding to modelling concepts, automated analysis of variability models, and interoperability between feature model and OVM tools. We have realised that most of the works focus on feature models, and that the analysis of attribute-aware variability models has received little attention by the research community. We have identified only one proposal providing automated analysis of OVMs without attributes; which does not provide support for specifying attribute-aware OVMs, for their automate analysis, nor for the automated analysis of OVMs with abstract elements. We have emphasised that our contribution provides support to denote abstract elements and attributes in OVMs, for the automated analysis of OVMs with, without attributes, and with abstract elements.

Chapter 5

Automated Analysis of OVMs

*If you don't know anything about computers,
just remember that they are machines that do exactly what you tell them
but often surprise you in the result.*

*Richard Dawkins, 1941 –
British evolutionary biologist and writer*

The aim of this chapter is to present a set of techniques to automate the analysis of OVMs. In Section §5.1, we describe the analysis process we propose to automate the analysis of OVMs. In Section §5.2, we provide information about abstract elements in OVMs. In Section §5.3, we introduce two different approaches to automate the analysis of OVMs, by providing two different mappings for translating an OVM into a CSP, namely: full mapping, and selective mapping. The former takes into account all the elements in the OVM, and the latter deals with abstract elements. In Section §5.4, we define a set of analysis operations on OVMs, apply them to the first approach, and give some examples. Then, in Section §5.5, we discuss the results of the operations defined in the previous section, when applied to the second approach. Finally, Section §5.6 summarises the chapter.

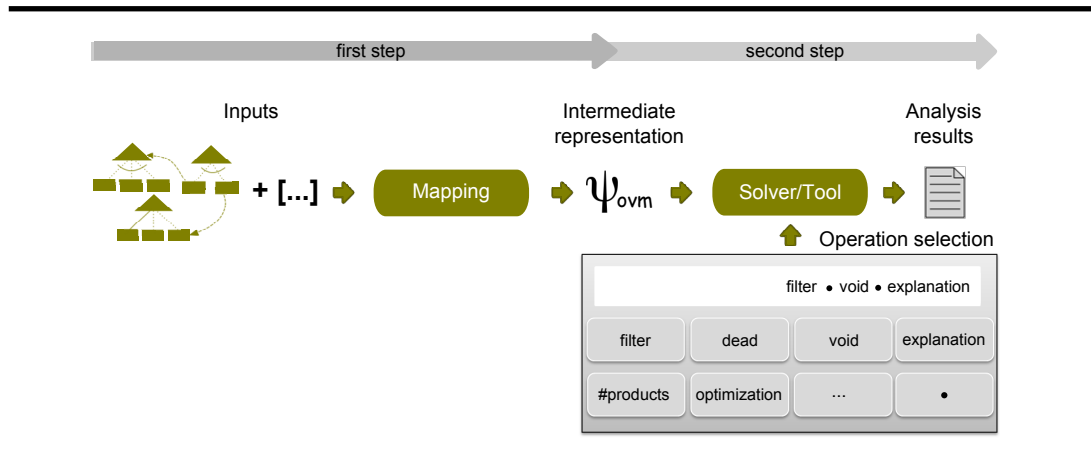


Figure 5.1: Process for the automated analysis of OVMs.

5.1 Introduction

The analysis of OVMs is performed in terms of analysis operations, which takes a set of parameter as input (e.g., an OVM) and returns a result as output. To automate this analysis, we follow the two-step process presented in Figure §5.1, which is similar to the process for the automated analysis of feature models. The process starts by translating the input parameters into a specific representation. Afterwards, an off-the-shelf solver is used to automatically analyse the representation of the input parameters and provide the result as an output.

In the next sections, we describe how to translate input parameters into a representation, in this case we use CSP as logical paradigm. Then, we define a set of analysis operations on OVM and give some examples.

5.2 Dealing with abstract elements

We consider that an OVM represents the set of all possible variations of a product line. When all elements are relevant for the analysis, variations are distinguished by the variability elements they have (i.e., variations points and variants). For example, the OVM example depicted in Figure §5.2 represents five different variations, namely: {Connectivity}, {Connectivity, Wifi},

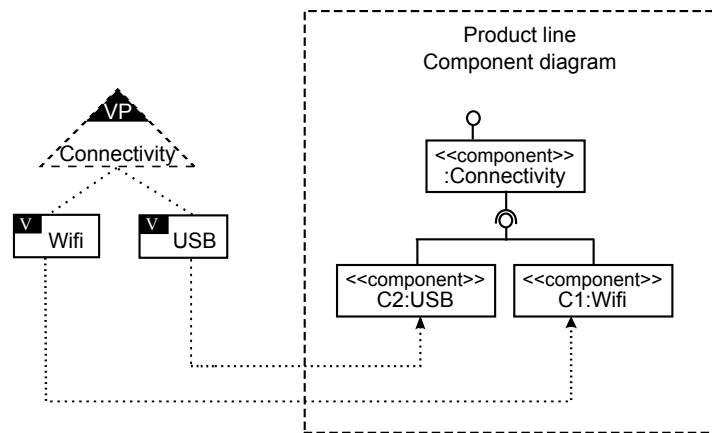


Figure 5.2: Product line component diagram with variability.

$\{\text{Connectivity, USB}\}$, $\{\text{Connectivity, Wifi, USB}\}$, and \emptyset . In this example, OVM documents the variability of a product line component diagram.

In some cases, as for example in testing activities, it is useful to be able to analyse the OVM taking into account only the elements that are relevant for the final product. A variability element is not always relevant for the final product. For example, variation $\{\text{Connectivity}\}$ would probably not have any impact on the design model, whereas $\{\text{Connectivity, Wifi}\}$, $\{\text{Connectivity, USB}\}$, and $\{\text{Connectivity, Wifi, USB}\}$ would result in three different models. In the example in Figure §5.2, variants *Wifi* and *USB* are realised by the components *C1:Wifi* and *C2:USB*, respectively. However, variation point *Connectivity* is used only to structure the different types of connectivity. In this case, the possible component diagrams derived from the OVM model depicted in Figure §5.2 would be as shown in Figure §5.3 (the grey area illustrate the variant being selected). The diagram depicted on the left hand-side results from the selection of variant *Wifi*, whereas the component diagram in the right hand-side represents the selection of variant *USB*, and, finally, the diagram shown in the bottom results from the selection of both variants *Wifi* and *USB*. Therefore, although the OVM depicted in Figure §5.2 represents five possible variations, only three of them are relevant at design model, namely: $\{\text{Connectivity, Wifi}\}$, $\{\text{Connectivity, USB}\}$, and $\{\text{Connectivity, Wifi, USB}\}$.

In OVM, the variability is structured by using two main elements, namely: variation points and variants. Considering that these two elements are already differentiated in the OVM abstract syntax, we propose to use variation

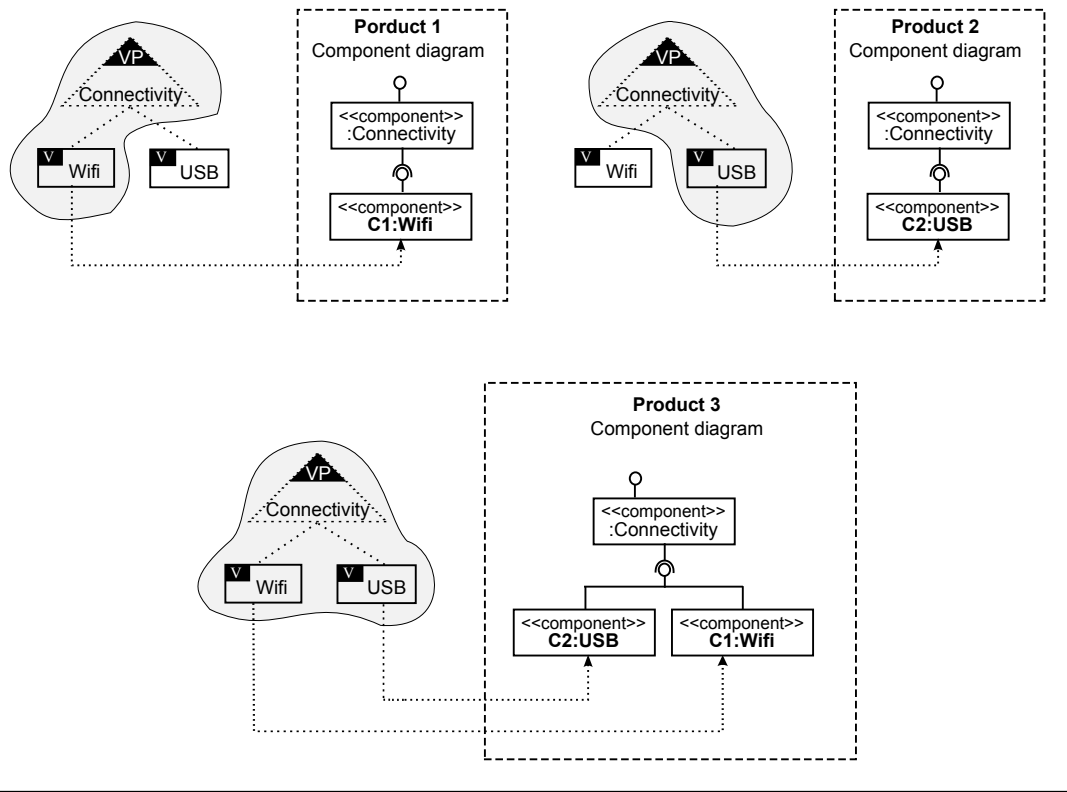


Figure 5.3: Product component diagrams.

points to make explicit the abstract elements in the OVM. Therefore, for analysis purposes, variation points can be considered concrete or abstract, resulting in two different OVM semantics, namely: *full* and *selective*. In the former, an OVM represents a set of sets of combinations of variation points and variants, meaning that all the variation points are concrete elements. In the latter, an OVM represents a set of sets of combinations of variants, meaning that all the variants are concrete elements and all the variation points are abstract. The decision about which semantics to use when analysing an OVM depends on which of them is more convenient to the user. Users may want to consider variation points only as a way of structuring variants, and therefore, do not consider these elements as part of variations. The set of variations obtained with *full* and *selective* semantics may not be equivalent.

5.3 Mapping OVM into Constraint Satisfaction Problem (CSP)

In order to be able to analyse OVMs using full and selective semantics, we provide two different mappings to translate an OVM into a CSP. In the first mapping, we consider every element as concrete which we denote as *Full mapping*. In the second, variants are concrete elements and variation points are abstract, which we denote as *Selective mapping*.

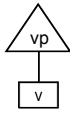
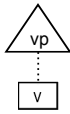
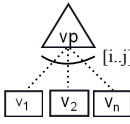
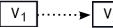
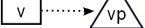
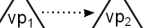

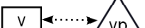
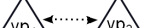
5.3.1 Full mapping

As previously explained, a CSP is defined as a set of variables, a set of domains for those variables, and a set of constraints restricting the values of those variables. Therefore, to carry out the full mapping from an OVM to CSP following the mapping in feature models, the 3-tuple $\psi_{\text{ovm}_f} = (V_{\text{ovm}_f}, D_{\text{ovm}_f}, C_{\text{ovm}_f})$ is built, where the variability elements (variation points and variants) in the OVM become variables in V_{ovm_f} with their respective domains in D_{ovm_f} , and the variability and constraint dependencies in the OVM become constraints in the C_{ovm_f} . To find solutions for the ψ_{ovm_f} , the CSP solver searches for a valid set of variable values that simultaneously satisfies all constraints.

The mapping of an OVM into a CSP can vary depending on the solver that is used later for the analysis. We use the generic CSP language provided by Benavides [8] to represent ψ_{ovm_f} . In order to represent cardinality of alternative relationship, we have added to this generic CSP language another operator called *inRangeOf*. The syntax of the operator *inRangeOf* is as follows: “*inRangeOf*([i..j], { v_1, v_2, \dots, v_n })”, where vp is the variation point, $v_k \mid k \in [1 \dots n]$ the set of optional variants in the relationship, and $[i \dots j] \mid 0 < i \leq j \leq n$ the cardinality. Its semantics is as follows:

$$i \leq \left(\sum_{k=1}^n v_k \right) \leq j$$

Table §5.1 lists the rules for the full mapping of an OVM into a generic CSP, and concrete rules using JaCoP-like notation, which is one of the specific CSP solvers used in FaMa framework as mentioned in Section §1.2.2. The mapping from an OVM into a CSP is similar to the one used for feature models [15], with the following differences:

		OVM relationships*	Generic CSP	JaCoP-like notation
Variability Dependency	MANDATORY		$biconditional(id(vp), id(v))$	$vp = v$
	OPTIONAL		$implies(id(v), id(vp))$	if $(vp = 0)$ $v = 0$
	ALTERNATIVE		$biconditional(id(vp), inRangeOf([i..j], \{v_1, v_2 \dots v_n\}))$	if $(vp > 0)$ sum (v_1, v_2, \dots, v_n) in $\{i..j\}$ else $v_1 = 0, v_2 = 0, \dots, v_n = 0$
Constraint Dependency	REQUIRES		$implies(id(v_1), id(v_2))$	if $(v_1 > 0)$ $v_2 > 0$
			$implies(id(v), id(vp))$	if $(v > 0)$ $vp > 0$
			$implies(id(vp_1), id(vp_2))$	if $(vp_1 > 0)$ $vp_2 > 0$
	EXCLUDES		$implies(id(v_1), not(id(v_2)))$	if $(v_1 > 0)$ $v_2 = 0$
			$implies(id(v), not(id(vp)))$	if $(v > 0)$ $vp = 0$
			$implies(id(vp_1), not(id(vp_2)))$	if $(vp_1 > 0)$ $vp_2 = 0$

* Variation points can be optional or mandatory

Table 5.1: Full mapping from an OVM to CSP.

- i. In OVM there are two types of nodes, namely: variation points and variants. These nodes differ from each other. Contrarily, in feature models, all nodes in the diagram are features;
- ii. There is no constraint for a root node, since there is no root node in the OVM;
- iii. Variation points can be mandatory or optional. For every mandatory variation point, we add a constraint assigning value 1 to the correspondent variable, whereas for optional variation points no constraints are added.

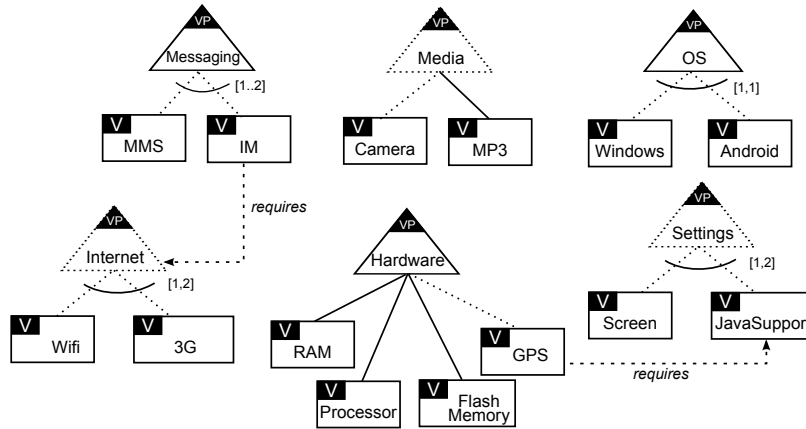


Figure 5.4: OVM for a mobile phone example.

- iv. Every alternative relationship is mapped into a constraint of the form “biconditional($\text{id}(\text{vp}), \text{inRangeOf}([i..j], \{v_1, v_2, \dots, v_n\})$ ”, which means that $\text{vp} \leftrightarrow i \leq (\sum_{k=1}^n v_k) \leq j$. This mapping is similar to cardinality-based feature models.

Table §5.2 shows how the example presented in Figure §5.4 can be represented as a CSP.

5.3.2 Selective mapping

There are two ways to approach the selective mapping. First, reducing the CSP generated from the full mapping rules, by removing all the variables equivalent to the variation points, as shown in Figure §5.5 (a). Second, defining a new set of mapping rules, enabling to directly map an OVM with abstract variation points into a CSP, as shown in Figure §5.5 (b).

The semantics of ψ'' and ψ''' are not necessarily equivalent. In the work presented by Thüm et al. [126], the authors followed the first approach, where they assume that the semantics of ψ'' is the result of removing abstract features from ψ' . This means that both semantics ψ' and ψ'' are strongly related. For example, when a feature model is void in ψ' , it is necessarily void in ψ'' .

	Mobile phone example	JaCoP-like notation
Variability Dependency		<p>Messaging = 1</p> <p>if (Messaging > 0) sum (MMS, IM) in {1..2} else MMS = 0, IM = 0</p>
		<p>Media = MP3</p> <p>if (Media = 0) Camera = 0</p>
		<p>OS = 1</p> <p>if (OS > 0) sum (Windows, Android) in {1..1} else Windows = 0, Android = 0</p>
		<p>if (Internet > 0) sum (Wifi, 3G) in {1..2} else Wifi = 0, 3G = 0</p>
		<p>Hardware = 1</p> <p>Hardware = RAM Hardware = Processor Hardware = FlashMemory</p> <p>if (Hardware = 0) GPS = 0</p>
		<p>if (Settings > 0) sum (Screen, JavaSupport) in {1..2} else Screen = 0, JavaSupport = 0</p>
Constraint Dependency		<p>if (IM > 0) Internet > 0</p>
		<p>if (GPS > 0) JavaSupport > 0</p>

Table 5.2: Full mapping from a mobile phone OVM to CSP.

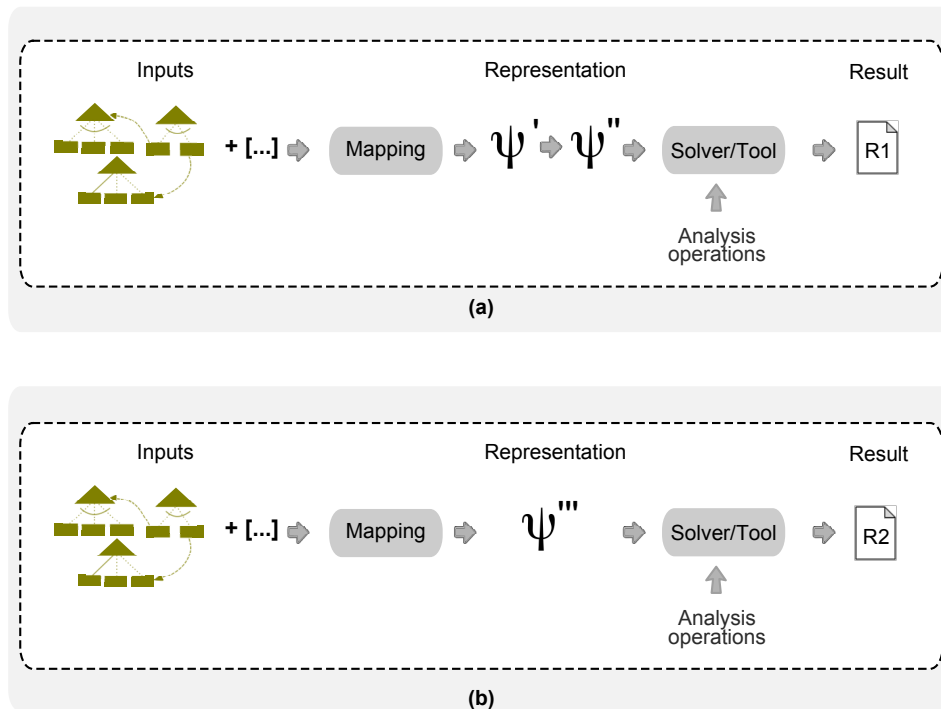


Figure 5.5: Two different approaches for the selective mapping.

However, we have realised that maybe, in the case of OVMs, it would be interesting to define a new OVM semantics to deal with abstract elements. To better illustrate this idea, we will use the scenario described below.

Let us assume a scenario in which a given mobile phone product line was devised to support one or more different mobile web browsers, namely: IEMobile, FirefoxMobile, and Safari. In addition, products from this product line optionally should support off-line GMail and Calendar, which are possible choices for the mandatory variation point GoogleGears, as shown in Figure §5.6. Now, let us suppose that as a consequence of a new business strategy, the company has decided to evolve the product line by adopting Safari as mandatory browser. This decision requires a modification in the OVM by changing the relationship between MobileWebBrowser and Safari to mandatory, as shown in Figure §5.7. In the meantime, the Safari was updated by its developers and has become incompatible with Google Gears applications. Thus, the product line had to be modified again to include an *excludes* constraint between Safari and GoogleGears, as shown in Figure §5.7.

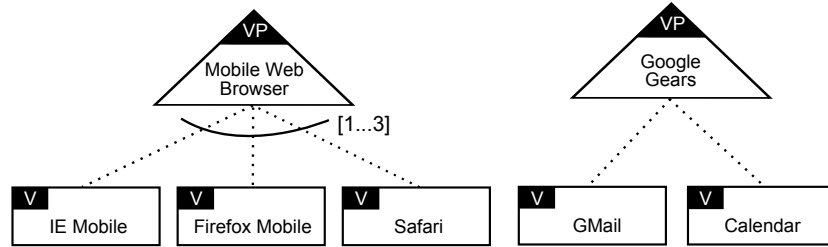


Figure 5.6: Variation points from the mobile phone product line.

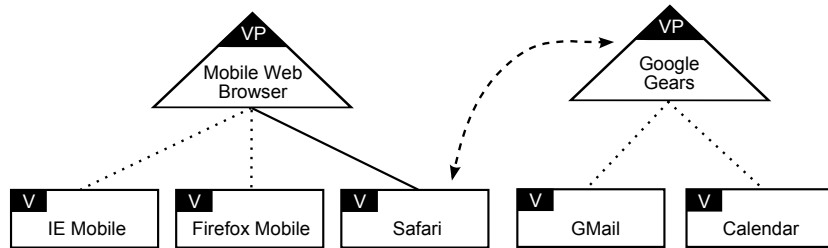


Figure 5.7: Evolution of the mobile phone product line.

Considering the previous scenario, if we apply the first approach depicted in Figure §5.5 (a) to the resulting OVM, the model is void, regardless if we consider abstract elements or not. This is due to the excludes constraint between two mandatory elements in the OVM, *i.e.*, Safari and GoogleGears. We argue that, when considering abstract elements, this model should not necessarily be void. The reason for this is that, GMail and Calendar are optionals and if none of them are selected, it should be possible to construct products without GoogleGears. In this way, the number of possible variations would be greater than zero, *i.e.*, the model would be not void. However, GMail and Calendar would be dead elements. Thus, we conclude that the second semantics seems to be more reasonable, since GoogleGears is abstract.

We propose to follow the second approach presented in Figure §5.5 (b) to define the selective mapping, and we conjecture that ψ'' and ψ''' may not be equivalent.

To carry out the selective mapping, we again have to build the 3-tuple, in this case $\psi_{\text{ovm}_s} = (V_{\text{ovm}_s}, D_{\text{ovm}_s}, C_{\text{ovm}_s})$. In the selective mapping, only the

variants in the OVM become variables in V_{ovm_s} with their respective domains in D_{ovm_s} . The selective mapping of an OVM into a CSP can vary depending on the solver that is used later for the analysis. The selective mapping of an OVM into a general CSP has the following general form:

- i. Every variant of the OVM is mapped into a variable of the CSP with a domain $\in [0, 1]$;
- ii. Every relationship of the model is mapped into a constraint depending on the type of the relationship; in such constraints only variants are allowed.

Unlike mapping rules for full mapping, the rules for selective mapping cannot consider variation points as part of constraints, but only variants. Therefore, each possible relationship between variation points and their child variants may result in a different constraint, leading to a greater number of rules. In order to know how many possible concrete rules we could obtain for this mapping, we have used the following scheme:

- i. There are two types of variation points: mandatory and optional. Each variation point can have three types of relationships with its child variants, namely: mandatory, optional, and alternative variability dependencies, shown in the OVM meta-model in Figure §2.7, on page 31. Therefore, there must be six possible variability dependency rules, which we denote by the set VD .
- ii. There are three types of requires constraints, namely: *VP requires VP*, *V requires VP*, and *V requires V*, and there are three types of excludes constraints, namely: *VP excludes VP*, *V excludes VP*, and *V excludes V*.
- iii. As said in item *i*, variation points can be related with their child variants in 6 different ways represented by VD . Therefore, *a*) the number of possible combinations in *VP requires VP* is $|VD \times VD|$, which means $6 \times 6 = 36$; *b*) the number of possible combinations in *V requires VP* is $|VD|$, since there are only one possible V , and *c*) the number of possible combinations in *V requires V* is one. Summarizing, we may state that the number of rules to transform requires constraints into CSP is $|VD \times VD| + |VD| + 1 = (6 \times 6) + 6 + 1 = 43$
- iv. The same reasoning is applied to *VP excludes VP*, *V excludes VP*, and *V excludes V*, in which the number of rules are respectively $|VD \times VD|$,

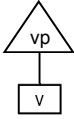
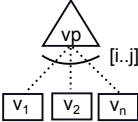
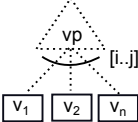
		Variability dependency	JaCoP-like notation
Mandatory	Variation Point	1	
	Mandatory		$v = 1$
Alternative	Variation Point	2	
	Alternative		$\text{sum}(v_1, v_2, v_n) \text{ in } \{i..j\}$
Optional	Variation Point	3	
Alternative	Alternative		$\text{sum}(v_1, v_2, v_n) \text{ in } \{0\} \cup \{i..j\}$

Table 5.3: Mapping variability dependencies into CSP.

$|VD|$, and one. Summarizing, we may state that the number of rules to transform excludes constraints into CSP is $|VD \times VD| + |VD| + 1 = (6 \times 6) + 6 + 1 = 43$

- v. Summarizing, we may state that the number of rules to transform an OVM into CSP, for the selective mapping, is the sum of the number of possible combinations presented in *i*, *iii* and *iv*, as follows:

$$|VD| + 2.(|VD \times VD| + |VD| + 1) = 6 + 2(36 + 6 + 1) = 92$$

All the 92 possible rules can be seen in Appendix §B. These mapping rules can be generalised resulting in 25 rules. Tables §5.3, §5.4, and §5.5, shows the 25 concrete rules for mapping variability dependencies, require constraints, and excludes constraints into a CSP, respectively.

In addition to these mapping rules, in Table §5.6 we show an example on how the product line depicted in Figure §5.4 can be mapped into a CSP, when considering selective mapping.

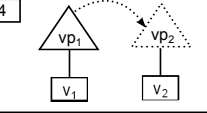
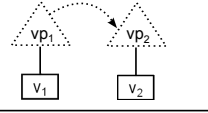
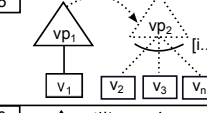
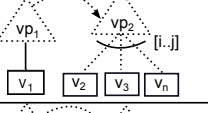
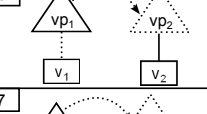
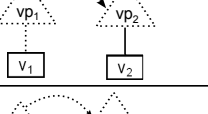
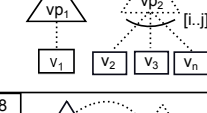
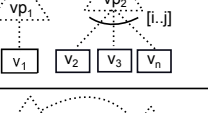
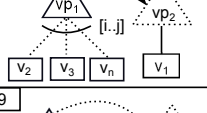
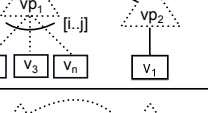
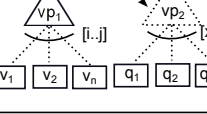
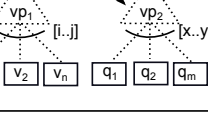
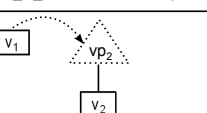
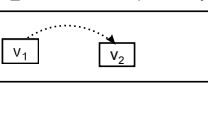
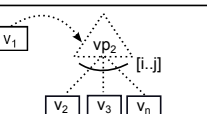
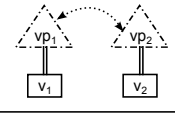
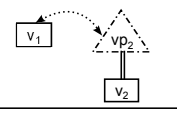
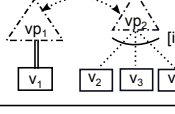
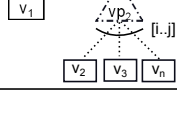
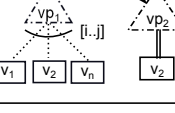
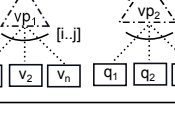
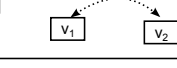
	Requires_VP_VP Constraint dependency	JaCoP-like notation		Requires_VP_VP Constraint dependency	JaCoP-like notation	
Mandatory Variation Point (left-hand side)	4		$v_2 = 1$	10		$\text{if } (v_1 > 0) \\ v_2 > 0$
	5		$\text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$	11		$\text{if } (v_1 > 0) \\ \text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$
	6		$v_2 = 1$	12		$\text{if } (v_1 > 0) \\ v_2 > 0$
	7		$\text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$	13		$\text{if } (v_1 > 0) \\ \text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$
	8		$v_1 = 1$	14		$\text{if } \text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\} \\ (v_1 > 0)$
9		$\text{sum}(q_1, q_2, q_m) \text{ in } \{x..y\}$	15		$\text{if } \text{sum}(v_1, v_2, v_n) \text{ in } \{i..j\} \\ \text{sum}(q_1, q_2, q_m) \text{ in } \{x..y\}$	
	16		$\text{if } (v_1 > 0) \\ v_2 > 0$	18		$\text{if } (v_1 > 0) \\ v_2 > 0$
	17		$\text{if } (v_1 > 0) \\ \text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$			

Table 5.4: Mapping requires constraints into CSP.

5.4 Analysis operations for the full mapping

Before going into the analysis operations details, we have to revisit some concepts described in Section §3.2, namely: *configuration*, *full configuration*, and *partial configuration*. In addition, we have to define a new concept to which we refer to as *variation*.

Excludes_VP_VP Constraint dependency		CSP	Excludes_V_VP Constraint dependency		CSP	
Mandatory or Optional Variation Point	19		if $(v_1 > 0)$ $v_2 = 0$	23		if $(v_1 > 0)$ $v_2 = 0$
	20		if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$	24		if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$
	21		if $\text{sum}(v_1, v_2, v_n)$ in $\{i..j\}$ $(v_2 = 0)$	Excludes_V_V Constraint dependency		CSP
	22		if $\text{sum}(v_1, v_2, v_n)$ in $\{i..j\}$ $q_1 = 0, q_2 = 0, q_m = 0$	25		if $(v_1 > 0)$ $v_2 = 0$

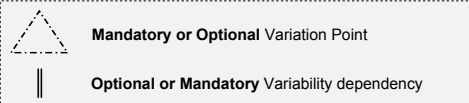


Table 5.5: Mapping excludes constraints into CSP.

- **Configuration.** Given an OVM with a set of variation points VP and variants V, and a set of variability elements VE such that $VE = VP \cup V$, a configuration is a 2-tuple of the form (S,R) such that $S, R \subseteq VE$ being S the set of elements to be selected and R the set of elements to be removed such that $S \cap R = \emptyset$. There are two types of configurations, as follows:

- ◊ **Full configuration.** If $S \cup R = VE$ the configuration is called *full configuration*. As an example, a possible full configuration (FC) for the model in Figure §5.4 can be as follows:

$$FC = (\{\text{Messaging, IM, OS, Android, Internet, 3G, Hardware, FlashMemory, Processor, RAM}\}, \{\text{GPS, Settings, Screen, JavaSupport, Wifi, MMS, Windows, Media, Camera, MP3}\})$$

- ◊ **Partial configuration.** If $S \cup R \subset VE$ the configuration is called *partial configuration*. As an example, a possible partial configuration (PC) for the model in Figure §5.4 can be as follows:

$$PC = (\{\text{Messaging, IM, OS, Android}\}, \{\text{GPS}\})$$

- **Variation.** A variation is equivalent to a full configuration where only selected variability elements are specified and omitted variability elements are implicitly removed. For instance, the following variation (VAR) is equivalent to the full configuration described above:

$$VAR = \{\text{Messaging, IM, OS, Android, Internet, 3G, Hardware, FlashMemory, Processor, RAM}\}$$

	OVM relationships	JaCoP-like notation
Variability Dependency		$\text{sum}(\text{MMS}, \text{IM}) \in \{1..2\}$
		no constraint
		$\text{sum}(\text{Windows}, \text{Android}) \in \{1..1\}$
		$\text{sum}(\text{Wifi}, \text{3G}) \in \{0\} \cup \{1..2\}$
		RAM = 1 Processor = 1 FlashMemory = 1
		$\text{sum}(\text{Screen}, \text{JavaSupport}) \in \{0\} \cup \{1..2\}$
Constraint Dependency		if (IM > 0) $\text{sum}(\text{Wifi}, \text{3G}) \in \{1..2\}$
		if (GPS > 0) JavaSupport > 0

Table 5.6: Selective mapping from a mobile phone OVM to CSP.

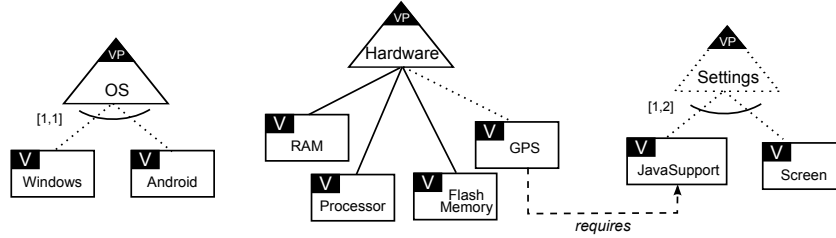


Figure 5.8: Excerpt of the OVM for the mobile phone product line.

In the following sections, we define a number of analysis operations on OVM, and provide some examples.

5.4.1 Variations

This operation takes an OVM as input and returns all the variations represented by the input model. Roughly speaking, an OVM is a compact representation of all the possible variations that can be derived from a set of variation points. In contrast to feature models, which represent a set of products (*i.e.*, combination of features), OVM represents the possible variations of software artefacts. Thus, when interpreting an OVM as a CSP, we can understand each solution of the CSP as a variation of the OVM, and the set of all solutions of this CSP as all possible variations of the OVM. Therefore, the *Variations* operation is defined as follows:

Operation 1 (Variations). Let ovm be an OVM and ψ_{ovm_f} be its equivalent CSP. Variations of ovm correspond to the set of solutions of ψ_{ovm_f} .

$$\text{variations}(ovm) = \text{sol}(\psi_{ovm_f})$$

The following variations are derived from the OVM in Figure §5.8:

- $\text{VAR}_1 = \{\text{OS, Hardware, Android, FlashMemory, Processor, RAM}\}$
- $\text{VAR}_2 = \{\text{OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, Screen}\}$
- $\text{VAR}_3 = \{\text{OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, JavaSupport}\}$
- $\text{VAR}_4 = \{\text{OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, Screen, JavaSupport}\}$
- $\text{VAR}_5 = \{\text{OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, JavaSupport, GPS}\}$

$VAR_6 = \{OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, Screen, JavaSupport, GPS\}$
 $VAR_7 = \{OS, Hardware, Windows, FlashMemory, Processor, RAM\}$
 $VAR_8 = \{OS, Hardware, Windows, FlashMemory, Processor, RAM, Settings, Screen\}$
 $VAR_9 = \{OS, Hardware, Windows, FlashMemory, Processor, RAM, Settings, JavaSupport\}$
 $VAR_{10} = \{OS, Hardware, Windows, FlashMemory, Processor, RAM, Settings, Screen, JavaSupport\}$
 $VAR_{11} = \{OS, Hardware, Windows, FlashMemory, Processor, RAM, Settings, JavaSupport, GPS\}$
 $VAR_{12} = \{OS, Hardware, Windows, FlashMemory, Processor, RAM, Settings, Screen, JavaSupport, GPS\}$

5.4.2 Number of variations

This operation takes an OVM as input and returns the number of variations represented by the input model. When the number of variations represented by the OVM is zero, the model is void.

Operation 2 (*Number of variations*). Let ovm be an OVM and ψ_{ovm_f} be its equivalent CSP. The number of variations of ovm corresponds to the number of solutions of ψ_{ovm_f} .

$$\#variations(ovm) = |\text{sol}(\psi_{ovm_f})|$$

As an example, the OVM depicted in Figure §5.8, on page 82, represents 12 variations, whereas the OVM depicted in Figure §5.4, on page 73, represents 360 variations.

5.4.3 Filter

This operation takes as inputs an OVM and a configuration (potentially partial) and returns the set of variations derived from the input model that match the given configuration. Thus, filtering an OVM can be interpreted as adding additional constraints to the CSP, which usually will reduce the set of solutions, and thus, the set of possible variations of the OVM. Therefore, the *Filter* operation is defined as follows:

Operation 3 (*Filter*). Let ovm be an OVM and ψ_{ovm_f} be its equivalent CSP, and let C be a configuration of the form (S,R) . The filtered model is the set of solutions of the conjunction of ψ_{ovm_f} and the equivalent constraints of a configuration.

$$\text{filter}(\text{ovm}, C) = \text{sol}(\psi_{\text{ovm}_f} \wedge \bigwedge_{e_i \in S} e_i = 1 \wedge \bigwedge_{e_i \in R} e_i = 0)$$

For example, the set of variations derived from the OVM in Figure §5.8, when applying a filter specified by partial configuration ($\{\text{Android}\}, \{\text{GPS}\}$), where **Android** is selected and **GPS** is removed, are as follows:

VAR₁ = {OS, Hardware, Android, FlashMemory, Processor, RAM}
 VAR₂ = {OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, Screen}
 VAR₃ = {OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, JavaSupport}
 VAR₄ = {OS, Hardware, Android, FlashMemory, Processor, RAM, Settings, Screen, JavaSupport}

5.4.4 Void OVM

This operation takes an OVM as input and returns a value (true or false) informing whether such OVM is void or not. An OVM is void if it does not represent any variation. An OVM can become void due to the wrong use of constraint dependencies, *i.e.*, requires and excludes.

Operation 4 (Void OVM). Let *ovm* be an OVM and ψ_{ovm_f} be its equivalent CSP. Then, *ovm* is void if the set of variations is empty.

$$\text{void}(\text{ovm}) \Leftrightarrow \text{variations}(\text{ovm}) = \emptyset$$

Figure §5.9 shows how an OVM example with two variation points can become void by the wrong use of the *excludes* constraint. Let us consider ψ_{ovm_1} and ψ_{ovm_2} the equivalent CSPs for the two models in Figure §5.9 (a) and Figure §5.9 (b), respectively. Therefore, the variations for both ψ_{ovm_1} and ψ_{ovm_2} are as follows:

$\text{sol}(\psi_{\text{ovm}_1}) = \{ \{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, MMS} \},$
 $\{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, IM} \},$
 $\{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, MMS, IM} \},$
 $\{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, MMS, GPS} \},$
 $\{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, IM, GPS} \},$
 $\{ \text{Hardware, FlashMemory, Processor, RAM, Messaging, MMS, IM, GPS} \} \}$

$\text{sol}(\psi_{\text{ovm}_2}) = \emptyset$

In Figure §5.9 (a), model *ovm*₁ is not void, but after adding an *excludes*, as shown in Figure §5.9 (b), it becomes void. The *excludes* between **Messaging** and **RAM** makes the selection of both variability elements impossible, which

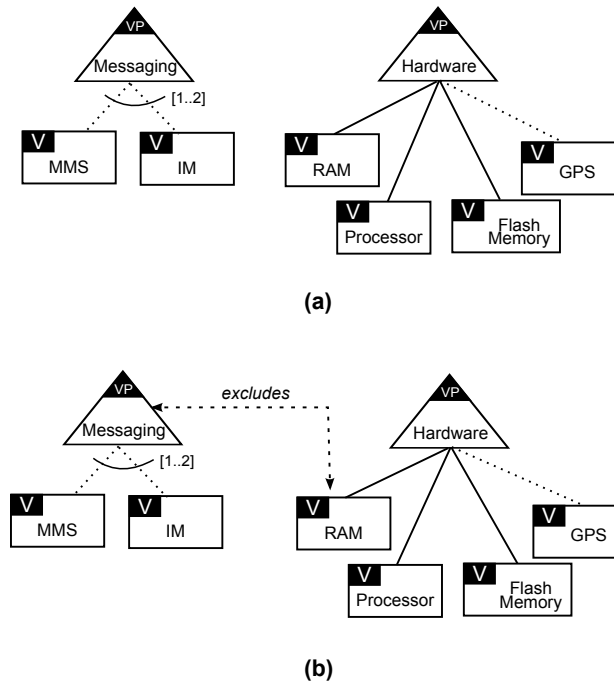


Figure 5.9: Example of an OVM becoming void.

leads the model to a contradiction since **Messaging** and **RAM** are both mandatory and thus must be in all variations.

5.4.5 Valid configuration

This operation takes an OVM and a configuration as inputs and returns a value (true or false) informing whether such configuration is valid to such OVM or not. A configuration, be it full or partial, is valid if when applied to an OVM, it is possible to derive at least one variation from this model.

Operation 5 (Valid configuration). Let ovm be a valid OVM and ψ_{ovm_f} be its equivalent CSP. A configuration C is valid if after applying a filter specified by C to ovm , the set of solutions of ψ_{ovm_f} is not empty.

$$\text{validConf}(ovm, C) \Leftrightarrow |\text{filter}(ovm, C)| > 0$$

As an example, consider the model in Figure §5.4 on page 73 and the following partial configurations, PC_1 and PC_2 :

$PC_1 = (\{IM, Android\}, \{Internet\})$
 $PC_2 = (\{MMS, Android\}, \{3G\})$

Partial configuration PC_1 is not valid since the selection of IM requires selection of Internet, thus PC_1 contradicts the relationship specified in the OVM. Contrarily, partial configuration PC_2 is valid since the selection of MMS and Android, and removal of 3G is allowed.

5.4.6 Valid variation

This operation takes an OVM and a variation as inputs and returns a value (true or false) informing whether such variation is valid or not. A variation is valid if it belongs to the set of variations represented by the OVM.

Operation 6 (Valid variation). Let ovm be a valid OVM and ψ_{ovm_f} be its equivalent CSP. A variation var is valid if var belongs to the set of variations of ovm .

$$\text{validVariation}(ovm, var) \Leftrightarrow var \in \text{variations}(ovm)$$

As an example, if we consider the OVM in Figure §5.4 on page 73, variations VAR_1 and VAR_2 , described below, both are not valid. Variation VAR_1 is not valid since it does not include all the mandatory variability elements; it misses out FlashMemory. Similarly, variation VAR_2 is not valid, since one and only one of the operating systems should be selected.

$VAR_1 = \{\text{Messaging, MMS, OS, Android, Hardware, RAM, Processor}\}$
 $VAR_2 = \{\text{Messaging, MMS, OS, Android, Windows, Hardware, RAM, Processor, FlashMemory}\}$

On the other hand, variations VAR_3 and VAR_4 , described below, do belong to the set of variations represented by the OVM in Figure §5.4.

$VAR_3 = \{\text{Messaging, MMS, OS, Android, Hardware, RAM, Processor, FlashMemory}\}$
 $VAR_4 = \{\text{Messaging, MMS, OS, Android, Hardware, RAM, Processor, FlashMemory, Internet, 3G}\}$

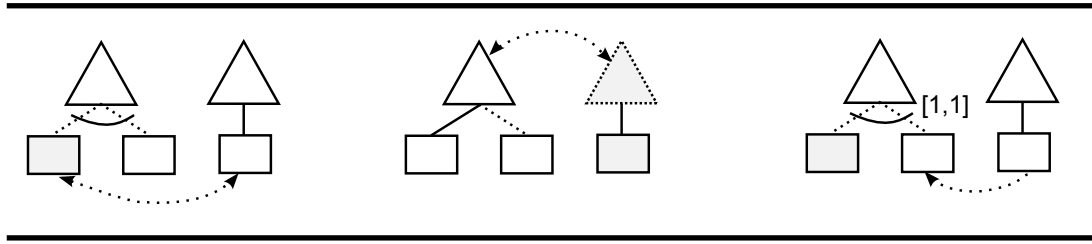


Figure 5.10: Typical cases of dead elements in OVM.

5.4.7 Dead elements

This operation takes an OVM as input and returns the set of dead elements (if any). A variability element is dead if it does not belong to any of the possible variations represented by the model. It can become dead due to the wrong use of constraint dependencies. This is an undesirable situation since the model gives to the user a false view of the product line variability.

Operation 7 (Dead element). Let ovm be an OVM, and let E be a variability element that belongs to ovm , and let ψ_{ovm_f} of the form $(V_{ovm_f}, D_{ovm_f}, C_{ovm_f})$ be the equivalent CSP and e the mapped variable. E is dead if there is no variation v in the set of variations of ovm such that e is an element of the set v .

$$isDead(ovm, E) \Leftrightarrow \nexists v \in variations(ovm) \bullet e \in v$$

Figure 5.10 shows some typical cases that generate dead elements in OVM (dead elements are depicted in grey).

5.4.8 False optional elements

This operation takes an OVM as input and returns the set of false optional elements (if any). A variability element is false optional if being modelled as optional, still appears in all variations derived from the model. Just as with dead elements, false optional elements are generated by the wrong use of constraint dependencies, and they give a wrong idea of the product line variability as well.

Operation 8 (False Optional). Let ovm be an OVM, and let E be a variability element that belongs to ovm , and let ψ_{ovm_f} of the form $(V_{ovm_f}, D_{ovm_f}, C_{ovm_f})$

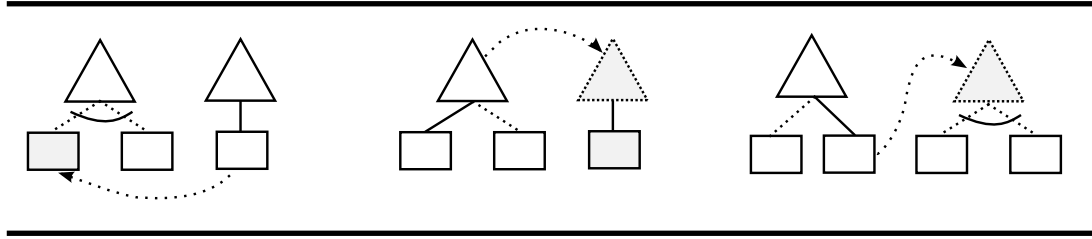


Figure 5.11: Typical cases of false optional elements in OVM.

be the equivalent CSP and e the mapped variable. E is false optional if E is optional and for all variations v in the set of variations of ovm , e is an element of the set v .

$$\text{isFalseOpt}(ovm, E) \Leftrightarrow \text{isOptional}(E) \wedge (\forall v \in \text{variations}(ovm) \bullet e \in v)$$

The predicate `isOptional` determines whether a given element has to be interpreted as an optional element or not. Figure §5.11 shows some typical cases that generate false optional elements in OVM (false optional elements are depicted in grey).

5.4.9 Commonality degree

This operation takes as inputs an OVM and a variability element and returns a value that represents the percentage of variations represented by the OVM in which the input variability element appears. For example, if there are 10 possible variations and a variant v appears in 5 variations, the commonality of v is 0.5.

Operation 9 (CommonalityDegree). Let ovm be an OVM, and let E be a variability element that belongs to ovm , and let ψ_{ovm_f} of the form $(V_{ovm_f}, D_{ovm_f}, C_{ovm_f})$ be the equivalent CSP and e the mapped variable. Commonality degree of E is the relation between the number of variations derived from ovm when applying a filter specified by a configuration $(\{E\}, \emptyset)$, and the number of all the possible variations derived from ovm .

$$\text{commonalityDegree}(ovm, E) = \frac{|\text{filter}(ovm, (\{E\}, \emptyset))|}{|\text{variations}(ovm)|}$$

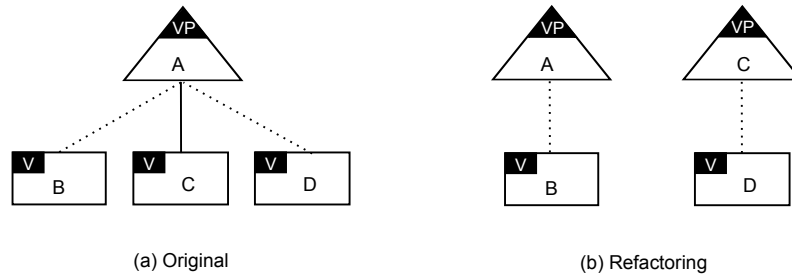


Figure 5.12: Refactoring between two OVMs.

For instance, the commonality degree of variant **Android** in the model in Figure §5.8 is 0.5 since **Android** is included in 6 out of 12 variations, as can be seen in Section §5.4.1. This means that **Android** appears in 50% of the variations of the product line. This operation can easily be generalised in order to receive as input a partial configuration instead of a single variability element.

5.4.10 Refactoring

Refactoring is an edit operation that takes two different OVMs as inputs and returns a value informing whether the models are equivalent or not. A model g is a refactoring of another model f when f and g represent the same set of variations while having a different structure. The set of variability elements in both models is not necessarily the same.

Operation 10 (Refactoring). Let f and g be two OVMs, where $f \neq g$, and ψ_f and ψ_g be their equivalent CSPs. Then, g is a refactoring of f if the set of variations represented by f is equal to the set of variations represented by g .

$$\text{isRefactoring}(f, g) \Leftrightarrow \text{variations}(f) = \text{variations}(g)$$

For instance, the OVM in Figure §5.12 (b) is a refactoring of the OVM in Figure §5.12 (a). Both models represent the same set of variations, *i.e.*, $\{\{A, C\}, \{A, C, B\}, \{A, C, D\}, \{A, C, B, D\}\}$. We may remark that in this case, there is no distinction between variation points and variants. Although **C** is a variant in the model represented in Figure §5.12 (a), and a variation point in the other, in terms of variations they are not distinguished.

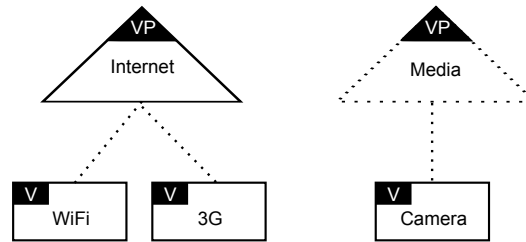


Figure 5.13: Sample OVM with two variation points.

5.5 Analysis operations for the selective mapping

The operations defined in Section §5.4 can also be used for the analysis of OVMs when using the selective mapping; they have the same definition for the two different mappings. What changes from one approach to another is how configuration and partial configuration are defined. Unlike in the full mapping, where the set of variability elements, VE , is such that $VE = VP \cup V$, in the selective mapping, $VE = V$, where V is the set of variants in the OVM. Although the operations are the same, their results may not be. In this section, we focus on those operations that may give us different results from those obtained when applying the full mapping.

5.5.1 Number of variations and all variations

The set of all possible variations will change when we consider only variants as members of variations. Let us consider the example OVM in Figure §5.13. In this example, when applying the full mapping, twelve variations can be derived, as shown in Table §5.7. However, when considering selective mapping, some of those variations are equivalent, namely, VAR1 and VAR2, VAR3 and VAR4, VAR5 and VAR6, and VAR7 and VAR8, since Internet and Media are abstract and thus do not form part of variations. Consequently, eight valid variations can be derived from the model when applying selective mapping.

Valid variations		
Full mapping	VAR1 = {Internet}	
	VAR2 = {Internet, Media}	
	VAR3 = {Internet, Wifi}	
	VAR4 = {Internet, Wifi, Media}	
	VAR5 = {Internet, 3G}	
	VAR6 = {Internet, 3G, Media}	
	VAR7 = {Internet, Wifi, 3G, }	
	VAR8 = {Internet, Wifi, 3G, Media}	
	VAR9 = {Internet, Media, Camera}	
	VAR10 = {Internet, Wifi, Media, Camera}	
	VAR11 = {Internet, 3G, Media, Camera}	
	VAR12 = {Internet, Wifi, 3G, Media, Camera}	
Selective mapping	VAR1 = {Internet}	VAR1 = \emptyset
	VAR2 = {Internet, Media}	VAR2 = {Wifi}
	VAR3 = {Internet, Wifi}	VAR3 = {3G}
	VAR4 = {Internet, Wifi, Media}	VAR7 = {Wifi, 3G}
	VAR5 = {Internet, 3G}	VAR4 = {Camera}
	VAR6 = {Internet, 3G, Media}	VAR5 = {Wifi, Camera}
	VAR7 = {Internet, Wifi, 3G}	VAR6 = {3G, Camera}
	VAR8 = {Internet, Wifi, 3G, Media}	VAR8 = {Wifi, 3G, Camera}
	VAR9 = {Internet, Media, Camera}	
	VAR10 = {Internet, Wifi, Media, Camera}	
	VAR11 = {Internet, 3G, Media, Camera}	
	VAR12 = {Internet, Wifi, 3G, Media, Camera}	

Table 5.7: Different set of variations depending on the approach.

5.5.2 Void OVM

An OVM is void if it does not represent any variation. A model can be void when applying full mapping, but not void when applying selective mapping. This is because not all constraints (excludes and requires) that lead to a void model in the first case, lead to a void model in the second one. Let us observe the OVM example in Figure §5.7. This OVM is void when applying the full mapping, but not void when applying the selective mapping. The full mapping of this model to a CSP results in eight rules, as shown in Table §5.8. Note that constraints C_1 , C_2 , and C_3 make constraint C_8 inconsistent, so that there is no solution for the resulting CSP. Therefore, the number of variations represented by this model is zero. Contrarily, when applying the selective mapping

	Full mapping	Selective mapping
Constraints	$C_1 : \text{MobileWebBrowser} = 1$	$C_1 : \text{Safari} = 1$
	$C_2 : \text{GoogleGears} = 1$	$C_2 : \text{if}(\text{Safari} > 0) \text{GMail} = 0$
	$C_3 : \text{Safari} = \text{MobileWebBrowser}$	$C_3 : \text{if}(\text{Safari} > 0) \text{Calendar} = 0$
	$C_4 : \text{if}(\text{MobileWebBrowser} = 0) \text{IEMobile} = 0$	
	$C_5 : \text{if}(\text{MobileWebBrowser} = 0) \text{FirefoxMobile} = 0$	
	$C_6 : \text{if}(\text{GoogleGears} = 0) \text{GMail} = 0$	
	$C_7 : \text{if}(\text{GoogleGears} = 0) \text{Calendar} = 0$	
	$C_8 : \text{if}(\text{Safari} > 0) \text{GoogleGears} = 0$	
Variations	\emptyset	$\text{VAR1} = \{\text{Safari}\}$ $\text{VAR2} = \{\text{FirefoxMobile}, \text{Safari}\}$ $\text{VAR3} = \{\text{IEMobile}, \text{Safari}\}$ $\text{VAR4} = \{\text{IEMobile}, \text{FirefoxMobile}, \text{Safari}\}$

Table 5.8: The set of variations represented by an OVM example.

to the same example, two rules are generated. These rules do not cause any inconsistency, thus resulting in a solution with four variations, namely: {Safari}, {FirefoxMobile, Safari}, {IEMobile, Safari}, and {IEMobile, FirefoxMobile, Safari}.

5.5.3 Dead and false optional

In the selective mapping, only variants can be identified as dead or false optional. Therefore, if there is any dead or false optional variation point in the model, there is no way to identify it when applying selective mapping. A variability element can be dead or false optional when applying the full mapping, but not when applying the selective mapping. This is because the set of variations derived from one model can vary from one approach to another. Figure §5.14 shows a new version of the previous model, but now the Excludes constraint is defined between FirefoxMobile and GoogleGears. If we analyse such model by using the full mapping, we find that FirefoxMobile is dead, since GoogleGears is part of all variants and it excludes FirefoxMobile, as can be seen in Table §5.9. However, FirefoxMobile is not dead when applying the selective mapping approach. In this case, as GoogleGears is abstract it is not part of all variants.

In Figure §5.15, we show another example in which some elements are

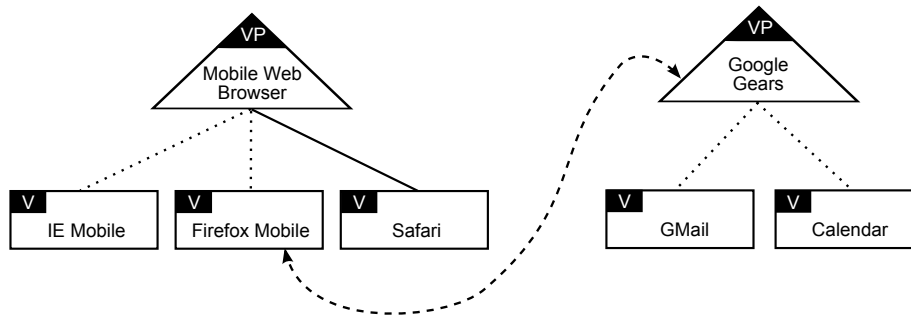


Figure 5.14: OVM with a dead element when applying the full mapping.

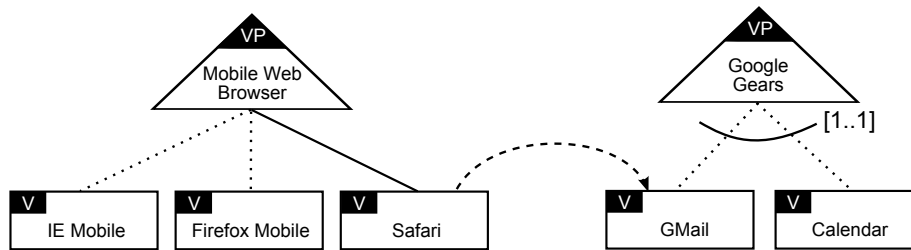


Figure 5.15: OVM with dead and false optional elements.

dead or false optional in both approaches. If we analyse this model by using both approaches, we find that *Calendar* is dead, since it does not appear in any variant; and *GMail* is false optional, since it is modelled as alternative but appears in all variations. Table §5.10 presents the set of variations represented by the model in Figure §5.15 and the dead and false optional elements.

5.5.4 Commonality degree

When applying the selective mapping, the commonality operation takes as inputs an OVM and a variability element and returns a value that represents the percentage of valid variations represented by the OVM in which such variability element appears. This definition is similar to that in Section §5.4.9, but here the variability element must be a variant. As an example, let us compute the commonality degree of each variability element of the sample in Fig-

	Full mapping	Selective mapping
Constraints	$C_1 : \text{MobileWebBrowser} = 1$	$C_1 : \text{Safari} = 1$
	$C_2 : \text{GoogleGears} = 1$	$C_2 : \text{if}(\text{FirefoxMobile} > 0) \text{GMail} = 0$
	$C_3 : \text{Safari} = \text{MobileWebBrowser}$	$C_3 : \text{if}(\text{FirefoxMobile} > 0) \text{Calendar} = 0$
	$C_4 : \text{if}(\text{MobileWebBrowser} = 0) \text{IEMobile} = 0$	
	$C_5 : \text{if}(\text{MobileWebBrowser} = 0) \text{FirefoxMobile} = 0$	
	$C_6 : \text{if}(\text{GoogleGears} = 0) \text{GMail} = 0$	
	$C_7 : \text{if}(\text{GoogleGears} = 0) \text{Calendar} = 0$	
	$C_8 : \text{if}(\text{FirefoxMobile} > 0) \text{GoogleGears} = 0$	
Variations	VAR1 = {MobileWebBrowser, GoogleGears, Safari}	VAR1 = {Safari}
	VAR2 = {MobileWebBrowser, GoogleGears, Safari, IEMobile}	VAR2 = {Safari, FirefoxMobile}
	VAR3 = {MobileWebBrowser, GoogleGears, Safari, Calendar}	VAR3 = {Safari, IEMobile}
	VAR4 = {MobileWebBrowser, GoogleGears, Safari, Calendar, IEMobile}	VAR4 = {Safari, IEMobile, FirefoxMobile}
	VAR5 = {MobileWebBrowser, GoogleGears, Safari, GMail}	VAR5 = {Safari, Calendar}
	VAR6 = {MobileWebBrowser, GoogleGears, Safari, GMail, IEMobile}	VAR6 = {Safari, Calendar, IEMobile}
	VAR7 = {MobileWebBrowser, GoogleGears, Safari, Calendar, GMail}	VAR7 = {Safari, GMail}
	VAR8 = {MobileWebBrowser, GoogleGears, Safari, Calendar, GMail, IEMobile}	VAR8 = {Safari, GMail, IEMobile}
		VAR9 = {Safari, Calendar, GMail}
		VAR10 = {Safari, Calendar, GMail, IEMobile}
D	FirefoxMobile	none
FO	none	none

D = Dead elements FO = False optional elements

Table 5.9: Dead elements.

ure §5.13 for the two approaches. As can be seen in Table §5.11, when applying the selective mapping, Wifi and 3G are more relevant for the product line than Camera, since their commonality degrees are 0.5, 0.5, and 0.33, respectively. However, in the selective mapping, the three variants have the same importance, since their commonality degree is 0.5. Therefore, the commonality degree of a given variant may change from one approach to another: the user has to decide which is the more appropriated approach before analysing the model.

	Full mapping	Selective mapping
Constraints	$C_1 : \text{MobileWebBrowser} = 1$	$C_1 : \text{Safari} = 1$
	$C_2 : \text{GoogleGears} = 1$	$C_2 : \text{if}(\text{Safari} > 0) \text{GMail} = 1$
	$C_3 : \text{Safari} = \text{MobileWebBrowser}$	
	$C_4 : \text{if}(\text{MobileWebBrowser} = 0) \text{IEMobile} = 0$	
	$C_5 : \text{if}(\text{MobileWebBrowser} = 0) \text{FirefoxMobile} = 0$	
	$C_6 : \text{if}(\text{GoogleGears} = 0) \text{GMail} = 0$	
	$C_7 : \text{if}(\text{GoogleGears} = 0) \text{Calendar} = 0$	
	$C_8 : \text{if}(\text{Safari} > 0) \text{GMail} = 1$	
Variations	$\text{VAR1} = \{\text{MobileWebBrowser}, \text{GoogleGears}, \text{Safari}, \text{GMail}\}$	$\text{VAR1} = \{\text{Safari}, \text{GMail}\}$
	$\text{VAR2} = \{\text{MobileWebBrowser}, \text{GoogleGears}, \text{Safari}, \text{GMail IEMobile}\}$	$\text{VAR2} = \{\text{Safari}, \text{GMail}, \text{FirefoxMobile}\}$
	$\text{VAR3} = \{\text{MobileWebBrowser}, \text{GoogleGears}, \text{Safari}, \text{GMail FirefoxMobile}\}$	$\text{VAR3} = \{\text{Safari}, \text{GMail}, \text{IEMobile}\}$
	$\text{VAR4} = \{\text{MobileWebBrowser}, \text{GoogleGears}, \text{Safari}, \text{GMail FirefoxMobile}, \text{IEMobile}\}$	$\text{VAR4} = \{\text{Safari}, \text{GMail}, \text{FirefoxMobile}, \text{IEMobile}\}$
D	Calendar	Calendar
FO	GMail	GMail

D = Dead elements FO = False optional elements

Table 5.10: Dead and false optional elements.

	Commonlaity				
	Internet	Media	Wifi	3G	Camera
Full mapping	$\frac{12}{12} = 1$	$\frac{8}{12} = 0.66$	$\frac{6}{12} = 0.5$	$\frac{6}{12} = 0.5$	$\frac{4}{12} = 0.33$
Selective mapping	-	-	$\frac{4}{8} = 0.5$	$\frac{4}{8} = 0.5$	$\frac{4}{8} = 0.5$

Table 5.11: Results of commonality operation.

5.5.5 Refactoring

This operation allows users to find out whether two OVMs are equivalent or not. In order to have a more accurate result, the user has to be aware that, depending on the approach, the result can be *yes* or *no*. For instance, let us analyse again the two models in Figure §5.12, on page 89, which were analysed previously and identified as being a refactoring, but now applying the selective mapping. In this case, the OVM in Figure §5.12 (b) is not a refactor-

	Model in Figure §5.12 (a)	Model in Figure §5.12 (b)
Variations	VAR1 = {C}	VAR1 = { \emptyset }
	VAR2 = {C, B}	VAR2 = {B}
	VAR3 = {C, D}	VAR3 = {D}
	VAR4 = {C, B, D}	VAR4 = {B, D}

Table 5.12: Models representing two different sets of variations.

ing of the OVM in Figure §5.12 (a), since these models do not represent the same set of variations, as can be seen in Table §5.12.

5.6 Summary

In this chapter, we have presented the analysis process we have used to automate the analysis of OVMS. This process is based on the mapping of OVM into CSP. We have provided two different approaches to map an OVM into CSP. These approaches differ mainly regarding which variability element can be part of a configuration, namely, variation points and variants, or only variants. Finally, we have defined a set of analysis operations that can be applied to both approaches, and we have discussed their possible results, which differ from one approach to another.

Chapter 6

Automated Analysis of Attribute-aware OVMs

It's more about good enough than it is about right or wrong.

*James Marcus Bach,
Software tester*

This chapter is devoted to presenting a technique to relate OVM and attributes – which we call Attribute-aware OVMs – and a technique to automate their analysis. In Section §6.1, we introduce the chapter. In Section §6.2 we discuss the challenges of relating variability in orthogonal variability models with attributes. In Section §6.3, we present the Attribute-based Model, which we use to document attributes and constraint on these attributes. Next, in Section §6.4, we describe the process to automate the analysis of Attribute-aware OVMs. In Section §6.5, we define a set of mapping rules to translate an Attribute-aware OVM into a CSP. In Section §6.6, we define a set of analysis operations to be performed on Attribute-aware OVMs. Finally, Section §6.7 summarises the chapter.

6.1 Introduction

In OVM, variability is defined by means of a set of rules which constraint the combination of reusable artefacts that compose a particular product within a product line. These rules are used to guide the derivation of products in application engineering. In the derivation process, variations are resolved by selecting options according to customer needs, but also respecting the variability modelling rules. For example, a mobile phone product line is planned to offer mobile phones with at least one and at most three types of connectivities, namely: *Bluetooth*, *USB*, and *Wifi*. Then, when deriving a specific product, one, two, or three of these options have to be selected.

Variability information may be not enough to guarantee the success of a product. Software engineers should be able to realise the impact of their choices on the attributes, such as memory consumption or development costs. Thus, decisions about the most suitable product to be built should take into account attributes. Furthermore, when there are limitations of resources such as memory and CPU capacity, the derivation of products that do not satisfy those conditions must be avoided.

In this chapter, we present the *Attribute-aware OVM* (AOVM), which is a technique we provide to express the relationship between OVM and attributes, and constraints on these attributes. In addition, we provide support to automate the analysis of AOVMs. This analysis aims to support the decisions of engineers, but it can be performed by any stakeholder that wants to obtain information from the AOVM.

6.2 Attribute-aware OVM

The specification of variability can be extended with measurable attributes and constraints on these attributes [13]. In feature models, the attributes annotate features, as illustrated in Figure §6.1. In this example, the feature model provides information about the transfer rate of each type of connectivity, and therefore *speed* is an attribute of features *Bluetooth*, *USB*, and *Wifi*.

Feature models document the variability of a product line in terms of features; they graphically represent all products of a product line. Therefore, when a feature model is extended with attributes, apart from variability, data information is also represented, such as features of a system. Contrarily, in

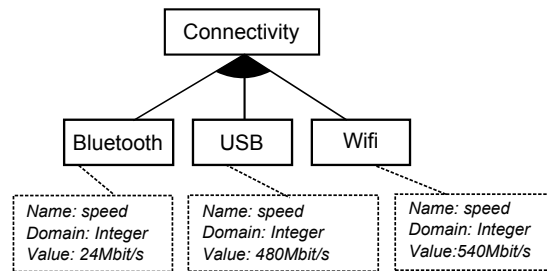


Figure 6.1: Excerpt of a feature model extended with attributes.

OVM the variability should be orthogonal to all models that contain data information, since OVM documents the variability of base models, as depicted in Figure §6.2. Consequently, there are two different possibilities when relating OVM to attributes, which represent two different problems:

- i. The OVM is directly related to attributes. In this case, OVM documents the variability of attributes, which are documented in a base model according to the OVM terminology. Figure §6.3 (a) shows an example of a base model (e.g., attribute information model) for representing three different levels of connectivity speed. In this example, the variability of the attribute information model is documented by the OVM, meaning that one and only one of the speed levels can be selected for each product.
- ii. The OVM is not directly related to attributes. The attributes refers to a base model (e.g., architecture, requirements, or configuration models), and the variability of this base model is documented by the OVM. In Figure §6.3 (b), we show an example of a configuration model with three components, namely Bluetooth, USB, and Wifi; and each component has an attribute Speed. Then, the variability of the configuration model is documented by the OVM.

Our contribution addresses the second problem, in which we consider that the base model is a configuration model and this model is related to attributes. Roughly speaking, a configuration model is a model that represents configuration problems [50, 53], and it is made up of a set of components (e.g., features or variants) and rules that constrain the possible combinations of these components. We assume that the rules of a configuration model can be represented in an OVM, and that attributes and constraints on attributes are expressed in

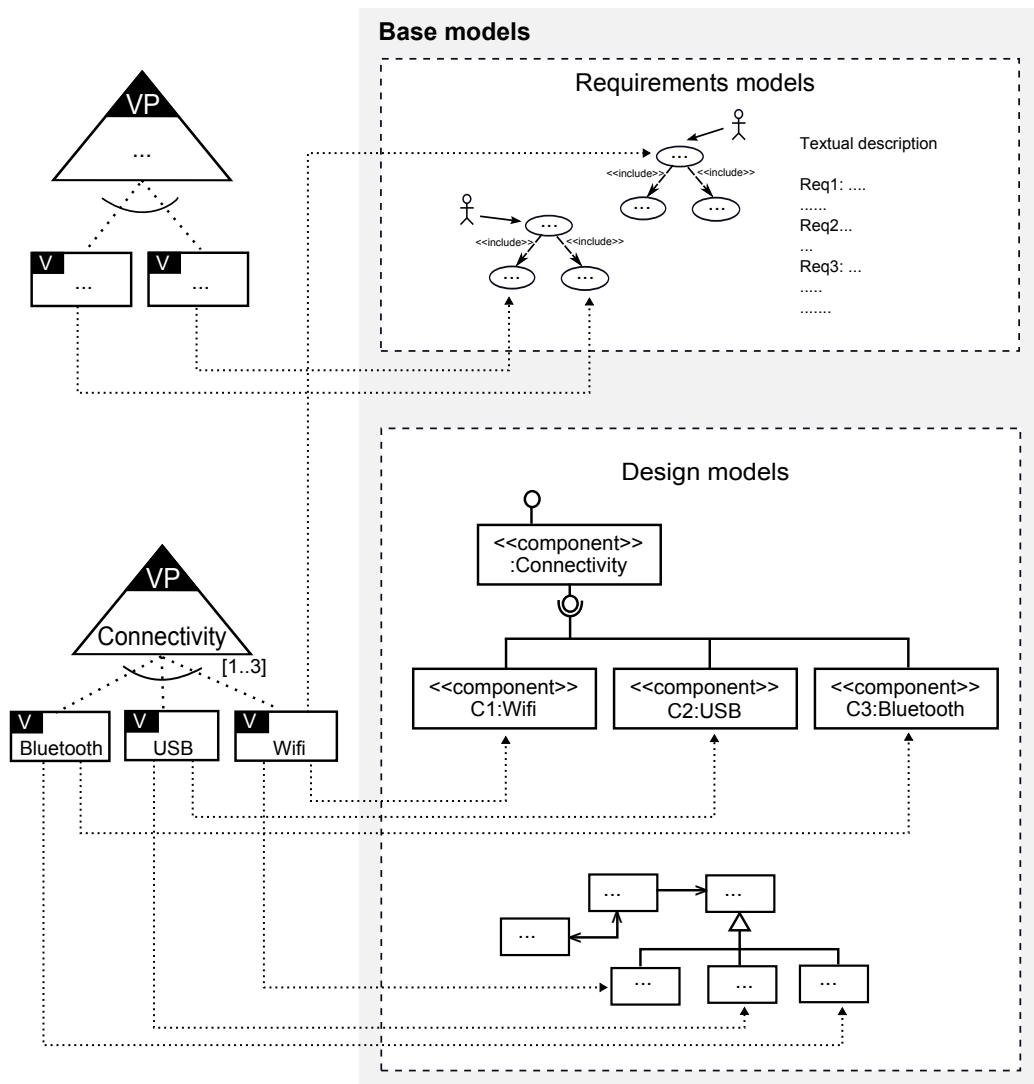


Figure 6.2: OVM documenting variability of base models.

the proposed Attribute-based Model, which is introduced in Section §6.3. The specific configuration model depends on the language used to model the configuration problem [50, 53]. A feature model is an example of a configuration model. Our approach is independent from the configuration model used.

We assume that the relationship between elements in both models (OVM and configuration model) is one-to-one. However, the relationship could be

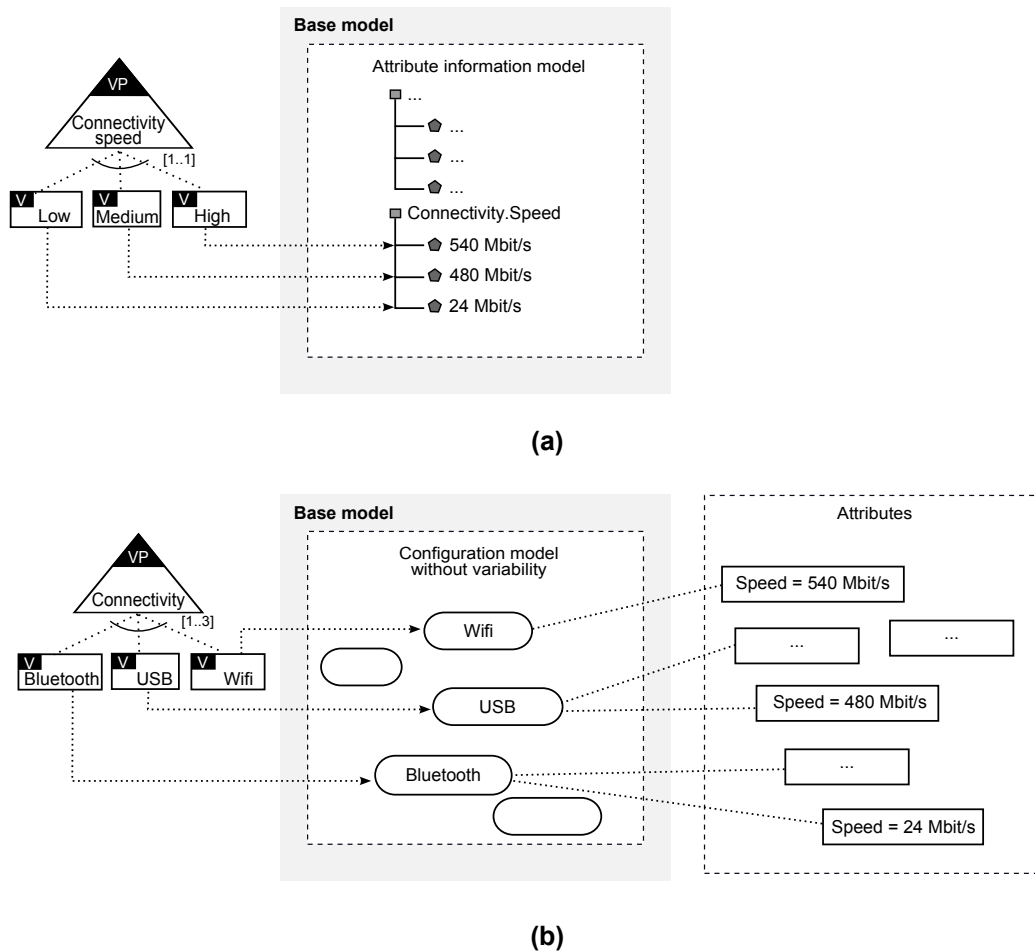


Figure 6.3: (a) Attributes as a base model, (b) Attributes of base model elements.

extended to be 1 : N with only minor changes. These changes concern to the mapping from OVM into a CSP, since in the process of automating the analysis of OVM we will map this specification into a CSP. In 1 : 1 relationships, we can omit the configuration model in the mapping process and then each relationship between an OVM element and an attribute becomes a variant in the CSP. If 1 : N is allowed, the configuration model cannot be omitted. Consequently, the mapping of OVM into CSP is changed and each relationship involving an OVM element, a configuration model element, and an attribute become a variable in the CSP. For the sake of simplicity, we have not addressed 1 : N relationships, however our approach is theoretically applicable.

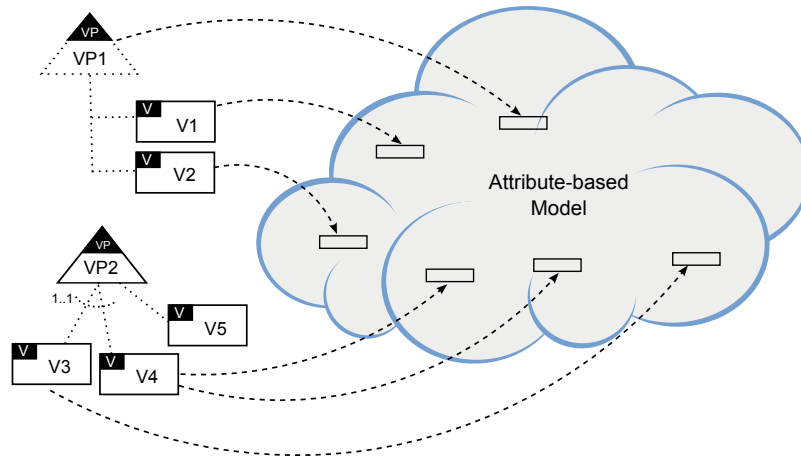


Figure 6.4: *Attribute-aware OVM.*

To simplify the presentation, we omit the configuration model in the subsequent figures and text, and relate the OVM directly to attributes in the Attribute-based Model, as depicted in Figure §6.4. We refer to the resulting relationship between OVM and the Attribute-based Model as *Attribute-aware OVM*. We remark that we still consider the second problem aforementioned, *i.e.*, we are not documenting the variability of the Attribute-based Model.

6.3 Attribute-based Model

In this section, we describe the Attribute-based Model and how we relate variability documented in an OVM to it. It is not our intention to provide a rigorous syntax and semantics of a language to define the Attribute-based Model, but we assume that such model should be defined so that it can be mapped into a constraint satisfaction problem, thus enabling the automation of the analysis of Attribute-aware OVMs. In Figure §6.5, we provide a high-level conceptual model to describe the main elements of the Attribute-based Model we use in our approach. An Attribute-based Model is composed of one or more attributes with their respective domains, and zero or more constraints on these attributes. In the following, we describe these elements and how they are related to the OVM.

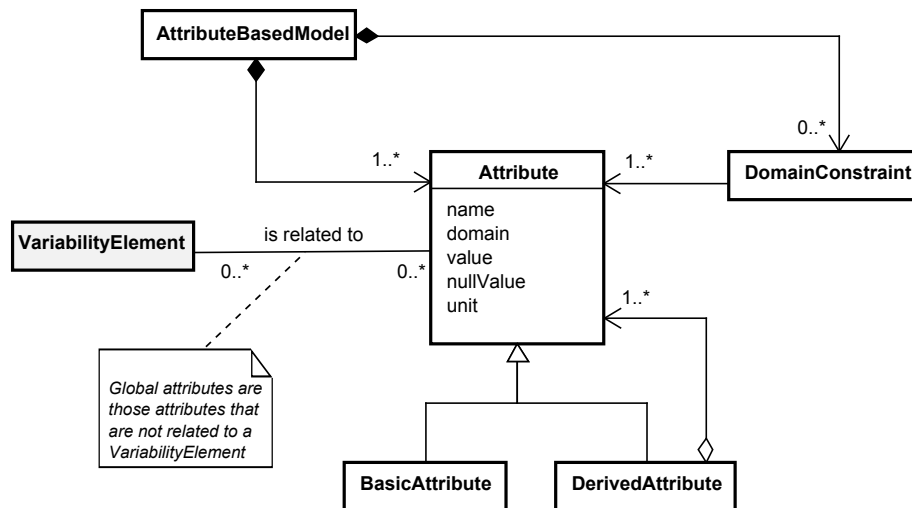


Figure 6.5: Attribute-based metamodel.

6.3.1 Attributes

Attributes are measurable properties of an artefact. We only consider those properties that can be quantified and technically defined. For example: the power consumption or the memory capacity of a feature in the mobile phone product line. As stated by [13], most proposals agree that an attribute should consist of a name, a domain, and a value. We have relied on this statement to specify the attributes used in our approach. An `Attribute` has a name, a domain, a value, a `nullValue`, and a unit. `name` denotes the name of the attribute which does not need to be unique since different artefacts can have different attributes with the same name. `domain` denotes the range of values that the attribute may hold such as Reals, Integers, and any range (e.g., [1..512]); `value` denotes the attribute value which will depend on the concrete type of attribute (we elaborate more on this later). `nullValue` denotes the value that must be taken by the attribute when the `VariabilityElement` with which the attribute is related is not selected. `unit` denotes a determinate quantity such as meters, seconds, currency, and kilobytes, adopted as a standard for measurement.

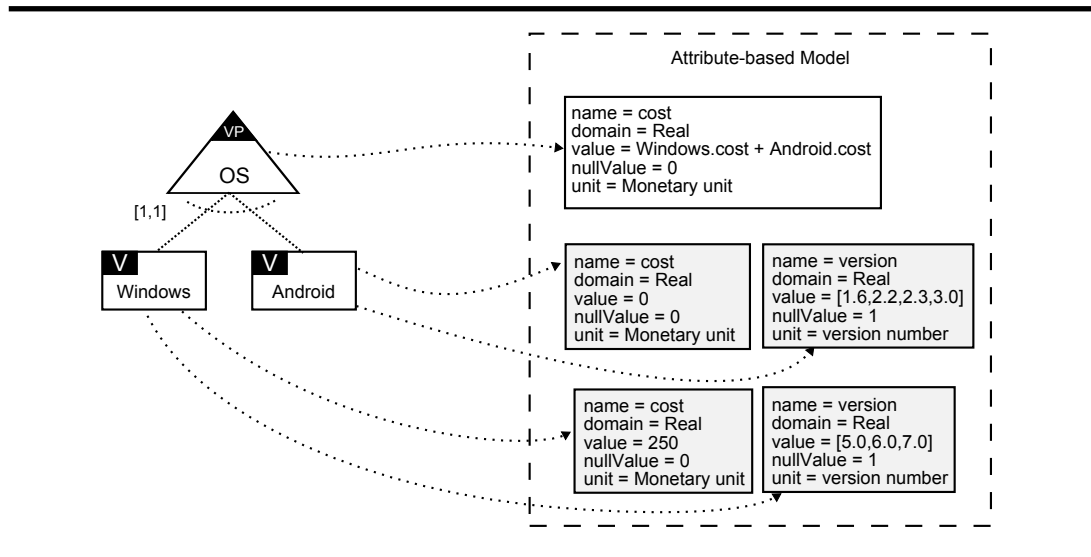


Figure 6.6: Example of basic and derived attributes.

We distinguish two kinds of attributes depending on how their values are calculated:

- **BasicAttribute.** The value of a basic attribute is a *base measure* [57], *i.e.*, a measure that does not depend upon any other measure. This value can be single or a range of values (e.g., [“high”, “medium”, “low”]).
- **DerivedAttribute.** The value of a derived attribute is determined by a function over other attribute values, including values of other derived attributes.

An Attribute can be related to zero or more VariabilityElement (*i.e.*, variation points and/or variants) in the OVM. In the same manner, a VariabilityElement can be related to zero or more Attribute. Note that, for the sake of simplicity, we relate attributes with variability elements; however, in practice variability elements are related with base model elements and these to attributes.

To illustrate how we specify basic and derived attributes, we use the example in Figure §6.6. In this example, the mobile phone product line has a variation point OS, meaning that the operating system can vary from one product to another. Products from this product line can have Windows or Android operating system. Furthermore, there are four basic attributes: *i)* cost, related

to Windows; *ii*) cost, related to Android; *iii*) version, related to Windows, and *iv*) version, related to Android (basic attributes are depicted in grey). These attributes determine that each operating system has a different version and system cost. In addition, there is one derived attribute: `cost`, related to variation point `OS`. This derived attribute expresses the system cost regarding the type of operating system selected, and it is the sum of costs of basic attributes `Windows.cost` and `Android.cost`. Note that we textually represent each attribute by relating the variability element and the attribute with the form `<variability-element.attribute>`, where `variability-element` denotes the name of the variability element which must be unique and `attribute` denotes the name of the attribute. For example, `Windows.cost` defines the relationship between variant `Windows` and attribute `cost`.

An attribute is not always related to a variability element, sometimes we may need to set it with no connection to elements in the OVM. For example, we may define an attribute `TotalCost` that represents the sum of attributes `cost` regarding to all or some of the variation points of the product line. When an attribute is not related to a variability element in the OVM we refer to it as a *global attribute*, and it can be basic or derived. Figure §6.7 shows an example of global attributes (the global attribute is depicted in grey). In this figure, we have included two more variation points, namely `Hardware` and `Settings`. `Hardware` has an attribute `cost`, which in turn is calculated by the sum of `RAM.cost`, `Processor.cost`, `FlashMemory.cost`, and `GPS.cost`. Thus, a global attribute called `TotalCost`, which is not connected to any variability element, is defined. The global attribute represents a property of the product line as a whole. Then, in the case of `TotalCost`, it represents the resulting cost of the product when variability is selected in the OVM and it is defined by adding the costs of selected operating system and selected hardware.

The function used to calculate the values of derived attributes depends on the solver used to automate the approach; furthermore, it is domain dependent. The resulting value of a function depends on whether the variability element related to it is selected or not. Therefore, when an attribute is involved in a function its `nullValue` must be neutral to such a function. Each function must be handled specifically and suitable neutral values must be defined. Let us observe, *e.g.*, the function defined in the `Hardware.cost` attribute. In the case where `GPS` is selected, the addition operation will assign the value 50 to the term `GPS.cost`. However, if `GPS` is not selected, the value of `GPS.cost` must have a neutral value with regard to the addition. In this case, we can use 0 as the `nullValue` since addition is associative.

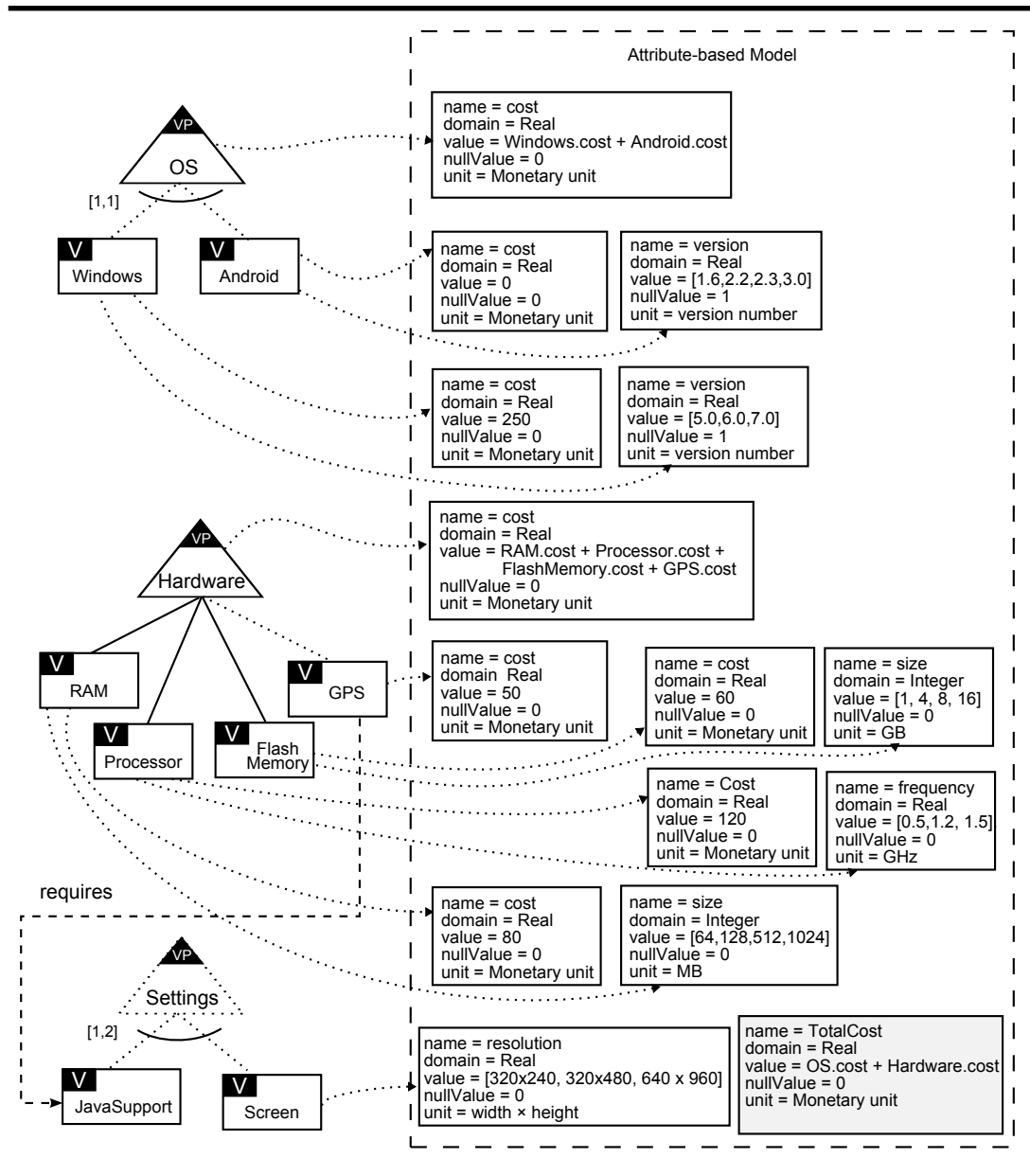


Figure 6.7: Example of global attributes.

6.3.2 Domain constraints

Domain constraints are constraints on attributes that limit the possible variations derived from an OVM. These constraints may come from resource

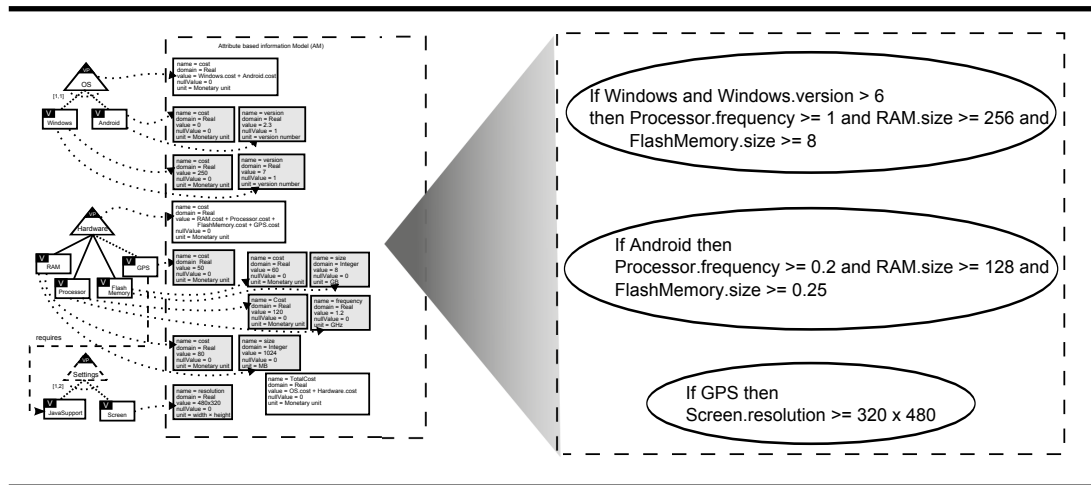


Figure 6.8: Domain constraints on attributes in the Attribute-aware OVM.

limitations (e.g., the maximum memory consumption allowed) or any domain relevant restriction (e.g., all variations derived from the mobile phone product line must provide at least 300 minutes of battery power in talk time, otherwise the system is useless). Therefore, domain constraints can be defined on attributes to avoid building unsuitable variations. We assume that domain constraints can be represented in terms of CSP.

In the case of the mobile phone product line, e.g., a given version of the operating system may require a minimum hardware device. This minimum must be fulfilled to allow the variation to provide a satisfiable performance. Therefore, some constraints on the attribute `Windows.version` and `Android.version` must be specified. To guarantee that a mobile phone variation can successfully run, if `Windows` is selected and `Windows.version` greater than 6, `Processor.frequency` must be greater than or equal to 1 GHz and `RAM.size` greater than or equal to 256 MB and `FlashMemory.size` greater than or equal to 8 GB. However, if `Android` operating system is selected, `Processor.frequency` must be greater than or equal to 0.2 GHz and `RAM.size` greater than or equal to 128 MB and `FlashMemory.size` greater than or equal to 0.25 GB. In addition, when the mobile phone has `GPS` resource, `Screen.resolution` must be greater than or equal to 320x480. Thus, the constraints on attributes represented within ellipses in Figure §6.8 are added to the OVM depicted in Figure §6.7 to prevent the configuration of unsatisfactory variations.

The syntax of domain constraints depends on the solver used to automate the analysis. Domain constraints are predicates over attributes that can be

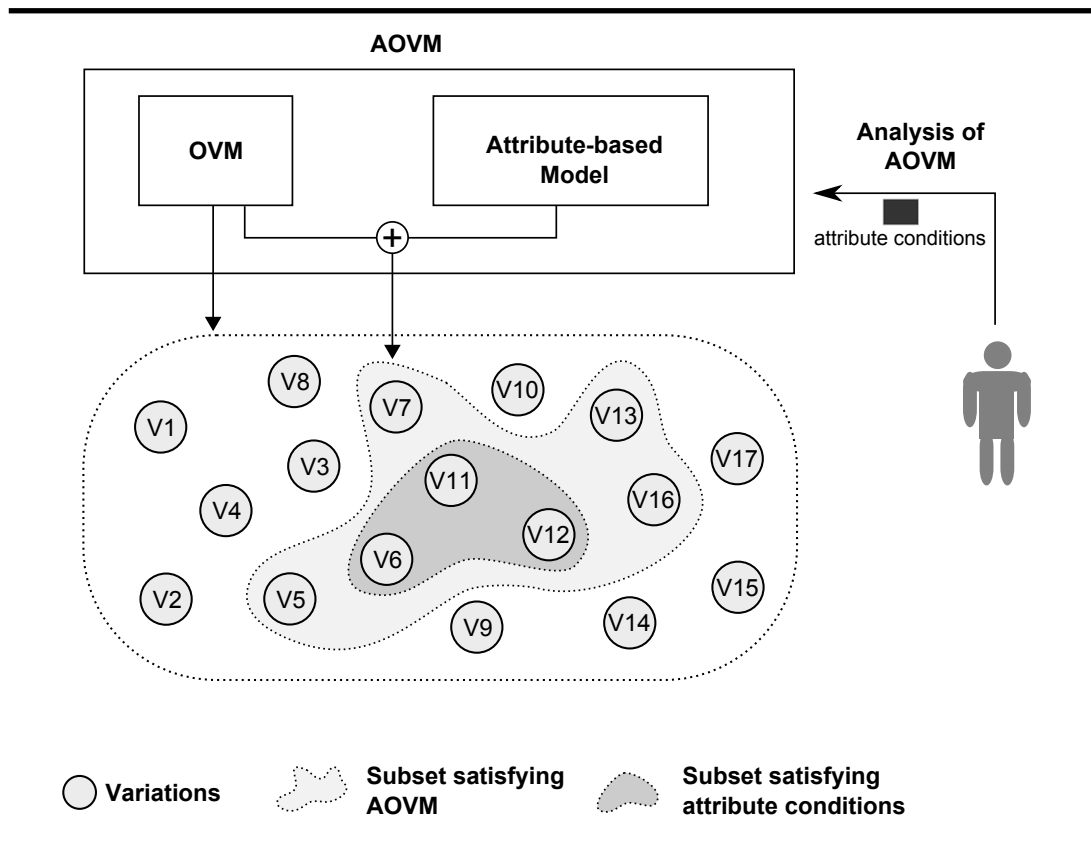


Figure 6.9: Analysis of Attribute-aware OVM.

evaluated to true or false depending on the attribute values. The neutral values for attributes must also be considered in constraint definitions.

6.4 The automated analysis process

In Figure §6.9, we show an overview of our approach for the analysis of AOVMs. An OVM represents a set of possible variations. When an OVM is associated with attributes, resulting in the AOVM, the set of variations can be reduced, since not all of them satisfy the required attributes. Based on this new set of variations, which takes attributes into account, engineers of a product line can carry out the analysis of AOVMs. On the one hand, the AOVM specification can be analysed in order to verify possible anomalies. On the other hand, engineers can execute other analysis operations on AOVM

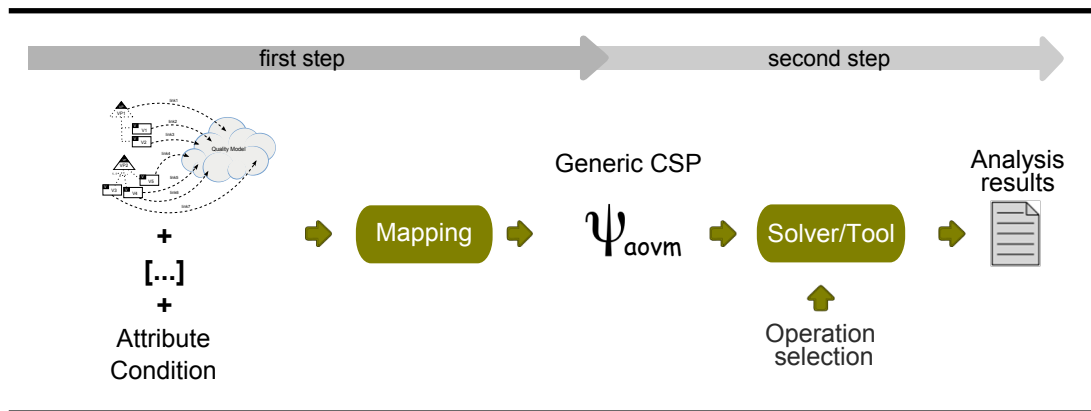


Figure 6.10: Process for the automated analysis of AOVMs.

to extract other useful information, such as to verify attribute conditions, and to ask for an optimal variation or the most representative one. An attribute condition is any constraint that restricts the value of attributes, defined by the engineers of the product line. Therefore, these conditions restrict the set of variations even more, as shown in Figure §6.9.

In our approach, we consider that the number of variations represented by an AOVM is less than or equals to the number of variations represented by an OVM. However, it is worth highlighting that, in some cases, the number of variations represented by an AOVM can be greater than those represented by an OVM; it depends on how variation is defined. In our approach, we define variations as a set of variations points and variants, meaning that variations differ only by functionalities, whether they are associated with attributes or not. However, variations can differ not only by functionality, but also by the values of attributes. Therefore, it would be possible to consider variation as a set of variation points, variants, and attribute values, meaning that different variations can have the same functionalities but different values for attributes. In this case, the number of variations represented by an AOVM can be greater than the number of variations represented by an OVM.

The analysis process previously introduced to automate the analysis of OVM is extended in order to consider attribute conditions (see Figure §6.10). In this process, an operation takes as input an AOVM. Apart from the target model and a possible partial configuration, an operation can also have as input an attribute condition. We define an attribute condition as a statement of what is required as part of a variation or a set of variations regarding attributes. For example, the engineer of the mobile phone product line may want to anal-

use if it is possible to derive a particular variation with development cost no higher than the assigned budget. The restrictions imposed by the engineer are expressed as an attribute condition and are used to get the answer from the AOVM specification.

6.5 Mapping Attribute-aware OVM into CSP

In this dissertation, the mapping from an AOVM into CSP follows the full mapping approach described in Section §5.3.1. As previously commented, the mapping of an AOVM into CSP can differ depending on the concrete solver that is used later for the analysis. In general, the mapping process goes through two main steps. First, the 3-tuple $\psi_{ovm_f} = (V_{ovm_f}, D_{ovm_f}, C_{ovm_f})$ is built, where the variability elements (variation points and variants) in the OVM become variables in V_{ovm_f} with their respective domains in D_{ovm_f} , and the variability and constraint dependencies in the OVM become constraints in the C_{ovm_f} . Second, the final mapping from an AOVM into a CSP is carried out by adding variables and constraints to the ψ_{ovm_f} , where the attributes become variables and the domain constraints become constraints, resulting in the 3-tuple $\psi_{aovm} = (V_{aovm}, D_{aovm}, C_{aovm})$. Next, we detail the complete mapping process.

The mapping from an OVM into a CSP was described in Section §5.3.1, where we have shown how to build the 3-tuple $\psi_{ovm_f} = (V_{ovm_f}, D_{ovm_f}, C_{ovm_f})$. In this section, we add the variables equivalent to each attribute in the AOVM and the needed constraints to the ψ_{ovm_f} , thus resulting in the ψ_{aovm} . The general mapping rules are presented in Table §6.1. In this table, we also show the mapping from the mobile phone example in Figure §6.11 into CSP.

The mapping follows these steps:

- i. Every attribute in the AOVM is mapped into a variable in ψ_{aovm} . The domain of these variables is defined by the union of the domain interval specified to the corresponding attribute and its `nullValue`.
 - When the attribute is related to a variable element, the equivalent variable in ψ_{aovm} is of the form “*ve.Aname*”, where *ve* is the variable element, and *Aname* is the name of the attribute.
 - When the attribute is global, the equivalent variable in ψ_{aovm} is of the form “*Aname*”. Note that, when the attribute is global the do-

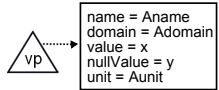
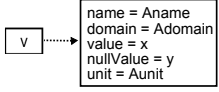
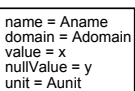
	AOVM		Ψ_{aovm} (JaCoP-like notation)	Mobile phone example
Attributes related to variation point		Domain	$vp.Aname \in \{Adomain\} \cup \{y\}$	$OS.cost \in \text{Real} \cup \{0\} \wedge$ $Hardware.cost \in \text{Real} \cup \{0\}$
		Constraint	If $vp = 1$ then $vp.Aname = x$ else $vp.Aname = y$	If OS = 1 then $OS.cost = Windows.cost + Android.cost$ else OS.cost = 0 If Hardware = 1 then $Hardware.cost = RAM.cost + Processor.cost +$ $FlashMemory.cost + GPS.cost$ else Hardware.cost = 0
Attributes related to variants		Domain	$v.Aname \in \{Adomain\} \cup \{y\}$	$Windows.cost \in \text{Real} \cup \{0\} \wedge$ $Android.cost \in \text{Real} \cup \{0\} \wedge$ $RAM.cost \in \text{Real} \cup \{0\} \wedge$ $Processor.cost \in \text{Real} \cup \{0\} \wedge$ $FlashMemory.cost \in \text{Real} \cup \{0\} \wedge$ $GPS.cost \in \text{Real} \cup \{0\} \wedge$ $Windows.version \in \text{Real} \cup \{0\} \wedge$ $Android.version \in \text{Real} \cup \{0\} \wedge$ $RAM.size \in \text{Integer} \cup \{0\} \wedge$ $Processor.frequency \in \text{Real} \cup \{0\} \wedge$ $FlashMemory.size \in \text{Integer} \cup \{0\}$
		Constraint	If $v = 1$ then $v.Aname = x$ else $v.Aname = y$	If Windows = 1 then $Windows.cost = 250$ $Windows.version = [5.0, 6.0, 7.0]$ else Windows.cost = 0 and Windows.version = 0 If Android = 1 then $Android.cost = 0$ $Android.version = [1.6, 2.2, 2.3, 3.0]$ else Android.cost = 0 and Android.version = 0 If RAM = 1 then $RAM.cost = 80$ $RAM.size = [64, 128, 512, 1024]$ else RAM.cost = 0 and RAM.size = 0 If Processor = 1 then $Processor.cost = 120$ $Processor.frequency = [0.5, 1.2, 1.5]$ else Processor.cost = 0 and Processor.frequency = 0 If FlashMemory = 1 then $FlashMemory.cost = 60$ $FlashMemory.size = [1, 4, 8, 16]$ else FlashMemory.cost = 0 and FlashMemory.size = 0 If GPS = 1 then GPS.cost = 50 else GPS.cost = 0
Global attribute		Domain	$Aname \in \{Adomain\}$	$TotalCost \in \text{Real}$
		Constraint	$Aname = x$	$TotalCost = OS.cost + Hardware.cost$
Domain constraint	domain constraint	constraint		If Windows and Winsows.version => 6 then $Process.frequency \geq 1$ and $RAM.size \geq 256$ and $FlashMemroy.size \geq 8$ If Android then $Process.frequency \geq 1$ and $RAM.size \geq 256$ and $FlashMemroy.size \geq 8$ If GPS then $Screen.resolution \geq 320 \times 480$

Table 6.1: Mapping AOVm into a CSP.

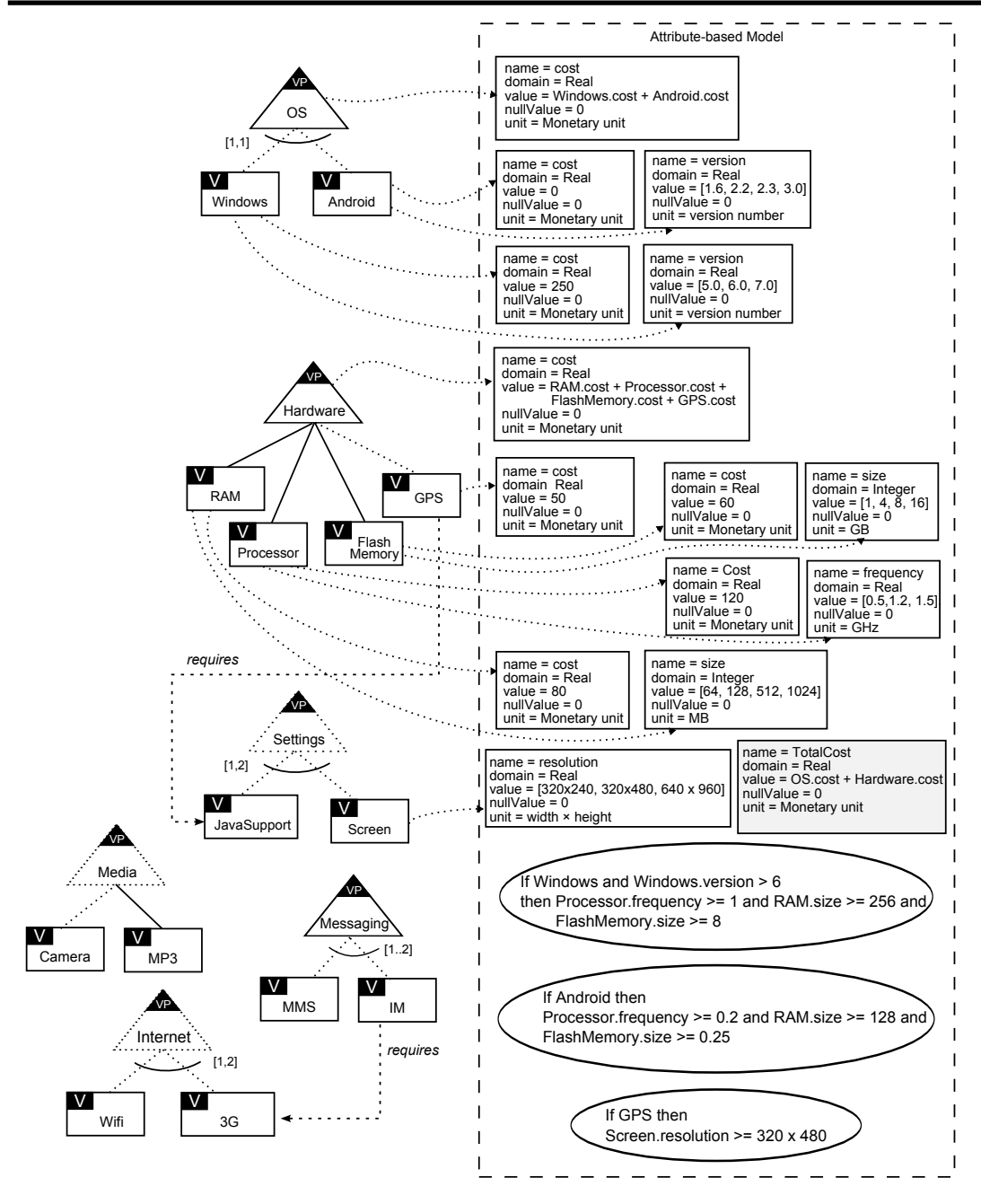


Figure 6.11: AOVm for a mobile phone example.

main of the equivalent variable in ψ_{aovm} is defined only by the domain of the attribute.

- ii. Every attribute in the AOVM is mapped into a constraint in ψ_{aovm} .
 - When the attribute is related to a variable element, its value is mapped into a constraint of the form “if($ve = 1$) then $ve.Aname = x$ else $ve.Aname = y$ ”, where ve is the variable element, $Aname$ is the name of the attribute, x is the value of the attribute, and y is the null value of the attribute.
 - When the attribute is global, its value is mapped into a constraint of the form “ $Aname = x$ ”, where $Aname$ is the name of the attribute, and x is the value of the attribute.
- iii. The domain constraints in the AOVM become constraints in ψ_{aovm} .

6.6 Analysis operations on Attribute-aware OVMs

In this section, we first present the analysis operations to detect anomalies in the AOVM. In addition to requires and excludes constraints, domain constraints can also cause anomalies. Therefore, before running any other attribute aware-analysis operation, we have to detect possible anomalies in the AOVM, such as “*the model does not allow the derivation of any variation that respects all the specified domain constraints*”. Next, we propose other analysis operations on AOVM.

6.6.1 Operations for detecting anomalies

The same operations applied to detect anomalies in the OVM, namely, void model, dead element and false optional element, can be used to detect anomalies in the AOVM. Their definitions are the same as those described in Sections §5.4.4, §5.4.7, and §5.4.8, respectively, except for the input model, which is the CSP equivalent to the AOVM, *i.e.*, ψ_{aovm} . Next, we detail each of them.

Void model. An AOVM is void when it does not represent any valid variation. An AOVM can become void due to the wrong use of constraint dependencies and/or domain constraints. For example, Figure §6.12

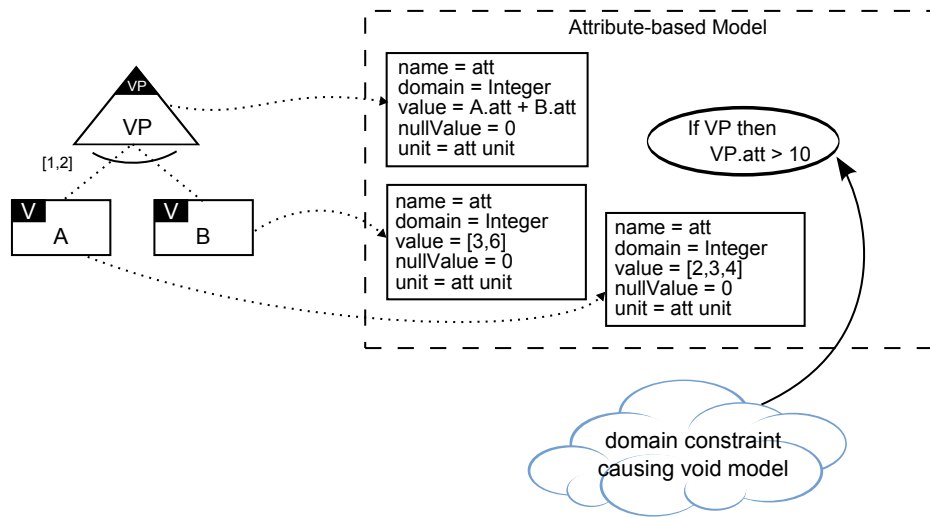


Figure 6.12: Example of void AOVM.

shows an example of a void AOVM, in which the anomaly is caused by a very restrictive domain constraint.

Dead element. A variability element in the AOVM is dead if it does not appear in any of the variations represented by the AOVM. It can become dead due to the wrong use of constraint dependencies and/or domain constraints. For example, Figure §6.13 shows an example of a dead variant (B), caused by a domain constraint.

False optional element. A variability element in the AOVM is false optional when it is modelled as optional, but it appears in every valid variations. It can become false optional due to the wrong use of constraint dependencies and/or domain constraints. For example, Figure §6.14 shows an example of a false optional variant (C), caused by a domain constraint.

6.6.2 Valid attribute condition

An engineer of a product line may want to verify if it is possible to configure a variation or a set of variations that satisfy a given attribute condition, and that at the same time satisfy functional variability and domain constraints. Additionally, taking into account the needs of the stakeholders, the engineers

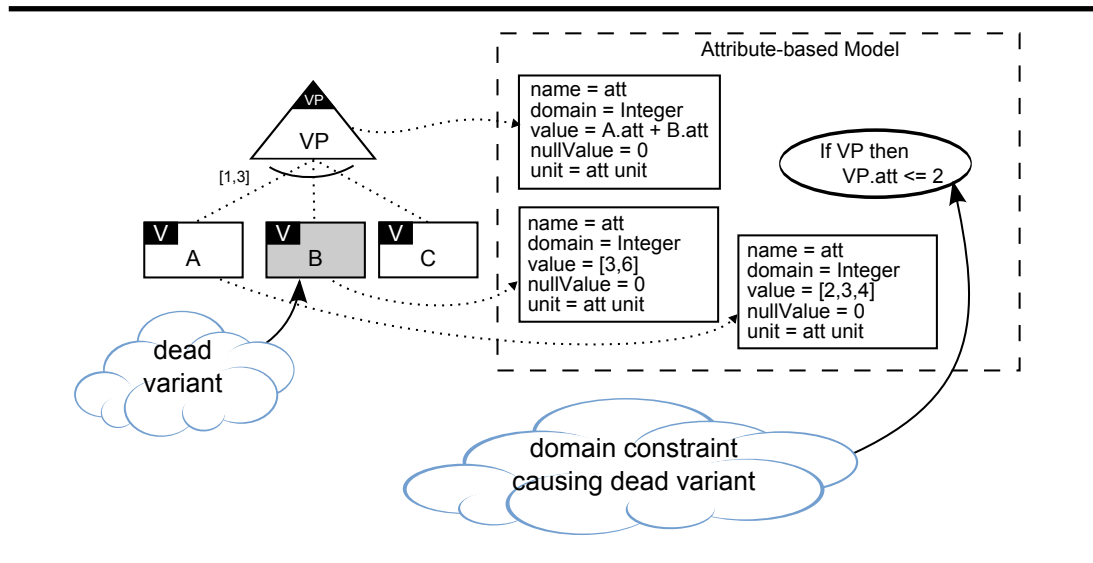


Figure 6.13: Example of a dead variant in the AOVM.

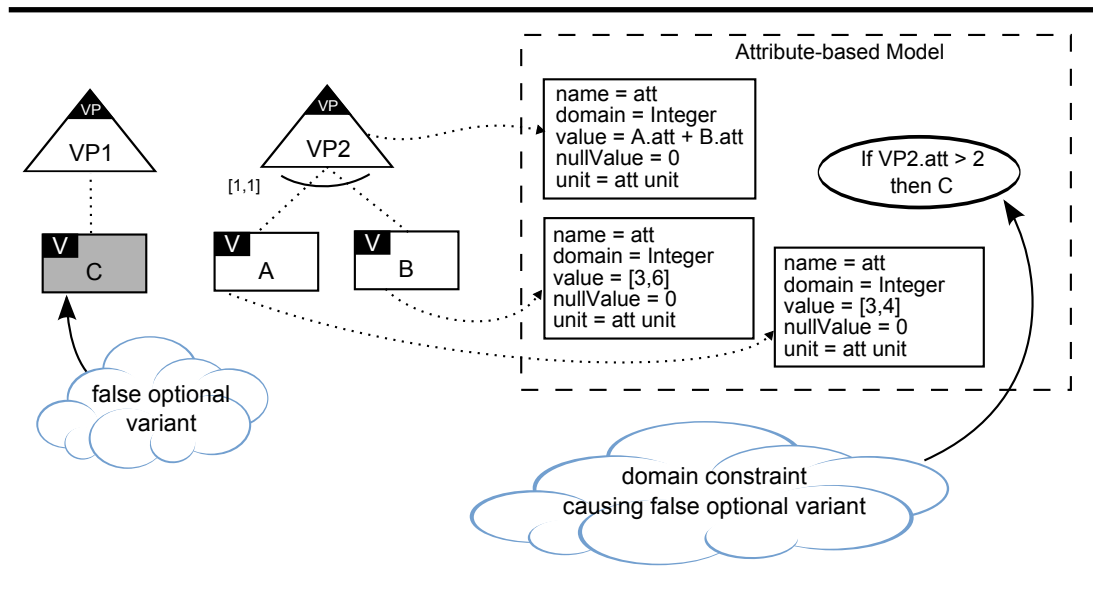


Figure 6.14: Example of a false optional variant in the AOVM.

can verify whether some configuration (*i.e.*, a set of variability elements to be selected and/or a set of variability elements to be removed) satisfies the variability expressed by the AOVM, and an attribute condition, as well. Therefore,

if necessary, the engineer can try to achieve the required configuration by relaxing or removing relationships in the variability model.

As our approach is automated by means of CSP, our basic assumption is that attribute conditions can be specified as a constraint on attributes, as follows:

Definition 1 (*Attribute condition*). Let $aovm$ be an AOVM specification, ψ_{aovm} be its equivalent CSP of the form $(V_{aovm}, D_{aovm}, C_{aovm})$, A be a set of attributes in $aovm$, and A_{aovm} its equivalent variables, such that $A_{aovm} \subset V_{aovm}$. An attribute condition AC is a constraint on one or more attributes, where $attributes \in A_{aovm}$.

To analyse if there is some variation that satisfies an attribute condition, we defined an operation called *Valid attribute condition*. This operation takes an AOVM and an attribute condition as inputs and returns a value (true or false) informing whether AOVM satisfies the given attribute condition or not.

Operation 11 (*Valid attribute condition*). Let $aovm$ be an AOVM specification, ψ_{aovm} be its equivalent CSP, ac be an attribute condition, and AC its equivalent CSP. The attribute condition ac is valid for $aovm$ if there is at least one solution for the conjunction of ψ_{aovm} and AC .

$$\text{validAC}(aovm, ac) \Leftrightarrow |\text{sol}(\psi_{aovm} \wedge AC)| > 0$$

Let us, consider $AC = \text{TotalCost} < 300$ as the attribute condition defined for the AOVM presented in Figure §6.11. This attribute condition is valid for the mobile phone product line, since there is at least one valid variation satisfying $\text{TotalCost} < 300$, such as {OS, Hardware, Android, FlashMemory, Processor, RAM} that has $\text{TotalCost} = 260$. Thus $\text{validAC}(aovm, \text{TotalCost} < 300) = \text{true}$. However, when applying another attribute condition, e.g., $AC = \text{TotalCost} < 100$, to the same example, this time the attribute condition is not valid. Thus $\text{validAC}(aovm, \text{TotalCost} < 100) = \text{false}$.

Now, let us consider that the engineer of the same mobile phone product line wants to verify if it is possible to derive a variation with GPS and Internet support, and without Camera resource, that satisfies his/her attribute condition, $AC = \text{TotalCost} < 550$. To provide support for this analysis, we define a new operation, in which we add the equivalent constraint of a configuration to the model. Then we check whether this configuration is valid when applying a given attribute condition.

Operation 12 (*Valid Configuration with attribute condition (ValidConfAC)*). Let aovm be an AOVm specification, ψ_{aovm} be its equivalent CSP, ac be an attribute condition, AC be its equivalent CSP, C be a configuration of the form (S, R) , E be a variability element $\in \text{SUR}$, and e be its equivalent variable $\in \psi_{\text{aovm}}$. The configuration C is valid for the aovm if there is at least one solution for the conjunction of its equivalent constraint and ψ_{aovm} and AC .

$$\text{ValidConfAC}(\text{aovm}, \text{ac}, C) \Leftrightarrow |\text{sol}(\psi_{\text{aovm}} \wedge \text{AC} \wedge (\bigwedge_{e_i \in S} e_i = 1 \wedge \bigwedge_{e_i \in R} e_i = 0))| > 0$$

Applying this operation to the example in Figure §6.11, we verify that if GPS and Internet are selected, and Camera is removed, there is no valid variation that satisfies $\text{TotalCost} < 300$, as follows:

$$\text{ValidConfAC}(\text{aovm}, (\{\text{GPS}, \text{Internet}\}, \{\text{Camera}\}), \text{TotalCost} < 550) = \text{false}$$

6.6.3 Optimal variation

We refer to optimal variation as the variation that satisfies AOVm, and also minimises or maximises a given objective function. When relating attribute information to OVM we are able to ask for an optimal variation, since the objective function takes into account values of attributes. The product line engineer may want to verify, e.g., which variation of the set of variations consumes less memory, or even to find the variation with the lowest development cost. Therefore, finding the optimal solution, as opposed to any possible solution, would be helpful for making attribute-aware decisions.

The Optimal operation takes an AOVm and an objective function as inputs and returns the set of variations fulfilling the criteria established by the function. Hence, to find the optimal solution, we can associate an objective function with an optimisation problem. Then, the solver has to find the best variation from all valid variations.

Definition 2 (*Optimal*). Let aovm be an AOVm specification, and O be an objective function, the optimal set of variations, hereinafter max and min , is equal to the optimal space of ψ_{aovm} .

$$\begin{aligned} & \max(\psi_{\text{aovm}}, O) \\ & \min(\psi_{\text{aovm}}, O) \end{aligned}$$

For example, to find the cheapest variation(s) represented by an AOVM, a possible objective function could be $O = \text{TotalCost}$, such that the sought solution is rendered by minimising O , as follows:

$$\text{CheapestVariation}(s) = \min(\psi_{\text{aovm}}, \text{TotalCost})$$

The engineer of a product line may want to verify which is the optimal variation that satisfies some attribute condition and a desired partial configuration. For example, which is the cheapest variation that offers $\text{RAM.size} < 128$ and has variant GPS? To find this optimal solution, we add a constraint equivalent to the configuration $(\{\text{GPS}\}, \{\})$ and another equivalent to the attribute condition to aovm . Then, we ask for the optimal variation(s), which minimise(s) the total cost. Thus the optimal variations V_{opt} are:

$$\begin{aligned} \psi'_{\text{aovm}} &= (\psi_{\text{aovm}} \wedge \text{GPS} = 1 \wedge \text{RAM.size} < 128) \\ O &= \text{TotalCost} \\ V_{\text{opt}} &= \min(\psi'_{\text{aovm}}, O) \end{aligned}$$

A possible application of the *Optimal Variation* operation can be to support the *Most Representative Variation* operation, which aims to find the most representative variation for a given product line. Assessing all possible variations of the product line is impracticable due to the usually very large number of variations in a product line. Therefore, there is a need for deciding which variations should be assessed, *i.e.*, the most representative variation for the product line.

There may be different ways of implementing the *Most Representative Variation (MRV)* operation. In this dissertation, we consider the MRV as those that have variants involved in a larger number of variations, however there are other possibilities (*e.g.*, the most expensive ones or those that have the largest number of variants). Therefore, the MRV operation is defined as the variation(s) of the software product line that maximise(s) the sum of commonality degrees of variations.

The commonality degree of a variation is determined by the sum of the

commonality of its variants and variation points. The commonality of a variability elements was already defined in Section §5.4.9, it represents the percentage of variations where the variability element appears, e.g., if there are 10 possible variations and a variant v appears in 5 variations, the commonality of v is 0.5. After the most representative variation has been found, it can be assessed by applying any evaluation technique employed in single-systems. MRV operation is defined as follows:

Operation 13 (*Most Representative Variation (MRV)*). Let $aovm$ be an AOVMM specification, ψ_{aovm} be its equivalent CSP of the form $(V_{aovm}, D_{aovm}, C_{aovm})$, E be a variability element $\in V_{aovm}$, and e be its equivalent variable $\in \psi_{aovm}$. The commonality degree (*CommDegree*) of e is the relation between the number of solutions of the conjunction of ψ_{aovm} and e , and the number of all the possible solutions of ψ_{aovm} . O is the summation of commonality degree of variants, and MRV is the variation(s) that maximise(s) O .

$$\text{CommDegree}(aovm, E) = \frac{|\text{sol}(\psi_{aovm} \wedge e = 1)|}{|\text{sol}(\psi_{aovm})|}$$

$$O = \sum_{v_i \in V_{aovm}} \text{CommDegree}(aovm, v_i)$$

$$\text{MRV}(\psi_{aovm}) = \max(\psi_{aovm}, O)$$

Note that, the MRV operation is defined based on an optimisation function represented by O . This function can be defined according to the user needs and it is attribute-aware. For example, O could be defined as the *sum* of the attributes cost and therefore MRV would return the most expensive variation(s).

6.7 Summary

In this chapter, we have presented the AOVMM, which is used to document the relationship between OVM and attributes, and constraints on these attributes. To build an AOVMM, we have introduced an Attribute-based Model, a model based on attributes and constraints on these attributes to capture measurable attributes of components in the base models of a product line. Then, we have described how we relate this Attribute-based Model to the variability documented in the OVM. In addition, we have presented the analysis process to automate the analysis of AOVMMs. We have provided a set of mapping rules

to translate an AOVM into a CSP, and defined a set of analysis operations to be performed on AOVMs. With the provided analysis, the engineers can verify anomalies in the specification, find an optimal variation, verify attribute conditions, and find the most representative variation.

Chapter 7

Evaluating the approach with FaMa-OVM

*The man of science has learned to believe in justification,
not by faith, but by verification.*

*Thomas Henry Huxley, 1825 - 1895
British biologist*

The goal of this chapter is to present an evaluation of our approach. To this purpose, we introduce a study of a case from the automotive domain we have used in our evaluation, and also introduce FaMa-OVM, a prototype tool we have developed to automatically analyse OVM. In Section §7.1, we introduce the sections that will be presented in this chapter. In Section §7.2, we describe the Radio Frequency Warner (RFW) product line, a study of a case we have used to evaluate our approach. In Section §7.3, we specify the RFW product line using OVM. In Section §7.4 we relate attributes and domain constraints with the RFW product line OVM. In Section §7.5, we introduce the FaMa-OVM tool, which we have used for automating the analysis of the RFW product line, and present the analysis results we have obtained from such analysis. Finally, Section §7.6 summarises the chapter.

7.1 Introduction

In this chapter, we first introduce the Radio Frequency Warner (RFW) product line, a study of a case from the automotive domain that we have used to evaluate our proposal. RFW product line is part of a case study created in a national project by a leading car company in Germany, and carried out by Paluno - The Ruhr Institute for Software Technology at the University of Duisburg-Essen. We show how the RFW product line was specified using OVM, describing the variation points, variants and their variability and constraints dependencies. We also describe and define the attributes and domain constraints identified by the RFW product line engineers using AOVM. We present the analysis results we have obtained using FaMa-OVM tool for the analysis of the RFW product line AOVM. FaMa-OVM is a prototype tool we have developed to illustrate the feasibility of our approach, enabling to automatically analyse both OVM and AOVM. It is worth mentioning that we have no intention of providing an industrial tool support for the analysis of OVM, but offer a proof of concepts of our approach.

7.2 Radio Frequency Warner (RFW) product line: a study of a case

In this section we describe the RFW system and its components. This description was provided by Paluno.

7.2.1 System overview

The motivation for developing the RFW product line is the increasing complexity of today's traffic. Systems derived from the RFW product line aim to give hints of relevant traffic signs to a driver of a car or truck. The product line is based on the fictional assumption that all traffic signs are equipped with a Radio-Frequency Identification (RFID) tag. This allows the identification of traffic signs when approaching a sign. The transmitted data includes the type of sign (maximum speed, no overtaking, etc.) and the direction of the sign. The functionality of the RFW is realised by a control unit in the car that interacts with other components in the car such as the display and sound system.

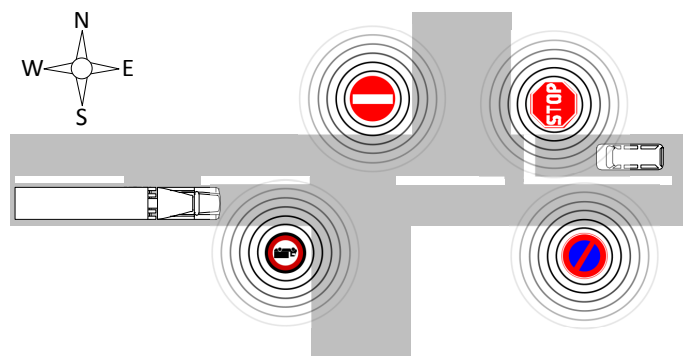


Figure 7.1: *Functionality of the RFW.*

An illustration of the functionality of the RFW system can be found in Figure §7.1: the car is arriving from the east heading west. Four different signs are in the area: a stop sign, a do-not-enter sign, a no-trucks sign, and a no-parking sign. All these signs are equipped with an active RFID transmitter and each sign knows its direction:

- the stop sign is relevant for all vehicles approaching from the east;
- the do-not-enter sign is relevant for all vehicles approaching from the east and west;
- the no-parking sign is relevant for all vehicles approaching from the west;
- the no-trucks sign is relevant for all trucks approaching from the west and the east.

The information about the direction is encoded in the signal of the traffic signs. If the car, for example, heads to the west, it will receive the signals of all signs. The RFW processes the signals and dismisses the no-parking sign, because it is only relevant for the opposite direction. It signals the stop sign to the driver, since this is the nearest sign to the car that is relevant. During the trip, the RFW system will also show the do-not-enter sign. The no-trucks sign is dismissed for the car. The truck, for example, that is approaching from the west receives the same signals including the no-parking sign in the opposite order, but the RFW does not dismiss the no-trucks sign, because it is relevant for the truck driver.

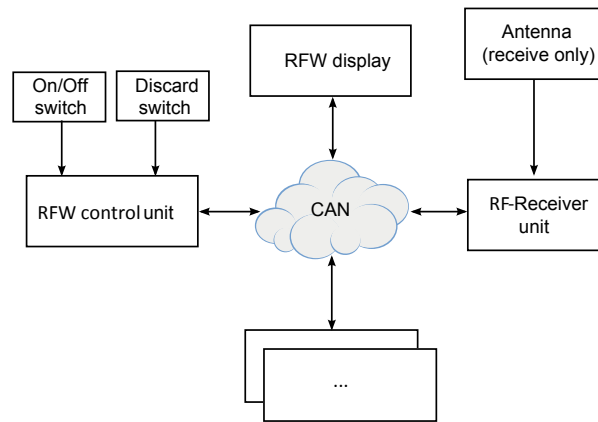


Figure 7.2: Overview of the RFW system.

7.2.2 System components

An overview of the system can be found in Figure §7.2. Roughly speaking, the system consists of three components: the RFW display, RF-receiver unit, and the RFW control unit. They communicate via a Controller Area Network (CAN) which is a standard interface in the automotive area, standardised by ISO (ISO 11898). However, the CAN is used as a transparent transport gateway. The components can be characterised as follows:

- **RFW control unit.** the control unit is the main part of the system. It receives the signals of the RF receiver and reacts specifically based on a set of rules.
 - ◇ **Discard switch.** the discard switch marks the current signal to be discarded. When the user presses the button, the actual symbol in the display is discarded and the actual warning sound is stopped, and all upcoming signs with the same RFID are dismissed for the next 60 seconds.
 - ◇ **On/Off switch.** the On/Off switch activates and deactivates the RFW system.
- **RFW display.** the RFW display shows the output of the RFW control unit.

- **RF receiver unit.** the RF receiver unit receives the signals and sends them to the RFW control unit.
 - ◊ **Antenna.** the antenna of the RF-receiver unit receives the signals of the active RFID transmitters and decodes them.

7.3 Specifying the RFW product line using OVM

In order to provide a RFW system to customers with different needs, the RFW product line has been created. Figure §7.3 shows an excerpt of the OVM corresponding to the RFW product line. In this figure, variation points VP7 : OtherSigns, VP8 : ProhibitionSigns, VP9 : WarningSigns, and VP10 : SignsGivingOrders subsume the different categories of signs that can be detected. For simplification, we show only two signs for each category. The complete OVM diagram with all variation points and variants can be found in Appendix §C, Figure §C.1. We may remark that in this figure we are omitting several requires and excludes dependencies since it would be too confusing to show all of them. We present a list with these constraints in Appendix §C, Table §C.1.

The main differentiation of the RFW system is made in variation point VP1 : TypeOfVehicle. There, one of four different vehicle types has to be chosen. Variation point VP2 : Activation determines whether the RFW is switchable (*i.e.*, whether it has a switch to turn it on or off) or whether it is turned on instantly and continuously. Variation point VP3 : comfortFunctions determines the additional functionality of the RFW. The following comfort functions are available:

- V7 : NoStoppingWarning: warns the driver if there is an active no stopping sign at the current position and therefore stopping is forbidden.
- V8 : OverSpeedWarning: warns the driver if there is a speed limit in effect and the car is going too fast. This requires additional information about the current speed that needs to be received via CAN.
- V9 : SoundAtWarningSign: if the car passes a warning sign, the RFW system warns the driver acoustically.
- V10 : HazardousSituationAlarm: warns the driver in a hazardous situation and may take over control, *e.g.*, by initiating an emergency brake.

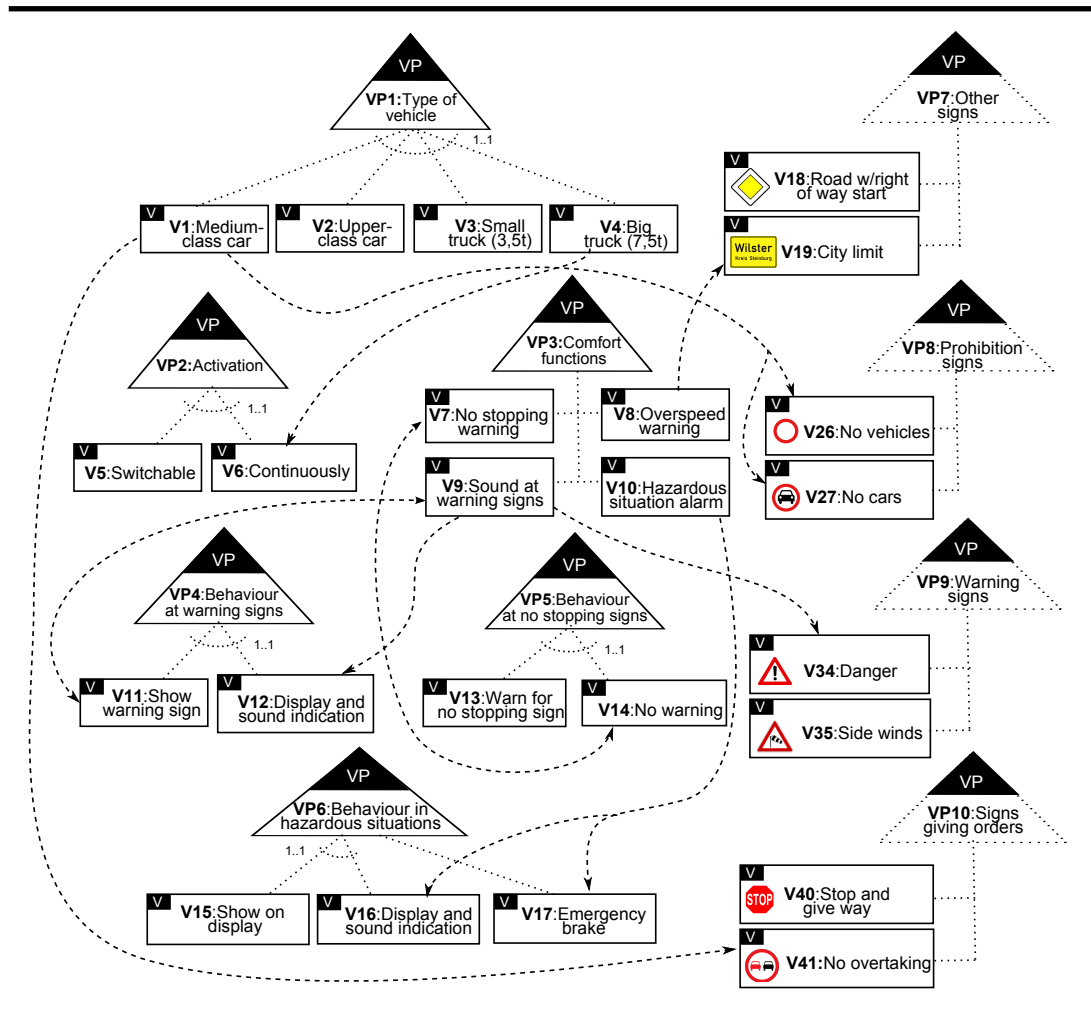


Figure 7.3: Excerpt of the RFW OVM.

This detection requires much external information, e.g., the actual speed, lateral acceleration, status of the wheels (*i.e.*, blocking, slippage, etc.).

Variation point VP4 : BehaviourAtWarningSigns determines the behaviour, if a relevant warning sign is passed. The system can show the warning sign in the display and it can additionally sound an acoustic warning. The behaviour of the RFW system at a relevant stopping sign is determined by variation point VP5 : BehaviourAtNoStoppingSign. The system may warn the driver or not.

The behaviour in a hazardous situation is defined by variation point VP6 : BehaviourInHazardousSituations. The RFW can show a warning in the

display and it can additionally warn the driver with an acoustic signal. Additionally, the system can initiate an emergency brake.

7.4 Expressing attributes for the RFW product line

The specification of RFW variability using OVM is important to guarantee that different customer needs are satisfied by the product line. However, attributes are also relevant when developing RFW systems, since information such as the specification of the development cost of the comfort function or the power of the sensor required by different signals have to be captured.

7.4.1 Attributes

In Figure §7.4, we give an example of attributes that were specified for the RFW product line. The RFW product line offers two different types of positioning systems: V53 : GPS and/or V54 : Galileo. The positioning systems have different accuracies, thus we define a basic attribute denoted as accuracy. The GPS system has an accuracy of 8 meters, while Galileo has 4 meters of accuracy. Furthermore, two derived attributes capture the relationships between basic attributes: Accuracy and AccuracyFactor, related to VP12 : PositioningSystem, and VP13 : Antenna, respectively. The former expresses the system accuracy regarding the type of positioning system selected, and it is the minimum value between V53 : GPS.Accuracy and V54 : Galileo.Accuracy; the latter expresses the accuracy factor of the selected antenna which is obtained by the maximum value between Small.AccuracyFactor, Medium.AccuracyFactor, and Big.AccuracyFactor. Finally, the TotalAccuracy is a global attribute because it is not connected to any variable element. The global attribute represents a property of the product line as a whole. Thus, TotalAccuracy represents the resulting accuracy of the system, which is the product of the selected positioning system accuracy and the selected quantifier. With this specification, for example, it would be possible to get the same overall accuracy with a bigger antenna and GPS, or a medium size antenna and Galileo.

Let us observe the function defined in the value of Accuracy attribute in VP12 : PositioningSystem variation point. In the case where both positioning systems, V54 : Galileo and V53 : GPS, are selected, the function will re-

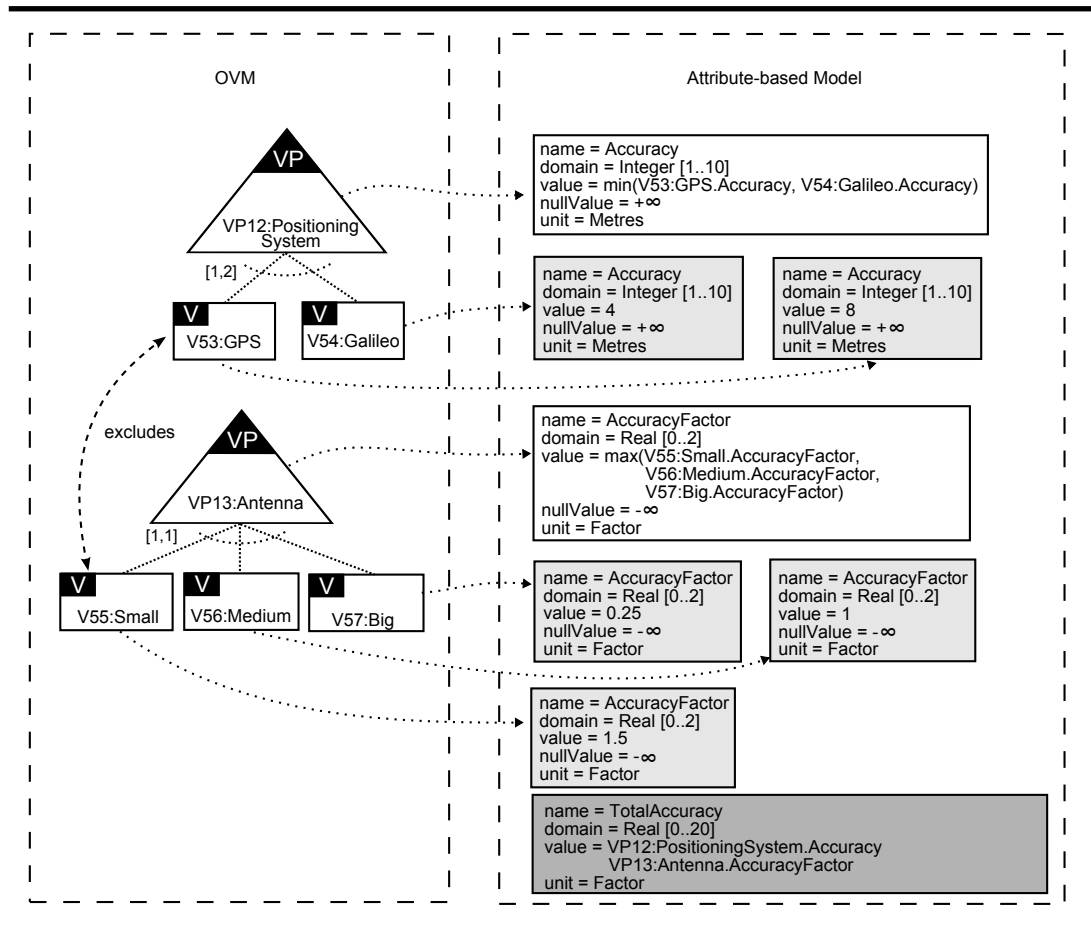


Figure 7.4: Attributes of Positioning system and Antenna.

turn the minimum value between their accuracy, resulting in value 4. However, if one of the variants is not selected, e.g., V53 : GPS, the value of V53 : GPS.Accuracy must have a neutral value with regard to the min function. In this case, we can use the $+\infty$ as a neutral value because it is bigger than any real number. In the RFW example, we use aggregate functions such as *sum*, *min* and *max* and also functions with the operators $+$ and $*$.

The list of all attributes identified for the RFW product line and their descriptions can be found in Table §7.1. Their values are not shown in this table because they depend on the association with the variable elements in the OVM. The values are shown in Tables §7.2, §7.3, and §7.4; they were defined by the engineers of the RFW product line. Table §7.2 shows the values taken by the basic attributes. The variants are listed in the first column of the table,

and the attribute names are listed along the first row. The cells indicate the values taken by each attribute when related to the corresponding variant. They are shown in the form *value | neutral-value*. Cells marked with “–” indicate that the attribute is not related to the variant.

In the RFW example, all derived attributes are related to variation points. Their values are shown in Table §7.3. The variation points are listed in the first column, and the attribute names are listed along the first row. The cells indicate which function is used to calculate each attribute value when related to the corresponding variation point. Those cells marked with “–” indicate that the attribute is not related to the corresponding variation point. Values are shown in the form *value|neutral-value*. As these attributes are involved in the values of the global attributes (see Table §7.4), their null values are defined as $+\infty$, $-\infty$ or 0. In the case that the variation point is selected, the attributes can take as values the functions *min*, *max* or *sum*. They are defined as follows:

$\min(v_1.\text{attribute}, \dots, v_n.\text{attribute})$, where $\{v_1, \dots, v_n\} \subseteq \text{childrenOf}(vp)$

$\max(v_1.\text{attribute}, \dots, v_n.\text{attribute})$, where $\{v_1, \dots, v_n\} \subseteq \text{childrenOf}(vp)$

$\sum_{i=1}^n v_i.\text{attribute}$, where $\{v_1, \dots, v_n\} \subseteq \text{childrenOf}(vp)$

Consider that $\text{childrenOf}(vp)$, with vp belonging to the set of variation points returns the set of children of vp , and $n \leq |\text{childrenOf}(vp)|$. Next, we provide some examples.

VP12 : PositioningSystem.Accuracy = $\min(\text{GPS.Accuracy}, \text{Galileo.Accuracy})$

VP13 : Antenna.AccuracyFactor = $\max(\text{Small.AccuracyFactor}, \text{Medium.AccuracyFactor}, \text{Big.AccuracyFactor})$

VP2 : Activation.Memory = V5 : Switchable.Memory + V6 : Continuously.Memory

The equations for the values of global attributes are shown in Table §7.4. Except for the **TotalAccuracy**, the other global attributes are calculated using an aggregate function in the set of variation points. In the following we elaborate on their meaning:

- **TotalAccuracy**: represents the overall accuracy of a given product which is computed by multiplying the accuracy of the positioning system by the antenna accuracy factor. For example, a variation of the RFW product line that has **GPS** and a **medium** antenna offers an overall accuracy of 8, since V53 : **GPS.Accuracy** = 8 and V56 : **Medium** = 1.
- **TotalMemory**: represents the total memory required by a given variation, which is computed by the sum of all attributes **Memory** related to variation points.

Name	Description	Domain	Unit
1. Accuracy	Specifies the accuracy of the positioning system to locate the position of the car	Integer [1..10]	meters
2. AccuracyFactor	Specifies a quantifier for the antenna	Real [0..2]	meters
3. TotalAccuracy	Specifies the accuracy offered by the system. It is calculated by relating the accuracy and the accuracyFactor attributes	Real[0..20]	meters
4. Memory	Specifies the memory size of the control unit that is needed to process the traffic sign	Integer [1..512]	kilobytes
5. TotalMemory	Specifies the total of memory required by a system. It is calculated by aggregating the memory attributes	Integer[1..512]	kilobytes
6. ROM	Specifies the ROM size of the control unit utilised by a specific variant	Integer [1..512]	kilobytes
7. TotalROM	Specifies the ROM size of the control unit. The more traffic signs are recognisable, the bigger the ROM size has to be to save the different types of signs and the required action for the traffic sign. It is calculated by aggregating the ROM attributes	Integer [1..512]	kilobytes
8. Range	It concerns the power of a sensor and specifies the distance from which the sensor is capable to detect a traffic sign. The higher the value is, the earlier a traffic sign can be detected	Integer	meters
9. Latency	Specifies the latency required by a variant. Latency means the elapsed time between the firing of an event and the feedback given to the user.	Integer [200..800]	milliseconds
10. TotalLatency	Specifies the latency that has to be guaranteed by the system. It is calculated by aggregating the latency attributes	Integer [200..800]	milliseconds
11. Cost	Cost of the specific variant	Real [1..500]	monetary unit
12. TotalCost	Specifies the total cost of a system. It is calculated by aggregating the cost attributes	Real [1..500]	monetary unit
13. Cycle	Specifies the maximum recognition time required by a variant.	Integer[10..500]	milliseconds
14. RecognitionTime	Specifies the maximum recognition time that the system has to ensure. It is calculated by aggregating the cycle attributes	Integer [10..500]	milliseconds

Table 7.1: Attributes in the RFW product line.

- TotalROM: represents the total ROM required by a given product, which is computed by the sum of all attributes ROM related to variation points.
- TotalLatency: represents the maximum time that a given variation takes to provide feedback, which corresponds to the minimum value amongst

	Accuracy	Accuracy Factor	Memory	ROM	Range	Latency	Cost	Cycle
V5:Switchable	-	-	2 0	2 0	-	-	2 0	-
V6:Continuously	-	-	2 0	2 0	-	-	0.2 0	-
V7:No stopping warning	-	-	-	8 0	-	-	0.5 0	-
V8:Overspeed warning	-	-	-	16 0	-	-	0.5 0	-
V9:Sound at warning signs	-	-	-	4 0	-	-	1 0	-
V10:Hazardous situation alarm	-	-	-	16 0	-	-	1 0	-
V11:Show warning sign	-	-	2 0	4 0	-	400 +∞	1 0	-
V12:Display and sound indication	-	-	2 0	4 0	-	400 +∞	1 0	-
V13:Warn for no stopping sign	-	-	2 0	4 0	-	500 +∞	0.5 0	-
V14:No warning	-	-	2 0	0 0	-	500 +∞	0.2 0	-
V15:Show on display	-	-	8 0	8 0	-	350 +∞	1 0	-
V16:Display and sound indication	-	-	8 0	8 0	-	350 +∞	1 0	-
V17:Emergency brake	-	-	16 0	32 0	-	350 +∞	3 0	-
V18:Road w/ right of way start	-	-	4 0	4 0	-	-	0.2 0	100 +∞
V19:City limit	-	-	2 0	4 0	-	-	0.2 0	50 +∞
V20:Crossroads	-	-	2 0	4 0	-	-	0.2 0	100 +∞
V21:Home zone entry	-	-	4 0	4 0	-	-	0.2 0	100 +∞
V22:Road w/ right of way end	-	-	4 0	4 0	-	-	0.2 0	100 +∞
V23:End of city limit	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V24:Traffic has priority	-	-	2 0	4 0	-	-	0.2 0	75 +∞
V25:Home zone end	-	-	4 0	4 0	-	-	0.2 0	100 +∞
V26:No vehicles	-	-	2 0	4 0	-	-	0.2 0	50 +∞
V27:No cars	-	-	2 0	4 0	-	-	0.2 0	100 +∞
V28:No vehicles over max width > Xm	-	-	8 0	8 0	-	-	0.2 0	200 +∞
V29:No vehicles w/ weight > 3.5t	-	-	2 0	4 0	-	-	0.2 0	100 +∞
V30:No vehicles over max gross weight g > Xt	-	-	8 0	8 0	-	-	0.2 0	200 +∞
V31:Do not enter	-	-	2 0	4 0	-	-	0.2 0	100 +∞

Table 7.2: Values of basic attributes when associated with variants.

	Accuracy	Accuracy Factor	Memory	ROM	Range	Latency	Cost	Cycle
V32:No vehicles over max height $h > X_m$	-	-	8 0	8 0	-	-	0.2 0	200 +∞
V33:No stopping	-	-	2 0	4 0	-	-	0.2 0	100 +∞
V34:Danger	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V35:Side winds	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V36:Slippery road	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V37:Risk of ice	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V38:Bend	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V39:Traffic queues	-	-	2 0	4 0	-	-	0.5 0	100 +∞
V40:Stop and give way	-	-	2 0	4 0	-	-	0.2 0	50 +∞
V41:No overtaking	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V42:No overtaking end	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V43:No overtaking vehicles > 3.5t	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V44:End of prohibitions	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V45:Yield	-	-	2 0	4 0	-	-	0.2 0	50 +∞
V46:Maximum speed X Km/h	-	-	8 0	8 0	-	-	0.2 0	200 +∞
V47:One way	-	-	2 0	4 0	-	-	0.2 0	50 +∞
V48:Maximum speed of X Km/h end	-	-	8 0	8 0	-	-	0.2 0	200 +∞
V49:No overtaking vehicles > 3.5t end	-	-	4 0	4 0	-	-	0.2 0	50 +∞
V50:Low	-	-	-	-	20 -∞	-	10 0	-
V51:Medium	-	-	-	-	45 -∞	-	35 0	-
V52:High	-	-	-	-	70 -∞	-	50 0	-
V53:GPS	8 +∞	-	-	-	-	-	-	-
V54:Galileo	4 +∞	-	-	-	-	-	-	-
V55:Small	-	1.5 -∞	-	-	-	-	15 0	-
V56:Medium	-	1 -∞	-	-	-	-	0.5 0	-
V57:Big	-	0.25 -∞	-	-	-	-	50 0	-

Table 7.2: Values of basic attributes when associated with variants (Cont'd).

	Accuracy	Accuracy Factor	Memory	ROM	Range	Latency	Cost	Cycle
VP2:Activation	-	-	sum 0	sum 0	-	-	sum 0	-
VP3:Comfort functions	-	-	-	sum 0	-	-	sum 0	-
VP4:Behaviour at warning signs	-	-	sum 0	sum 0	-	min + ∞	sum 0	-
VP5:Behaviour at no stopping signs	-	-	sum 0	sum 0	-	min + ∞	sum 0	-
VP6:Behaviour in hazardous situations	-	-	sum 0	sum 0	-	min + ∞	sum 0	-
VP7:Other signs	-	-	sum 0	sum 0	-	-	sum 0	min + ∞
VP8:Prohibitions signs	-	-	sum 0	sum 0	-	-	sum 0	min + ∞
VP9:Warning signs	-	-	sum 0	sum 0	-	-	sum 0	min + ∞
VP10:Signs giving orders	-	-	sum 0	sum 0	-	-	sum 0	min + ∞
VP11:Sensor power	-	-	-	-	max - ∞	-	sum 0	-
VP12:Positioning system	min + ∞	-	-	-	-	-	-	-
VP13:Antenna	-	max - ∞	-	-	-	-	sum 0	-

Table 7.3: Values of derived attributes when associated with variation points.

Name	Value
TotalAccuracy	PositioningSystem.Accuracy * Antenna.AccuracyFactor
TotalMemory	$\sum_{j=1}^k vp_j.Memory$, where $vp_j.Memory \in AOV M$
TotalROM	$\sum_{j=1}^k vp_j.ROM$, where $vp_j.ROM \in AOV M$
TotalLatency	$\min(vp_1.Latency, \dots, vp_k.Latency)$, where $vp_1.Latency, \dots, vp_k.Latency \in AOV M$
TotalCost	$\sum_{j=1}^k vp_j.Cost$, where $vp_j.Cost \in AOV M$
RecognitionTime	$\min(vp_1.Cycle, \dots, vp_k.Cycle)$, where $vp_1.Cycle, \dots, vp_k.Cycle \in AOV M$

$k \leq$ number of variation points $\in AOV M$

Table 7.4: Equations for the values of global attributes.

the attributes Latency related to variation points. For example, the minimum value between $VP1.Latency = 350$ and $VP2.Latency = 500$ is 350. Then, the maximum time this variation should take to provide feedback to the user is 350 milliseconds.

- TotalCost: represents the cost of a given variation, which is computed by the sum of all attributes Cost related to variation points.
- RecognitionTime: represents the maximum time that a given variation takes to recognise a signal, which corresponds to the minimum value amongst the attributes Cycle related to variation points. For example, the minimum value between $VP1.Cycle = 50$ and $VP2.Cycle = 100$ is 50. Then, the maximum time this variation should take to recognise a signal is 50 milliseconds.

7.4.2 Domain constraints

In the RFW product line, the traffic signs must be detected by the sensor from a given distance before passing the traffic sign. This distance must be reasonable to allow the product to provide feedback to the user within an expected time. Therefore, some constraints on the attribute $VP11 : SensorPower.range$ must be defined. To guarantee that a RFW variation can successfully detect the $V44 : EndOfProhibitions$ sign, the sensor has to

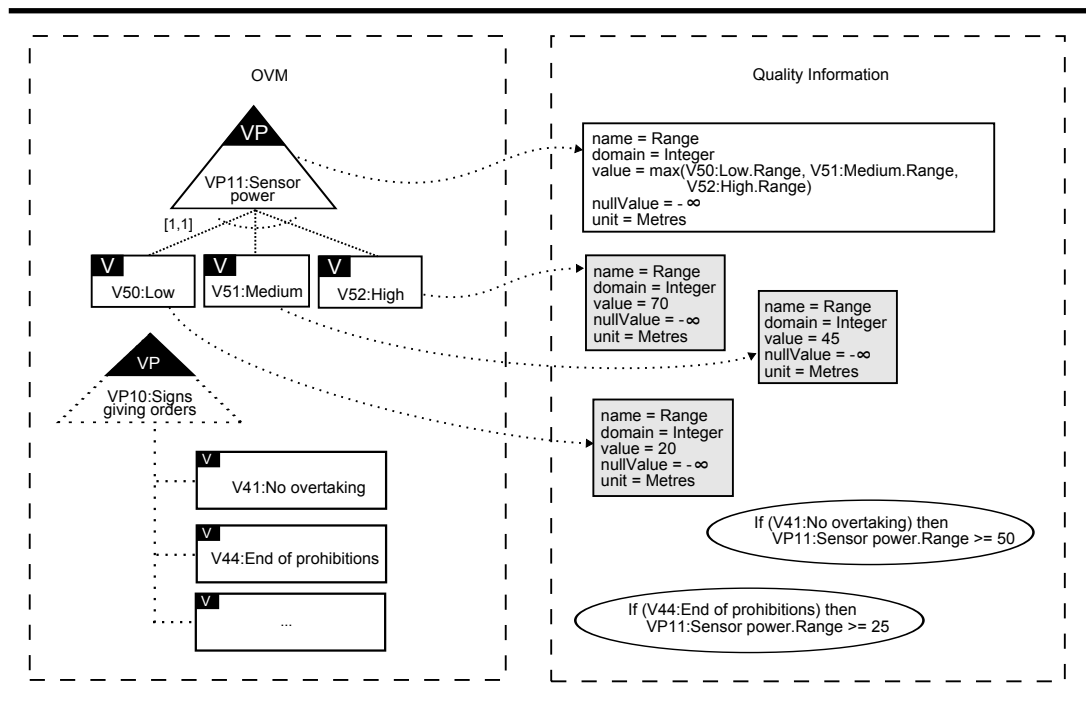


Figure 7.5: Domain constraints on RFW AOVM.

have a range of at least 25 meters. However, to detect the V41 : NoOvertaking sign, the sensor must be able to detect the signal at least 50 meters before passing the sign. Thus, the constraints on attributes represented within ellipses in Figure §7.5 are specified to prevent the configuration of unsatisfactory products.

In the RFW product line, we have identified a number of required domain constraints. A complete list of these constraints can be found in Appendix §C, Table §C.2.

The RFW product line has 13 variations points, 4 out of which are optional, and 57 variants. Consequently, there are 57 variability dependencies between variation points and variants, 65% of them are optional and 35% are alternative. Furthermore, the product line has 34 requires and 4 excludes relationships between variable elements, 225 attributes (18 out of them are derived and 6 are global), and 72 domain constraints. Consequently, to manually detect anomalies and extract information from this model is a tedious and error-prone task. This task is even more complicated if we consider that the products should respect domain constraints.

7.5 Automating the analysis using FaMa-OVM

In the following subsections we give an overview of the FaMa-OVM tool, introduce the FaMa-OVM textual format for the RFW product line, and present the analysis results obtained.

7.5.1 The FaMa-OVM tool

FaMa-OVM provides support for the analysis of both OVM and AOVM. This tool allows performing all the analysis operations defined in this dissertation, such as detecting anomalies, checking satisfiability of quality conditions, and finding an optimal variation or the most representative one. FaMa-OVM is freely distributed under LGPL v3 license and can be downloaded from the FaMa-OVM Web site ^{†1} (see Figure §7.6).

FaMa-OVM is an open source Java tool, that is implemented based on FaMa-Framework (FaMa-FW) [130]. FaMa-FW is an open source Java framework designed to facilitate the development of analysis tools for diverse variability modelling languages. This framework provides a number of extension points to plug in new components, such as metamodels, readers/writers and reasoners. Figure §7.7 shows an overview of FaMa-OVM components. The dark components are the extensions of the original framework. In the following, we report on those components we have implemented:

The *OVM metamodel* implements the description of the different variability elements, and the rules that constraint the combination of these elements. Furthermore, it describes the attributes and their relationship with variability elements, as well as the domain constraints on attributes.

The *OVM reader* implements a reader to both OVM and AOVM textual format, which we have defined for representing OVM and AOVM specification, respectively.

The *OVM reasoner* implements the specific solver for the analysis. FaMa-FW provides by default JavaBDD, Choco, JaCoP GPL, and SAT4j for the analysis of feature models. For the analysis of both OVM and AOVM, we have implemented Choco solver, which is CSP solver. This solver enables working with numerical values, such as integers, which allows to deal with attributes:

^{†1}FaMa-OVM website (www.isa.us.es/fama-ovm)

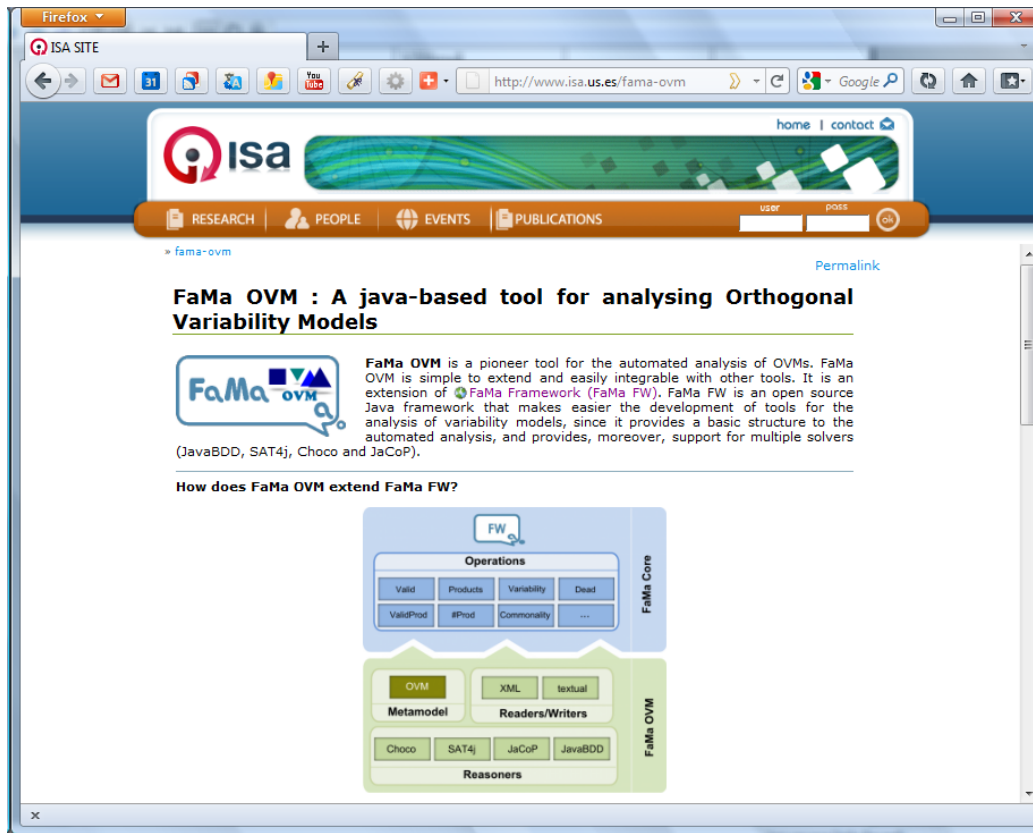


Figure 7.6: FaMa-OVM web Site.

enabling it to maximise or minimise values. We may remark that we could also have used a SAT solver, since the transformation from a CSP into SAT is reasonably easy. In fact, we have implemented in FaMa-OVM a SAT solver for the analysis of OVM without attributes, since this solver is more efficient for most of the operations defined in this dissertation. For the sake of simplicity, in this dissertation, we will only address the analysis results obtained by using Choco solver.

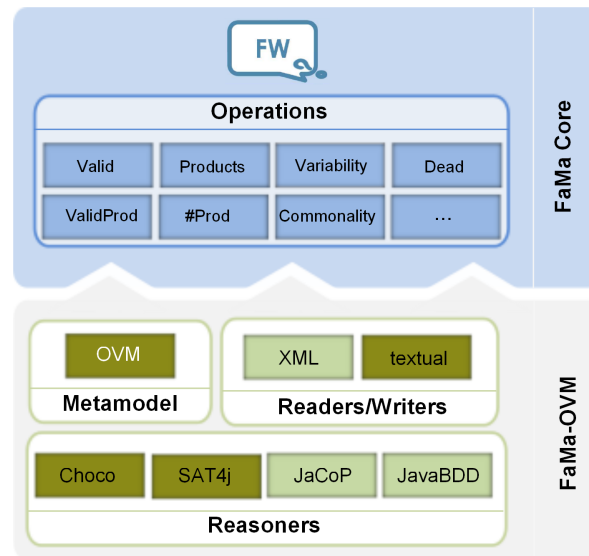


Figure 7.7: FaMa-OVM extending FaMa-FW.

7.5.2 The textual format for the RFW product line

We have defined a FaMa-OVM textual format to represent both OVM and AOVm specifications^{†2}. This textual format consists of four main parts, namely: Relationships, Attributes, Global Attributes, and Constraints. Relationships specifies the variability dependencies between variation points and variants. Attributes specifies basic and derived attributes. Global Attributes specifies global attributes. Constraints specifies excludes and requires relationships, and quality conditions. Attributes are defined in the form of `<name>:<domain>,<value>,<nullValue>;` where the terms are separated by commas, and lines are finished with semicolon. When the value of an attribute is a function, a semicolon after `<value>` is used. Figure §7.8 shows a single textual format for the examples in Figures §7.4 and §7.5.

Because we have used Choco solver, all variables in the CSP must belong to a finite domain, which implies that attributes must have a finite domain. Due to this limitation, functions can only involve integer values. Consequently, we were unable to use real numbers as we intended. Subsequently, real numbers

^{†2}The FaMa-OVM tool and the textual AOVm for the RFW model used in our evaluation are available at www.isa.us.es/fama-ovm

```

1: %Relationships
2: VP12PositioningSystem : [1,2]V53GPS V54Galileo;
3: VP13Antenna : [1,1]V55Small V56Medium V57Big;
4: VP11SensorPower: [1,1]V50Low V51Medium V52High;
5: [VP10SignsGivingOrders] : [V41NoOvertaking]
   [V44EndOfProhibitions];

6: %Attributes
7: V53GPS.Accuracy: Integer[1 to 10], 8, INF;
8: V54Galileo.Accuracy: Integer[1 to 10], 4, INF;
9: VP12PositioningSystem.Accuracy: Integer[1 to 10],
   min(V53GPS.Accuracy,V54Galileo.Accuracy);,INF;

10: V55Small.AccuracyFactor: Integer [1 to 4], 4, MINF;
11: V56Medium.AccuracyFactor: Integer [1 to 4], 3, MINF;
12: V57Big.AccuracyFactor: Integer [1 to 4], 1, MINF;
13: VP13Antenna.AccuracyFactor: Integer [1 to 4],
   max(V55Small.AccuracyFactor,
   V56Medium.AccuracyFactor,V57Big.AccuracyFactor);,MINF;

14: V50Low.Range: Integer [20 to 30], 20, MINF;
15: V51Medium.Range: Integer [45 to 60], 45, MINF;
16: V52High.Range: Integer [70 to 100], 70, MINF;
17: VP11SensorPower.Range: Integer [1 to 100],
   max(V50Low.Range,V51Medium.Range,V52High.Range);,MINF;

18: %GlobalAttributes
19: TotalAccuracy: Integer [1 to 40],
   VP12PositioningSystem.Accuracy *
   VP13Antenna.AccuracyFactor;, 0;

20: %Constraints
21: V53GPS EXCLUDES V55Small;
22: V41NoOvertaking IMPLIES VP11SensorPower.Range >= 50;
23: V44EndOfProhibitions IMPLIES VP11SensorPower.Range >= 25;

```

Figure 7.8: RFW in FaMa-OVM textual format.

were mapped to integers. The values of attributes `V55Small.AccuracyFactor`, `V56Medium.AccuracyFactor`, and `V57Big.AccuracyFactor` were mapped from `[1.5, 1, 0.25]` to `[4, 3, 1]`, respectively.

7.5.3 Analysis results

We have used FaMa-OVM to execute the analysis of AOVM defined in Chapter §6. In our evaluation we have analysed two models: *i)* the *RFW model*, which represents the RFW product line with all attributes and domain constraints that were defined through this chapter, and *ii)* the *Excerpt of RFW model*, which is the RFW excerpt depicted in Figure §7.3.

When specifying quality attributes for the RFW product line we have defined a certain domain for them. The domain sets the limits of the attribute values. We addressed the analysis problem as a CSP, then the higher the range of the domain is, the more complex the problem becomes. In our implementation, when mapping the RFW product line to a CSP, we have replaced the domain range of attributes as much as possible in order to reduce the problem complexity. For example, in the case of `V11.Latency`, we have replaced the range `Integer[200..800]` by `[400]`, and in the case of `VP4BehaviorAtWarningSigns.Latency`, we have replaced `Integer[200..800]` by `350, 400, 500`. An effort was made to preserve consistency amongst the assigned values. In the following we present the obtained analysis operations results:

Detecting anomalies. None of the two models were identified as void, but other anomalies were detected. The excerpt model does not have any anomaly, however the RFW model has seven dead elements and six false optional elements, as can be seen in Table §7.5. These anomalies were caused by the wrong use of constraints by the product line engineer. We were able to identify that the constraints involving `TotalAccuracy <= 10` cause some of the dead elements as well as the false optional. Therefore, it allows us to conclude that the bigger the model is, the more error-prone the modelling task becomes.

Satisfiability with quality condition. The RFW product line does not satisfy the quality condition `TotalAccuracy < 10 ∧ TotalCost < 30`. When we have relaxed this quality condition, by changing values to `TotalAccuracy < 30 ∧ TotalCost <= 50`, we have found that there are variations which satisfy the relaxed quality condition. As can be

		RFW model ^{†±}	Excerpt of RFW ^{‡≠}
Detect anomalies	Void	False	False
	Dead	V38Bend V39Traffic queues V45Yield V50Low V51Medium V55Small V56Medium	None
	False Optional	VP7Other signs VP8Prohibition signs VP9Warning signs VP10Signs given orders V41No overtaking V57Big	None
Satisfiability	Satisfies(QC)	False	True
	Satisfies(QC+PC)	False	True

[†] QC = TotalAccuracy < 10 \wedge TotalCosts < 30

[‡] QC = TotalAccuracy < 10

[±] PC = {{V53 : GPS, V52 : High, V1 : Medium – ClassCar}, {}}

[≠] PC = {{V53 : GPS, V52 : High, {}}

Table 7.5: Analysis operations results.

seen in Table §7.5, the excerpt model satisfies the quality condition TotalAccuracy < 10. The number of variations found in the excerpt model when no quality conditions were defined is 70, but when analysing it using the quality condition TotalAccuracy < 10, this number was reduced to 30.

Satisfiability with quality condition and configuration. We have analysed the RFW model to verify whether it satisfies the quality condition TotalAccuracy < 10 \wedge TotalCost < 30 associated with the partial configuration {{V53GPS, V52High, V1MediumClassCar}, {}}. As shown in Table §7.5, there are no variations which have V53GPS, V52High,

and `V1MediumClassCar`, and satisfy such quality condition. In the case of the excerpt model, we have verified that there are variations which have `V53GPS` and `V52High`, and satisfy the quality condition `TotalAccuracy < 10`. When we have associated the partial configuration with the quality condition and verified satisfiability, the number of variations found in the excerpt model was reduced to 10.

Optimal variation. This operation needs to compute all possible solutions beforehand to be able to find a solution. Therefore the problem to be solved is more complex. We have observed that for the RFW model this operation has taken much more time. The Choco solver was not able to find the optimal variation in a reasonable time; however, it found a solution to this operation when analysing the excerpt model. In this model, we were able to find the most accurate variations. For this purpose we have defined that the optimal variations minimise the *TotalAccuracy* global attribute. Thus, we have found that 20 variations are able to provide the minimum accuracy, which is 4. In the following, we present the 20 most accurate variations (`OptVar`).

`OptVar1 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High}`

`OptVar2 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders}`

`OptVar3 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V44EndOfProhibitions}`

`OptVar4 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking}`

`OptVar5 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking, V44EndOfProhibitions}`

`OptVar6 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium}`

`OptVar7 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium, VP10SignsGivingOrders}`

`OptVar8 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium, VP10SignsGivingOrders, V44EndOfProhibitions}`

`OptVar9 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V50Low}`

`OptVar10 = {VP12PositioningSystem, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V50Low, VP10SignsGivingOrders}`

`OptVar11 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High}`

`OptVar12 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders}`

OptVar13 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V44EndOfProhibitions}

OptVar14 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking}

OptVar15 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking, V44EndOfProhibitions}

OptVar16 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium}

OptVar17 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium, VP10SignsGivingOrders}

OptVar18 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V51Medium, VP10SignsGivingOrders, V44EndOfProhibitions}

OptVar19 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V50Low}

OptVar20 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V50Low, VP10SignsGivingOrders}

Optimal variation with quality condition. We have analysed the excerpt model by executing the optimal operation associated with the quality condition $TotalAccuracy < 10$ and the partial configuration $\{\{V53GPS, V52High\}, \{\}\}$, and still using $TotalAccuracy$ as the objective function. In this case, five variations were found as the most accurate variations, as follows:

OptVar1 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High}

OptVar2 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders}

OptVar3 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V44EndOfProhibitions}

OptVar4 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking}

OptVar5 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower, V52High, VP10SignsGivingOrders, V41NoOvertaking, V44EndOfProhibitions}

Most representative variation. Similarly to the optimal variation, the most representative operation also needs to compute all possible solutions beforehand to be able to find a solution. Therefore, the Choco solver was also not able to find the most representative variations in a reasonable time; however, it found a solution when analysing the excerpt model, which is as follows:

```
MRV1 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V56Medium, VP11SensorPower,  
        V52High, VP10SignsGivingOrders, V41NoOvertaking, V44EndOfProhibitions}
```

```
MRV2 = {VP12PositioningSystem, V53GPS, V54Galileo, VP13Antenna, V57Big, VP11SensorPower,  
        V52High, VP10SignsGivingOrders, V41NoOvertaking, V44EndOfProhibitions}
```

We ran our experiments on a computer that is equipped with a Dual core AMD Opteron 1218 processor running at 2.6 GHz, 2GB of RAM, Ubuntu 10.10 with Kernel 2.6.35.28, and the 1.6.0 version of the Java Runtime Environment.

7.6 Summary

In this chapter, we have evaluated our approach by using a study of a case from the automotive domain. First, we have specified the RFW product line using OVM, and then specified attributes and domain constraints of the RFW product line using AOVM. Afterwards, we have presented FaMa-OVM tool, which we have used to analyse the RFW AOVM. FaMa-OVM is a prototype tool we have developed that works a proof of concepts of our approach. With this tool we were able to perform the analysis operations described in Section §6.6 on the RFW product line. From our results we have verified that two of those analysis operations on RFW AOVM are too complex for the solver we have used, namely Choco. The complexity of finding the optimal variation(s) and finding the most representative variation(s) increases exponentially with the number of variability elements on the model.

Part IV

Final Remarks

Chapter 8

Conclusions and Future Work

All progress is precarious, and the solution of one problem brings us face to face with another problem.

*Martin Luther King Jr., 1929 - 1968
US civil rights leader and clergyman*

8.1 Conclusions

In this dissertation we have shown that:

We can enrich orthogonal variability models with attributes and constraints on attributes, and based on a framework developed for the automated analysis of feature models, provide an automated support for the analysis of software product lines by means of attribute-aware orthogonal variability models.

The automated analysis of variability models is an active research field. The importance of variability modelling activity to the software product line development has led to an increased interest of researchers in extracting useful information from its resulting models. The automated analysis of software product lines by means of variability models deals with the computer-aided

extraction of information from variability models. Currently, there are several works providing automated support for the analysis of feature models in particular. However, the emergence of other techniques for the variability modelling is leading to the need for an automated support of their analysis. Current efforts are mainly driven towards feature modelling approaches, and the analysis of variability models taking into account attributes is hardly supported. This limits the scope and applicability of current variability modelling analysis tools to feature models domain.

In this dissertation, we have presented a set of techniques and tools to support the automated analysis of OVMs, and attribute-aware OVMs as well. These contributions are the result of applying the analysis process from the feature models community to the analysis of OVM. Our main results are:

- i. Improvements on modelling concepts of the OVM language. These improvements consist of a way to document abstract elements explicitly and to specify attribute-aware OVMs, which allows us to express attributes and constraints on attributes, and their relationship with an OVM;
- ii. Support for the automated analysis of OVMs. To this purpose, we provided a set of mapping rules to translate an OVM and also an attribute-aware OVM into a CSP. Then, we provided a set of analysis operations to be performed on the resulting CSP specification.
- iii. The FaMa-OVM, a tool support for the automated analysis of both, OVMs and attribute-aware OVMs.

To illustrate the feasibility of our approach, we have presented a study of a case from the automotive domain, showing that we were able to perform the analysis of a non-trivial real-world product line, allowing to detect false optional and dead elements in the OVM that represents the variability of such product line, and to verify attribute conditions as well. FaMa-OVM is based on a framework for the analysis of feature models, which simplified the development of our tool.

Based on our experience, we have observed that although feature models and OVM are similar, and the same techniques from the feature models context can be applied to automate the analysis of OVM, the realisation of our approach was not straightforward. Next, we point out some differences in automating the analysis of OVMs with respect to automating the analysis of feature models.

- With regards to the modelling concepts. First, we have used the variation points to make explicit the abstract elements in the OVM. We have taken the advantage that its syntax distinguishes between variation points and variants, and the former is usually used to structure variability. Second, unlike feature models, OVM only documents information about the variability of product line artefacts, but not data information. Therefore, we could not annotate OVM elements with attributes as in feature models, instead we had to relate OVMs with external models in which attributes are specified.
- With regards to the selective mapping of OVM into CSP. In feature models, the selective mapping is done in a two-step process. First, a feature model is transformed into a propositional formula that contains the variables equivalent to all features. Second, the resulting formula is transformed into a propositional formula in which there is no variable equivalent to abstract features. Contrarily, we have done the selective mapping of OVM using a single-step, *i.e.*, we transform OVMs directly into a CSP with no variables equivalent to abstract elements.
- With regards to the FaMa-OVM implementation, we have implemented the full and the selective mapping using a specific solver. The implementation is based on the FaMa framework, which has simplified the development of our tool, since key components, such as reasoners, readers, and writers were partially reused. With FaMa-OVM we have demonstrated that the analysis of OVMs, as in the case of feature models, can be computationally expensive. This is particularly true for those operations that need to compute all possible solutions beforehand to be able to find a solution, such as the *number of variations* operation. Furthermore, the analysis of variability models with attributes can be even more complex, as for example when asking for the *optimal* solution or for the *most representative variation*, since there is the need to compute the optimal solution from the total set of solutions.

8.2 Discussion, limitations and extensions

We next discuss some of the decisions that we have made in this dissertation highlighting its main limitations and possible extensions.

- **Is our Attribute-based Model adequate and sufficiently comprehen-**

sive? Although we have provided a model to express attributes and constraints on attributes, we did not rigorously define its elements and neither have we investigated how it should be defined in order to be a comprehensive model, so that it can be used for any other variability modelling technique.

Extension: Study the domain of attributes of product line artefacts in order to provide a domain-specific language that can be used to express attributes and constraints on attributes, regardless of the variability model being used for expressing variability.

- **Which type of attributes can be modelled with our approach and how are their values defined?** The compositionality of certain attributes, such as security, usability, and performance is not obvious and often hard to define. We deal with measurable attributes that are technically known and that can be composed by means of functions on individual values. These attributes can be functional or non-functional. A limitation of our approach is that the same attribute can only be involved in multiple functions when it is possible to find a common neutral value for these functions. Furthermore, we did not provide a systematic way to assign values to attributes.

Extension: Complement our approach with a systematic way of defining attribute values. For this purpose we intend to use some of the works identified in the literature [149–151].

- **Are all operations contemplated in our approach?** We have not defined to OVM all the operations identified for the analysis of feature models. In addition, we deal with operations that extract information from models without modifying them. To provide support for staged configuration of OVM, operations that modify the model have to be studied. This operation is very useful to support the user in deriving variations of OVM. This operation would have an impact on the states of the variability elements, which in our approach are only selected and removed; other states such as decided and undecided have to be considered.

Extension:

- ◇ Extending our approach with modification operations.
 - ◇ Provide new analysis operations that can be applied to OVM.
- **OVM semantics.** Although we have discussed two different semantics for OVMs, namely: Full and Selective. We have only addressed full se-

antics when providing support for the automated analysis of attribute-aware OVMs.

Extension: Provide selective mapping from attribute-aware OVMs into CSP, and support for analysis operations as well.

- **Are CSPs the most appropriated intermediate representation?** We have chosen CSP as the intermediate representation for two reasons: *i)* it offers a high level of abstraction in the specification, since any CSP can always be translated into a propositional formula [21], and besides, the language of CSP solvers is more intuitive for developers than propositional-based solvers; and, *ii)* using CSP solvers enables to work with numerical values, such as integer, which allows dealing with attributes. One of the limitations of have chosen this paradigm is that it is well-known that CSPs are in general NP-complete problems. We have identified that, similar to the analysis of feature models, determining whether an OVM is void is an NP-complete problem as well. In addition, with our proof of concepts, we have determined that the addition of attributes to the analysis process highly increases the complexity of the problem when trying to find an optimal solution.

Conclusion: We believe that it is important to work in collaboration with other research areas (*e.g.*, Constraint Programming, Artificial Intelligence) in order to find other techniques that could better solve the problem.

- **Should we use formal language for specifying our approach?** We have not formally specify our approach. Although we have based our approach on the OVM semantics defined in [91], in this dissertation, we have presented an informal definition of the OVM semantics, the Attribute-based Model, and the analysis process itself. This has brought us some benefits in terms of ease of understanding for regular software engineers and developers, since they are more used to UML models and metamodels. However, using informal specification our approach lacks rigour.

Extension: Formalise our specification using a formal language.

8.3 Other future work

- **Relationship between OVM and base models.** In our approach we consider 1 : 1 relationships between OVM and configuration model elements. However, these relationships could be extended to be 1 : N with minor changes. In addition, we have considered that base model is a configuration model, which has limited the approach.

Future work:

- ◊ Extend our approach to allow 1:N relationships between OVM and configuration models.
 - ◊ Study how our approach could be applied to analyse OVM related to other kinds of base model.
- **Should variations also be distinguished by attributes?** In our approach, variations are distinguished by variability elements, but not by attributes. In other words, variations that have the same set of variability elements are considered equivalent variations, regardless their attribute values. However, in some cases, could be appropriate to differentiate variations by they attribute values as well. This means that different variations can have the same set of variability elements, but different values for their attributes.

Conclusion: At the time of writing this dissertation, the distinction of variations only by variability elements seems to be the most appropriate approach for the analysis of OVM.

Future work: Study the practical significance of considering attribute values as a way of distinguishing variations.

- **Tool support.** Although FaMa-OVM is already functional, it is only a prototype and there is much improvements to be done.

Future work:

- ◊ Perform functional testing of FaMa-OVM. For this purpose, we intend to use metamorphic testing techniques, such the one presented in [118].
 - ◊ Provide an user Web interface for FaMa-OVM.
- **Survey and research agenda.** During the development of this dissertation a question always came to light, “Which is the most appropriate variability modelling technique and in which context?” The state of the

art on this field has to be reviewed in order to summarise the differences between variability modelling techniques. More rigorous surveys are missing and this is a field that we plan to explore as well.

Future work: Revise the state of the art and the challenges ahead on the domain of variability modelling techniques.

- **MDD and transformation of OVM.** We have already started researching in this field, but we only have preliminary results. We believe that this is an open research area that has to be investigated to provide interoperability between feature models and OVM tools, and to leverage automated analysis of OVM.

Future work:

- ◇ Investigate more about the possibility of using model transformation to provide interoperability between feature models and OVM tools.
- ◇ Mapping OVM selection into base models and verifying that specific variation model is consequent with its OVM.

Part V
Appendices

Appendix A

Interoperability Between OVM and FM Tools

In this appendix, we report on the implementation of an algorithm to carry out FM to OVM transformations. The algorithm transforms the variable part of a feature model into an OVM, thus providing an explicit view of variability of a product line. We have devised model-to-model transformations using MOMENT2 [22, 23]. The transformations we have devised transform models described with the metamodel presented in Figure §A.1 into models described with the metamodel depicted in Figure §A.2.

In the following, we provide the transformation rules, *cf.* Programs §A.2–§A.18. Each rule performs an specific transformation of feature model elements into OVM elements. Every rule must be written inside Program §A.1.

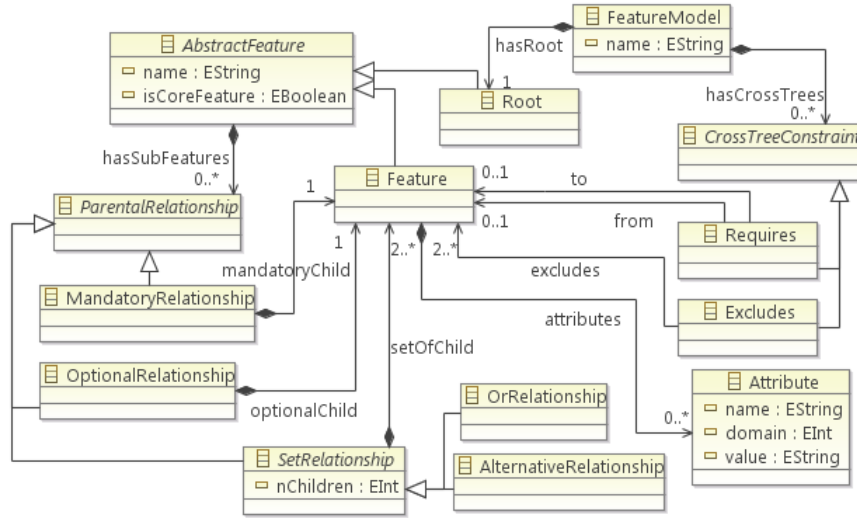


Figure A.1: The feature model metamodel.

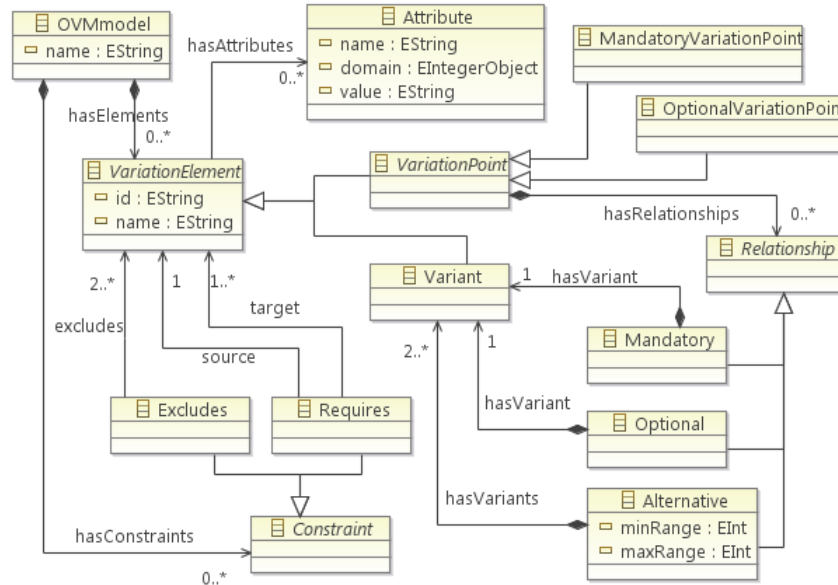


Figure A.2: The OVM metamodel.

```
import 'platform:/resource/fm2ovm/metamodels/FM_METAMODEL.ecore';
import 'platform:/resource/fm2ovm/metamodels/OVM_METAMODEL.ecore';

transformation fm2ovm( fm: FM_METAMODEL; ovm: OVM_METAMODEL ) {
  < ... Rules come here... >
}
```

Program A.1: *Main transformation method that contains every rule.*

```
rl FeatureModelToVmodel {
  nac ovm noVmodel {
    vm: Vmodel { }
  };

  lhs {
    fm {
      f: FeatureModel {
        name = n
      }
    }
    ovm { }
  };

  rhs {
    fm {
      f: FeatureModel {
        name = n
      }
    }

    ovm {
      vm: Vmodel {
        name = n
      }
    }
  };
}
```

Program A.2: *Map each feature model to an OVM.*

```
rl OptFeatureToMandatoryVP {
  nac ovm noVariationPoint {
    mdt: MandatoryVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = true,
        hasSubFeatures = op: OptionalRelationship {
          optionalChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    vm: Vmodel { }
  }
};
```

Program A.3: Map each core feature with optional child to a mandatory VP with an optional child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = true,
      hasSubFeatures = op: OptionalRelationship {
        optionalChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasElements = mdt: MandatoryVariationPoint {
      id = nf1,
      name = nf1,
      hasRelationships = optRel: Optional {
        hasVariant = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  }
}
};
}
```

Program A.3: Map each core feature with optional child to a mandatory VP with an optional child variant (Cont'd).

```

rl OptFeatureToOptionalVP {
  nac ovm noVariationPoint {
    opt: OptionalVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = op: OptionalRelationship {
          optionalChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    v1: Variant {
      id = nf1
    }
    vm: Vmodel { }
  }
};

rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = op: OptionalRelationship {
        optionalChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

```

Program A.4: Map each non-core feature with optional child to an optional VP with an optional child variant.

```
ovm {
  v1: Variant {
    id = nf1,
    name = nf1 + "_V"
  }
  vm: Vmodel {
    hasElements = opt: OptionalVariationPoint {
      id = nf1,
      name = nf1 + "_VP",
      hasRelationships = optRel: Optional {
        hasVariant = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  },
  hasConstraints = req1: Requires {
    source = v1: Variant { },
    target = opt: OptionalVariationPoint { }
  },
  hasConstraints = req2: Requires {
    source = opt: OptionalVariationPoint { },
    target = v1: Variant { }
  }
}
};
```

Program A.4: *Map each non-core feature with optional child to an optional VP with an optional child variant (Cont'd).*

```
rl OptFeatureToOptVariant {
  nac ovm noVariant {
    v: Variant {
      id = nf2
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = true,
        hasSubFeatures = op: OptionalRelationship {
          optionalChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    vm: Vmodel {
      hasElements = mdt: MandatoryVariationPoint {
        id = nf1
      }
    }
  }
};
```

Program A.5: Map each optional feature with parent core feature to an optional variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = true,
      hasSubFeatures = op: OptionalRelationship {
        optionalChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasElements = mdt: MandatoryVariationPoint {
      id = nf1,
      hasRelationships = optRel: Optional {
        hasVariant = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  }
}
};
}
```

Program A.5: *Map each optional feature with parent core feature to an optional variant (Cont'd).*

```
rl OptFeatureToOptVariantNotCore {
  nac ovm noVariant {
    v: Variant {
      id = nf2
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = op: OptionalRelationship {
          optionalChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    opt: OptionalVariationPoint {
      id = nf1
    }
  }
};
```

Program A.6: *Map each optional feature with parent non-core feature to an optional variant.*

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = op: OptionalRelationship {
        optionalChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  opt: OptionalVariationPoint {
    id = nf1,
    hasRelationships = optRel: Optional {
      hasVariant = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  }
}
};
}
```

Program A.6: Map each optional feature with parent non-core feature to an optional variant (Cont'd).

```
rl MdtFeatureToOptionalVP {
  nac ovm noVariationPoint {
    opt: OptionalVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = mdt: MandatoryRelationship {
          mandatoryChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    v1: Variant {
      id = nf1
    }

    vm: Vmodel { }
  }
};
```

Program A.7: Map each non-core feature with mandatory child feature to an optional VP with mandatory child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = mdt: MandatoryRelationship {
        mandatoryChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  v1: Variant {
    id = nf1,
    name = nf1 + "_V"
  }
}

vm: Vmodel {
  hasElements = opt: OptionalVariationPoint {
    id = nf1,
    name = nf1 + "_VP",
    hasRelationships = mdtRel: Mandatory {
      hasVariant = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  },
  hasConstraints = req1: Requires {
    source = v1: Variant { },
    target = opt: OptionalVariationPoint { }
  },
  hasConstraints = req2: Requires {
    source = opt: OptionalVariationPoint { },
    target = v1: Variant { }
  }
}
};
}
```

Program A.7: *Map each non-core feature with mandatory child feature to an optional VP with mandatory child variant (Cont'd).*

```
rl MdtFeatureToMdtVariant {
  nac ovm noVariant {
    v: Variant {
      id = nf2
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = mdt: MandatoryRelationship {
          mandatoryChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    vm: Vmodel {
      hasElements = opt: OptionalVariationPoint {
        id = nf1
      }
    }
  }
};
```

Program A.8: Map each mandatory feature with non-core parent feature to a mandatory variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = mdt: MandatoryRelationship {
        mandatoryChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasElements = opt: OptionalVariationPoint {
      id = nf1,
      hasRelationships = mdtRel: Mandatory {
        hasVariant = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  }
}
};
}
```

Program A.8: *Map each mandatory feature with non-core parent feature to a mandatory variant (Cont'd).*

```
rl AlternativeCoreToMandatoryVP {
  nac ovm noVariationPoint {
    vp: MandatoryVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = true,
        hasSubFeatures = alt: AlternativeRelationship {
          setOfChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    vm: Vmodel { }
  }
};
```

Program A.9: Map each core feature with alternative child feature to a mandatory VP with an alternative child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = true,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasElements = vp: MandatoryVariationPoint {
      id = nf1,
      name = nf1,
      hasRelationships = alt1: Alternative {
        minRange = 1,
        maxRange = 1,
        hasVariants = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  }
}
};
}
```

Program A.9: Map each core feature with alternative child feature to a mandatory VP with an alternative child variant (Cont'd).

```
rl AlternativeToOptionalVP {
  nac ovm noVariationPoint {
    vp: OptionalVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = alt: AlternativeRelationship {
          setOfChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    v1: Variant {
      id = nf1
    }

    vm: Vmodel { }
  }
};
```

Program A.10: Map each non-core feature with alternative child feature to an optional VP with an alternative child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  v1: Variant {
    id = nf1,
    name = nf1 + "_V"
  }
}

vm: Vmodel {
  hasElements = opt: OptionalVariationPoint {
    id = nf1,
    name = nf1 + "_VP",
    hasRelationships = alt1: Alternative {
      minRange = 1,
      maxRange = 1,
      hasVariants = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  },
  hasConstraints = req1: Requires {
    source = v1: Variant { },
    target = opt: OptionalVariationPoint { }
  },
  hasConstraints = req2: Requires {
    source = opt: OptionalVariationPoint { },
    target = v1: Variant { }
  }
}
};
}
```

Program A.10: Map each non-core feature with alternative child feature to an optional VP with an alternative child variant (Cont'd).

```

rl AlternativeCoreToAlternative {
  nac ovm noAlternative {
    vp: VariationPoint {
      id = nf1,
      hasRelationships = alt1: Alternative {
        minRange = 1,
        maxRange = 1
      }
    }
  }
};

lhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1
  }
}
};

```

Program A.11: *Map each alternative feature to an alternative relationship and an alternative child variant.*

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1,
    hasRelationships = alt1: Alternative {
      minRange = 1,
      maxRange = 1,
      hasVariants = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  }
}
};
}
```

Program A.11: Map each alternative feature to an alternative relationship and an alternative child variant (Cont'd).

```
rl AlternativeToVariants {
  nac ovm noVariant {
    v: Variant {
      id = nf2
    }
  }
};

lhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

  ovm {
    vp: VariationPoint {
      id = nf1,
      hasRelationships = alt1: Alternative {
        maxRange = 1
      }
    }
  }
};
```

Program A.12: *Map each alternative feature to an alternative variant.*

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = alt: AlternativeRelationship {
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1,
    hasRelationships = alt1: Alternative {
      maxRange = 1,
      hasVariants = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  }
}
};
}
```

Program A.12: *Map each alternative feature to an alternative variant (Cont'd).*

```
r1 OrCoreToMandatoryVP {
  nac ovm noVariationPoint {
    vp: MandatoryVariationPoint {
      id = nf1
    }
  }
};

lhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = true,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel { }
}
};
```

Program A.13: Map each core feature with an OR child feature to a mandatory VP with an alternative child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = true,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasElements = vp: MandatoryVariationPoint {
      id = nf1,
      name = nf1,
      hasRelationships = alt1: Alternative {
        minRange = 1,
        maxRange = n,
        hasVariants = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    }
  }
}
};
}
```

Program A.13: *Map each core feature with an OR child feature to a mandatory VP with an alternative child variant (Cont'd).*

```
rl OrToOptionalVP {
  nac ovm noVariationPoint {
    opt: OptionalVariationPoint {
      id = nf1
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        isCoreFeature = false,
        hasSubFeatures = orRel: OrRelationship {
          nChildren = n,
          setOfChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    v1: Variant {
      id = nf1
    }
    vm: Vmodel { }
  }
};
```

Program A.14: Map each non-core feature with an OR child feature to an optional VP with an alternative child variant.

```

rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      isCoreFeature = false,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  v1: Variant {
    id = nf1,
    name = nf1 + "_V"
  }
  vm: Vmodel {
    hasElements = opt: OptionalVariationPoint {
      id = nf1,
      name = nf1 + "_VP",
      hasRelationships = alt1: Alternative {
        minRange = 1,
        maxRange = n,
        hasVariants = v: Variant {
          id = nf2,
          name = nf2
        }
      }
    },
    hasConstraints = req1: Requires {
      source = v1: Variant { },
      target = opt: OptionalVariationPoint { }
    },
    hasConstraints = req2: Requires {
      source = opt: OptionalVariationPoint { },
      target = v1: Variant { }
    }
  }
}
};
}

```

Program A.14: Map each non-core feature with an OR child feature to an optional VP with an alternative child variant (Cont'd).

```
rl OrToAlternative {
  nac ovm noAlternative {
    vp: VariationPoint {
      id = nf1,
      hasRelationships = alt1: Alternative {
        minRange = 1,
        maxRange = n
      }
    }
  }
};

lhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1
  }
}
};
```

Program A.15: Map each OR feature to an alternative relationship and an alternative child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1,
    hasRelationships = alt1: Alternative {
      minRange = 1,
      maxRange = n,
      hasVariants = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  }
}
};
}
```

Program A.15: Map each OR feature to an alternative relationship and an alternative child variant (Cont'd).

```
rl OrToVariants {
  nac ovm noVariant {
    v: Variant {
      id = nf2
    }
  };

  lhs {
    fm {
      f: AbstractFeature {
        name = nf1,
        hasSubFeatures = orRel: OrRelationship {
          nChildren = n,
          setOfChild = f2: Feature {
            name = nf2
          }
        }
      }
    }
  }

  ovm {
    vp: VariationPoint {
      id = nf1,
      hasRelationships = alt1: Alternative {
        maxRange = n
      }
    }
  }
};
```

Program A.16: Map each OR feature to an alternative child variant.

```
rhs {
  fm {
    f: AbstractFeature {
      name = nf1,
      hasSubFeatures = orRel: OrRelationship {
        nChildren = n,
        setOfChild = f2: Feature {
          name = nf2
        }
      }
    }
  }
}

ovm {
  vp: VariationPoint {
    id = nf1,
    hasRelationships = alt1: Alternative {
      maxRange = n,
      hasVariants = v: Variant {
        id = nf2,
        name = nf2
      }
    }
  }
}
};
}
```

Program A.16: *Map each OR feature to an alternative child variant (Cont'd).*

```
rl RequiresToRequires {
  nac ovm noRequires {
    req2: Requires {
      source = ve1: VariationElement {
        id = nf1
      },
      target = ve2: VariationElement {
        id = nf2
      }
    }
  }
};

lhs {
  fm {
    req: Requires {
      from = f1: Feature {
        name = nf1
      },
      to = f2: Feature {
        name = nf2
      }
    }
  }
}

ovm {
  vm: Vmodel { }
  ve1: VariationElement {
    id = nf1
  }
  ve2: VariationElement {
    id = nf2
  }
}
};
```

Program A.17: Map each requires constraint to a requires constraint.

```
rhs {
  fm {
    req: Requires {
      from = f1: Feature {
        name = nf1
      },
      to = f2: Feature {
        name = nf2
      }
    }
  }
}

ovm {
  vm: Vmodel {
    hasConstraints = req2: Requires {
      source = ve1: VariationElement {
        id = nf1
      },
      target = ve2: VariationElement {
        id = nf2
      }
    }
  }
}
};
}
```

Program A.17: *Map each requires constraint to a requires constraint (Cont'd).*

```
rl ExcludesToExcludes {
  nac ovm noExcludes {
    req2: Excludes {
      excludes = ve1: VariationElement {
        id = nf1
      },
      excludes = ve2: VariationElement {
        id = nf2
      }
    }
  }
};

lhs {
  fm {
    exc: Excludes {
      excludes = f1: Feature {
        name = nf1
      },
      excludes = f2: Feature {
        name = nf2
      }
    }
  }
}

ovm {
  vm: Vmodel { }
  ve1: VariationElement {
    id = nf1
  }
  ve2: VariationElement {
    id = nf2
  }
}
};
```

Program A.18: Map each *excludes* constraint to an *excludes* constraint.

```
rhs {
  fm {
    exc: Excludes {
      excludes = f1: Feature {
        name = nf1
      },
      excludes = f2: Feature {
        name = nf2
      }
    }
  }
}



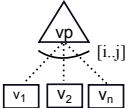
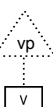

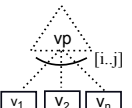
ovm {
  vm: Vmodel {
    hasConstraints = req2: Excludes {
      excludes = ve1: VariationElement {
        id = nf1
      },
      excludes = ve2: VariationElement {
        id = nf2
      }
    }
  }
}
};
}
```

Program A.18: *Map each excludes constraint to an excludes constraint (Cont'd).*

Appendix B

Selective Mapping Rules

This appendix presents the 92 rules for translating an OVM into a CSP applying the selective mapping, which were presented in Section [§5.3.2](#).

		Variability dependency	JaCoP-like notation
MANDATORY Variation Point	Mandatory	1  ▲	$v=1$
	Optional	2  +	No constraint
	Alternative	3  *	$\text{sum}(v_1, v_2, v_n) \text{ in } \{i..j\}$
OPTIONAL Variation Point	Mandatory	4  +	No constraint
	Optional	5  +	No constraint
	Alternative	6  *	$\text{sum}(v_1, v_2, v_n) \text{ in } \{0\} \cup \{i..j\}$

▲ when a vp and v are mandatory, v will be always equal to 1.

+ When a vp or v is optional, no constraint is generated.

* when a mandatory vp has an alternative relationship with its child variants, the sum of its child variants will be always in $\{i..j\}$. Since they must be selected. However, when a vp is optional, the sum of its child variants will be always in $\{i..j\}$ or zero, since they may be selected or not.

Table B.1: Mapping rules for variability dependencies.

Requires_VP_VP Constraint dependency		JaCoP-like notation	Requires_VP_VP Constraint dependency		JaCoP-like notation
MANDATORY Variation Point (left-hand side)	Mandatory variability dependency (left-hand side)	1		▲	No constraint
		2			$v_2=1$
		3		+	No constraint
		4		+	No constraint
		5		★	No constraint
		6			$\text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$
	Optional variability dependency (left-hand side)	7		▲	No constraint
		8			$v_2=1$
		9		+	No constraint
		10		+	No constraint
		11		★	No constraint
		12			$\text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$
MANDATORY Variation Point (left-hand side)	Alternative variability dependency (left-hand side)	13		▲	No constraint
		14			$v_1=1$
		15		+	No constraint
		16		+	No constraint
		17		★	No constraint
		18			$\text{sum}(q_1, q_2, q_m) \text{ in } \{x..y\}$

▲ when vp and v in the right-hand side are mandatory, there is no constraint because v will be always equal to 1 according to variability dependency rule number 1.

+

When vp or v in the right-hand side is optional, there is no constraint, according to variability dependency rules numbers 2, 4 and 5.

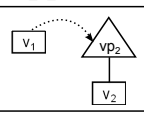
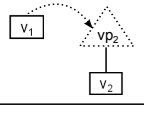
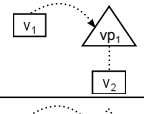
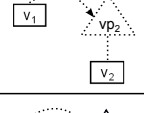
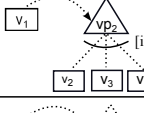
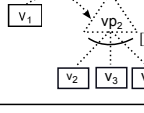
★ when vp in the right-hand side is mandatory and the relationship with child variants is alternative, there is no constraint because the sum of its child variants will be always in the cardinality, according to variability dependency rule number 3.

Table B.2: VP requires VP mapping rules for mandatory VPs.

		Requires_VP_VP Constraint dependency	JaCoP-like notation			Requires_VP_VP Constraint dependency	JaCoP-like notation		
OPTIONAL Variation Point (left-hand side)	Mandatory variability dependency (left-hand side)	19		No constraint	OPTIONAL Variation Point (left-hand side)	Alternative variability dependency (left-hand side)	31		No constraint
		20		if $(v_1 > 0)$ $v_2 > 0$			32		if $\text{sum}(v_2, v_3, v_n)$ in $\{i..j\}$ $(v_1 > 0)$
		21		No constraint			33		No constraint
		22		No constraint			34		No constraint
		23		No constraint			35		No constraint
		24		if $(v_1 > 0)$ $\text{sum}(v_2, v_3, v_n)$ in $\{i..j\}$			36		if $\text{sum}(v_1, v_2, v_n)$ in $\{i..j\}$ $\text{sum}(q_1, q_2, q_m)$ in $\{x..y\}$
		25		No constraint					
		26		if $(v_1 > 0)$ $v_2 > 0$					
		27		No constraint					
		28		No constraint					
OPTIONAL Variation Point (left-hand side)	Optional variability dependency (left-hand side)	29		No constraint					
		30		if $(v_1 > 0)$ $\text{sum}(v_2, v_3, v_n)$ in $\{i..j\}$					

▲ when vp and v in the right-hand side are mandatory, there is no constraint because v will be always equal to 1 according to variability rule number 1.
 + When vp or v in the right-hand side is optional, there is no constraint, according to variability dependency rules numbers 2, 4 and 5.
 ★ when vp in the right-hand side is mandatory and the relationship with child variants is alternative, there is no constraint because the sum of its child variants will be always in the cardinality, according to variability dependency rule number 3.

Table B.3: VP requires VP mapping rules for optional VPs.

	Requires_V_VP Constraint dependency	JaCoP-like notation
VARIANT (left-hand side)	1 	▲ No constraint
	2 	if ($v_1 > 0$) $v_2 > 0$
	3 	+ No constraint
	4 	+ No constraint
	5 	* No constraint
	6 	if ($v_1 > 0$) $\text{sum}(v_2, v_3, v_n) \text{ in } \{i..j\}$

▲ when vp and v in the right-hand side are mandatory, there is no constraint because v will be always equal to 1 according to variability dependency rule number 1.

+ When vp or v in the right-hand side is optional, there is no constraint, according to variability dependency rules numbers 2, 4 and 5.

* when vp in the right-hand side is mandatory and the relationship with child variants is alternative, there is no constraint because the sum of its child variants will be always in the cardinality, according to variability dependency rule number 3.

Table B.4: V requires VP mapping rules.


Requires_V_V Constraint dependency	JaCoP-like notation
1 	if ($v_1 > 0$) $v_2 > 0$

Table B.5: V requires V mapping rules.

MANDATORY Variation Point (left-hand side)		Excludes_VP_VP Constraint dependency	JaCoP-like notation	MANDATORY Variation Point (left-hand side)		Excludes_VP_VP Constraint dependency	JaCoP-like notation	
MANDATORY variability dependency (left-hand side)	1			MANDATORY variability dependency (left-hand side)	13			
	2				14			
	3		if $(v_1 > 0)$ $v_2 = 0$		15		if $\text{sum}(v_2, v_3, v_n)$ in $\{i..j\}$ $(v_1 = 0)$	
	4				16			
	5			Alternative variability dependency (left-hand side)	17			
	6		if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$		18		if $\text{sum}(v_1, v_2, v_n)$ in $\{i..j\}$ $q_1 = 0, q_2 = 0, q_m = 0$	
	7							
	8							
	Optional variability dependency (left-hand side)	9		if $(v_1 > 0)$ $v_2 = 0$				
		10						
		11						
		12		if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$				

Table B.6: VP excludes VP mapping rules for mandatory VPs.

		Excludes_VP_VP Constraint dependency	JaCoP-like notation			Excludes_VP_VP Constraint dependency	JaCoP-like notation	
OPTIONAL Variation Point (left-hand side)	Mandatory variability dependency (left-hand side)	19			OPTIONAL Variation Point (left-hand side)	31		
		20				32		
		21		if $(v_1 > 0)$ $v_2 = 0$		33		if $\text{sum}(v_2, v_3, v_n)$ in $\{i..j\}$ $(v_1 = 0)$
		22				34		
		23				35		if $\text{sum}(v_1, v_2, v_n)$ in $\{i..j\}$ $q_1 = 0, q_2 = 0, q_m = 0$
	24		if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$	36				
	25							
	Optional variability dependency (left-hand side)	26		if $(v_1 > 0)$ $v_2 = 0$				
		27						
		28						
29			if $(v_1 > 0)$ $v_2 = 0, v_3 = 0, v_n = 0$					
30								

Table B.7: VP excludes VP mapping rules for optional VPs.

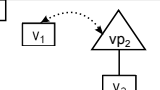
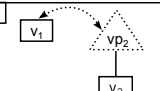
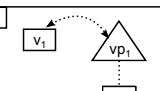
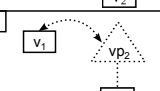
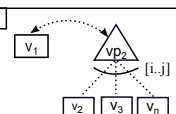
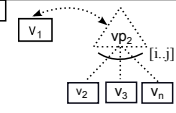
	Excludes_V_VP Constraint dependency	JaCoP-like notation
VARIANT (left-hand side)	1 	if ($v_1 > 0$) $v_2 = 0$
	2 	
	3 	
	4 	
	5 	if ($v_1 > 0$) $v_2 = 0, v_3 = 0, v_n = 0$
	6 	

Table B.8: *V* excludes *VP* mapping rules.

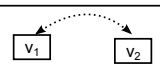
	Excludes_V_V Constraint dependency	JaCoP-like notation
1		if ($v_1 > 0$) $v_2 = 0$

Table B.9: *V* excludes *V* mapping rules.

Appendix C

RFW Product Line Specification

This appendix describes the complete AOVm specification for the RFW product line presented in Chapter §7.

Domain constraints	
V10Hazardous situation alarm	IMPLIES VP11.Range >= 50;
V17Emergency brake	IMPLIES VP11.Range >= 50;
V18Road w/ right of way start	IMPLIES VP11.Range >= 25;
V19City limit	IMPLIES VP11.Range >= 50;
V20Crossroads	IMPLIES VP11.Range >= 25;
V21Home zone entry	IMPLIES VP11.Range >= 25;
V22Road w/ right of way end	IMPLIES VP11.Range >= 10;
V23End of city limit	IMPLIES VP11.Range >= 10;
V24Traffic has priority	IMPLIES VP11.Range >= 10;
V25Home zone end	IMPLIES VP11.Range >= 10;
V26No vehicles	IMPLIES VP11.Range >= 25;
V27No cars	IMPLIES VP11.Range >= 25;
V28No vehicles over max width > Xm	IMPLIES VP11.Range >= 25;
V29No vehicles w/ weight > 3.5t	IMPLIES VP11.Range >= 25;
V30No vehicles over max gross weight g > Xt	IMPLIES VP11.Range >= 25;
V31Do not enter	IMPLIES VP11.Range >= 25;
V32No vehicles over max height h > Xm	IMPLIES VP11.Range >= 25;
V33No stopping	IMPLIES VP11.Range >= 10;
V34Danger	IMPLIES VP11.Range >= 50;
V35Side winds	IMPLIES VP11.Range >= 50;
V36Slippery road	IMPLIES VP11.Range >= 50;
V37Risk of ice	IMPLIES VP11.Range >= 50;
V38Bend	IMPLIES VP11.Range >= 80;
V39Traffic queues	IMPLIES VP11.Range >= 80;

Domain constraints	
V40Stop and give way	IMPLIES VP11.Range >= 50;
V41No overtaking	IMPLIES VP11.Range >= 50;
V42No overtaking end	IMPLIES VP11.Range >= 50;
V43No overtaking vehicles > 3.5t	IMPLIES VP11.Range >= 50;
V44End of prohibitions	IMPLIES VP11.Range >= 25;
V45Yield	IMPLIES VP11.Range >= 80;
V46Maximum speed X Km/h	IMPLIES VP11.Range >= 50;
V47One way	IMPLIES VP11.Range >= 25;
V48Maximum speed of X Km/h end	IMPLIES VP11.Range >= 10;
V49No overtaking vehicles > 3.5t end	IMPLIES VP11.Range >= 10;
V11Show warning sign	IMPLIES TotalAccuracy <= 30;
V12Display and sound indication	IMPLIES TotalAccuracy <= 30;
V13Warn for no stopping sign	IMPLIES TotalAccuracy <= 10;
V15Show on display	IMPLIES TotalAccuracy <= 30;
V16Display and sound indication	IMPLIES TotalAccuracy <= 30;
V17Emergency brake	IMPLIES TotalAccuracy <= 10;
V18Road w/ right of way start	IMPLIES TotalAccuracy <= 30;
V19City limit	IMPLIES TotalAccuracy <= 30;
V20Crossroads	IMPLIES TotalAccuracy <= 30;
V21Home zone entry	IMPLIES TotalAccuracy <= 30;
V22Road w/ right of way end	IMPLIES TotalAccuracy <= 30;
V23End of city limit	IMPLIES TotalAccuracy <= 30;
V24Traffic has priority	IMPLIES TotalAccuracy <= 30;
V25Home zone end	IMPLIES TotalAccuracy <= 30;
V26No vehicles	IMPLIES TotalAccuracy <= 10;
V27No cars	IMPLIES TotalAccuracy <= 10;
V28No vehicles over max width > Xm	IMPLIES TotalAccuracy <= 10;
V29No vehicles w/ weight > 3.5t	IMPLIES TotalAccuracy <= 10;
V30No vehicles over max gross weight g > Xt	IMPLIES TotalAccuracy <= 10;
V31Do not enter	IMPLIES TotalAccuracy <= 10;
V32No vehicles over max height h > Xm	IMPLIES TotalAccuracy <= 10;
V33No stopping	IMPLIES TotalAccuracy <= 10;
V34Danger	IMPLIES TotalAccuracy <= 30;
V35Side winds	IMPLIES TotalAccuracy <= 30;
V36Slippery road	IMPLIES TotalAccuracy <= 30;
V37Risk of ice	IMPLIES TotalAccuracy <= 30;
V38Bend	IMPLIES TotalAccuracy <= 10;
V39Traffic queues	IMPLIES TotalAccuracy <= 30;
V40Stop and give way	IMPLIES TotalAccuracy <= 10;
V41No overtaking	IMPLIES TotalAccuracy <= 30;
V42No overtaking end	IMPLIES TotalAccuracy <= 30;
V43No overtaking vehicles > 3.5t	IMPLIES TotalAccuracy <= 30;
V44End of prohibitions	IMPLIES TotalAccuracy <= 30;

Domain constraints	
V45Yield	IMPLIES TotalAccuracy <= 10;
V46Maximum speed X Km/h	IMPLIES TotalAccuracy <= 30;
V48Maximum speed of X Km/h end	IMPLIES TotalAccuracy <= 30;
V49No overtaking vehicles > 3.5t	IMPLIES TotalAccuracy <= 30;

Table C.2: RFW domain constraints

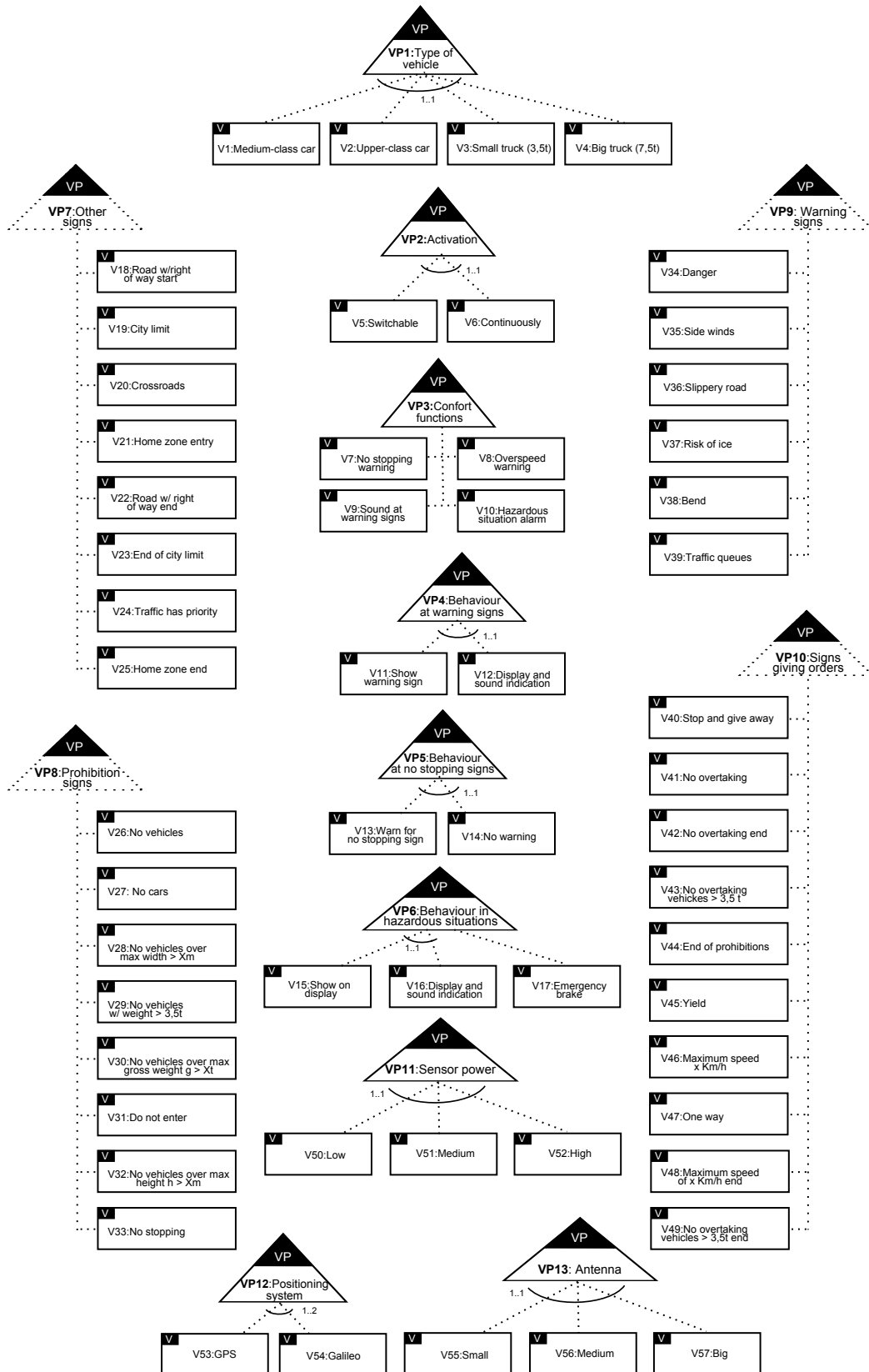


Figure C.1: RFW OVM without excludes and requires dependencies.

Variation Point	Variant	Type	Variation Point	Variant
VP1:Type of vehicle		requires	VP9:Warning signs	
VP1:Type of vehicle		requires	VP10:Signs giving orders	
VP1:Type of vehicle		requires	VP8:Prohibition signs	
VP1:Type of vehicle		requires	VP7:Other signs	
	V10:Hazardous situation alarm	requires		V16:Display and sound indication
	V15:Show on display	excludes		V10:Hazardous situation alarm
	V9:Sound at warning signs	requires		V12:Display and sound indication
	V9:Sound at warning signs	requires		V34:Danger
	V11:Show warning sign	excludes		V9:Sound at warning signs
	V8:Overspeed warning	requires		V48:Maximum speed of x km/h end
	V8:Overspeed warning	requires		V19:City limit
	V8:Overspeed warning	requires		V21:Home zone entry
	V7:No stopping warning	requires		V33:No stopping
	V7:No stopping warning	requires		V13:Warn for no stopping sign
	V7:No stopping warning	excludes		V14:No warning
	V1:Medium-class car	requires		V26:No vehicles
	V1:Medium-class car	requires		V27:No cars
	V1:Medium-class car	requires		V31:Do not enter
	V1:Medium-class car	requires		V41:No overtaking
	V1:Medium-class car	requires		V5:Switchable
	V2:Upper-class car	requires		V26:No vehicles
	V2:Upper-class car	requires		V27:No cars
	V2:Upper-class car	requires		V31:Do not enter
	V2:Upper-class car	requires		V41:No overtaking
	V2:Upper-class car	requires		V6:Continuously
	V2:Upper-class car	requires		V8:Overspeed warning
	V3:Small truck (3,5t)	requires		V26:No vehicles
	V3:Small truck (3,5t)	requires		V27:No cars
	V3:Small truck (3,5t)	requires		V31:Do not enter
	V3:Small truck (3,5t)	requires		V41:No overtaking
	V3:Small truck (3,5t)	requires		V5:Switchable
	V4:Big truck (7,5t)	requires		V29:No vehicles w/ weight > 3,5t
	V4:Big truck (7,5t)	requires		V30:No vehicles over max gross weight g > x
	V4:Big truck (7,5t)	requires		V43:No overtaking vehicles > 3,5t
	V4:Big truck (7,5t)	requires		V41:No overtaking
	V4:Big truck (7,5t)	requires		V6:Continuously
	V17:Emergency brake	requires		V10:Hazardous situation alarm
	V53:GPS	excludes		V55:Small

Table C.1: *RFW excludes and requires dependencies.*

Appendix D

Acronyms

BDD. Binary Decision Diagram.

BeTTy. BEnc Benchmarking and TesTing on the analYsis of feature models.

CNF. Conjunctive Normal Form.

COVAMOF. ConIPF Variability Modelling Framework.

CSP. Constraint Satisfaction Problem.

FaMa. FeAture Model Analyzer.

OVM. Orthogonal Variability Model.

SAT. Satisfiability Problem.

OMG. Object Management Group.

MOF. MetaObject Facility.

CVL. Common Variability Language.

MDD. Model-Driven Development.

VM. Variability model.

UML. Unified Modelling Language.

Bibliography

- [1] K. R. Apt. *Principles of constraint programming*. Cambridge University Press, Cambridge, United Kingdom, 2003
- [2] F. Bachmann, M. Goedicke, J. Leite, R. Nord, K. Pohl, B. Ramesh, and A. Vilbig. [A meta-model for representing variability in product family development](#). In F. van der Linden, editor, *Software Product-Family Engineering*, volume 3014 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2004.
- [3] R. Bachmeyer and H. Delugach. [A conceptual graph approach to feature modeling](#). In U. Priss, S. Polovina, and R. Hill, editors, *Conceptual Structures: Knowledge Architectures for Smart Applications*, volume 4604 of *Lecture Notes in Computer Science*, pages 179–191. Springer, 2007.
- [4] E. Bagheri, T. Di Noia, A. Ragone, and D. Gasevic. [Configuring software product line feature models based on stakeholders' soft and hard requirements](#). In *Proceedings of the 14th international conference on Software product lines (SPLC'10)*, volume 6287 of *Lecture Notes in Computer Science*, pages 16–31. Springer, 2010.
- [5] D. Batory. *Feature models, grammars, and propositional formulas*. In *Proceedings of the 9th International Conference on Software Product Lines (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, September 2005
- [6] D. Batory, D. Benavides, and A. Ruiz-Cortes. [Automated analysis of feature models: challenges ahead](#). *Communications of the ACM*, 49: 45–47, December 2006.
- [7] D. Batory, R. E. Lopez Herrejon, and J. P. Martin. [Generating Product-Lines of Product-Families](#). In *Proceedings of the 17th IEEE international*

- conference on Automated software engineering, ASE '02, Washington, DC, USA, 2002. IEEE CS.
- [8] D. Benavides. *On the automated analysis of software product lines using feature models*. PhD thesis, University of Sevilla, Spain, 2007
- [9] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. *Coping with automatic reasoning on software product lines*. In *Proceedings of the 2nd Groningen Workshop on Software Variability Management*, November 2004
- [10] D. Benavides, A. Ruiz-Cortés, and P. Trinidad. *Using constraint programming to reason on feature models*. In *The 17th International Conference on Software Engineering and Knowledge Engineering (SEKE'05)*, pages 677–682, 2005
- [11] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. *A first step towards a framework for the automated analysis of feature models*. In *Workshop on Managing Variability for Software Product Lines: Working With Variability Mechanisms*, pages 39–45, 2006
- [12] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. *FAMA: Tooling a framework for the automated analysis of feature models*. In *First International Workshop on Variability Modelling of Software-intensive Systems*, pages 129–134, 2007
- [13] D. Benavides, S. Segura, and A. Ruiz-Cortés. *Automated analysis of feature models 20 years later: A literature review*. *Information Systems*, 35(6):615 – 636, 2010
- [14] D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. *Using java csp solvers in the automated analyses of feature models*. In R. Lämmel, J. Saraiva, and J. Visser, editors, *Generative and Transformational Techniques in Software Engineering*, volume 4143 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2006.
- [15] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. *Automated reasoning on feature models*. In O. Pastor and J. F. e Cunha, editors, *Advanced Information Systems Engineering (CAISE'05)*, volume 3520 of *Lecture Notes in Computer Science*, pages 381–390. Springer, 2005.
- [16] N. Bencomo. *Supporting the modelling and generation of reflective middleware families and applications using dynamic variability*. PhD thesis, Lancaster University, UK, 2008

- [17] N. Bencomo, P. Grace, C. A. Flores-Cortés, D. Hughes, and G. S. Blair. *Genie: supporting the model driven development of reflective, component-based adaptive systems*. In *30th International Conference on Software Engineering (ICSE'08)*, pages 811–814, 2008
- [18] D. L. Berre and A. Parrain. SAT4j solver. <http://sat4j.org/>, accessed July 2011
- [19] BeTTY Framework. <http://www.isa.us.es/betty>, accessed August 2011
- [20] B. Bollig and I. Wegener. *Improving the variable ordering of OBDDs is NP-complete*. *IEEE Transactions on Computers*, 45(9):993–1002, 1996.
- [21] L. Bordeaux, Y. Hamadi, and L. Zhang. *Propositional satisfiability and constraint programming: A comparative survey*. *ACM Comput. Surv.*, 38(12), December 2006.
- [22] A. Boronat, R. Heckel, and J. Meseguer. *Rewriting logic semantics and verification of model transformations*. In M. Chechik and M. Wirsing, editors, *Fundamental Approaches to Software Engineering*, volume 5503 of *Lecture Notes in Computer Science*, pages 18–33. Springer, Berlin, Heidelberg, 2009.
- [23] A. Boronat and J. Meseguer. *An algebraic semantics for MOF*. *Formal Aspects of Computing*, 22:269–296, 2010.
- [24] J. Bosch. *From software product lines to software ecosystems*. In *Proceedings of the 13th International Software Product Line Conference (SPLC'09)*, pages 111–119, Pittsburgh, PA, USA, 2009
- [25] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, J. Obbink, and K. Pohl. *Variability issues in software product lines*. In F. van der Linden, editor, *Software Product-Family Engineering*, volume 2290 of *Lecture Notes in Computer Science*, pages 303–338. Springer, April 2002.
- [26] R. Bryant. *Graph-based algorithms for boolean function manipulation*. *IEEE Transactions on Computers*, 35(8):677–691, 1986
- [27] S. Bühne, K. Lauenroth, and K. Pohl. *Why is it not sufficient to model requirements variability with feature models?* In Aoyama, M.; Houdek, F.; Shigematsu, T. (Eds.) *Proceedings of Workshop: Automotive Requirements Engineering (AURE04)*, Los Alamitos, 2004. IEEE CS

- [28] G. H. Campbell, S. R. Faulk, and D. M. Weiss. *Introduction to synthesis*. Technical report INTRO-SYNTHESIS-PROCESS 90019-N, Software Productivity Consortium, Herndon, VA, USA, 1990
- [29] F. Cao, B. Bryant, C. Burt, Z. Huang, R. Raje, A. Olson, and M. Auguston. *Automating feature-oriented domain analysis*. In *International Conference on Software Engineering Research and Practice (SERP'03)*, pages 944–949, June 2003
- [30] M. Carlsson, G. Ottosson, and B. Carlson. *An open-ended finite domain constraint solver*. In *Proceedings of the 9th International Symposium on Programming Languages: Implementations, Logics, and Programs: Including a Special Track on Declarative Programming Languages in Education, PLILP '97*, pages 191–206. Springer, 1997.
- [31] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All about Maude - a high-performance logical framework: how to specify, program and verify systems in rewriting logic*. Springer, Berlin, Heidelberg, 2007
- [32] P. Clements and L. Northrop. *Software product lines: Practices and patterns*. SEI Series in Software Engineering. Addison–Wesley, Reading, Massachusetts, USA, 2001
- [33] S. P. Consortium. *Reuse-driven software processes*. Technical report SPC-92019-CMC, Version 02.00.03, Software Productivity Consortium Services Corporation, November 1993
- [34] S. Cook. *The complexity of theorem-proving procedures*. In *Conference Record of Third Annual ACM Symposium on Theory of Computing*, pages 151–158, 1971
- [35] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. *Generative programming for embedded software: An industrial experience report*. In *1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering*, pages 156–172. Springer, October 2002
- [36] K. Czarnecki, S. Helsen, and U. Eisenecker. *Staged configuration using feature models*. In *Third Software Product Line Conference*, volume 3154 of *Lecture Notes in Computer Science*, pages 266–282. Springer, September 2004

- [37] K. Czarnecki, S. Helsen, and U. Eisenecker. *Formalizing cardinality-based feature models and their specialization*. *Software Process: Improvement and Practice*, 10(1):7–29, 2005
- [38] K. Czarnecki and C. H. P. Kim. *Cardinality-based feature modeling and constraints: a progress report*. In *International Workshop on Software Factories at OOPSLA'05*. ACM, 2005
- [39] D. Dhungana, P. Heymans, and R. Rabiser. *A formal semantics for decision-oriented variability modeling with dopler*. In *Fourth International Workshop on Variability Modelling of Software-Intensive Systems*, pages 29–35, 2010
- [40] D. Dhungana, R. Rabiser, P. Grünbacher, and T. Neumayer. *Integrated tool support for software product line engineering*. In *22nd IEEE/ACM International Conference on Automated Software Engineering*, pages 533–534, New York, NY, USA, 2007. ACM
- [41] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. Galindo. *Configuration of multi product lines by bridging heterogeneous variability modeling approaches*. In *Proc. of the 15th International Software Product Line Conference (SPLC 2011)*, pages 120–129, August 2011
- [42] D. Dhungana, P. Grünbacher, and R. Rabiser. *The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study*. *Automated Software Engineering*, 18(1):77–114, March 2011.
- [43] O. Djebbi, C. Salinesi, and D. Diaz. *Deriving product line requirements: the RED-PL guidance approach*. *Asia-Pacific Software Engineering Conference (ASPEC'07)*, pages 494–501, 2007.
- [44] A. Durán, D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. *Flame: A formal framework for the automated analysis of software product lines using feature models*. *ACM Transactions on Software Engineering and Methodology*, 2012, submitted
- [45] A. Elfaki, S. Phon-Amnuaisuk, and C. Ho. *Investigating inconsistency detection as a validation operation in software product line*. In R. Lee and N. Ishii, editors, *Software Engineering Research, Management and Applications 2009*, volume 253 of *Studies in Computational Intelligence*, pages 159–168. Springer, 2009.

- [46] L. Etxeberria and G. Sagardui. *Evaluation of quality attribute variability in software product families*. In *15th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2008)*, pages 255–264. IEEE, March 2008.
- [47] FaMa Tool Suite. <http://www.isa.us.es/fama/>, accessed August 2011
- [48] FaMa-OVM. <http://www.isa.us.es/fama-ovm/>, accessed August 2011
- [49] S. Fan and N. Zhang. *Feature model based on description logics*. In *Knowledge-Based Intelligent Information and Engineering Systems*, volume 4252, pages 1144–1151. Springer, 2006
- [50] A. Felfernig, G. E. Friedrich, and D. Jannach. *UML as domain specific language for the construction of Knowledge-Based configuration systems*. *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, 10(4):449–469, 2000
- [51] D. Fernandez-Amoros, R. Heradio, and J. Cerrada. *Inferring information from feature diagrams to product line economic models*. In *Proceedings of the Software Product Line Conference*, 2009
- [52] D. Fey, R. Fajta, and A. Boros. *Feature modeling: A Meta-Model to enhance usability and usefulness*. In G. Chastek, editor, *Software Product Lines*, volume 2379 of *Lecture Notes in Computer Science*, pages 198–216. Springer, July 2002
- [53] R. Finkel and B. O’Sullivan. *Reasoning about conditional constraint specification problems and feature models*. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 25(2):163–174, 2011
- [54] F. Fleurey, Ø. Haugen, B. Møller-Pedersen, G. K. Olsen, A. Svendsen, and X. Zhang. *A generic language and tool for variability modeling*. Technical report A13505, SINTEF, Oslo, Norway, 2009
- [55] T. Forster, D. Muthig, and D. Pech. *Understanding decision models – visualization and complexity reduction of software variability*. In *Proceedings of the 2nd Int. Workshop on Variability Modelling of Software-intensive Systems*, Essen, Germany, January 2008
- [56] J. A. Galindo, D. Benavides, and S. Segura. *Debian packages repositories as software product line models. towards automated analysis*. In *Proceeding of the First International Workshop on Automated Configuration and Tailoring of Applications (ACOTA)*, 2010

- [57] F. Garcia, M. Bertoa, C. Calero, A. Vallecillo, F. Ruiz, M. Piattini, and M. Genero. *Towards a consistent terminology for software measurement*. *Information and Software Technology*, 48(8):631–644, August 2006
- [58] R. Gheyi, T. Massoni, and P. Borba. *A theory for feature models in Alloy*. In *Proceedings of the ACM SIGSOFT First Alloy Workshop*, pages 71–80, Portland, United States, 2006
- [59] R. Gheyi, T. Massoni, and P. Borba. *Algebraic laws for feature models*. *Journal of Universal Computer Science*, 14(21):3573–3591, 2008.
- [60] GNU Prolog, <http://www.gprolog.org>, accessed July 2011
- [61] H. Gomaa. *Designing software product lines with UML 2.0: From use cases to Pattern-Based software architectures*. In *Proceedings of the 10th International Software Product Line Conference (SPLC'06)*. IEEE, 2006.
- [62] H. Gomaa and M. E. Shin. *Multiple-view modelling and meta-modelling of software product lines*. *Software, IET*, 2(2):94–122, April 2008
- [63] H. Gomaa. *Designing software product lines with UML: From use cases to pattern-based software architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004
- [64] M. Griss, J. Favaro, and M. d'Alessandro. *Integrating feature modeling with the RSEB*. In *5th Intl. Conference on Software Reuse*, pages 76–85. IEEE CS, June 1998
- [65] G. Halmans, K. Pohl, and E. Sikora. *Documenting Application-Specific adaptations in software product line engineering*. In Z. Bellahsène and M. Léonard, editors, *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 109–123. Springer, 2008.
- [66] A. Hemakumar. *Finding contradictions in feature models*. In *First International Workshop on Analyses of Software Product Lines (ASPL)*, pages 183–190, 2008
- [67] A. Heuer, K. Lauenroth, M. Müller, and J.-N. Scheele. *Towards effective visual modeling of complex software product lines*. In *3rd International Workshop on Visualisation in Software Product Line Engineering (VIS-PLE) in Proceedings of the SPLC'10*, volume 2, pages 229–237, 2010

- [68] P. Istoan, J. Klein, G. Perouin, and J.-M. Jézéquel. *A metamodel-based classification of variability modeling approaches*. In *Proceedings of VARY International Workshop affiliated with ACM/IEEE 14th International Conference on Model Driven Engineering Languages and Systems (MODELS'11)*, pages 23–32, 2011
- [69] I. Jacobson, M. Griss, and P. Jonsson. *Software reuse. architecture, process and organization for business success*. Addison-Wesley, 1997
- [70] *Java Constraint Programming solver (JaCoP)*. <http://jacop.osolpro.com/>, accessed July 2011
- [71] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990
- [72] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. *FORM: A feature-oriented reuse method with domain-specific reference architectures*. *Annals of Software Engineering*, 5(1):143–168, 1998
- [73] K. C. Kang, J. Lee, and P. Donohoe. *Feature-oriented product line engineering*. *Software, IEEE*, 19(4):58–65, July 2002
- [74] A. Karataş, H. Oğuztüzün, and A. Doğru. *Mapping extended feature models to constraint logic programming over finite domains*. In J. Bosch and J. Lee, editors, *Proceedings of the 14th international conference on Software product lines (SPLC'10)*, volume 6287 of *Lecture Notes in Computer Science*, pages 286–299. Springer, 2010
- [75] F. Laburthe, N. Jussien, G. Rochart, H. Cambazard, C. Prud'homme, A. Malapert, and J. Menana. *CHOCO solver*. <http://choco.emn.fr/>, accessed July 2011
- [76] K. Lauenroth and K. Pohl. *Dynamic consistency checking of domain requirements in product line engineering*. In *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE 2008, 8-12 September 2008, Barcelona, Catalunya, Spain*, pages 193–202. IEEE CS, 2008
- [77] F. J. v. d. Linden, K. Schmid, and E. Rommes. *Software product lines in action: The best industrial practice in product line engineering*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007

- [78] F. Loesch and E. Ploedereder. *Optimization of variability in software product lines*. In *Proceedings of the 11th International Software Product Line Conference (SPLC'07)*, pages 151–162. IEEE CS, 2007.
- [79] N. Loughran, P. Sánchez, A. Garcia, and L. Fuentes. *Language support for managing variability in architectural models*. In *Software Composition*, pages 36–51, 2008.
- [80] H. M. Mærsk-Møller and B. N. Jørgensen. *Cardinality-dependent variability in orthogonal variability models*. In *Sixth International Workshop on Variability Modelling of Software-Intensive Systems*, 2012
- [81] M. Mannion. *Using first-order logic for product line model validation*. In *Second Software Product Line Conference*, volume 2379 of *Lecture Notes in Computer Science*, pages 176–187. Springer, 2002
- [82] M. Mannion and J. Camara. *Theorem proving for product line model verification*. In *Software Product-Family Engineering (PFE'03)*, volume 3014 of *Lecture Notes in Computer Science*, pages 211–224. Springer, 2003.
- [83] F. Marić. *Formalization and implementation of modern SAT solvers*. *Journal of Automated Reasoning*, 43(1):81–119, June 2009.
- [84] R. Mazo, P. Grünbacher, W. Heider, R. Rabiser, C. Salinesi, and D. Diaz. *Using constraint programming to verify DOPLER variability models*. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '11*, pages 97–103. ACM, 2011.
- [85] D. McIlroy. *Mass-Produced software components*. In *Proceedings of the 1st International Conference on Software Engineering*, pages 88–98, 1968
- [86] M. Mendonça, D. Cowan, W. Malyk, and T. Oliveira. *Collaborative product configuration: Formalization and efficient algorithms for dependency analysis*. *Journal of Software*, 3(2):69–82, 2008
- [87] M. Mendonça, A. Wasowski, and K. Czarnecki. *SAT-based analysis of feature models is easy*. In *Proceedings of the International Software Product Line Conference (SPLC'09)*, 2009
- [88] M. Mendonça, A. Wasowski, K. Czarnecki, and D. Cowan. *Efficient compilation techniques for large scale feature models*. In *7th International Conference on Generative Programming and Component Engineering (GPCE)*, pages 13–22, 2008

- [89] D. Messerschmitt and C. Szyperski. *Software ecosystem: Understanding an indispensable technology and industry*. The MIT Press, edition 1, 2005
- [90] A. Metzger and K. Pohl. *Variability management in software product line engineering*. In *29th International Conference on Software Engineering (ICSE Companion)*, pages 186–187. IEEE CS, 2007
- [91] A. Metzger, K. Pohl, P. Heymans, P. Schobbens, and G. Saval. *Disambiguating the documentation of variability in software product lines: A separation of concerns, formalization and automated analysis*. In *15th Intl. Requirements Engineering Conference*, pages 243–253, October 2007
- [92] R. Mietzner, A. Metzger, F. Leymann, and K. Pohl. *Variability modeling to support customization and deployment of multi-tenant-aware software as a service applications*. In *2009 ICSE Workshop on Principles of Engineering Service Oriented Systems (PESOS 2009)*, pages 18–25. IEEE, May 2009.
- [93] S. Montagud and S. Abrahão. *Gathering current knowledge about quality evaluation in software product lines*. In *SPLC '09: Proceedings of the 13th International Software Product Line Conference*, pages 91–100, Pittsburgh, PA, USA, 2009. Carnegie Mellon University
- [94] *Moskitt Feature Modeler*. <http://www.pros.upv.es/mfm>, accessed August 2011
- [95] *OPL studio*, <http://www.ilog.com/products/oplstudio/>, accessed July 2011
- [96] A. Osman, S. Phon-Amnuaisuk, and C. Ho. *Knowledge based method to validate feature models*. In *First International Workshop on Analyses of Software Product Lines*, pages 217–225, 2008
- [97] A. Osman, S. Phon-Amnuaisuk, and C. Ho. *Using first order logic to validate feature model*. In *Third International Workshop on Variability Modelling in Software-intensive Systems (VaMoS)*, pages 169–172, 2009
- [98] D. Parnas. *On the design and development of program families*. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, march 1976
- [99] J. Peña, M. Hinchey, A. Ruiz-Cortés, and P. Trinidad. *Building the core architecture of a multiagent system product line: With an example from*

- a future NASA mission. In *7th International Workshop on Agent Oriented Software Engineering*. Lecture Notes in Computer Science, 2006
- [100] R. M. Peña. *A generic approach for automated verification of product line models*. PhD thesis, Sorbonne University, Paris, France, 2011
- [101] K. Petersen, J. M. Zaha, and A. Metzger. *Variability-driven selection of services for service compositions*. In *Service-Oriented Computing - ICSOC Workshops*, pages 388–400. Springer, September 2007
- [102] K. Pohl, G. Böckle, and F. J. van der Linden. *Software product line engineering: Foundations, principles and techniques*. Springer, Berlin Heidelberg New York, 2005
- [103] pure::variants. <http://www.pure-systems.com/>, accessed August 2011
- [104] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow. *Extending feature diagrams with uml multiplicities*. In *6th World Conference on Integrated Design & Process Technology (IDPT)*, June 2002
- [105] M. Riebisch, D. Streitferft, and I. Pashov. *Modeling variability for object-oriented product lines*. In *ECOOP 2003 Workshop Reader*, volume 3013 of *Lecture Notes in Computer Science*. Springer, July 2004
- [106] F. Roos-Frantz. *A preliminary comparison of formal properties on orthogonal variability model and feature models*. In *Third International Workshop on Variability Modelling of Software-Intensive Systems*, pages 121–126, 2009
- [107] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés. *Feature Model to Orthogonal Variability Model Transformations. A First Step*. In *VI Taller sobre Desarrollo de Software Dirigido por Modelos*, volume 3, 2009
- [108] F. Roos-Frantz and S. Segura. *Automated analysis of orthogonal variability models. a first step*. In *Workshop on Analyses of Software Product Lines*, pages 243–248, 2008
- [109] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés. *Feature model to orthogonal variability model transformation towards interoperability between tools*. In *Knowledge Industry Survival Strategy Initiative, KISS workshop in ASE'09*. <http://www.industrialized-software.org/kiss-ase-2009>, Auckland, New Zealand, Nov 2009

- [110] F. Roos-Frantz, D. Benavides, and A. Ruiz-Cortés. *Automated analysis of orthogonal variability models using constraint programming*. In *XV Jornadas de Ingeniería del Software y Bases de Datos (JISBD 2010)*, Valencia, España, Sep 2010
- [111] F. Roos-Frantz, D. Benavides, A. Ruiz-Cortés, A. Heuer, and K. Lauenroth. *Quality-aware analysis in product line engineering with the orthogonal variability model*. *Software Quality Journal*, pages 1–47, August 2011.
- [112] C. Salinesi, C. Rolland, and R. Mazo. *VMWare: Tool support for automatic verification of structural and semantic correctness in product line models*. In *Third International Workshop on Variability Modelling of Software-intensive Systems*, pages 173–176, 2009
- [113] M. Sannella. *The skyblue constraint solver and its applications*. In *Proceedings of the 1993 Workshop on Principles and Practice of Constraint Programming*, pages 385–406. MIT Press, 1993
- [114] K. Schmid and I. John. *A customizable approach to full-life cycle variability management*. *Science of Computer Programming, Special Issue on Variability Management*, 53(3):259–284, 2004
- [115] K. Schmid, R. Rabiser, and P. Grünbacher. *A comparison of decision modeling approaches in product lines*. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems, VaMoS '11*, pages 119–126. ACM, 2011.
- [116] P.-Y. Schobbens, P. Heymans, J.-C. Trigaux, and Y. Bontemps. *Generic semantics of feature diagrams*. *Computer Networks*, 51(2):456–479, 2007
- [117] S. Segura. *Automated analysis of feature models using atomic sets*. In *First Workshop on Analyses of Software Product Lines (ASPL)*, pages 201–207, Limerick, Ireland, September 2008
- [118] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cortés. *Automated metamorphic testing on the analyses of feature models*. *Information and Software Technology*, 53(3):245 – 258, 2011.
- [119] M. Sinnema and S. Deelstra. *Classifying variability modeling techniques*. *Information & Software Technology*, 49(7):717–739, 2007
- [120] M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. *COVAMOF: A framework for modeling variability in software product families*. In *Third*

- Software Product Line Conference*, volume 3154 of *Lecture Notes in Computer Science*, pages 197–213. Springer, September 2004
- [121] M. Steger, C. Tischer, B. Boss, A. Müller, O. Pertler, W. Stolz, and S. Ferber. *Introducing pla at bosch gasoline systems: Experiences and practices*. In *International Software Product Line Conference (SPLC'04)*, pages 34–50, 2004
- [122] V. Sugumaran, S. Park, and K. C. Kang. *Software product line engineering*. *Commun. ACM*, 49(12):28–32, December 2006
- [123] C. Sun, R. Rossing, M. Sinnema, P. Bulanov, and M. Aiello. [Modeling and managing the variability of web service-based systems](#). *Journal of Systems and Software*, 83(3):502–516, 2010.
- [124] J. Sun, H. Zhang, Y. Li, and H. H. Wang. *Formal semantics and verification for feature modeling*. In *10th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 303–312. IEEE CS, June 2005
- [125] T. Thüm, D. Batory, and C. Kastner. *Reasoning about edits to feature models*. In *31st International Conference on Software Engineering*, pages 254–264. IEEE CS, 2009
- [126] T. Thüm, C. Kastner, S. Erdweg, and N. Siegmund. [Abstract features in feature modeling](#). In *Software Product Line Conference (SPLC'11), 2011 15th International*, pages 191–200. IEEE, August 2011.
- [127] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro. *Automated error analysis for the agilization of feature modeling*. *Journal of Systems and Software*, 81(6):883–896, 2008
- [128] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. *Improving decision making in software product lines product plan management*. In J. Dolado, I. Ramos, and J. Cuadrado-Gallego, editors, *Proceedings of the 5th ADIS Workshop on Decision Support in Software Engineering*, volume 120. CEUR Workshop Proceedings (CEUR-WS.org), 2004
- [129] P. Trinidad, D. Benavides, and A. Ruiz-Cortés. *A first step detecting inconsistencies in feature models*. In *CAiSE Short Paper Proceedings*, 2006
- [130] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. *Fama framework*. In *12th Intl. Software Product Line Conference - Tool Demonstrations*, pages 359–359. IEEE CS, September 2008

- [131] E. Tsang. *Foundations of constraint satisfaction*. Academic Press, London and San Diego, 1993
- [132] M. Tseng and J. Jiao. *Handbook of industrial engineering: Technology and operations management*. Wiley, 2001
- [133] T. T. Tun, Q. Boucher, A. Classen, A. Hubaux, and P. Heymans. *Relating requirements and feature configurations: a systematic approach*. In *SPLC, ACM International Conference Proceeding Series*, pages 201–210. ACM, 2009
- [134] P. van den Broek and I. Galvao. *Analysis of feature models using generalised feature trees*. In *Third International Workshop on Variability Modelling of Software-intensive Systems*, number 29. In ICB-Research Report, pages 29–35, Essen, Germany, January 2009. Universität Duisburg-Essen
- [135] T. van der Storm. *Variability and component composition*. In *8th International Conference on Software Reuse: Methods, Techniques and Tools (ICSR)*, volume 3107 of *Lecture Notes in Computer Sciences*, pages 157–166. Springer, July 2004
- [136] T. van der Storm. *Generic feature-based software composition*. In *Software Composition*, volume 4829 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2007
- [137] A. van Deursen and P. Klint. *Domain-specific language design requires feature descriptions*. *Journal of Computing and Information Technology*, 10(1):1–17, 2002
- [138] T. von der Massen and H. Lichter. *Requiline: A requirements engineering tool for software product lines*. In F. van der Linden, editor, *Proceedings of the Fifth International Workshop on Product Family Engineering (PFE'03)*, Lecture Notes in Computer Science, Siena, Italy, 2003. Springer
- [139] T. von der Massen and H. Lichter. *Determining the variation degree of feature models*. In *Proceedings of the 9th International Conference on Software Product Lines (SPLC'05)*, volume 3714 of *Lecture Notes in Computer Science*, pages 82–88, 2005
- [140] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan. *A semantic web approach to feature modeling and verification*. In *Workshop on Semantic Web Enabled Software Engineering (SWESE'05)*, November 2005

- [141] H. Wang, Y. Li, J. Sun, H. Zhang, and J. Pan. *Verifying feature models using OWL*. *Journal of Web Semantics*, 5(2):117–129, 2007
- [142] J. Whaley. *JavaBDD solver*. <http://javabdd.sourceforge.net/>, accessed July 2011
- [143] J. White, B. Dougherty, and D. C. Schmidt. *Selecting highly optimal architectural feature sets with filtered cartesian flattening*. *Journal of Systems and Software*, 82(8):1268–1284, August 2009
- [144] J. White, B. Dougherty, D. C. Schmidt, and D. Benavides. *Automated reasoning for multi-step feature model configuration problems*. In *13th Intl. Software Product Line Conference*, pages 11–20. IEEE CS, Aug 2009
- [145] J. White and D. Schmidt. *Filtered cartesian flattening: An approximation technique for optimally selecting features while adhering to resource constraints*. In *First International Workshop on Analyses of Software Product Lines (ASPL)*, pages 209–216, 2008
- [146] J. White, D. Schmidt, D. B. P. Trinidad, and Ruiz-Cortés. *Automated diagnosis of product-line configuration errors in feature models*. In *Proceedings of the 12th Software Product Line Conference (SPLC'08)*, Limerick, Ireland, September 2008
- [147] H. Yan, W. Zhang, H. Zhao, and H. Mei. *An optimization strategy to feature models' verification by eliminating verification-irrelevant features and constraints*. In *ICSR*, pages 65–75, 2009
- [148] L. A. Zaid, F. Kleinermann, and O. D. Troyer. *Applying semantic web technology to feature modeling*. In *ACM symposium on Applied Computing (SAC'09)*, pages 1252–1256, New York, NY, USA, 2009. ACM
- [149] G. Zhang, H. Ye, and Y. Lin. *Quality attributes assessment for feature-based product configuration in software product line*. In *17th Asia Pacific Software Engineering Conference (APSEC)*, pages 137–146, 2010
- [150] G. Zhang, H. Ye, and Y. Lin. *Modelling quality attributes in feature models in software product line engineering*. In *6th International Conference on Software and Data Technologies (ICSOF 2011)*, pages 249–254, 2011
- [151] G. Zhang, H. Ye, and Y. Lin. *Using knowledge-based systems to manage quality attributes in software product lines*. In *15th International Software Product Line Conference, Volume 2, SPLC '11*, New York, NY, USA, 2011. ACM.

- [152] H. Zhang, S. Jarzabek, and B. Yang. *Quality prediction and assessment for product lines*. In *15th Int. Conf. Advanced Information Systems Engineering*, volume 2681 of *Lecture Notes in Computer Science*. Springer, June 2003
- [153] W. Zhang, H. Mei, and H. Zhao. *Feature-driven requirement dependency analysis and high-level software design*. *Requirements Engineering*, 11(3):205–220, June 2006.
- [154] W. Zhang, H. Zhao, and H. Mei. *A propositional logic-based method for verification of feature models*. In *6th International Conference on Formal Engineering Methods*, volume 3308 of *Lecture Notes in Computer Science*, pages 115–130. Springer, November 2004
- [155] W. Zhang, H. Yan, H. Zhao, and Z. Jin. *A BDD-based approach to verifying clone-enabled feature models' constraints and customization*. In *10th International Conference on Software Reuse (ICSR)*, *Lecture Notes in Computer Science*, pages 186–199. Springer, 2008.
- [156] T. Ziadi, L. H elou et, and J.-M. J ez equel. *Towards a UML profile for software product lines*. In *Software Product-Family Engineering*, pages 129–139, 2003.

This document was typeset using class RC-BOOK α 2.12 for $\text{\LaTeX}2_{\epsilon}$. As of the time of writing this document, this class is not publicly available since it is in alpha version. Only members of The Distributed Group are using it to typeset their documents. Should you be interested in giving forthcoming public versions a try, please, do contact us at contact@tdg-seville.info

Software product line engineering is a software development paradigm that aims to build a family of software products by reusing a common set of core assets. In this paradigm, variability models are central artefacts, since they document the variability amongst products in a product line. Over the past twenty years, a number of variability modelling approaches have been proposed in order to document and manage variability, such as feature modelling, decision modelling, and orthogonal variability modelling. Amongst them, feature modelling is the most popular. In this approach, feature models are used to provide a compact representation of all the products of a product line in terms of features.

The automated analysis of variability models is defined as the computer-aided extraction of information from variability models. This is an active research topic that has received the attention of many researchers during the last twenty years. Most of this research has been focused on feature models, resulting in a set of analysis operations, techniques, and tools to automate the analysis of this kind of models. The existence of other variability models is naturally leading to the need for new techniques and tools to support their automated analysis as well. Furthermore, there is a need for extending variability with attributes, so that the analysis can take into account not only variability in terms of functional features, but also in terms of attributes.

Variability models usually contain elements that are used only to structure the variability of the product line, and therefore do not have any impact on the generated models, such as requirements, design, or implementation models. We refer to these elements as abstract elements. Most of the variability modelling languages do not provide an explicit way to express abstract elements. Furthermore, the majority of the current approaches for the automated analysis of variability models can only reason about the combinations of all the elements in the variability model, but not about those that may be relevant for the user, i.e., those that have some impact on other models of the product line. Therefore, abstract elements should be made explicit in the variability models, so that the analysis that only considers relevant elements can be performed.

The Orthogonal Variability Model is a modelling language to define the variability of a software product line. It is a known standard of the product line community that interrelates the variability in base models such as requirement models, design models, component models, and test models. In this dissertation, we provide a set of techniques and tools to support the automated analysis of Orthogonal Variability Models. An important strength of our contribution lies in the fact that we provide support for dealing with attributes and abstract elements. First, we make abstract elements explicit in the orthogonal variability models, and then we provide two techniques to automate the analysis of such models, one omitting abstract elements and other considering all the elements of the model. Second, we provide a technique to enrich orthogonal variability models with attributes and to automate their analysis.

Our contributions have been integrated into a tool that is built as part of the FaMa ecosystem, which is a framework for the analysis of variability models developed by our research group. In order to demonstrate the effectiveness of our techniques and analysis tool we present an evaluation using a product line in the automotive domain, which was created in a German project by a leading car company. Such evaluation allowed the detection of false optional and dead elements in the orthogonal variability model that represents the variability of such product line, and the verification of attribute conditions as well.