

Verification of Feature Anomalies in Software Product Line Feature Models

Birat Saha

2014-2-60-086

Md. Fahim Shahrier Rasel

2014-2-60-088

Syed Nazmus Shakib

2014-2-60-125

A thesis submitted in partial fulfillment of the requirements for the
degree of Bachelor of Science in Computer Science and Engineering



Department of Computer Science and Engineering
East West University
Dhaka-1212, Bangladesh

April, 2018

Declaration

We, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Shamim H. Ripon, Associate Professor, Department of Computer Science and engineering, East West University. We also declare that no part of this thesis has been or is being submitted elsewhere for the award of any degree or diploma.

.....
Dr. Shamim H. Ripon
Supervisor

.....
Birat Saha
2014-2-60-086

.....
Md. Fahim Shahrier Rasel
2014-2-60-088

.....
Syed Nazmus Shakib
2014-2-60-125

Letter of Acceptance

This project entitled “**Verification of Feature Anomalies in Software Product Line Feature Models**” submitted by Birat Saha (2014-2-60-086), Md. Fahim Shahrier Rasel (2014-2-60-088) and Syed Nazmus Shakib (2014-2-60-125) to the Computer Science and Engineering Department, East West University, Dhaka-1212, Bangladesh is accepted as satisfactory for partial fulfillment of requirements for the degree of Bachelors of Science(B. Sc.) in Computer Science and Engineering.

Board of Examiners

.....

Dr. Shamim H. Ripon

Associate Professor

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

.....

Dr. Ahmed Wasif Reza

Associate Professor and Chairperson

Department of Computer Science and Engineering

East West University, Dhaka, Bangladesh

Abstract

Products with new features need to be introduced on the market in a prompt step and organizations need to speed up their development process. Reuse has been suggested as a solution, but to achieve effective reuse within an organization a planned and preemptive effort must be used. Software Product lines are the most promising technique and it increases productivity and software quality and decreases time-to-market. In SPL, a feature model shows various types of features and seizes the relationships among them. As different configuration found in feature model it has high probability that some configuration is not correct. There could be anomalies in feature model which could lead invalid configuration every time. Number of rules proposed to identify those anomalies. Some tools can identifies those anomalies. Verifying those rule with different tools ensure that those rules are correct and universal on every tools.

Acknowledgments

First of all, We would like to express our deepest gratitude to The Almighty for His blessings on us. Next, our special thanks go to thank our thesis advisor Dr. Shamim H. Ripon of Department of Computer Science and Engineering at East West University. The door to his office was always open whenever we ran into a trouble spot or had a question about our thesis. He consistently allowed this thesis to be our own work, but steered us in the right the direction whenever he thought we needed it.

We would also like to acknowledge all of the faculty members of Department of Computer Science and Engineering for their unconditional help and support whenever we needed it.

Finally, we must express our very profound gratitude to our parents for providing unfailing support and continuous encouragement throughout our years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you

Birat Saha

April, 2018

Md. Fahim Shahrier Rasel

April, 2018

Syed Nazmus Shakib

April, 2018

Table of Contents

Declaration of Authorship	i
Letter of Acceptance	ii
Abstract	iii
Acknowledgment	iv
Table of Contents	v
List of Figures	vii
Chapter List of Tables	x
Chapter 1 Introduction	1
1.1 Introduction	1
1.2 Problems and Motivation	2
1.3 Objectives	3
1.4 Contribution	4
1.5 Outline	4
Chapter 2 Background	6
2.1 Software Product Line (SPL)	6
2.2 Feature Models	7
2.2.1 Basic Feature Models	9

2.3	Logical Representation	11
2.3.1	Logical operators	11
2.4	FaMa	14
2.4.1	General Description	15
2.5	FeatureIDE	16
Chapter 3 Domain Model		18
3.1	Overview of CAD Domain	18
3.2	CAD Domain Model using FODA	20
Chapter 4 Analysis Rules		23
4.1	Introduction	23
4.2	False Optional	25
4.3	Dead Feature	26
Chapter 5 Tool Implementation		28
5.1	Represent Feature Models Using FaMa	28
5.2	FaMa GUI	29
5.2.1	Checking Model Validity	31
5.2.2	Products by Feature Model	31
5.2.3	Number of Product by Feature Model	33
5.2.4	Variability of Feature Model	33
5.2.5	Errors on Feature Model	34
5.2.6	Error Explanation of Feature Model	35
5.2.7	Export FaMa Model to Graphical Model of FeatureIDE	37
Chapter 6 Case Study		39
6.1	Analysis Rules Verification on CAD Domain	41
6.1.1	False Optional 1	41
6.1.2	False Optional 2	42
6.1.3	Dead Feature 1	43

Chapter 7 Conclusion	45
7.1 Future Work	45
Bibliography	45

List of Figures

2.1	A Simple Feature Model	8
3.1	Basic Operational Scenario in a CAD System for Police	19
3.2	CAD Domain Feature Model	21
4.1	The types of Analysis Rules in Feature Model	24
4.2	False Optional 1	25
4.3	False Optional 2	26
4.4	Dead Feature 1	27
5.1	Simple FaMa GUI	30
5.2	Model Validation	31
5.3	Products by Model	32
5.4	Number of Product by Model	33
5.5	Variability of the Model	34
5.6	Errors on Model	35
5.7	Error Explanation on Model	36
5.8	Export FaMa Model	37
5.9	Exported Model in FeatureIDE	38
6.1	CAD Domain Validation	40
6.2	CAD Domain Errors	41
6.3	Analysis of False Optional 1 in CAD	42
6.4	Analysis of False Optional 2 in CAD	43

6.5 Analysis of Dead Feature 1 in CAD 44

List of Tables

2.1	Logical notations for feature model	10
2.2	Negation Operator	11
2.3	Conjunction Operator	12
2.4	Disjunction Operator	12
2.5	Exclusive Or Operator	13
2.6	Implication Operator	14
2.7	Biconditional Operator	14
3.1	Key entities of CAD domain	19

Chapter 1

Introduction

1.1 Introduction

Designing, developing and maintaining a good software system is a challenge still in this 21st century. The approach of reusing existing good solutions for developing any new application is now one of the central focuses of software engineers. Building software systems from previously developed components saves cost and time of redundant work and improves the system and its maintainability. A new software development paradigm, software product line [1], is emerging to produce multiple systems by reusing the common assets across the systems in the product line. However, the idea of product line is not new. In 1976 Parnas [2] proposed modularization criteria and information hiding for handling product line.

The increase competitiveness in the software development sector with immense economic considerations such as cost, time to market, etc. motivates the transition from single product development to product-line development approach. Software product line is a set of software intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or missions and that are developed from a common set of core assets in a prescribed way [1].

A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [1]. Core assets are the basis for software product line. The core assets often include the architecture, reusable software components, domain models, requirements statements,

documentation and specifications, performance model, etc. Different product line members may differ in functional and non-functional requirements, design decisions, run-time architecture and interoperability (component structure, component invocation, synchronization, and data communication), platform, etc. The product line approach integrates two basic processes: the abstraction of the commonalities and variabilities of the products considered (development for reuse) and the derivation of product variants from these abstractions (development with reuse) [3].

The main idea of software product line is to explicitly identify all the requirements that are common to all members of the family as well as those that varies among products in the family. This implies a huge model that help the stakeholders to be able to trace any design choices and variability decision. A particular product is then derived by selecting the required variants and configuring them according to the product requirements.

1.2 Problems and Motivation

Both industry and the scholarly community have indicated much enthusiasm for taking care of product line in application domains such as business systems, avionics, command and control systems etc. Today most of the effort in product line development are relating to architecture, detail design and code. Basic necessities among all relatives are anything but difficult to deal with as they just can be incorporated into the family design and are a piece of each relative.

But problem arises from the variant requirements among family members. In a product line, currently variants are modeled using feature diagram, inheritance, templates and other techniques. In contrast with analysis of a solitary framework, demonstrating variations includes an additional level of intricacy to the domain analysis. In any product line model, the same variant has occurrences in different domain model views. Different variants have dependencies on each other. Tracing multiple occurrences in different model views of any variant and understanding the mutual dependencies among variants are major challenges during domain modeling. While each step in modeling variant may

be simple but problem arises when the volume of information grows. When the volume of information grows the domain models become difficult to understand.

The main problems are the possible explosion of variant combinations, complex dependencies among variants and difficulty in tracing variants from the domain model down to the specification of a particular product. As feature model now used in Software Product Line for making different configuration of same product but the anomalies in the feature model can cause invalid configuration. In invalid configuration is used for product development then the software will not be much helpful for customer as well as the developer because it will cost much time and money.

Our motivation is to detect anomalies in feature model based on some rules which is theoretically proven. To use this rule for detect anomalies in feature model we have to verify those rule using existing tools which is used to create feature model and detect anomalies.

1.3 Objectives

In developing product offering, the variations are to be overseen in space designing stage, which scopes the product offering and builds up the way to quickly deliver the individuals from the family. It fills two particular yet related needs, firstly, it can record choices about the product in general including distinguishing the variations for every part and furthermore, it can bolster application designing by giving legitimate data and instrument to the required variations amid product age.

The goal of this work is to verify the existing rule which are used in detect anomalies in feature model theoretically. Our work focuses on two specific analysis operations: false optional and dead features of feature model. We also tried to get some new rules which can detect and explain the anomalies in the feature model

1.4 Contribution

The particular emphasis of this thesis is to verify the feature anomalies in feature model.

To achieve this goal in this thesis we have made the following contributions,

- Our work is to verify the existing rule we have to convert all the rule in such format which is compatible with verification method and tool
- We also proposed some new rules for False Optional and Dead Feature and written its logic in First Order.
- We also verified our proposed rule using the same method we used to verified the existing rule. We verified our rule using existing domain model.
- We developed a simple interface for a feature model tool which can be used as verification of rules and as well to get details about the verification failure
- We extended the feature model tool by adding functionality to export its format to use in other feature model verification tool.

1.5 Outline

The thesis is organized as follows,

Chapter 2 gives a brief overview of the Software Product Line(SPL), Feature Model, Logical representation of feature model and Feature model tool like FaMa and FeatureIDE.

Chapter 3 gives an overview of the CAD domain. We briefly describe the activities of the domain and give an example of a Police CAD domain.

Chapter 4 introduces new rules for detection feature model anomalies with Scenarios, Diagram and Logical representation in First Order.

Chapter 5 tells about the tools used for verification how its implemented how we convert existing rule to compatible format and different use of the feature model tools.

Chapter 6 discusses about the case study of new rule in our domain model and its verification.

Chapter 7 concludes the thesis by summarizing our work. Finally we outline our future plan.

2.1 Software Product Line (SPL)

Software Product Line (SPL) is a set of related softwares, also known as software family, where the member products of the family share some common features and each member is characterized by their varying features [4]. The main objectives behind SPL are reusability, time to market, increased quality [4, 5]. The common and varying features of a SPL are arranged in model that helps the stakeholder to select their required product feature configuration. Common requirements among all family members are easy to handle as they simply can be integrated into the family architecture and are part of every family member. But problem arises from the variant requirements among family members as modeling variants adds an extra level of complexity to the domain analysis. Thus management of variants is considered to be one of the critical areas in SPL.

Actually, Software product line (SPL) is a software intensive system sharing a common and managed set of features that satisfy the needs of a particular market segment or mission and that are developed from a set of core assets in a prescribed way [6]. Product line technology is a way of improving the software development lifecycle and reuse by providing facilities to reuse the model of the system family. By reusing rather than recreating the work products of the system families, it is possible to increase the productivity and decrease the possible errors significantly.[4] The main idea of software product line is to explicitly identify all the activities which are common to all members of the family as well as which are different and then arrange them in a model. This implies a huge model which will help the stakeholders to be able to trace any design choices and variability

decisions as well. Finally, the derivation of the product is done by selecting the required variants from the model and configuring them according to product requirements.

Manufacturers have long employed analogous engineering techniques to create a product line of similar products using a common factory that assembles and configures parts designed to be reused across the product line. For example, automotive manufacturers can create unique variations of one car model using a single pool of carefully designed parts and a factory specifically designed to configure and assemble those parts. The characteristic that distinguishes software product lines from previous efforts is predictive versus opportunistic software reuse. Rather than put general software components into a library in the hope that opportunities for reuse will arise, software product lines only call for software artifacts to be created when reuse is predicted in one or more products in a well defined product line. Recent advances in the software product line field have demonstrated that narrow and strategic application of these concepts can yield order of magnitude improvements in software engineering capability. The result is often a discontinuous jump in competitive business advantage, similar to that seen when manufacturers adopt mass production and mass customization paradigms.

While early software product line methods at the genesis of the field provided the best software engineering improvement metrics seen in four decades, the latest generation of software product line methods and tools are exhibiting even greater improvements. New generation methods are extending benefits beyond product creation into maintenance and evolution, lowering the overall complexity of product line development, increasing the scalability of product line portfolios, and enabling organizations to make the transition to software product line practice with orders of magnitude less time, cost and effort. Domain and application engineering are the two main phases of SPL development [5].

2.2 Feature Models

A feature model represents the information of all possible products of a software product line in terms of features and relationships among them. Feature models are a special

type of information model widely used in software product line engineering. A feature model is represented as a hierarchically arranged set of features composed by:

1. relationships between a parent (or compound) feature and its child features (or subfeatures).
2. crosstree (or crosshierarchy) constraints that are typically inclusion or exclusion statements in the form: if feature F is included, then features A and B must also be included (or excluded).

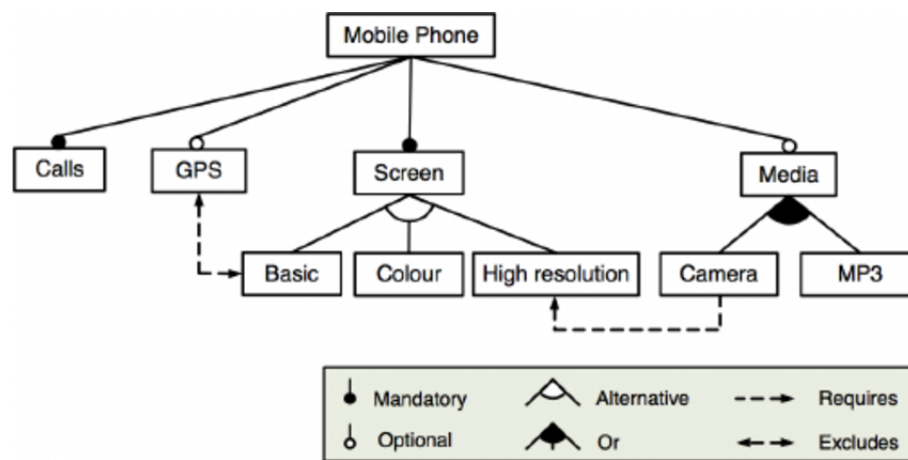


Figure 2.1: A Simple Feature Model

Figure 2.1 depicts a simplified feature model inspired by the mobile phone industry. The model illustrates how features are used to specify and build software for mobile phones. The software loaded in the phone is determined by the features that it supports. According to the model, all phones must include support for calls, and displaying information in either a basic, colour or high resolution screen. Furthermore, the software for mobile phones may optionally include support for GPS and multimedia devices such as camera, MP3 player or both of them.

Feature models are used in different scenarios of software production ranging from model driven development [6], feature oriented programming [7], software factories [8] or generative programming, all of them around software product line development. Although

feature models are studied in software product line engineering, these information models can be used in different contexts ranging from requirements gathering to data model structures, hence the potential importance of feature models in the information systems domain. The term feature model was coined by Kang et al. In the FODA report back in 1990[6] and has been one of the main topics of research in software product lines since then. There are different feature model languages. We refer the reader to for a detailed survey on the different feature model languages. Below, we review the most well known notations for those languages

2.2.1 Basic Feature Models

We group as basic feature models those allowing the following relationships among features:

- **Mandatory.** A child feature has a mandatory relationships with its parent when the child is included in all products in which its parent feature appears. For instance, every mobile phone system in our example must provide support for calls.
- **Optional.** A child feature has an optional relationship with its parent when the child can be optionally included in all products in which its parent feature appears. In the example, software for mobile phones may optionally include support for GPS.
- **Alternative.** A set of child features have an alternative relationship with their parent when only one feature of the children can be selected when its parent feature is part of the product. In the example, mobile phones may include support for a basic, colour or high resolution screen but only one of them.
- **Or.** A set of child features have an or-relationship with their parent when one or more of them can be included in the products in which its parent feature appears. In Figure 1, whenever Media is selected, Camera, MP3 or both can be selected.

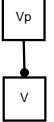
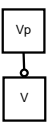
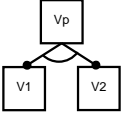
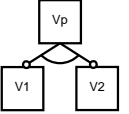
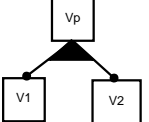
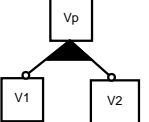
Type	Logic Expression	Type	Logic Expression
 Mandatory	$v_p \Leftrightarrow v$	 Optional	$v \Rightarrow v_p$
 Alternative	$v_p \Leftrightarrow (v_1 \oplus v_2)$	 Optional Alternative	$(v_1 \oplus v_2) \Rightarrow v_p$
 Or	$v_p \Leftrightarrow (v_1 \vee v_2)$	 Optional Or	$(v_1 \vee v_2) \Rightarrow v_p$

Table 2.1: Logical notations for feature model

Notice that a child feature can only appear in a product if its parent feature does. The root feature is a part of all the products within the software product line. In addition to the parental relationships between features, a feature model can also contain cross-tree constraints between features. These are typically in the form:

- **Requires.** If a feature A requires a feature B, the inclusion of A in a product implies the inclusion of B in such product. Mobile phones including a camera must include support for a high resolution screen.
- **Excludes.** If a feature A excludes a feature B, both features cannot be part of the same product. GPS and basic screen are incompatible features.

More complex cross-tree relationships have been proposed later in the literature [5] allowing constraints in the form of generic propositional formulas, e.g. A and B implies not C.

2.3 Logical Representation

Logic has been studied since the classical Greek period (600-300BC). The Greeks, most notably Thales, were the first to formally analyze the reasoning process. Aristotle (384-322BC), “the father of logic”, and many other Greeks searched for universal truths that were irrefutable. A second great period for logic came with the use of symbols to simplify complicated logical arguments. Gottfried Leibniz (1646-1716) began this work at age 14, but failed to provide a workable foundation for symbolic logic. George Boole (1815-1864) is considered the “father of symbolic logic”. He developed logic as an abstract mathematical system consisting of defined terms (propositions), operations (conjunction, disjunction, and negation), and rules for using the operations. Boole’s basic idea was that if simple propositions could be represented by precise symbols, the relation between the propositions could be read as precisely as an algebraic equation. Boole developed an “algebra of logic” in which certain types of reasoning were reduced to manipulations of symbols.

2.3.1 Logical operators

1. **Negation Operator:** “not”, has symbol “ \neg ” :

Example: p : This book is interesting. Then $\neg p$ can be read as “This book is not interesting”.

Truth Table:

P	$\neg P$
T	F
F	T

Table 2.2: Negation Operator

The negation operator is a unary operator which, when applied to a proposition p , changes the truth value of p . That is, the negation of a proposition p , denoted by $\neg p$, is the proposition that is false when p is true and true when p is false.

2. **Conjunction Operator:** “and”, has symbol “ \wedge ”. Example: p: This book is interesting.

q: I am staying at home.

$p \wedge q$: This book is interesting and I am staying at home.

Truth Table:

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

Table 2.3: Conjunction Operator

The conjunction operator is the binary operator which, when applied to two propositions p and q , yields the proposition “ p and q ”, denoted $p \wedge q$. The conjunction $p \wedge q$ of p and q is the proposition that is true when both p and q are true and false otherwise.

3. **Disjunction Operator:** inclusive “or”, has symbol “ \vee ”. Example:

p: This book is interesting

q: I am staying at home.

$p \vee q$: This book is interesting, or I am staying at home.

Truth Table:

P	Q	$P \vee Q$
T	T	T
T	F	T
F	T	T
F	F	F

Table 2.4: Disjunction Operator

The disjunction operator is the binary operator which, when applied to two propositions p and q , yields the proposition “ p or q ”, denoted $p \vee q$. The disjunction $p \vee q$ of p and q is the proposition that is true when either p is true, q is true, or both are true, and is false otherwise

4. **Exclusive Or Operator:** “xor”, has symbol \oplus .

Example:

p : This book is interesting

q : I am staying at home.

$p \oplus q$: Either this book is interesting or I am staying at home, but not both.

Truth table:

P	Q	$P \oplus Q$
T	T	F
T	F	T
F	T	T
F	F	F

Table 2.5: Exclusive Or Operator

The exclusive or is the binary operator which, when applied to two propositions p and q yields the proposition “ p xor q ”, denoted $p \otimes q$, which is true if exactly one of p or q is true, but not both. It is false if both are true or if both are false.

5. **Implication Operator:** “if...then...”, has symbol “ \Rightarrow ” . Example:

p : This book is interesting.

q : I am staying at home.

$p \Rightarrow q$: If this book is interesting, then I am staying at home.

Truth Table:

The implication $p \Rightarrow q$ is the proposition that is often read as “if p then q ”. If “ p then q ” is false precisely when p is true but q is false

P	Q	$P \Rightarrow Q$
T	T	T
T	F	F
F	T	T
F	F	T

Table 2.6: Implication Operator

6. **Biconditional Operator:** “if and only if”, has symbol “ \Leftrightarrow ” Example:

p: This book is interesting.

q: I am staying at home.

$p \Leftrightarrow q$: This book is interesting if and only if I am staying at home.

Truth table:

P	Q	$P \Leftrightarrow Q$
T	T	T
T	F	F
F	T	F
F	F	T

Table 2.7: Biconditional Operator

The bi-conditional statement is equivalent to $(p \Rightarrow q) \wedge (q \Rightarrow p)$. In other words: For $p \Leftrightarrow q$ to be true we must have both p and q true.

2.4 FaMa

FaMa (FeAture Model Analyser) is a tool to analyse feature models. FaMa is a tool to analyse Software Product Lines, represented as feature models. Software product line is a recent software development paradigm, based on the principle of “customization in mass”. But how we have presented, FaMa has uses further than analysing software product lines strictly. FaMa can analyse any system that can be expressed as a set of

features on a hie

FAMA is an extensible framework for the automated analysis of feature models. FAMA allows the integration of different logic representations and solvers in order to optimize the analysis process. It can be configured to select automatically in execution time the most efficient of the available solvers according to the operation requested by the user. The current implementation of FAMA integrates three of the most promising logic representations proposed in the area of the automated analysis of feature models: CSP, SAT and BDD, but more solvers can be added if needed. The implementation is based on an Eclipse plug in and uses XML to represent FMs so it can inter operate with other tools that support it.

FAMA has been implemented as a complete tool for the edition and analysis of FMs. FAMA supports cardinality- based feature modelling (that includes traditional feature models, e.g FODA, FeatureRSEB, and so on), export/import of FMs from XML and XMI and analysis operations on FMs. In order to make our tool multiplatform and easy to access we implemented FAMA as a plugin for the Eclipse Platform1. In the next sections we overview of the functionality offered by the framework and we describe some of the most relevant design and implementation details.

2.4.1 General Description

FAMA offers two main functionalities: visual model edition/creation and automated model analysis. Once the user has created or imported (from XML/XMI) a cardinality based FM, the analysis capability can be used. Most of the operation identified on FMs [6] are being currently implemented. At the moment of writing this article the operations fully supported by FAMA are:

- Finding out if an FM is valid, i.e. it exists a a product satisfying all the constraints.
- Finding the total number of possible products of an FM (number of products).
- List all the possible products of a feature model (list of products).

- Calculate the commonality of a feature, i.e. the number of products where a feature appears in.

FAMA integrates different solvers in order to combine the best of all of them in terms of performance. The actual version of the framework integrates CSP 2, SAT 3 and BDD 4 Java solvers to perform the analysis tasks. However, an unlimited number of new analysis operations and solvers could be added. One of the advantages of FAMA is the ability to select automatically, in execution time, the most efficient solver according to the operation requested by the user. The mapping from a FM onto the correspondent solver is done on demand. Therefore, if the user asks for the number of possible combinations of features of an FM the framework will select automatically the BDD solver to get it (the most efficient known approach for this operation). The automated selection of a solver is based on the value of some configuration parameters establishing the priority between the available solvers for each operation. The values of these parameters were set according to the results from a complete performance test of the solvers integrated in the framework [8].

2.5 FeatureIDE

FeatureIDE, an open source framework of an IDE for software product line engineering based on Feature-Oriented Software Development (FOSD) [13, 5]. FeatureIDE supports the entire life-cycle of a product line in a coherent tool infrastructure, starting with domain analysis and feature modeling [6], but also covering design, implementation and maintenance with FOSD.

FeatureIDE does not only cover a single language (e.g., Jak from the AHEAD tool suite [5]), but several languages based on the same foundation: the concept of FOSD. At the point of writing, FeatureIDE supports a multitude of different tools including AHEAD [5], FeatureC++ [3], FeatureHouse [2], and CIDE [9]; this way FeatureIDE supports FOSD in many languages, including Java, C++, Haskell, C, C#, JavaCC, and XML. As we will show, also other parts of FeatureIDE are opened up for extensions, which makes

it possible to extend FeatureIDE further, either toward specific needs in an industrial setting or to showcase research results in a full IDE and make them quickly available to users in academia and industry (as in [16]).

We envision FeatureIDE as open source project that provides a broad foundation, but that can be used and extended by different parties to teach and productively use FOSD. In this demonstration, we give an overview of FeatureIDEs design and present recent developments on the background of current software product line projects.

Chapter 3

Domain Model

Our Domain Model is based on the Computer Aided Dispatch System (CAD) domain. We consider this domain as our case study because we have got supporting documents of this domain from the company who is working on this domain and also working in collaboration with our research team. Several research works have already been done on different aspects of this domain which help to gain better knowledge of this domain. An overview of the Computer Aided Dispatch System (CAD) and its basic domain model is presented in this chapter.

3.1 Overview of CAD Domain

A Computer Aided Dispatch system (CAD) is a mission-critical system that is used by police, fire and rescue, health service, port operation, taxi booking and others. However, the basic operational scenarios are similar in all the CAD systems. Figure 3.1 depicts a basic operational scenario and roles in a CAD system.

When an incident is happened in a place a Caller reports the incident to the command and control center of the police unit. A Call Taker in the command and control center captures the details about the incident and the Caller, and creates a task for the incident. There is a Dispatcher in the system whose task is to dispatch resources to handle any incident. The system shows the Dispatcher a list of un-dispatched tasks. The Dispatcher examines the situation, selects suitable Resources (e.g. police units) and dispatches them to execute the task. The Resources carry out the task instructions and report to the Task Manager. The Task Manager monitors the situation and at the end when the resources finished the task- closes the task. Different CAD members have different resources and

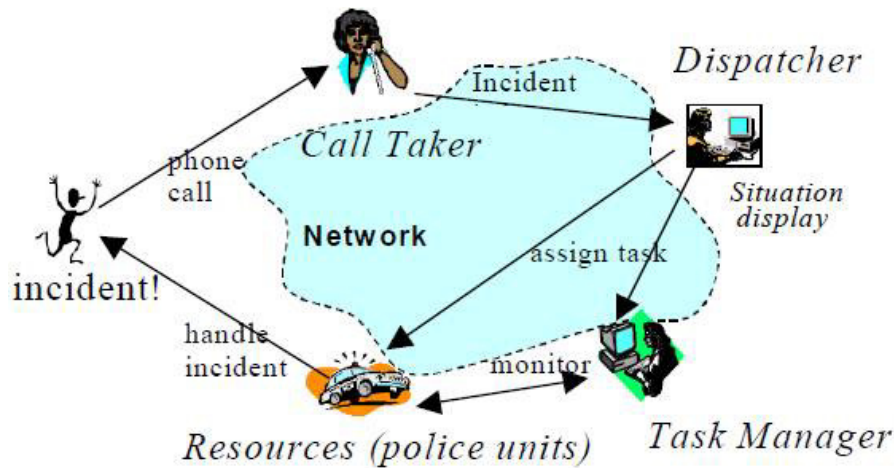


Figure 3.1: Basic Operational Scenario in a CAD System for Police

tasks for their system. The key entities of CAD domain will interact with each other according to the system requirements. For example, some resources are free of charge (e.g. police) whereas some are not (e.g. taxi). The basic key entities of CAD domain are listed in Table 3.1.

Key	Definitions
Task	A task holds together the information regarding a particular incident (incident location, type, priority, urgency, status, caller's detail etc). Two separate entities are included within it - incident's detail and caller detail.
Resource	A resource handles the task. In police CAD, the resource would be police car, in taxi CAD, the resource would be taxi. It is presumed that the resources are equipped with necessary hardware to communicate with CAD system.
Command	Command instructs the resources to complete a task. The person who is responsible for dispatching task to resources sends the command.

Table 3.1: Key entities of CAD domain

At the basic operational level, all CAD systems are similar; basically they support

the dispatcher units to handle the incidents. However, there are differences across the CAD systems. The specific context of operation results in many variations on the basic operational theme. Here are some of the variants identified in CAD domain: then call taker informs the dispatcher of the newly created task but if merged then without informing to dispatcher he/she can dispatch resources directly to the incidents.

Validation of caller and task information differs across CAD systems. In some CAD systems basic validation (i.e., checking the completeness of caller information and the task information) is sufficient while in other CAD systems validation includes duplicate task checking, etc. in yet other CAD systems no validation is required at all.

Un-dispatched task selection rule in certain situation at any given time there might be more than one task to be dispatched, then there is a need to decide which task will be dispatched next. A number of algorithms are available for this purpose and different CAD system use different algorithm. In Ambulance CAD system task may be selected based on task urgency or priority whereas in taxi system different algorithm will be applied.

This simple description of CAD variants hints us about numerous variants and variant dependencies, which focus the importance of managing them properly.

3.2 CAD Domain Model using FODA

Modeling variants is an important process during designing software product line. The feature oriented domain analysis (FODA) method was developed at the Software Engineering Institute (SEI) [7]. FODA focuses on identifying features that characterize a domain. Features are user visible aspects or characteristics of a system and are organized into And/Or graph in order to identify the commonalities and variants of the application domain. Feature modeling is an integral part of the FODA method and the Feature Oriented Domain Reuse Method (FORM) [9]. The commonalities and variants within features are exploited to create a set of models that is used to implement any member product of that family.

Features are represented in graphical form as trees. The internal nodes of a tree represent the variants and their leaves represent values of corresponding variants. Graphical symbols are used to indicate the categories of features. The root node of a feature tree always represents the domain whose features are modeled. The remaining nodes represent features which are classified into three types:

- Mandatory features are always part of the system.
- Optional features may be selected as a part of the system if their parent feature is in the system. The decision whether an optional feature is part of the system or not can be made independently from the selection of other features.
- Alternative features of a variant are related to each other as exclusive-or relationship, i.e. exactly one feature out of a set of features is to be selected.

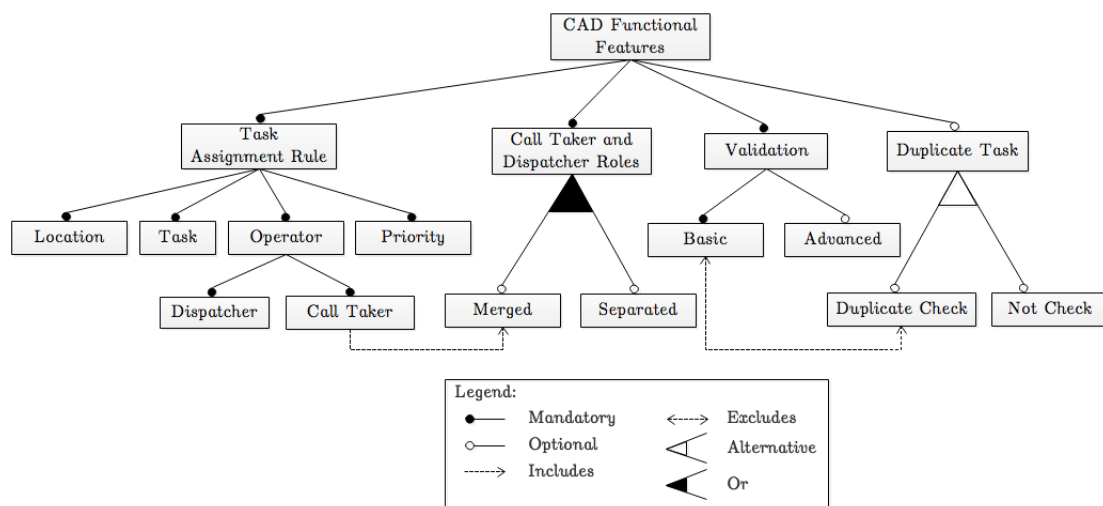


Figure 3.2: CAD Domain Feature Model

The feature diagram depicts the classification of mandatory features and variant features as well as their dependencies. Mandatory features are those which are present in all products in the respective domain. Variant features appear only some members of the domain which differentiate one product from others. There are more relationships

between features. One is Or-feature by [10], which connects a set of optional features with a parent feature, either common or variant. The meaning is that whenever the parent feature is selected then at least one of the optional features will be selected. Feature diagram also depicts the interdependencies among the variants which describes the selection of one variant depends on the selection of the dependency connected variants. A partial CAD feature diagram is given in the Figure 5.1. In this feature diagram the root represents the functional features of CAD. Task Assignment Rule, Call Taker & Dispatcher Roles and Validation are linked to the root via mandatory link as these are mandatory features of CAD. However, Checking Duplicate Task is linked via optional link as this feature is optional. We use extensions described in [10].

Chapter 4

Analysis Rules

4.1 Introduction

With the increasing nature of number of features in a feature model the potential complexity incorporated with the analysis of the model is becoming a task of extensive hardship. The additional level of complexity is making the analysis task error-prone [11] and, hence, number of defects may get introduced at the modeling stage. Therefore, care need to be taken in constructing a feature model so that the model represents the domain precisely. However, correct modeling of the domain is a critical task [12] and identification of the defects at the proper stage of the software development life cycle is a prerequisite of developing valid software that fulfills all the requirements of stakeholders. Defects found in a SPL feature model has been termed as anomalies. In order to detect and correct inconsistencies as well as redundancies addressing these anomalies by exploiting intelligent techniques and tools has been suggested by many experts [12, 13]. In our work we carry out on two specific types of anomalies namely False Optional and Dead Feature.

False Optional: A feature becomes a false optional (FO) feature when it has been defined as an optional feature but it is selected for all the valid products.

Dead Feature: A feature becomes a dead feature (DF) when it has been defined as an optional feature but it is never selected for any of the valid products.

For defining the rules we use following predicates.

- ***variation_point(v)***: This predicate indicates that feature v has one or more child feature(s).

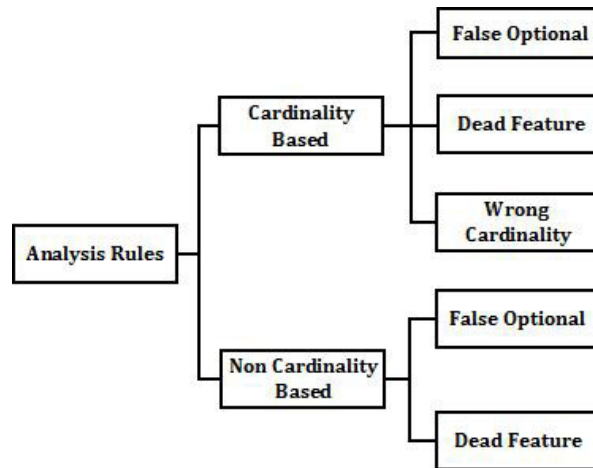


Figure 4.1: The types of Analysis Rules in Feature Model

- *mandatory_variant*(v, x): This predicate indicates that feature x is a mandatory child of feature v .
- *optional_variant*(v, x): This predicate indicates that feature x is an optional child of feature v .
- *requires*(x, y): This predicates indicates that x requires y to be selected for a valid product.
- *excludes*(x, y): This predicates indicates that x excludes y from being selected for a valid product.
- *cardinality*(G, k, m, n): This predicate indicates G is a group cardinality with total number of features k , lower bound m and upper bound n .
- *parent_feature*(G, f): This predicate indicates f is the parent feature in the group cardinality G .
- *child_features*(G, a, d): This predicate indicates that $a, .., d$ are the child features of the parent under the group cardinality G .

- *dead_features(a,, d)*: This predicate indicates that feature a,, d are dead features.
- *false_optional(a,, d)*: This predicate indicates that feature a,, d are false optional features.
- *select(x)*: This predicate indicates the selection of feature x.

4.2 False Optional

Rule 1. An optional feature become false optional when a mandatory feature require it's child feature that is also optional.

In the Figure 4.2 *x* is a mandatory feature which requires an optional feature *z* where *z* is the descendant of *y* which is also optional. As mandatory feature *x* requires *z* this results both *y* and *z* false optional feature.

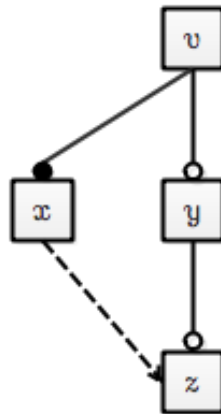


Figure 4.2: False Optional 1

$$\begin{aligned} & \forall v, x, y, z \cdot \text{variation_point}(v) \wedge \text{mandatory_variant}(v, x) \wedge \text{optional_variant}(v, y) \\ & \wedge \text{variation_point}(y) \wedge \text{mandatory_variant}(y, z) \wedge \text{requires}(x, z) \\ & \wedge \text{select}(x) \Rightarrow \text{select}(z) \wedge \text{select}(y) \end{aligned}$$

Rule 2. An optional feature become false optional when a mandatory feature require it's child which is mandatory feature.

In the Figure 4.3 x is a mandatory feature which requires an optional feature z where z is the descendant of y which is mandatory feature. As mandatory feature x requires z this results y as a false optional feature.

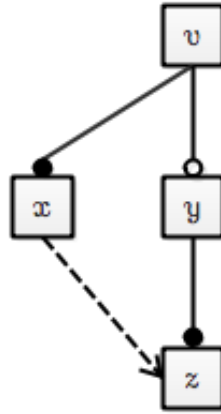


Figure 4.3: False Optional 2

$$\begin{aligned} & \forall v, x, y, z \cdot \text{variation_point}(v) \wedge \text{mandatory_variant}(v, x) \wedge \text{optional_variant}(v, y) \\ & \wedge \text{variation_point}(y) \wedge \text{optional_variant}(y, z) \wedge \text{requires}(x, z) \\ & \wedge \text{select}(x) \Rightarrow \text{select}(z) \wedge \text{select}(y) \end{aligned}$$

4.3 Dead Feature

Rule 1. An optional feature and its child become dead when one of its child is mandatory and excluded by a mandatory feature

In the Figure 4.4 x is a mandatory feature which requires an mandatory feature z where z is the descendant of y which is an optional feature. As mandatory feature x requires z this makes y dead and this results y and all of its child as a dead feature.

$$\begin{aligned} & \forall v, x, y, z \cdot \text{variation_point}(v) \wedge \text{mandatory_variant}(v, x) \wedge \text{optional_variant}(v, y) \\ & \wedge \text{variation_point}(y) \wedge \text{mandatory_variant}(y, z) \wedge \text{excludes}(x, z) \\ & \wedge \text{select}(x) \Rightarrow \neg \text{select}(y) \wedge \neg \text{select}(z) \end{aligned}$$

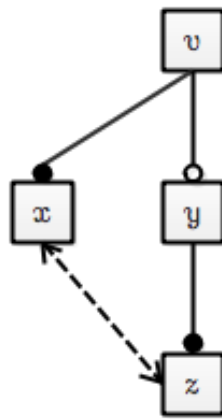


Figure 4.4: Dead Feature 1

Chapter 5

Tool Implementation

5.1 Represent Feature Models Using FaMa

FaMa accepts two input file formats for models: xml and plain text. For our work we selected plain text format. Feature Model Format (FMF) is a plain text format where we can define standard or extended feature models.

a feature model is very similar to a tree. It has a root, and except this node (feature since now), every feature has a parent. Each feature may have three types of relationships with their children:

- Mandatory relationship (1 to 1): if parent is present in a product, child must be too
- Optional relationship (1 to 1): if parent is present in a product, child may be or not.
- Group relationship (1 to n)
 - Set relationship: if parent is present in a product, one (and only one) child must be present.
 - Or relationship: if parent is present in a product, one or more children must be present.
 - Cardinality-based relationship: if a parent is present in a product, user can define how many children must be present.

Moreover, there is a special type of relationship between non parent-child nodes. These

relationships, called Cross Tree Constraints, are from one feature to another feature (1 to 1). There are two types:

- Excludes: if A excludes B, B and A cant be at the same time in a product.
- Requires: if A requires B, if feature A is present in a product, feature B must be too

For basic models, FaMa have two sections. In %Relationships, we define tree structure and relationships types. First defined feature is the root of the feature model. Any feature after colon is a child of the feature before the colon. Name of the feature can contain letters and “_” character. Mandatory relationship is the default one. Optional can be specified with the feature between brackets [], and group relationships with cardinality and a set of features ([n,m]FeatA FeatB...).

We can define cross-tree-constraints too in %Constraints section (REQUIRES and EXCLUDES among features) Basic models have **.fm** or **.fmf** file extension. We can use these extensions to avoid errors when using FaMa. Here an example of basic feature model in this notation.

```
%Relationships
HIS: SUPERVISION_SYSTEM CONTROL [SERVICES];
SUPERVISION_SYSTEM: FIRE INTRUSION [FLOOD];
CONTROL: LIGHT_CONTROL [APPLIANCES_CONTROL] TEMPERATURE;
SERVICES: [1,2]{VIDEO INTERNET};
INTERNET: [1,1]{POWER_LINE ADSL WIRELESS};

%Constraints
LIGHT_CONTROL EXCLUDES POWER_LINE;
```

5.2 FaMa GUI

FaMa standalone API provides and implements several analysis operations on feature models. FaMa can analyze any system that can be expressed as a set of feature on a

hierarchy, with or without attributes. Some of FaMa's operations are:

- Validation
- Products
- Number of Product
- Variability
- Error Detection
- Error Explanations

FaMa provides a shell front-end to final users, and three ways to integrate application with FaMa: a java standalone version, a SOAP/WSDL web service, and a set of OSGi bundles. We used java standalone version in our FaMa GUI.

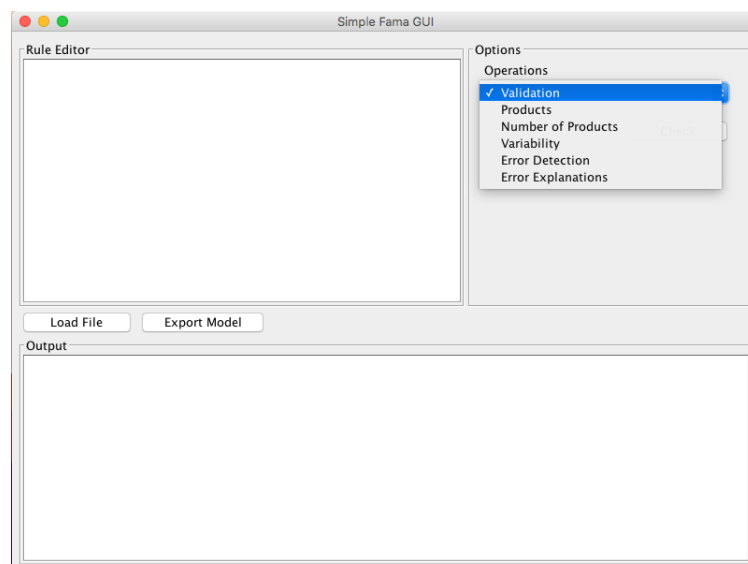


Figure 5.1: Simple FaMa GUI

In Figure 5.1 we can see the Simple FaMa GUI interface. In the interface we can see the GUI has Rule Editor Output box and the operations Combo Box where we can see all the operation offered by this tool. There also has two button for selecting the Rule File(*.fm) and for exporting the model to FeatureIDE model.

5.2.1 Checking Model Validity

Checks the feature model validity. It checks if the feature model is not empty, or other words, it has at least one product.

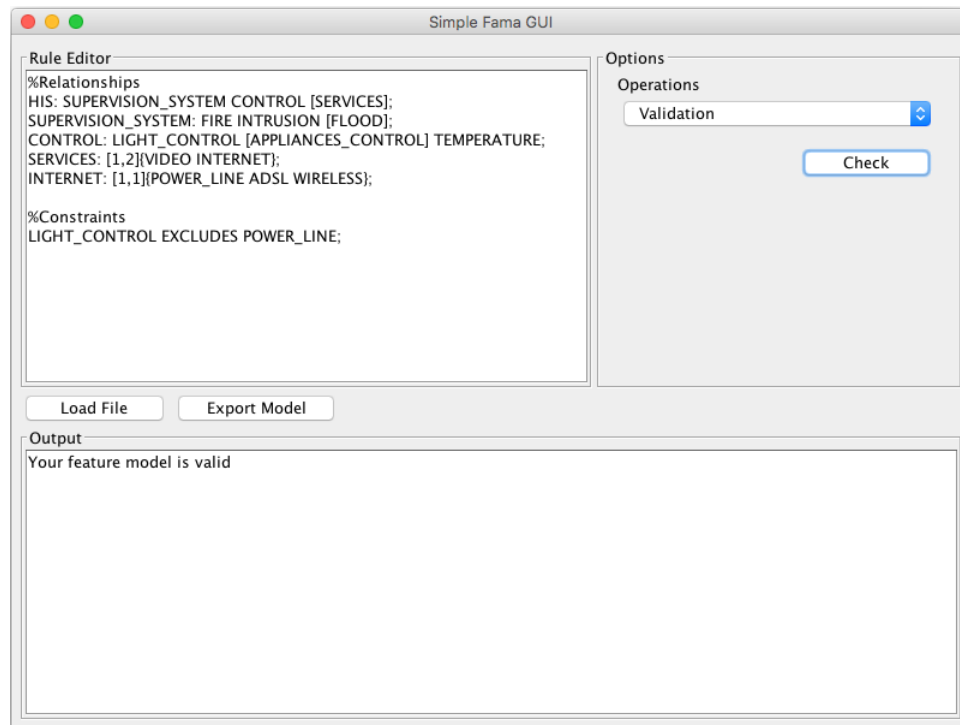


Figure 5.2: Model Validation

If we input the Rule for HIS(Home Integration System) from the section 5.1 in the tool and check for validity. The tool will correctly identified that this is valid.

5.2.2 Products by Feature Model

Generates all valid products of the feature model.

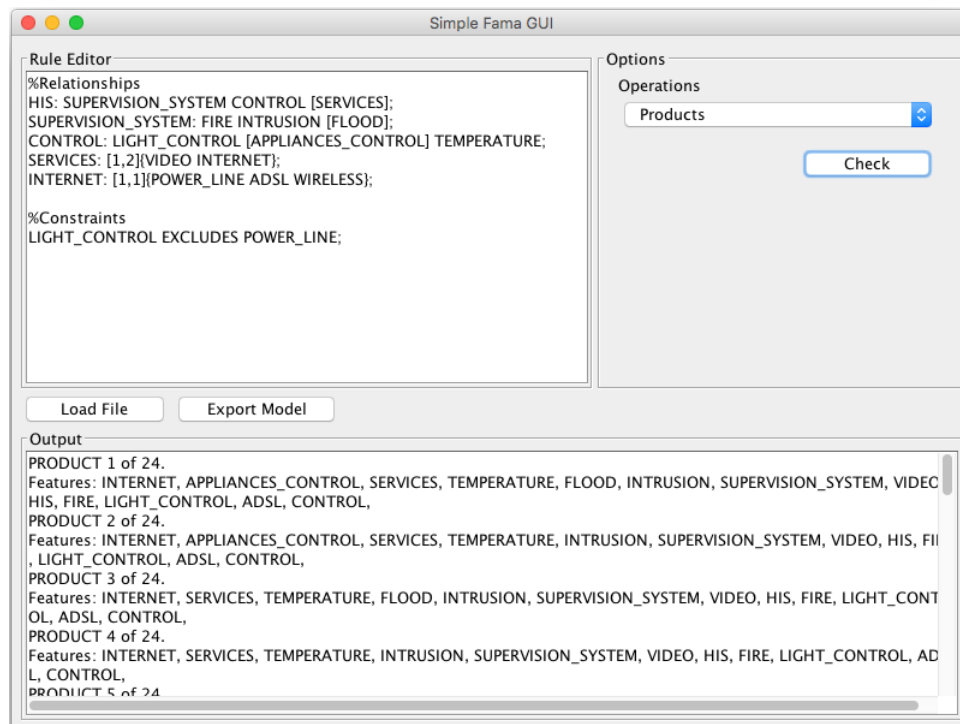


Figure 5.3: Products by Model

If we input the Rule of HIS from section 5.1 in the rule editor we can check all the valid products configuration of HIS. For HIS domain we can see there are 24 products can be configured. All the configuration is valid and all 24 product is an individual product. Using this tool we can also see which feature we need to select for valid product configuration. By generating products from model we can also see if there has some defects in existing configured product.

5.2.3 Number of Product by Feature Model

Calculate total number of valid product of the feature model.

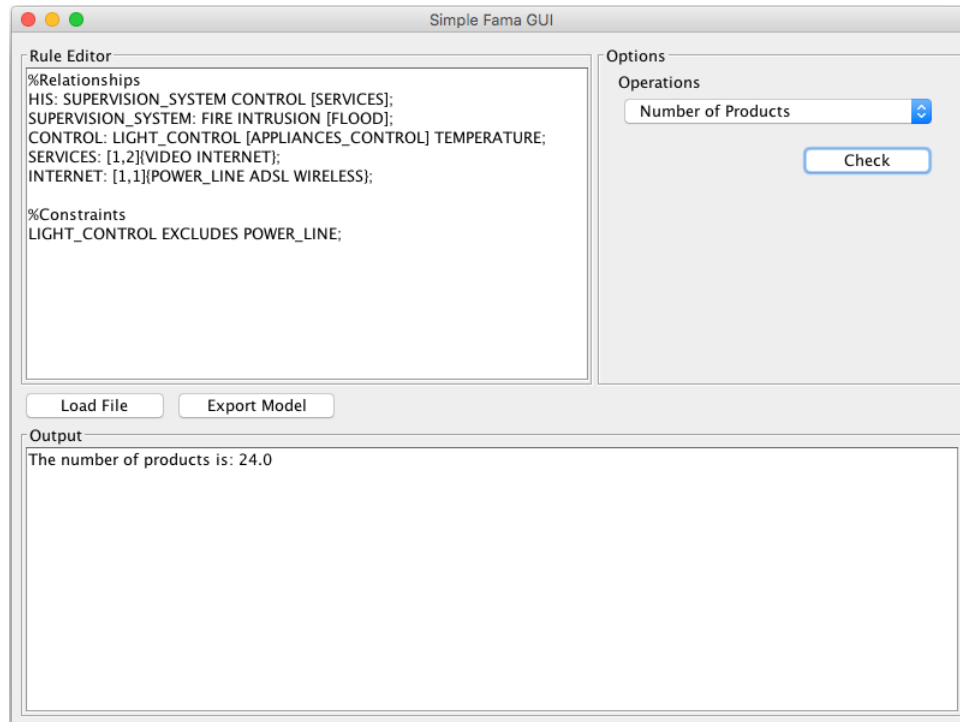


Figure 5.4: Number of Product by Model

One of the questions to be answered is how many potential products a FM contains. This is a key question in SPL engineering because if the number of products increases the SPL becomes more flexible as well as more complex. From Figure 5.4 we can see how many products we can configured by HIS model from 5.1. Sometimes we don't need to see all the valid product configuration but we need to see how many valid products we can generate. For this case number of product is very important. Total number of product counts all valid configuration of the products.

5.2.4 Variability of Feature Model

Calculates variability degree of the feature model

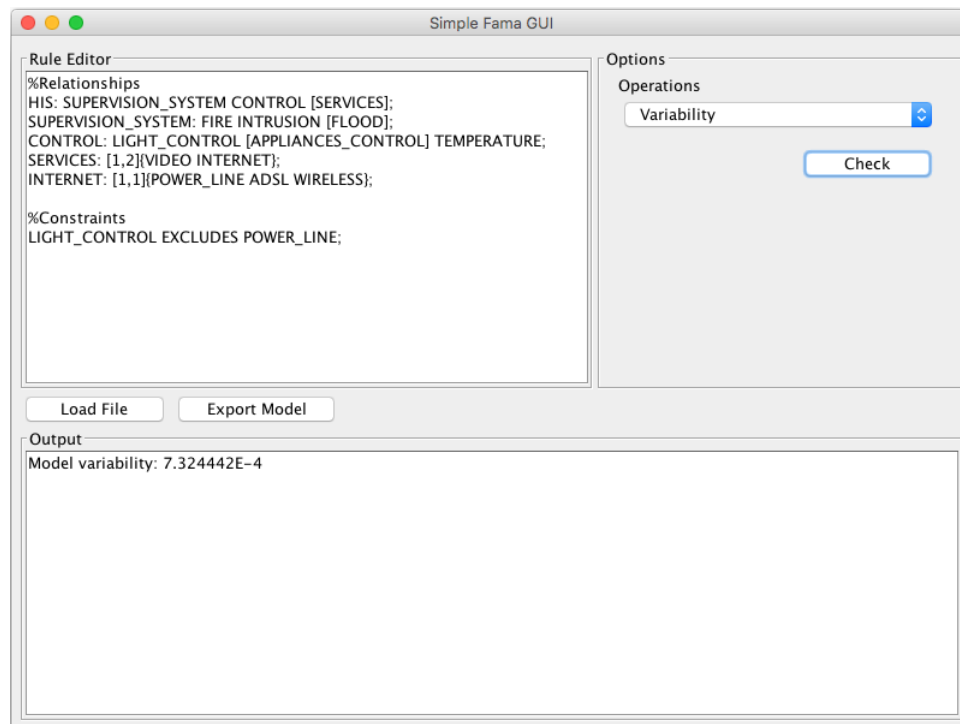


Figure 5.5: Variability of the Model

As mentioned previously, feature models are composed of a set of features and relations among them. If relations restrict the number of products to only one, we are considering the lowest variability while a feature model defining no possible product would be considered a non-valid model. On the other hand, considering no relations, the number of products within the feature model would be the highest. This case would represent the highest variability. Relations restrict the number of potential products, so variability depends on relation types.

5.2.5 Errors on Feature Model

Looks for errors on feature model

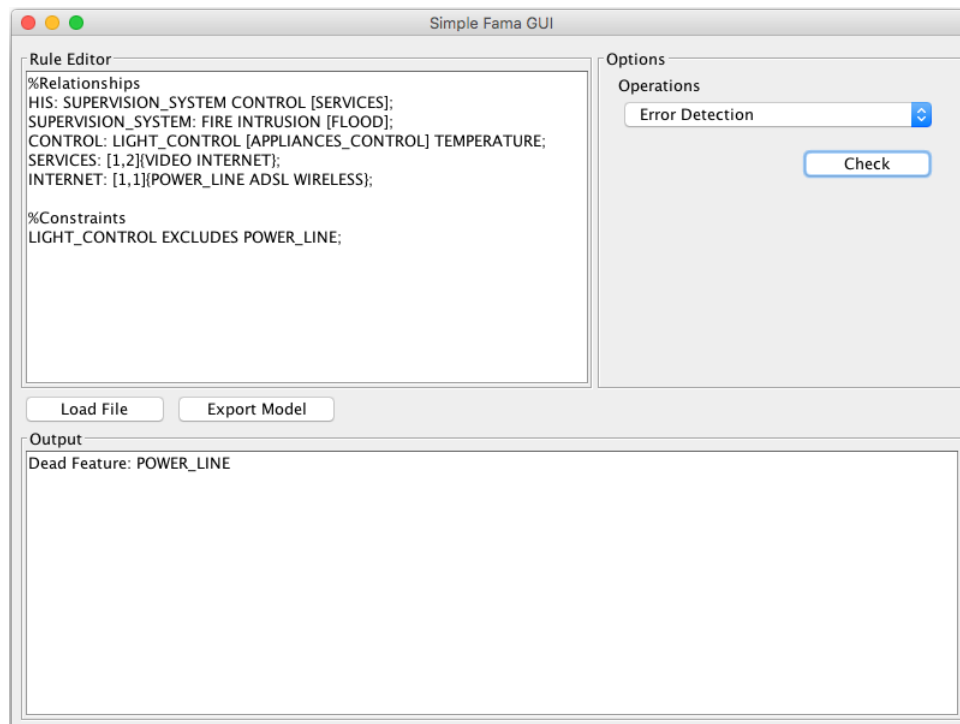


Figure 5.6: Errors on Model

From Figure 5.6 we can see that if we give the model in our tool we can find the error/defects of the model. From previous we know that there are two kind of defects in the feature model one is False Optional and other one is Dead Feature. A feature becomes false optional when that feature will always picked for all valid configurations. And a feature becomes Dead feature when a optional feature always unselected for all valid configuration.

5.2.6 Error Explanation of Feature Model

If a feature model has errors, this operation will look for proper explanation for the errors.

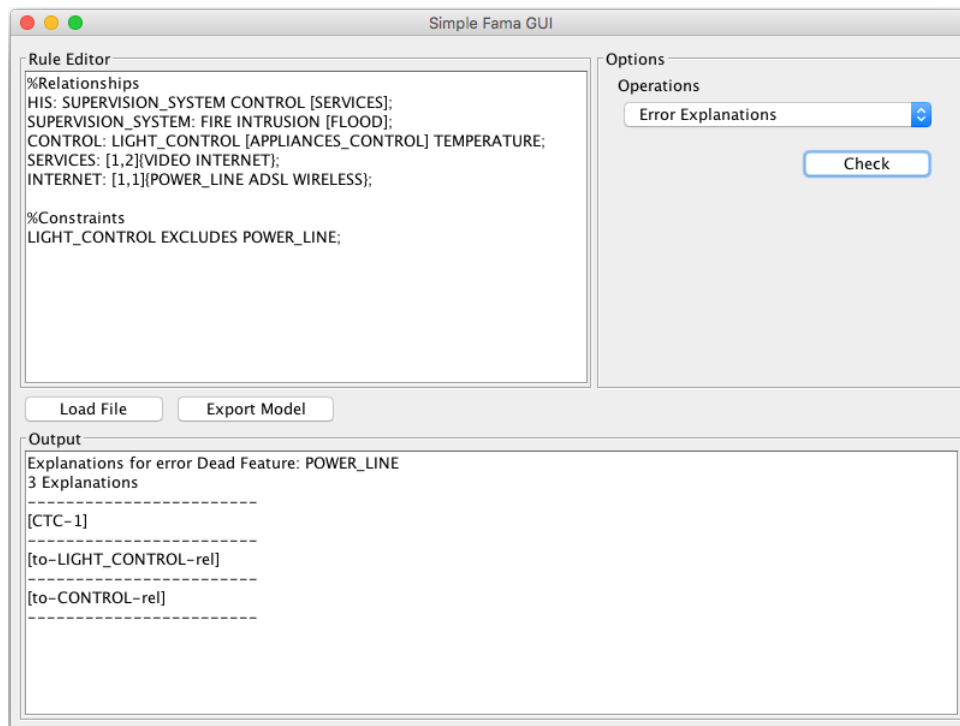


Figure 5.7: Error Explanation on Model

From Figure 5.7 we can see the error as well as the reason. If we input the HIS model rule from section 5.1 we can see the defects as well as the the reason behind the error. It's very important to find the defects because if we find the defects on the early stage of software requirement then we can minimize failure cost.

5.2.7 Export FaMa Model to Graphical Model of FeatureIDE

FaMa feature model can be exported as graphical feature model of FeatureIDE and then visual configure the model.

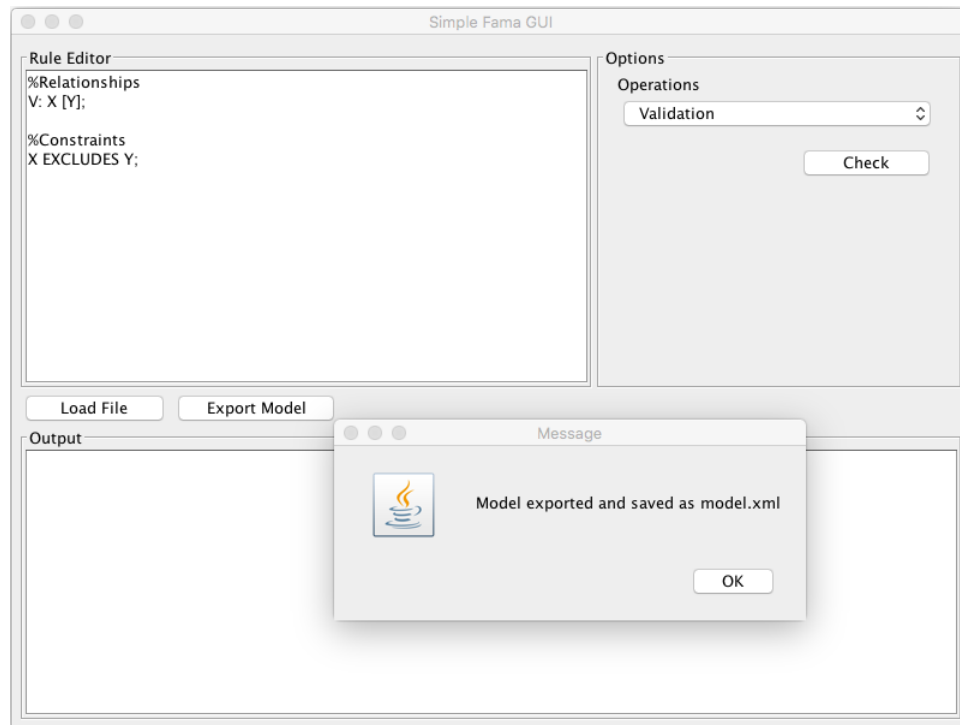


Figure 5.8: Export FaMa Model

The exported model will be shown in FeatureIDE like Figure 5.9. The export model we used is native to FeatureIDE. The exported model is XML file when we copy this model.xml file in FeatureIDE. In the visual model we can see that V is the parent and X and Y is the child where X is the mandatory feature and Y is the optional feature and X excludes Y. So Y becomes dead feature here. By exporting the model we can visually see all the defects and we can also select all the valid configuration.

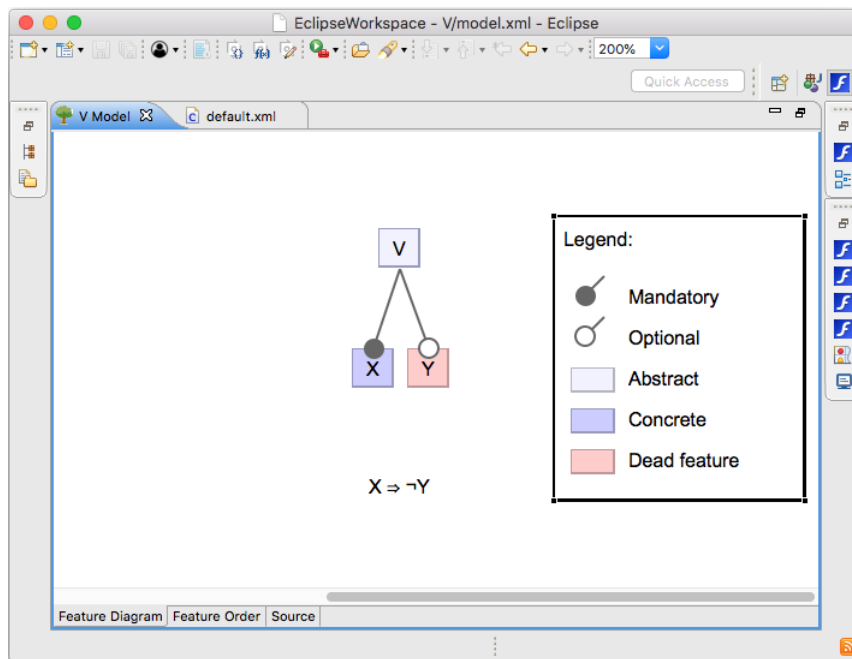


Figure 5.9: Exported Model in FeatureIDE

Chapter 6

Case Study

In this thesis we tried to verify the existing rules of the feature model using tools like FaMa and FeatureIDE. Our CAD Domain Figure 3.2 in Fama format

%Relationships

CAD: TASK_ASSIGNMENT CALL_TAKER_DISPATCHER VALIDATION [DUPLICATE_TASK];

TASK_ASSIGNMENT: LOCATION TASK OPERATOR PRIORITY;

OPERATOR: DISPATCHER CALL_TAKER;

CALL_TAKER_DISPATCHER: [1,2] {MERGED SEPARATED};

VALIDATION: BASIC [ADVANCED];

DUPLICATE_TASK: [1,1] {DUPLICATE_CHECK NOT_CHECK};

%Constraints

CALL_TAKER REQUIRES MERGED;

BASIC EXCLUDES DUPLICATE_CHECK;

If we check this using our tool we will find the model is valid. Error does not always make the model invalid. Error in model will make the configuration for specific combination invalid. Total number of configuration will be reduce because error in model makes configuration invalid.

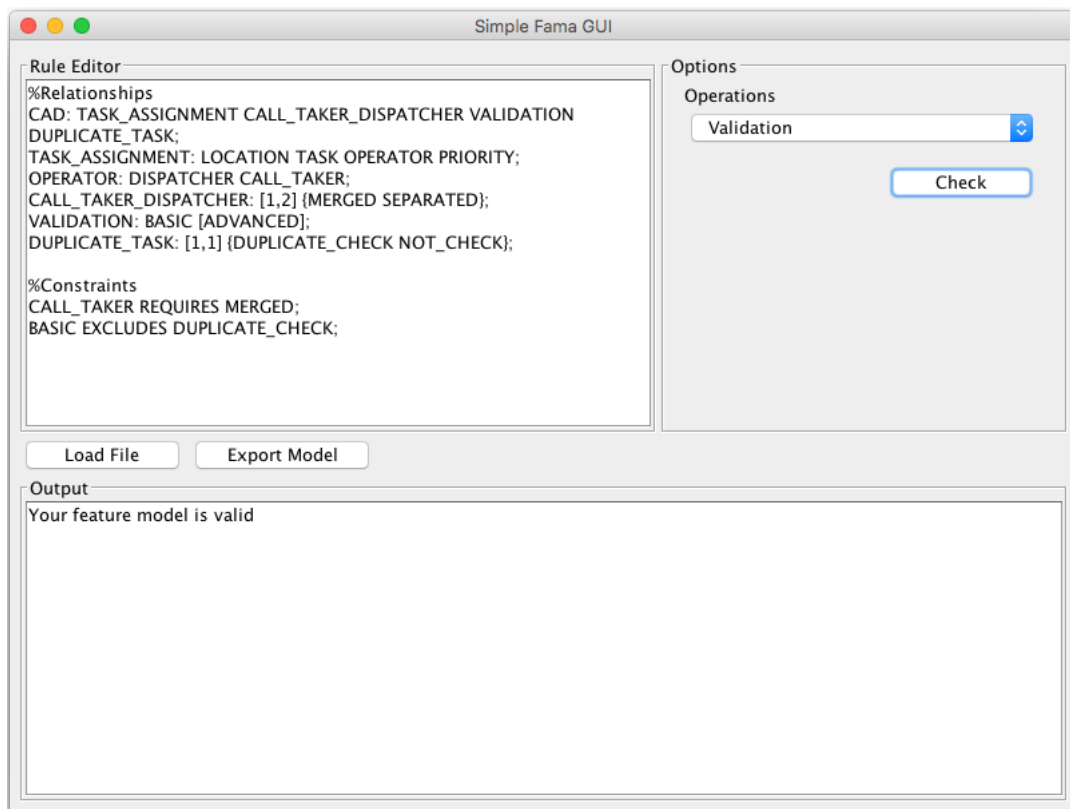


Figure 6.1: CAD Domain Validation

but the model has some error like False Optional and Dead Feature. As from Figure 6.2 we found that CAD Domain has three error two of them is False Optional and other one is Dead Feature. MERGED and NOT_CHECK are the False Optional feature and DUPLICATE_CHECK is the Dead feature.

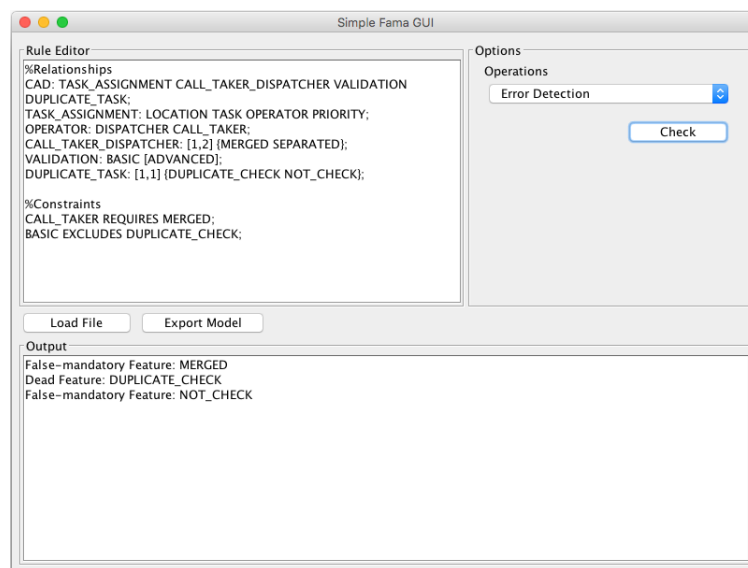


Figure 6.2: CAD Domain Errors

6.1 Analysis Rules Verification on CAD Domain

6.1.1 False Optional 1

For verifying the False Optional feature of analysis rule shown in 4.2 we have to modify the CAD Domain for make this scenario. So the CAD Domain modified Fama Format will be:

```
%Relationships
CAD: TASK_ASSIGNMENT CALL_TAKER_DISPATCHER VALIDATION [DUPLICATE_TASK];
TASK_ASSIGNMENT: LOCATION TASK OPERATOR PRIORITY;
OPERATOR: DISPATCHER CALL_TAKER;
CALL_TAKER_DISPATCHER: [1,2] {MERGED SEPARATED};
VALIDATION: BASIC [ADVANCED];
DUPLICATE_TASK: [DUPLICATE_CHECK] [NOT_CHECK];

%Constraints
BASIC_REQUIRES_DUPLICATE_CHECK;
```

If we check for the error in our tools we will find that `DUPLICATE_TASK` and `DUPLICATE_CHECK` become False Optional by analysis rule False Optional 1.

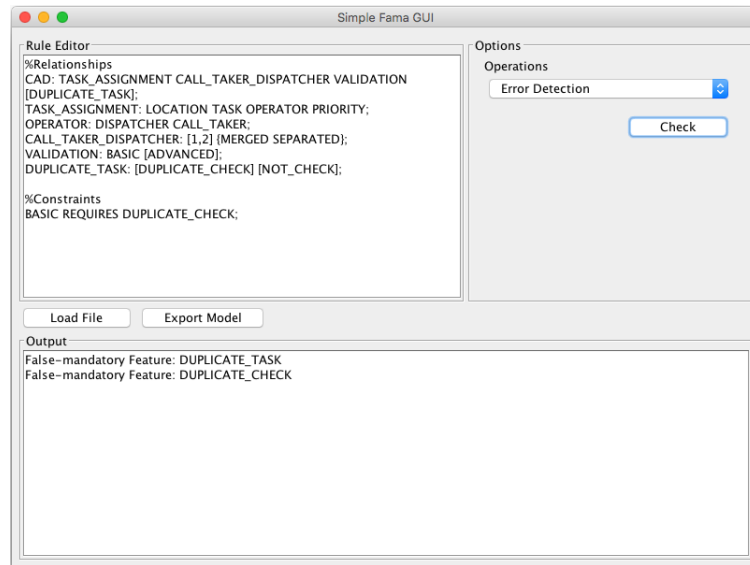


Figure 6.3: Analysis of False Optional 1 in CAD

In Figure 6.3 we verified that analysis rule False Option 1 is correct.

6.1.2 False Optional 2

For verifying the False Optional feature of analysis rule shown in 4.3 we have to modify the CAD Domain for make this scenario. So the CAD Domain modified Fama Format will be:

```
%Relationships
CAD: TASK_ASSIGNMENT CALL_TAKER_DISPATCHER VALIDATION [DUPLICATE_TASK];
TASK_ASSIGNMENT: LOCATION TASK OPERATOR PRIORITY;
OPERATOR: DISPATCHER CALL_TAKER;
CALL_TAKER_DISPATCHER: [1,2] {MERGED SEPARATED};
VALIDATION: BASIC [ADVANCED];
DUPLICATE_TASK: DUPLICATE_CHECK [NOT_CHECK];

%Constraints
```

BASIC REQUIRES DUPLICATE_CHECK;

If we check for the error in our tools we will find that DUPLICATE_TASK become False Optional by analysis rule False Optional 2.

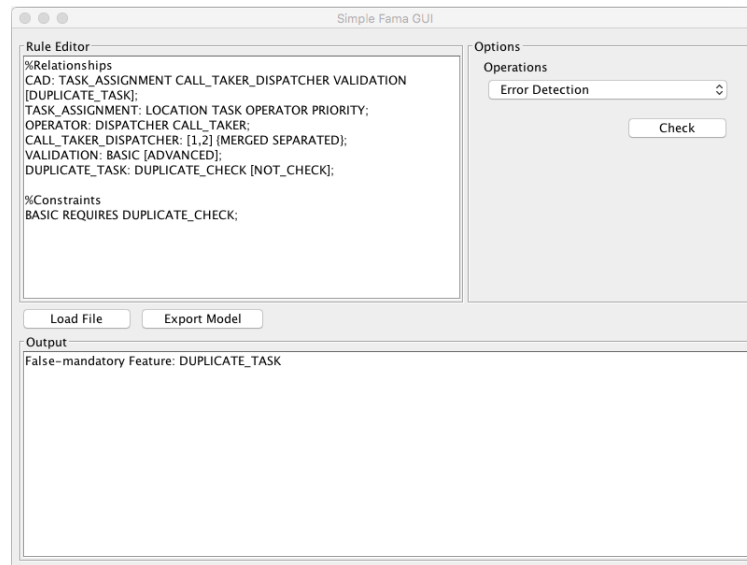


Figure 6.4: Analysis of False Optional 2 in CAD

In Figure 6.4 we verified that analysis rule False Option 2 is correct.

6.1.3 Dead Feature 1

For verifying the Dead feature of analysis rule shown in 4.4 we have to modify the CAD Domain for make this scenario. So the CAD Domain modified Fama Format will be:

```
%Relationships
CAD: TASK_ASSIGNMENT CALL_TAKER_DISPATCHER VALIDATION [DUPLICATE_TASK];
TASK_ASSIGNMENT: LOCATION TASK OPERATOR PRIORITY;
OPERATOR: DISPATCHER CALL_TAKER;
CALL_TAKER_DISPATCHER: [1,2] {MERGED SEPARATED};
VALIDATION: BASIC [ADVANCED];
DUPLICATE_TASK: DUPLICATE_CHECK [NOT_CHECK];
```

%Constraints

```
BASIC EXCLUDES DUPLICATE_CHECK;
```

If we check for the error in our tools we will find that `DUPLICATE_TASK`, `DUPLICATE_CHECK` and `NOT_CHECK` become Feature by analysis rule Dead Feature 1.

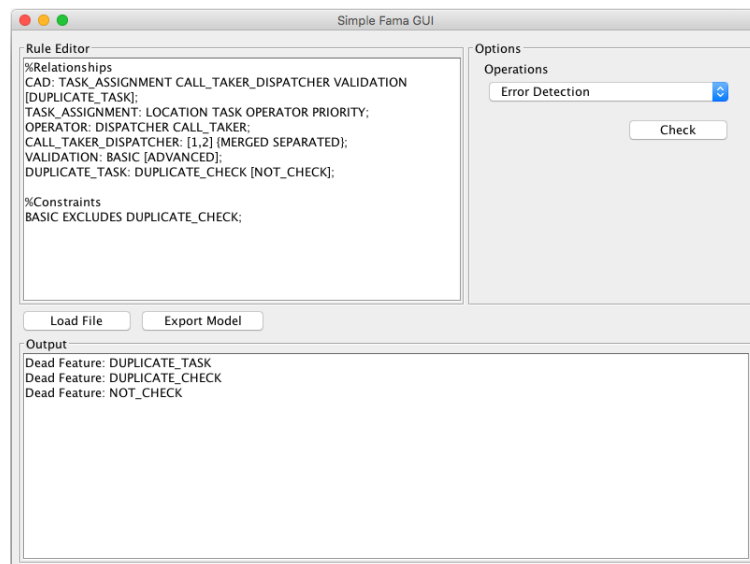


Figure 6.5: Analysis of Dead Feature 1 in CAD

In Figure 6.5 we verified that analysis rule Dead Feature 1 is correct.

Chapter 7

Conclusion

Successful development of software product line requires appropriate organization and management of products requirements. In feature model, when the volume of information grows the possible of feature model anomalies is high. In this thesis we proposed some feature analysis rule for feature model and tried to verify existing and proposed software product line feature analysis rules. we developed a tool to verify the feature models and for doing other operations. In the tool we also implemented a method to export the model in such format which is compatible with other feature model analysis tools.

7.1 Future Work

We only analyzed False Optional and Dead Feature in feature model. In Future we will extend our research to verify cardinality rules also. Our developed a tool which can export FaMa feature model to other feature model analysis tool however our tools is now only limited for generating 2-3 level of feature model. In our feature work we will updated the tool which can export n-Level of feature model and support new feature model analysis tool.

Bibliography

- [1] *Software product lines: practices and patterns*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.
- [2] D. L. Parnas, “Software fundamentals,” D. M. Hoffman and D. M. Weiss, Eds. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, ch. On the design and development of program families, pp. 193–213. [Online]. Available: <http://dl.acm.org/citation.cfm?id=376584.376628>
- [3] A. Hein, J. MacGregor, and S. Thiel, “Configuring software product line features,” in *Proceedings of the ECOOP 2001 Workshop on Feature Interaction in Composed Systems (FICS 2001), Budapest, Hungary, June 18-22, 2001*.
- [4] P. C. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, ser. SEI Series in Software Engineering. Addison-Wesley, August 2001.
- [5] J. Bosch, *Design and use of software architectures: adopting and evolving a product-line approach*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.
- [6] K. Czarnecki, S. She, and A. Wasowski, “Sample spaces and feature models: There and back again,” *Software Product Line Conference, International*, vol. 0, pp. 22–31, 2008.
- [7] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, “Feature-oriented domain analysis (foda) feasibility study,” Carnegie-Mellon University Software Engineering Institute, Tech. Rep., November 1990.

-
- [8] P. Trinidad, D. Benavides, A. Durán, A. Ruiz-Cortés, and M. Toro, “Automated error analysis for the agilization of feature modeling,” *J. Syst. Softw.*, vol. 81, no. 6, pp. 883–896, Jun. 2008.
- [9] 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,” *Ann. Softw. Eng.*, vol. 5, pp. 143–168, Jan. 1998.
- [10] K. Czarnecki and U. W. Eisenecker, *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000.
- [11] A. Felfernig, D. Benavides, J. A. Galindo, and F. Reinfrank, “Towards Anomaly Explanation in Feature Models,” in *Proceedings of the 15th International Configuration Workshop, Vienna, Austria, August 29-30, 2013*.
- [12] D. Benavides, S. Segura, and A. R. Cortés, “Automated analysis of feature models 20 years later: A literature review,” *Inf. Syst.*, vol. 35, no. 6, pp. 615–636, 2010.
- [13] T. von der Massen and H. Lichter, “Deficiencies in Feature Models,” in *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, T. Mannisto and J. Bosch, Eds., 2004.