CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

# Improving Use Case Based Feature Model Construction for Software Product Lines

by

Asra Ishtiaq

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing
Department of Computer Science

2018

**Dedication**

I dedicate my dissertation work to my supervisor, family, all other teachers and friends. A special feeling of gratitude is for my loving family for their love, endless support and encouragement.

CAPITAL UNIVERSITY OF SCIENCE & TECHNOLOGY
ISLAMABAD

## CERTIFICATE OF APPROVAL

**Improving Use Case Based Feature Model Construction for Software Product Lines**

by

Asra Ishtiaq

MCS163017

**THESIS EXAMINING COMMITTEE**

| S. No. | Examiner | Name | Organization |
|---|---|---|---|
| (a) | External Examiner | Dr. Rizwan Bin Faiz | RIU, Islamabad |
| (b) | Internal Examiner | Dr. Muhammad Tanvir Afzal | CUST, Islamabad |
| (c) | Supervisor | Dr. Aamer Nadeem | CUST, Islamabad |

Dr. Aamer Nadeem
Thesis Supervisor
October, 2018

Dr. Nayyer Masood
Head
Dept. of Computer Science
October, 2018

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
October, 2018

# *Author's Declaration*

I, **Asra Ishtiaq** hereby state that my MS thesis titled "**Improving Use Case Based Feature Model Construction for Software Product Lines**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

**Asra Ishtiaq**

Registration No: MCS163017

# *Plagiarism Undertaking*

I solemnly declare that research work presented in this thesis titled "**Improving Use Case Based Feature Model Construction for Software Product Lines**" is solely my research work with no significant contribution from any other person. Small contribution/help wherever taken has been dully acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Capital University of Science and Technology towards plagiarism. Therefore, I as an author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred/cited.

I undertake that if I am found guilty of any formal plagiarism in the above titled thesis even after award of MS Degree, the University reserves the right to withdraw/revoke my MS degree and that HEC and the University have the right to publish my name on the HEC/University website on which names of students are placed who submitted plagiarized work.

**Asra Ishtiaq**

Registration No: MCS163017

# *Acknowledgements*

I praise and thank ALLAH (S.W.T) for His greatness and for giving me the strength and courage to complete this research work. I would like to thank all who in one way or another contributed to the completion of this research work. I would like to express my greatest gratitude to my supervisor Dr. Aamer Nadeem for his unwavering support, encouragement, advice and patience. It is because of his constant support that I was able to complete my thesis. Thanks to all the CSD (Center For Software Dependability) members especially Mr. Qamar uz Zaman for his valuable feedback which helped me in improving my thesis. I also like to thank Dr. Muhammad Arshad Islam for introducing me to the world of research. His guidance was quite helpful during this research work.

I am extremely thankful to my family for their endless support, encouragement, and prayers throughout the completion of this Master of Science degree. Shout out to my family for tolerating my mood swings for the past year and being patient with me. To my sister for believing in me and for her constant love, support, and care. To my sister-in-law for moral support, tea, and fries. Thank you, my dear friend Sundus Ali, for helping me survive all the stress from this year and not letting me give up.

Finally and without hesitation, I would like to thank coffee, Harry Potter, BTS and Pinterest for helping me out in my darkest days.

# *Abstract*

Software Product Line (SPL) is a set of software products that share a common set of assets known as core assets satisfying need of particular domain. Software Product Line Engineering (SPLE) provides low cost and efficient development of diverse highly related products. SPLE provides efficient development of products by reusing core assets shared by the products rather than starting from the scratch. SPL variability management is a vital part of SPLE. SPL variability management provides information about the core assets shared by all the products of SPL as well as about the assets that differentiate each product from other products. Feature modeling is the most frequently used technique for managing the variability of SPL. FM construction can be done in two ways: top-down or bottom-up. The top-down approach is better than the bottom-up approach due to the fact that it can be used for the construction of FM of new SPL. Top-down approach constructs FM in the early stages of software development life cycle as compared to the bottom-up approach. FM constructed in early stages of the software development life cycle can be used for testing purpose. For top-down FM construction use case based approach is better than textual requirement based techniques as it does not require deep natural language processing (NLP) analysis which is a time-consuming process.

The focus of this research is on the improvement of use case based FM construction. A number of use case based FM construction techniques have been introduced. These techniques perform well in terms of feature and hierarchy extraction but fail to do so for constraint extraction. In addition to that existing approaches do not cover constraints generated by sequential dependencies. To overcome deficiencies of existing approaches, we propose an algorithm that uses an activity diagram for the better extraction of the constraints. Using activity diagram for FM construction helps in extracting constraints for such scenarios in which existing techniques failed to extract constraints. Using an activity diagram also helps in extracting constraints based on sequential dependencies. We have evaluated and compared our approach with the existing approach using different case studies. The result shows that our approach performs better than existing approaches in terms of the number of extracted constraint and quality of constraints extracted for FM.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **SPL** | Software Product Line |
| **FM** | Feature Model |
| **CTC** | Cross Tree Constraint |
| **SPLE** | Software Product Line Engineering |
| **AFT** | Aplication Feature Tree |
| **DFT** | Domain Feature Tree |
| **UML** | Unified Modeling Language |
| **XML** | Extensible Markup Language |

# List of Symbols

| | |
|---|---|
| $\Rightarrow$ | Requires constraint |
| $\Leftrightarrow$ | Excludes constraint |
| $FM_{simple}$ | FM without constraints |
| $FM_{constraint}$ | FM with constraints |

# Chapter 1

# Introduction

Software Product Line (SPL) is a set of software products that share a common set of assets known as core assets satisfying need of particular domain. In SPL products are derived from these core assets in a prescribed way [1]. SPL supports mass customization by allowing generation of variable products that share core assets. The main goal of the SPL is to speed up the process of the production and to reduce the time to market. In previous production approaches, each product is developed from the scratch that is a time-consuming process. SPL gives an advantage by allowing software reuse for production of new products using core assets [2]. Core assets refer to assets that are part of every product of SPL. When creating a new product in SPL variable asset will be added to the core assets rather than starting from the scratch and creating core assets again for the new product [3]. Consider an example of Mobile phone SPL, In Mobile phone SPL the core assets will be call function, message function, and setting function as every mobile phone must support these functions. The variable assets in mobile phone SPL can be Bluetooth, Wi-Fi and media option as these assets are not part of every mobile phone. Software product line engineering (SPLE) is a paradigm for the development of software from a common set of features satisfying need of particular domain. SPLE helps organizations in developing their products from reusable core assets. SPLE assist proficient, high caliber and low-cost development of software by reusing core assets [4]. SPLE concerns two concepts: commonality and variability. Commonality refers to characteristics that are shared by all products of the SPL. Variability refers to differences between products of the SPL. For the development of core assets, software engineering must exploit commonalities and manage variability. Different variability models have been proposed for managing SPL variability [5]. Feature modeling is the most common technique used for managing the variability of SPL products. Feature modeling can be done using table or graph [6]. The most common technique is graph based. In graph-based technique feature models (FM) are utilized. The concept of FM was first introduced by [7] in FODA report for modeling variability of SPLs.

## 1.1 Feature Model

FM represents commonalities and variability of SPL in terms of features and relationships between them. A feature is referred to increment in product functionality. In FM, a feature may have one or more child features. The top feature of the FM is called root feature and this feature usually represent the concept of FM [8]. FM represents a hierarchical and structural relationship between features [9]. A legal combination of features forms a product of SPL. FM is a hierarchical structure that represents all valid products or legal combinations of features. Modeling variability with FM helps in Construction of new SPL as well as derivation of new products for already existing SPLs. FMs can be broadly categorized into Basic FMs, Cardinality-based FMs and extended FMs. All these three FMs are discussed below.

### 1.1.1 Basic Feature Model

According to Benavides et al. in basic FM, two kinds of relationships exist between features [10].

- relationship between parent-child features

- cross-tree constraints between features

In basic FM, root feature represents the concept of the FM. All other features in the FM can be either optional or mandatory. All child features can have "Alternative" or "OR" relationship with the parent feature. Basic FM also has cross-tree constraints (CTCs) between features. CTCs models feature dependencies. CTCs avoid generation of invalid products for SPL.

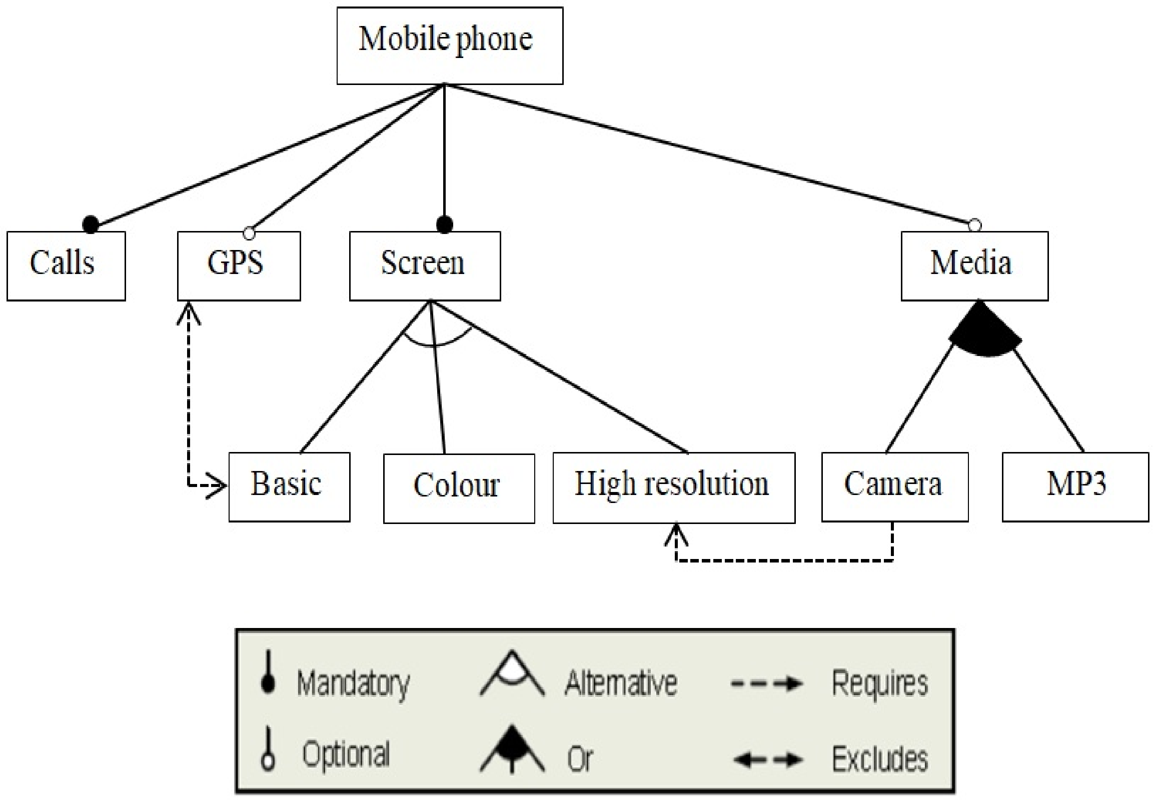


FIGURE 1.1: Feature Model Example

Parent-Child relationships between feature and sub features in basic FM are categorized as [11]:

1. **Mandatory** child feature has a mandatory relationship with its parent feature if child feature must be included in every product that contains parent feature [11]. Calls and screen are mandatory features so every mobile phone product generated by FM shown in Figure 1.1 must contain feature calls and screen.

2. **Optional** child feature has an optional relationship with its parent feature when it is not required to include child feature in every product in which parent feature is included [12]. GPS and Media are optional features in FM shown in Figure 1.1 so, GPS and Media can be optionally included in each generated mobile phone product.

3. **Alternative** set of child features has an alternative relationship with its parent feature when only one of the child feature can be included in a product in which parent feature is selected [13]. Basic, Color and High-resolution features have an alternative relationship with parent feature Screen as shown in Figure 1.1 so, whenever feature Screen is selected only one of the child features: basic, Color, High resolution can be selected for each product.

4. **OR** set of child features has an or-relationship with parent feature when at least one of the child features must be selected when parent feature is included in a product [14]. Camera and MP3 has OR relation with parent feature Media in FM shown in Figure 1.1. Feature MP3, camera or both can be selected when feature media is selected in a product.

In addition to parent-child relationships, there are also CTCs between features. CTCs are used to avoid generation of invalid configurations. CTCs between features are recognized as:

1. **Requires** if feature A requires feature B then selection of feature A in product implies selection of feature B in such product [15]. Feature Camera requires feature High resolution in FM shown in Figure 1.1 so whenever camera feature is selected in a product then feature high resolution must be selected in that product.

2. **Excludes** feature A excludes feature B implies that both features cannot be included in the same product [16]. Feature GPS and Basic has an exclusion relationship as shown in Figure 1.1 so, If feature GPS is selected in a product then feature Basic should not be selected in that product and vice versa.

### 1.1.2 Cardinality-based Feature Model

Cardinality-based FM is basically an extension of original FODA FM [17, 18]. In cardinality-based FM, FM relationships are expressed in the form of UML multiplicities. In cardinality-based FM relationships are divided into feature cardinality and group cardinality.

**Feature cardinality** shows the optionality of the feature, weather feature is optional or mandatory. Feature cardinality is represented by [n..m] where n represents lower bound and m represents upper bound. The optional feature is represented by [0..1] and mandatory by [1..1] [19].

**Group cardinality** represent or and alternative relationships of FM. Group cardinalities are represented by <n..m> where n is lower bound and m is upper bound that gives information about the number of features that are selected whenever a parent is chosen for the product. Or relationship is represented by <1..N> where N is the number of child features so, it shows that a minimum 1 and maximum N number of child features can be selected for or relationship. An alternative relationship is represented by <1..1> so one and only one child feature must be selected for an Alternative relationship.

### 1.1.3 Extended Feature Model

Another FM notation that has been proposed is extended feature models. In extended feature models, feature attributes are added in the FM that gives extra information about the features. Example of feature attributes includes: memory, space and speed. Different feature attribute element has been used by proposed approaches [20, 21]. Benavides et al. argue that the attribute must consist of name, domain and value [10]. In the FM in Figure 1.2 attribute Memory and speed for each feature: USB, Bluetooth and Wi-Fi are included in FM. Each attribute has a name, domain, and a value. For feature Wi-Fi, there are two attributes Memory and Speed as indicated by the name of the attributes. Domain of all two attributes for Wi-Fi feature is ***Real*** and value for each attribute of Wi-Fi is also given. For the Wi-Fi feature, a value of memory attribute is 725 and value of Max speed is 3.6 as shown in Figure 1.2.

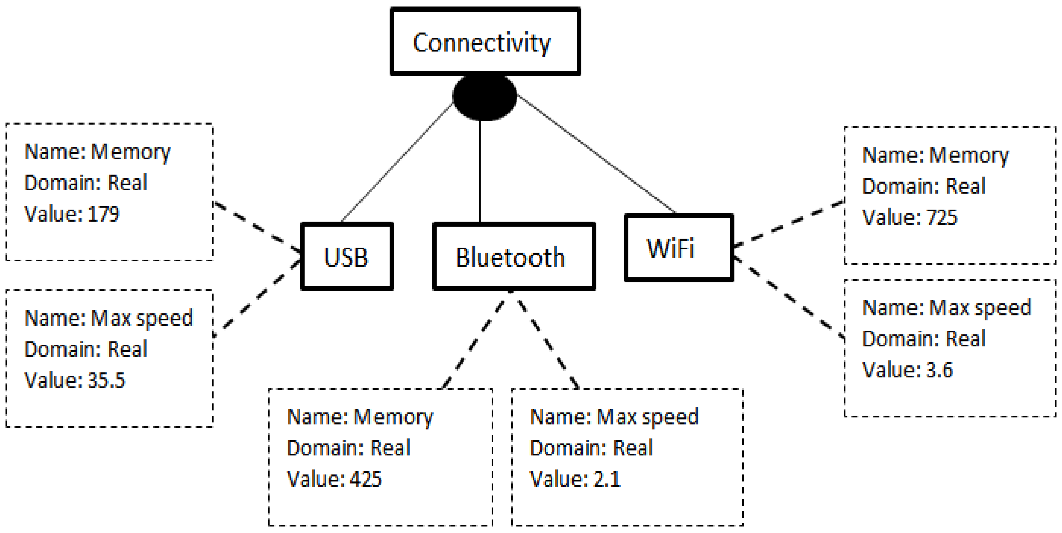


FIGURE 1.2: Example of extended feature model

## 1.2 Feature Model Construction

FM play important role in modeling variability of SPL. FM presents all the possible configurations of the SPL with the help of features and relationships between them. Each valid configuration presented by FM refers to product of SPL. FM gives information about all the valid products that can be generated for SPL so, helps in generation of new product of SPL. FM construction also helps in SPL testing process. FM construction process can be divided in to three main steps: features identification, Features organization and constraints identification. Feature is a product functionality that is used for capturing commonality and variability between products of SPL. In feature identification step, features of different SPL products are extracted either from requirements of the products or configurations of products [22] depending upon the FM construction approach used. In feature organization step, hierarchy of features and parent-child relationships between features are determined. In constraints identification step, cross tree constraints between features are determined. FM can be constructed from undeveloped products of SPL as well as developed products of SPL. These two feature model construction approaches use different input assets for FM construction and has been discussed below:

### 1.2.1 Top-down Approach

In top-down approach, FM is constructed by exploring functional requirements of undeveloped products of SPL. This approach is used when creating new SPL. In top-down approach, FM is constructed from the functional requirements and then products are derived from the constructed FM [23]. Using requirements for FM construction is really helpful because requirements are radially available. Using top down approach for FM construction results in development of FM in early stages of Software Development Life Cycle (SDLC). FM created in early stages of SDLC can be used in designing and testing phase. Different input artifacts can be used for the construction of FM by top-down approach. On the basis of input artifacts top-down approach can be categorized as:

**Textual documentation based construction** approach use requirements of product variants for the construction of FM. Features and hierarchy of FM is extracted by analyzing textual documents that contains requirements. Requirements or product descriptions written in Natural language (NL) are given as input for construction of FM. In this approach natural language processing (NLP) techniques are used for extraction of FM from textual documents. FM construction from Textual documents usually requires documentation of requirements in some standard format for accurate extraction of features. For FM construction from this approach requires transformation of extracted requirements in to some standard format without which features extraction is not possible. Apart from that deep

NLP is required in this technique for the extraction of features which is usually quite expensive.

**Use case based construction** approach utilize use case diagrams for the construction of FM. This technique utilizes use cases and relationships between them for extraction of features and structure of FM. Usually includes and excludes relationships of use cases are utilized for use case based construction of FM [24]. Using use cases for FM construction makes feature extraction quite easy by mapping each use case on to feature.

### 1.2.2 Bottom-up Approach

In bottom-up approach, FM is constructed by examining configurations of already developed product variants of SPL. This approach assumes that SPL is created after the development of several product variants of SPL [25]. This technique cannot be used for creating new SPL. In this approach, codes or configurations of already developed product variants are analyzed and feature model is constructed from them [26].

Using top-down approach for FM construction has number of advantages as compared to bottom-up approach. Bottom-up approach can only be used for construction of FM for already developed SPLs. Bottom-up approach will be of no help if FM construction is required for new SPL. Apart from that in bottom-up approach code or configurations of product variants of SPL are used which are usually not available or hard to get hold of, on the other hand requirements are easily available and can be easily extracted. Another important advantage of top-down approach is that FM can be constructed in early phases of SDLC. Requirements are available in early stages of SDLC as compared to product configurations. Using requirement specifications rather than code or configurations will result in early construction of FM. FM constructed in early phase of SDLC can be used for designing as well as in testing phase for test case generation.

## 1.3 Anomalies in FM

Anomalies refer to redundant or contradictory information presented by FM. Anomalies can occur in FM if wrong CTCs are included in the FM [27]. Anomalies can affect the quality of the constructed FM. FM with anomalies gives the wrong idea of domain which is undesirable.

Most commonly four types of anomalies can occur in FM due to wrong usage of CTCs as listed below:

### 1.3.1 Void FM

Void FM anomaly exists if FM does not generate any valid configuration [28]. The void FM anomaly occurs due to wrong usage of the CTCs. FM with valid CTCs can never be void. Void FM anomaly is very critical as no valid configuration of SPL can be generated form FM which makes FM useless. Void FM anomaly can occur if there is a *excludes* constraint between two mandatory features. In Figure 1.3 there is *excludes* constraint between mandatory feature A and mandatory feature B. Due to *excludes* constraint between feature A and B, no valid configuration can be generated by FM given in Figure 1.3 because each mandatory feature should be part of every valid configuration of FM, whereas *excludes* constraint in FM shown in Figure 1.3 implies that two mandatory features A and B cannot be part of same product which is contradictory.

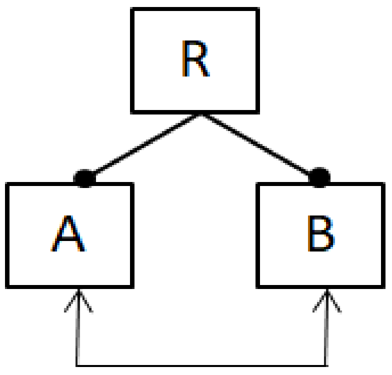


FIGURE 1.3: Void FM anomaly

### 1.3.2 Dead Feature

Dead feature refers to the feature that is a part of FM but due to wrong usage of the CTCs it is not included in any of the products generated by the FM [29]. Dead features are present in FM but are not used in any of the product. If a mandatory feature excludes optional feature then optional feature becomes dead feature. Mandatory feature will be a part of every product of SPL and excluded optional constraints will not be a part of any SPL product. In Figure 1.4 feature A is a mandatory feature and it excludes optional feature B due to this CTC feature B will never be a part of any product of SPL and will be the dead feature.

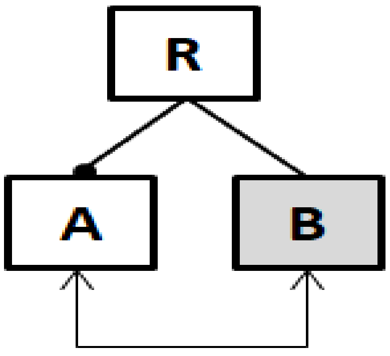


FIGURE 1.4: dead feature anomaly

### 1.3.3 False Optional Feature

False optional feature refers to a feature that is modeled as optional in FM but is a part of every product derived by SPL [30]. If a mandatory feature in FM requires an optional feature then that optional feature will be false optional feature. The mandatory feature is a part of every product so, optional feature required by mandatory feature will also be a part of every product hence will be modeled as false optional feature. In Figure 1.5 Mandatory feature A requires optional feature B. As feature A is mandatory it will be part of every product and feature B is required by feature A so it will also be a part of every product. Feature B despite being part of every product is modeled as optional so it is false optional feature.

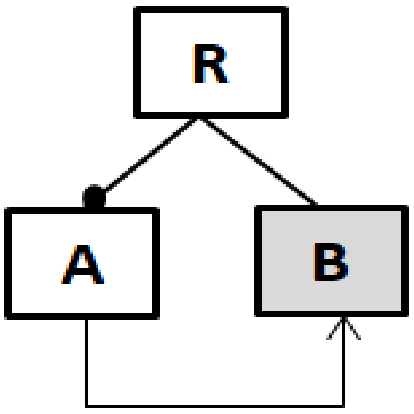


FIGURE 1.5: false optional feature anomaly

### 1.3.4 Redundancies

Redundancies occur in FM if same information of FM is modeled in multiple ways. Redundancies in FM can decrease the maintainability of FM and are usually considered as negative point. Redundant features as well as redundant constraint can occur in FM. redundant constraints occur in FM due to improper use of CTCs. If there is a *requires* constraint from optional feature to mandatory feature in a FM then this *requires* constraint is redundant constraint [31]. Mandatory feature is already a part of every product. *Requires* constraints from optional feature to mandatory feature implies inclusion of mandatory feature in every product which is already being fulfilled, there is no need for that *requires* constraint. In Figure 1.6 optional feature B *require* mandatory feature A , this *requires* constraint is redundant constraint.

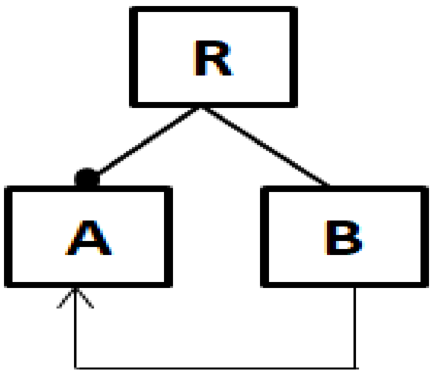


FIGURE 1.6: Redundant constraint

FM with anomalies in it gives the wrong idea of the domain. Anomalies effect the number of product derived from the SPL and even leads to null configuration extraction from FM. FM must be free from anomalies for the correct depiction of the domain.

## 1.4 Problem Statement

A number of techniques have been proposed for FM construction from use cases. Most of the proposed techniques do not extract CTCs while constructing FM. Some approaches that support CTCs identification deduce constraints between features using “extend” and “include” relationships of use cases. Utilizing only relationships between use cases for the extraction of FM constraints is not enough. Extraction of FM constraints from use cases that does not have “extend” and “include” relationship will not be possible if extraction only depends on these relationships. To counter this problem such approach is required for extraction of constraints that exploit resources other than use cases relationships. It will help in extracting FM constraints even if relationships between use cases are not available. Other techniques that support identification of CTCs utilize use case variability modeling techniques. These techniques first model variability in to use case model by using use case variability modeling techniques then use these variabilitys for extraction of CTCs. use case variability modeling is itself a complex task so using these techniques can be very expensive. In addition to that, use case sequential dependencies between use cases have never been exploited by any of the proposed techniques for the extraction of CTCs.

## 1.5 Research Questions

In this research, we will propose use case based construction of FM for modeling commonality and variability of SPL. However the following questions must be taken into account.

- **RQ 1:** Can we extract feature model constraints for certain scenarios not covered by existing techniques?

  The literature survey is carried out and deep analysis of existing techniques is done to find the gaps in previously proposed techniques. Through literature survey, we identify that an algorithm should be designed that use sequential dependencies of the use cases for extraction of CTCs.

- **RQ 2:** Are constraints identified by proposed approach higher than existing techniques?

  To answer this question, evaluation and comparison of proposed algorithm with existing technique is done using different case studies. In the end, number and quality of constraints extracted by both approaches are compared.

Our research will be focused to answer these research questions

## 1.6 Research Methodology

1. First of all, we have done the literature review to identify existing approaches for construction of FM from use case model. After studying various techniques, we have reached the conclusion that previously proposed techniques failed in extracting constraint for certain scenarios.

2. To overcome the research gap we will implement new technique for construction of FM from use case model. Proposed technique will make use of activity diagram of use cases to overcome gaps in previously proposed techniques.

3. Following steps will be performed for implementation of our proposed approach:

   (a) In the first phase, we collect all data including use cases and FM of SPL's without constraints.

   (b) After collecting the data, we create system level activity diagram from use cases that model sequential dependencies of use cases.

   (c) After having created the activity diagram, features are identified from activity diagram as well as from input FM of SPL. Features identified by both are compared and matching features are stored.

   (d) After identification of the matching features, constraints between these features are identified using constructed activity diagram. Identified constraints are added in to the feature model.

   (e) FM is checked for anomalies after adding constraints. If anomalies are detected in FM, wrong constraints causing anomalies in FM are removed from FM.

4. In the end, evaluation of proposed technique will be done. Use case studies from [32] will be used for construction of FM. we will evaluate our approach by comparing it with existing approach [33] as well as with benchmark FM. Using selected case studies, two FMs will be constructed. One FM will be constructed using existing approach other will be constructed using proposed

approach. Comparative evaluation of number of constraints and quality of constraints extracted by both techniques will be done to check the improvement in FM construction by proposed approach.

## 1.7 Thesis Organization

Rest of the thesis is organized as; in Chapter two existing approaches for construction of FM from use cases are discussed. Third chapter is about proposed solution, proposed methodology and algorithm are discussed in this chapter. Fourth chapter is on implementation details of proposed algorithm, fifth chapter is about results and discussion and sixth chapter is on conclusions which we have made from this research and future work on how we can extend this research work.

# Chapter 2

# Literature Review

In this chapter, we discuss the related work that has been done in the field of FM construction from use case models. We conducted a detailed survey analysis to find the gap and comparison is performed among the existing approaches.

SPL is a set of related software's that are created from commons set of core assets instead of starting from the scratch. SPLE process helps in efficient development of the software. Feature is used for differentiating products of the SPL by showing commonalities and variabilitys of SPL products. Different variability models have been proposed for managing commonalities and variabilitys of SPL [34]. FM is considered as de facto standard for managing variability of SPL [35].

FM plays an important role in providing reusability information of specific domain. FM displays information of all products of SPL with the help of features and relationship between them. In F8M, features are arranged in hierarchy with the help of relationships and constraints between them.

FM construction process includes three main steps: feature identification, constraints identification and hierarchy identification. Construction of complete FM is quite important for derivation of all valid product configurations of domain. Different input sources are used for FM construction including product configurations and requirement specifications. Model construction is done in early phases of SDLC. Using requirement specifications rather than product configuration makes early construction of FM possible. FM constructed in early phase of SDLC can be used for designing as well as in testing phase.

## 2.1 Use Case Based FM Construction Techniques

In object oriented requirements are modeled using use case diagrams and use case descriptions. Number of techniques has been proposed for construction of FM using use cases as input asset. Some of the existing use case based FM construction techniques do not extract constraints of FM while other techniques construct FM with constraints. Both FM construction techniques with and without constraints extraction are discussed below.

### 2.1.1 FM Construction Without Constraint Extraction

**Griss et al., 1998** introduced the idea for construction of FM from the use case domain model [24]. Use case models are constructed for the each application in the domain. Constructed use case models are merged to form a domain use case model. Domain use case model is used for the construction of FM. Features are identified from domain use case model, in domain use case model each use case correspondent to a feature. Mandatory and optional features are identified by frequency of use cases in domain application. If the use case of a particular feature has higher occurrence rate in domain applications then this feature is considered as mandatory else considered as optional. Features are decomposed in to sub features according to the relationships (use and extend) of corresponding use case in the domain use case model. If corresponding use case of a feature is a variation point in use case model then sub features of that feature are created which correspond to variation points of use case. In the end constructed FM is restructured by analysts and CTC's are identified by them manually.

In this approach each use case maps to feature but in reality on use case can maps to number of features and number of use cases can form one feature. CTC's are also identified manually by the analysts so; quality of FM will heavily depend on the knowledge of analyst.

**Braganca and Machado, 2007** proposed automated mapping between use case and FM [36]. For modeling variability use case Meta-model is extended. In this technique each use case is mapped to the feature. Use case variability modeling technique is used for modeling variability in "Include" and "extend" relationships of use cases. Modeled variabilities are then used for identification of features and sub features of FM. The *include* relationship of use cases is used for identification of sub features. Using *include* relationship for identification of sub features can result in generation of redundant constraints.

This technique does not consider inner description of use cases for FM construction. In addition to that, it does not provide any method for identification of CTCs. Both *requires* and *excludes* constraints are not identified by this approach.

**Wang et al., 2009** extract FM relations with out considering inner descriptions of use cases. Wang et al. [37] proposed semi-automatic technique for the FM construction that use inner descriptions of use cases for arranging features in FM. This technique takes a set of use cases that are described with use case diagrams and use case scenarios as input and generate DFM as output. FM for each application called application feature model (AFM) is constructed using input use cases. Use case descriptions are used for extraction of features and relationships between features. Features are discovered by identifying operation and object in an action and by identifying method of an action. After feature discovery, refinement relationships: decomposition, characterization and specialization are identified between features. Decomposition relationship is identified by exploring sub flows, exploring system operations, identifying features referencing to same object, identifying "is a " relationship between objects and identifying "has a" relationship between objects. Characterization/ specialization are identified between features by identifying method of an action. After identifications of features and relationships between features AFM for each application is constructed. Afterwards, DFM is constructed by adjusting and merging set of AFMs generated earlier. The Automatic construction of AFMs and DFM is done with the help of provided rules and algorithms.

In this research, only relationships between use cases are used for FM construction. This approach does not exploit semantics for constraint derivation. FM constraints are also not identified by this approach.

### 2.1.2 FM Construction With Constraint Extraction

**Lin and Zhou, 2012** FM construction technique proposed by Wang et al. [37] does not extract FM constraints. In another study, construction of FM using system use case models is conducted by Lin and Zhou [38] that extracts Requires constraint of FM. In this technique use case diagrams as well as use case descriptions are used for FM construction. First, system use cases with specific characteristics are selected and domain use case diagram (DUCD) is constructed by merging set of use case diagrams (UCDs). Second, primary FM is constructed using DUCD that involve two steps: grouping of use cases according to resources manipulated by them and tracing relationship between use cases and features. For FM construction use cases are mapped to features. Include relationship between use cases is used to model decomposition relationship in FM. Generalize relationship from use case diagram is used to model specialization relationship in FM.

*Requires* constraint in FM is captured by using extension scenario of extend relationship from use case diagrams. Extension scenario are usually not available in use case diagrams, *requires* constraint will not be extracted if extension scenario is not available. *Excludes* constraint is not identified by this approach.

**Casalnguida and Duran, 2012** proposed construction of FMs from UML requirement models [22]. Use case diagrams, activity diagrams and use case descriptions are utilized for FM construction. First, UML models are constructed from the requirements. Use case variability modeling technique is used for modeling variability in use case diagram. Activity diagram of each use case is also constructed and variability is modeled in to it as well. Afterwards, FM is automatically constructed from use case diagram with variability. Generation of FM involves two transformations: UCD2FM and Reqs2FM. Construction of initial FM from use case diagrams is done using UCD2FM with the help of 8 rules. In UCD2FM transformation use case are mapped to features and variability modeled in use case diagram utilized for extraction of constraints in FM. Reqs2FM transformation is applied on activity diagrams and initial FM for extracting extra features to build a final FM. For evaluation of proposed approach online library application case study is used. Results show that 15% of features are extracted by application of UCD2FM and 40% by applying Reqs2FM transformation. Results further illustrated that 50% of the features introduced by Reqs2FM are critical.

This approach requires modeling of variability in to use case diagram as well as in to activity diagrams of each use case. Modeling variability in to UML is quite expensive. This approach uses activity diagrams of use cases which will result in generation of low granularity features.

**Mefteh and Bouassida, 2014** proposed fully automated approach for extraction of FM from documented use cases [39]. This approach deals with incomplete use case diagrams. Incomplete use cases diagrams are completed and refined using information from use cases documentation. Each use case is treated as feature for FM construction. Hierarchy of features is identified using the Formal concept analysis (FCA). Relationships between features are identified using the hypernym and synonym semantics. Or relationship is identified using "Meronymy" relationship. XOR relationship is identified using "Synonyms" relationship, if two features name has synonym relationship and have same parents then there is XOR relationship between them. Afterwards, Constraints among features are identified using the semantics criteria and "includes" relation between use cases. If name of features are synonyms and belong to different parents, then there is "excludes" constraint between these features. If two use cases has "include" relationship, then features extracted from these use cases has requires relationship.

This technique requires use cases with detailed description of "goal in context" field for extraction of features. Constraints identified in this approach use relationships of use cases, unavailability of these relationship will result in construction of incomplete FM.

**Mefteh et al., 2015** proposed implementation of their previous work [33]. UC2FM-tool is proposed for automatic construction of FM from documented use case diagrams. For evaluation, five FMs from different domains are used. The quality metric of FMs generated from proposed tool are compared with the FMs constructed by experts.

## 2.2 Comparison

In this section use case based FM construction approaches from literature are compared. All the above discussed techniques utilizes use cases for FM construction. Techniques proposed by Wang et al. and Braganca and Machado do not extract constraints as no constraints extraction method is given by these techniques [36, 37]. In another technique proposed by Griss et al., no method for constraint extraction is given and CTCs are manually identified by the analysts [24]. The output FM generated by these techniques do not have CTCs in it. Technique proposed by Lin and Zhou extracts *requires* constraint [38]. The *excludes* constraint is not identified by the technique as well as *requires* constraint cannot be identified by this technique if extension scenarios are not present in use case diagrams. Both CTCs are identified by the technique proposed by Casalnguida and Duran [22]. The down side of this approach is that, for constraints extraction variability modeling technique is used to modeling variability in use case diagrams and activity diagrams. Use of variability modeling is quite expensive in addition to that, features generated by activity diagrams are low granularity features. The technique proposed by Mefteh and Bouassida is the only technique that extracts both CTCs without modeling variability [39]. Using this technique *Requires* constraint cannot be extracted if use case relationships are not present in input use case diagrams.

Overall comparison of all the above discussed techniques is given in Table 2.1. First column of Table 2.1 contains reference of research paper. Source asset used for FM construction by each approach is given in second column. Type of FM notation used by each approach is given in third column and following six columns give information about FM features and relationships extracted by each approach. ***MF*** stands for mandatory feature, ***OF*** for optional feature, ***Alt*** for *alternative* relationship, ***OR*** for *or* relationship, ***Req*** for *requires* constraint and ***EX*** for *excludes* constraint. Last two columns contain information about tool supported by each technique and limitations of techniques respectively.

Table 2.1: Comparison of use case based FM construction approaches

| **Author** | **Source** | **FM notation** | **MF** | **OF** | **Alt** | **OR** | **Req** | **Ex** | **Tool supp-orted** | **Limitations** |
|---|---|---|---|---|---|---|---|---|---|---|
| Griss et al. (1998) [24] | Use case diagrams | Basic Feature model | yes | yes | yes | yes | no | no | no | Requires and excludes constraints are manually identified |
| Braganca and Machado et al. (2007) [36] | use case diagrams | Cardinality based | yes | yes | yes | yes | no | no | no | Requires and excludes constraints are not extracted. |
| Wang et al. (2009) [37] | use case diagrams | Basic Feature model | yes | yes | yes | yes | no | no | no | Requires and excludes constraints are not extracted. |
| Lin & Zhou (2009) [38] | use case diagrams and use case discriptions | Basic Feature model | yes | yes | yes | yes | yes | no | no | Excludes constraint is not identified |
| Casalng -uida & Durn (2012) [22] | use case diagrams, use case descriptions and activity diagrams | Cardina -lity based | yes | yes | yes | yes | yes | yes | no | Dependen -cies between high level features are not identified |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Mefteh & Bouas -sida (2014) [39] | complete and incomplete documented UC diagrams ,scenarios in NL | Basic Feature model | yes | yes | yes | yes | yes | yes | no | Requires constraint can not be identified if include relationship is not present in Input |
| Mefteh et al. (2015) [33] | complete and incomplete documented UC diagrams, scenarios in NL | Basic Feature model | yes | yes | yes | yes | yes | yes | yes | Requires constraint can not be identified if include relationship is not present in Input |

## 2.3 Gap Analysis

Existing use case based techniques for FM construction use different source for the extraction of features and constraints from the use cases. Constraints are vital part of any FM so, extraction of constraints is quite important for FM construction. Comparison of existing techniques is shown in Table 2.1. Most of the proposed techniques do not extracted CTCs. Some techniques utilize only "include "and "extend" relationship between use cases for extraction of FM constraints from use cases. Relying only on include and extend relationship for constraints extraction adversely affects FM construction process. If the relationships between use cases are not mentioned in source use case diagrams then extraction of constraints will not be possible. The resulting FM will not be complete. In other techniques variability need to be explicitly modeled in UML models for extraction of CTCs.

Activity diagram is only used by one of the existing technique for FM construction. Activity diagrams of individual use cases are used by existing approach that results in generation of low level features. For better extraction of constraints, FM construction using activity diagram that shows dependencies between use cases is required. Only one of the existing techniques [33] extract both CTCs without modeling variability in to UML models. In addition to that, it is also the only technique that is tested and evaluated on case studies. This approach is also compared to five other approaches.

# Chapter 3

# Proposed Solution

From state of the art, we have observed that most of the existing techniques either use used case diagrams or use case descriptions for FM construction. Existing techniques make use of "extend" and "include" relationships of use cases to deduce constraints between features. These techniques cannot extract constraints when "extend" and "include" relationships of use cases are not mentioned in the input. These techniques are also unable to extract constraint based on sequential dependencies. In FM, constraints avoid generation of invalid configurations. As constraints are vital part of FM so, identification of all the constraints during FM construction process is quite important. We have proposed an approach for better extraction of FM constraints. Our proposed approach does not rely on relationships of use cases for constraint extraction and also is able to extract sequential dependencies based constraints.

The sequential dependencies of use cases can be extracted using system activity diagram or preconditions and postconditions of use cases. The pre conditions of use cases can be written in natural language (NL) or object constraint language (OCL). The extraction of use case dependencies from pre conditions of use cases written in NL is quiet expensive as it requires deep NLP analysis. Pre conditions of use cases written in OCL are usually not available due to the fact that experts avoid writing pre conditions of use cases in OCL due to its unfamiliar syntax. The most convenient way for extraction of use case dependencies is to use system activity diagram. The use case dependencies can be easily extracted from activity diagram using flow information from one activity to another.

Our proposed technique uses activity diagram for the better extraction of constraints. Using activity diagram will improve constraints extraction process by extracting constraints even for such scenarios in which previously proposed techniques failed. Using activity diagram will also extract constraints based on sequential dependencies of use cases which are not covered by previously proposed techniques.

## 3.1 Activity Diagram Based Constraint Extraction

In activity diagram based constraints extraction, Activity diagram is used for the extraction of features as well as constraints. As we are using system level activity diagram that represent sequential dependencies of use cases. Each use case in activity diagram represents the feature of FM. Constraints are extracted by utilizing the sequential dependencies present between use cases in activity diagram. If use case B comes after use case A in flow than use case B is considered to be dependent on use case A.

An activity diagram of system and initial FM without constraints are given as input to proposed approach. Constraints are identified using input activity diagram. Features are identified from the input FM. After the identification of features and constraints, features in both are compared. If identified constraints exist between features that are present in FM, constraint will be added in FM otherwise not. In the end identified constraints are checked for anomalies. If anomalies are found in FM, constraints are filtered using proposed rules and constraints causing anomalies are removed from FM. Output of Proposed technique is FM with remaining constraints. Proposed solution diagram is shown in Figure 3.1.

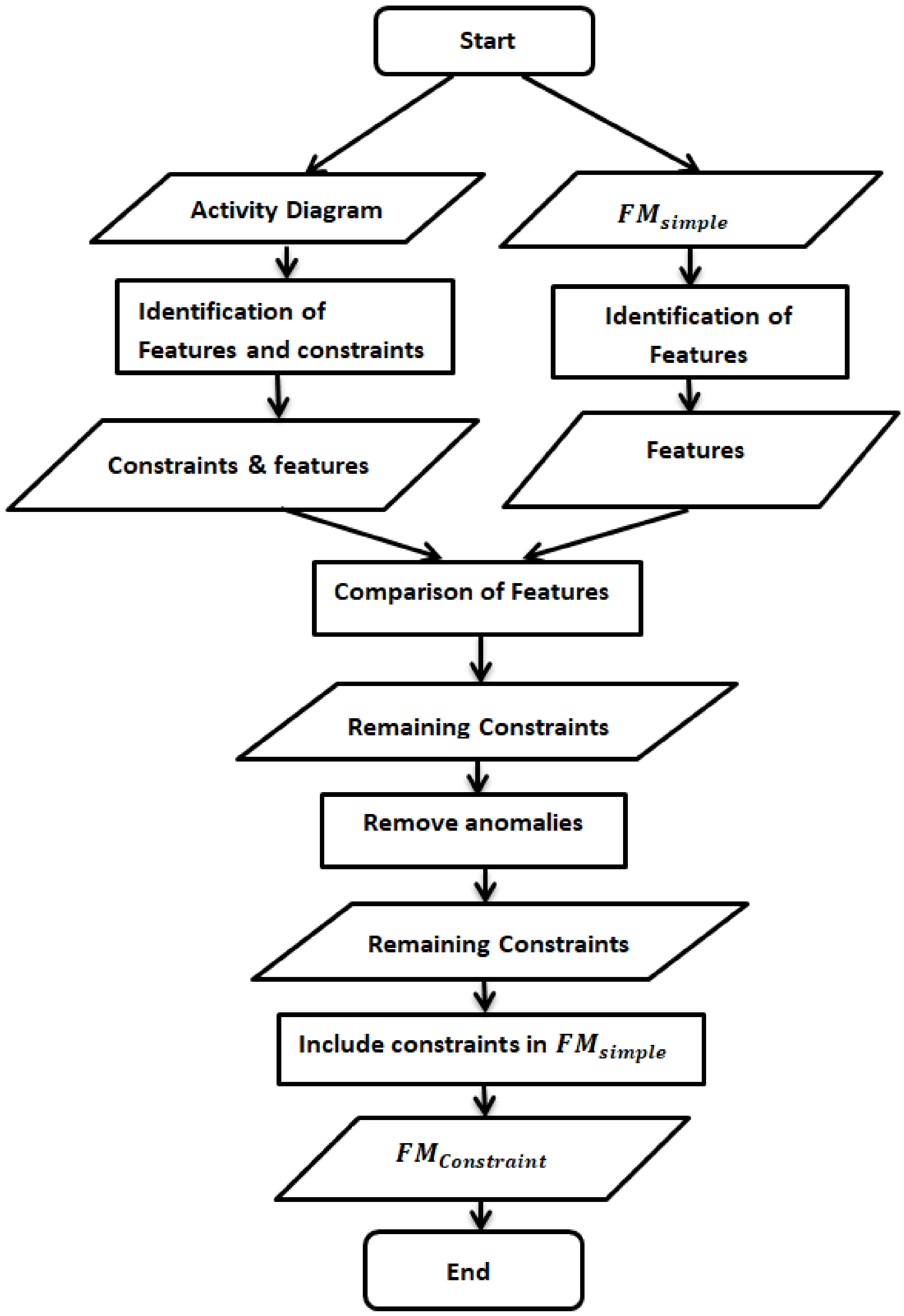


FIGURE 3.1: An illustration of proposed solution

## 3.2 Algorithm for Constraint Extraction

**Algorithm 1** Procedure-for-Constraints extraction

```
Require: $ACT = \{N, N_0 \in N, \delta N_\mathrm{i} \rightarrow N_\mathrm{j}\}, FM_\mathrm{simple} = \{F\}$
Ensure: $FM_\mathrm{Constraints} = \{F, C\}$
Declarations: $C = \phi, F_\mathrm{target} = \phi, F_\mathrm{source} = \phi$
1: procedure ACT2FM($ACT, FM_\mathrm{simple}$)
2:     for all $ein\delta N_\mathrm{i} \rightarrow N_\mathrm{j}$ do
3:         for all $NinACT$ do
4:             if $e_\mathrm{i}.N_\mathrm{j}isJoinNode$ then
5:                 $\mathrm{F_{source}} = \mathrm{F_{source}}$ U $\mathrm{N_i}$
6:                 for all $ein\delta N_\mathrm{i} \rightarrow N_\mathrm{j}$ do
7:                     if $e_\mathrm{i}.N_\mathrm{i}isJoinNode$ then
8:                         $\mathrm{F_{target}} = \mathrm{T_{target}}$ U $\mathrm{N_j}$
9:                     end if
10:                end for
11:            end if
12:            if $e_\mathrm{i}.N_\mathrm{i}isForkNode$ then
13:                $\mathrm{F_{target}} = \mathrm{F_{target}}$ U $\mathrm{N_j}$
14:                for all $ein\delta N_\mathrm{i} \rightarrow N_\mathrm{j}$ do
15:                    if $e_\mathrm{i}.N_\mathrm{j}isForkNode$ then
16:                        $\mathrm{F_{source}} = \mathrm{F_{source}}$ U $\mathrm{N_i}$
17:                    end if
18:                end for
19:            end if
20:            if $e_\mathrm{i}.N_\mathrm{i}isDecisionNode$ then
21:                $\mathrm{F_{target}} = \mathrm{F_{target}}$ U $\mathrm{N_j}$
22:                for all $ein\delta N_\mathrm{i} \rightarrow N_\mathrm{j}$ do
23:                    if $e_\mathrm{i}.N_\mathrm{j}isDecisionNode$ then
24:                        $\mathrm{F_{source}} = \mathrm{F_{source}}$ U $\mathrm{N_i}$
```

```
25:                 end if
26:             end for
27:         end if
28:         if $e_i.N_i isActionNode \&\& e_i.N_j isActionNode$ then
29:             $F_{source}$= $F_{source}$ U $N_i$
30:             $F_{target}$= $F_{target}$ U $N_j$
31:         end if
32:     end for
33: end for
34: for i=1 to Length of $F_{source}$ do
35:     if $F contains F_{sourcei} \&\& F contains F_{targeti}$ then
36:         C= C U $F_{targeti}$ Requires $F_{sourcei}$
37:     end if
38: end for
39: $FM_{constraints}$=$FM_{simple}$ U C
40: return $FM_{constraints}$
41: end procedure
```

Section 3.2 describes the proposed algorithm that is built to extract constraint of FM. Proposed algorithm takes activity diagram and initial FM without constraints as input. Proposed algorithm extracts sequential dependencies of use cases and maps them to constraint. Time complexity of proposed algorithm is T(n,m)= n+(m-a)+ma where n is number of nodes in activity diagram, m is number of edges in activity diagram and a is number of edges in activity diagram which are not between action- action nodes. For the best case scenario time complexity of algorithm is linear O(m) where m is number of edges in activity diagram. For worst case scenario time complexity of algorithm is quadratic O($m^2$).

After the extraction of constraints, proposed algorithm compares features that have constraint between them to features of Input FM. If match is found for both features in which constraint is present then this constraint is added in to the input

FM else not. In the end, FM is checked for anomalies wrong constraints causing anomalies in FM are removed.

Listed below steps are involved to achieve our goal:

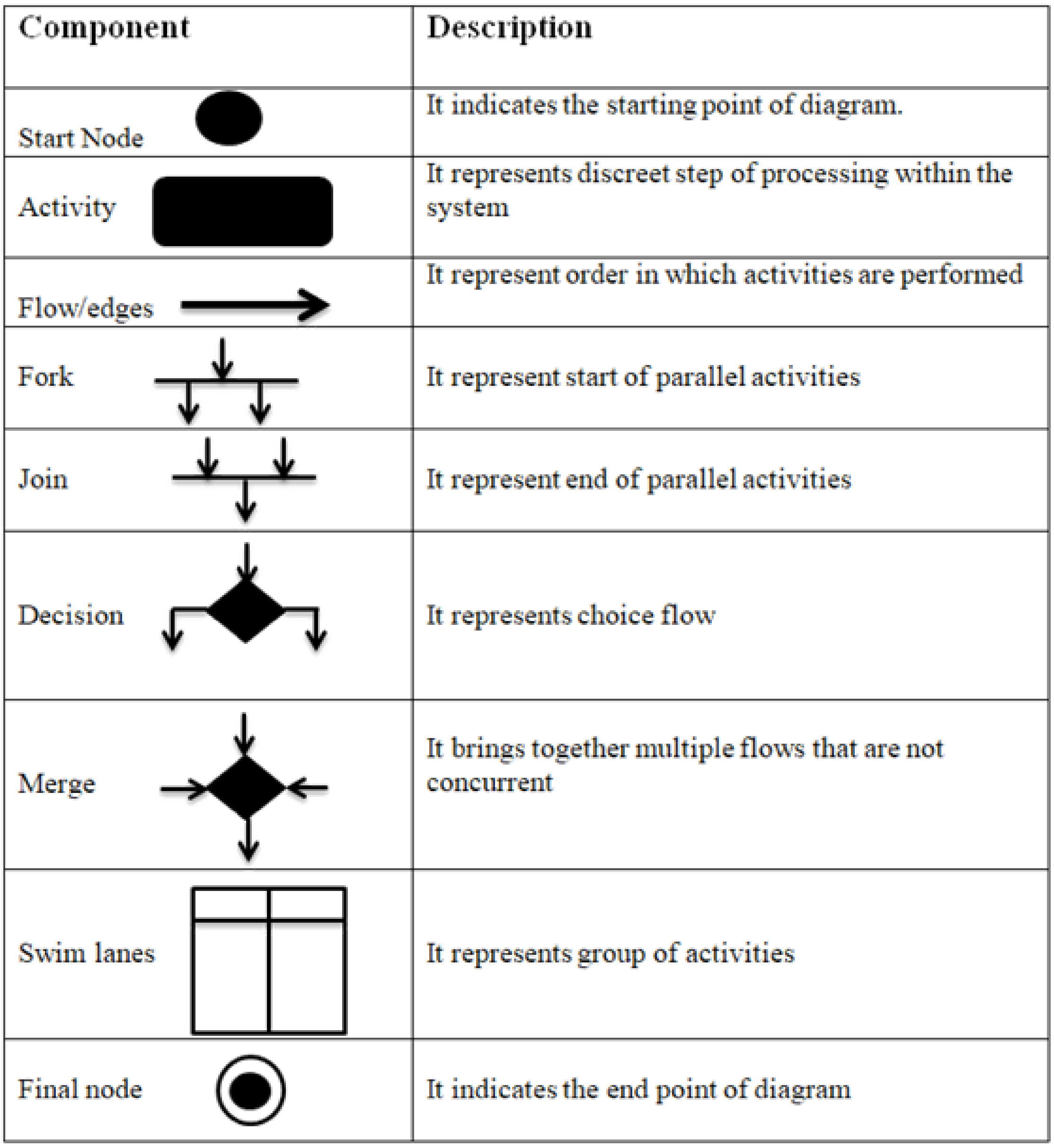
1. **Identification of features and constraints**
2. **Comparison of features extracted from activity diagram and FM features**
3. **Dealing with anomalies**
4. **Inclusion of constraint in FM**

### 3.2.1 Identification of Features and Constraints

In our approach, activity diagram is used for the extraction of features and constraints. Use cases in activity diagram are mapped to features and sequential dependencies of use cases are mapped to constraints between features.

#### 3.2.1.1 Activity Diagram

Activity diagram is a UML diagram. Activity diagram is used to represent stepwise flow of activities in a process. The flow of execution is represented as activity nodes connected by edges. Control flows are used to represent flow form one activity to other activity. In activity diagram control flow can be sequential, branched, or concurrent [40]. Different types of symbols are used in activity diagram for representation of all type of control flows. Sequential flow is represented by arrow. Fork node in activity diagram represents start of concurrent activities and join node represent end of concurrent activities. Branched flows are represented in activity diagram with the help of decision symbol [41]. Different activity diagram symbols and their descriptions are mentioned in Figure 3.2



| Component | Description |
|---|---|
| Start Node | It indicates the starting point of diagram. |
| Activity | It represents discreet step of processing within the system |
| Flow/edges | It represent order in which activities are performed |
| Fork | It represent start of parallel activities |
| Join | It represent end of parallel activities |
| Decision | It represents choice flow |
| Merge | It brings together multiple flows that are not concurrent |
| Swim lanes | It represents group of activities |
| Final node | It indicates the end point of diagram |

FIGURE 3.2: Activity diagram components

#### 3.2.1.2 Use Case Identification

A list of use cases is obtained by reading activity diagram. Each use case in activity diagram is considered as feature in FM and included in the use case list. For the extraction of use case list, first all the nodes of activity diagram are identified. Use cases in activity diagram are represented by activity nodes. For the extraction of use cases list, all the activity nodes are extracted from node list and included in the use case list.

For example in Figure 3.3 E-shop activity diagram is illustrated. The node list for E-shop activity diagram in Figure 3.3 is given below.

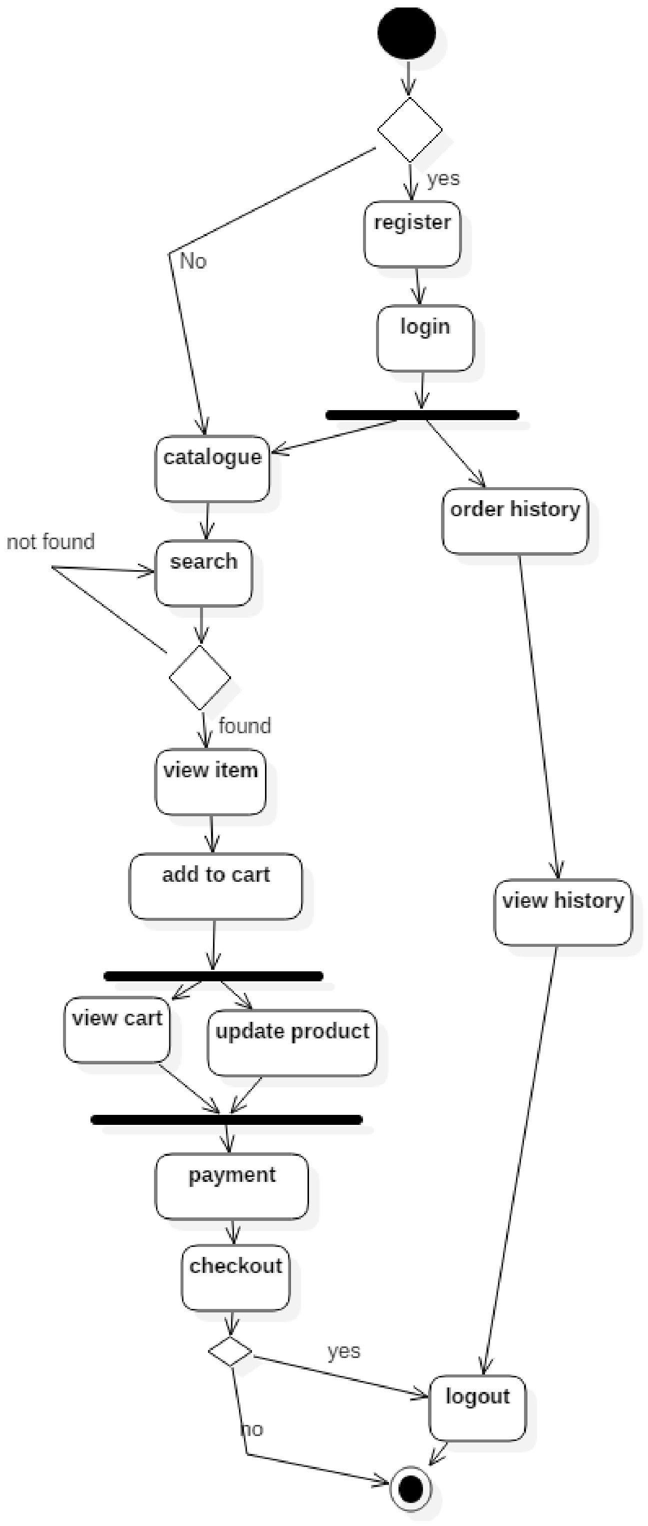


FIGURE 3.3: E-shop activity diagram

Node List: [start node,DecisionNode1, register, login, ForkNode1, search, DecisionNode2, view item, add to cart, ForkNode2, view cart, update product, JoinNode1, checkout, order history, view history, logout, payment, catalogue, DecisionNode3, final node]

The use case list for E-shop activity diagram in Figure 3.3 is given below.

use case list:[ register, login, search, view item, add to cart, view cart, update product, checkout, order history, view history, logout, payment, catalogue]

#### 3.2.1.3 Constraints Identification

For the constraints, sequential dependencies between use cases are used. For each action node in activity diagram, the action node that comes after that action node is considered to be dependent on it. By reading edges of activity diagram, source and target lists are generated. Source and target lists are generated in such a way that every action node **i** in target list depends on action node **i** in source list. In other words, each node **i** in target list has a *requires* constraint with node **i** in source list. By utilizing source and target lists, requires constraints are identified.

The source list and target list generated for activity diagram in Figure 3.3 are given below.

Source list: [register, login, search, search, view item,add to cart, add to cart, view cart, update product, payment, login, catalogue, checkout, view history, order history]

Target list: [login, order history, view item, search, add to cart, view cart, update product,payment, payment, checkout, catalogue, search,logout,logout, view history]

The activity nodes in source list and target list are considered as features in FM and each target feature **i** has a requires constraint with source feature **i**. After the generation of source list and target list, identified constraints are given in Table 3.1

Table 3.1: Identified Constraints

| Feature | Constraint | Feature |
|---|---|---|
| login | Requires | register |
| order history | Requires | login |
| view item | Requires | search |
| search | Requires | search |
| add to cart | Requires | view item |
| view cart | Requires | add to cart |
| update product | Requires | add to cart |
| payment | Requires | view cart |
| payment | Requires | update product |
| checkout | Requires | payment |
| catalogue | Requires | login |
| search | Requires | catalogue |
| logout | Requires | checkout |

### 3.2.2 Comparison of Features Extracted from Activity Diagram and FM Features

For the correct inclusion of the requires constraints in FM, features comparison is required. For the comparison, first the features from the input FM are extracted. Features from FM are extracted by reading the FM. Each feature that is read from the FM is included in the FM feature list.

To give an example E-shop feature model is given in Figure 3.4.

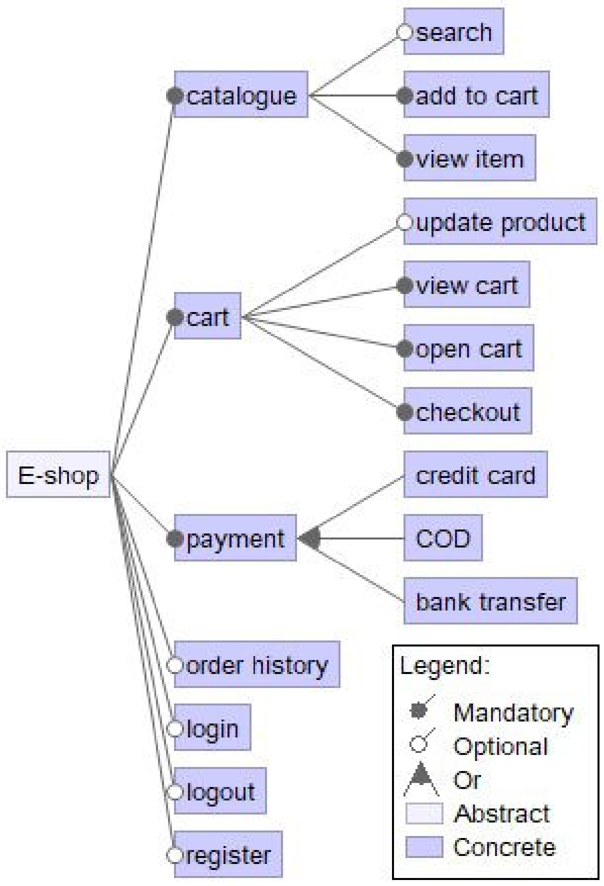


FIGURE 3.4: E-shop FM

The list of features obtained from the feature model in Figure 3.4 is given below

Features List: [search, add to cart, view item, update product, view cart, open cart, checkout, credit card, COD, bank transfer, order history, login, logout, register, E-shop, catalogue, cart, payment]

Before including requires constraints identified during previous step in FM, we need to make sure that identified constraint can be added in the FM. If the identified constraint is between such features that are not present in FM then inclusion of such constraint in FM will not be possible.

For the identification of such constraints that can be included in FM, for each constraint both the features between which the constraint is present are compared with FM feature list. After the comparison if both features exist in the list, constraint can be included otherwise not. In E-shop example the constraints identified by activity diagram given in Table 3.1 contains requires constraints including features view history. View history feature is not present in the Input FM given in Figure 3.4. As this feature is not present in FM, constraints that involve this feature cannot be included in the FM. Identified constraints are filtered by removing constraints that involve view history. The remaining constraints of E-shop FM after comparison are shown in Figure 3.5. The *requires* constraint in FM is represented by ⇒ symbol.

### 3.2.3 Dealing with Anomalies

Constraints are vital part of any FM. Constraint maps dependencies between features and helps in avoiding generation of invalid products. Improper usage of constraints in FM can also result in propagation of anomalies in FM. FM anomalies include: Void FM, redundancies, false optional features and dead feature [42]. Void feature model refers to a FM which does not generate any valid product. Redundancies in FM can be of two types: feature redundancy and constraint redundancy. Feature redundancy exists in FM if same feature is used more than once in FM. Redundant constraint refers to constraints which if included in FM does not affect number of generated products. False optional feature is such a feature in FM which is despite being part of every product of SPL is not marked mandatory. Dead feature refers to such feature which is not used in any of the product of SPL [43].

Using activity diagram for extraction of constraints between features can result in extraction of redundant or wrong constraints. To avoid the problem of anomaly that can occur due to wrong usage of CTCs. we filter the constraints extracted by using sequential dependencies of use cases from activity diagram. Constraints are filtered by using four restriction rules that avoid generation of wrong constraint which in return will help in avoiding occurrence of anomalies in FM.

The used restriction rules are listed below:

- Constraint should be between two optional features

- Constraint should not be between parent and child feature
- Self constraints are not allowed
- One feature in OR group should not be required by all other features in the group

Using above mentioned rules, extracted constraints are filtered to avoid generation of anomalies in FM. Above mentioned rules remove wrong constraints that cause anomalies in FM.

The list of constraints given in Table 3.1 is filtered using these rules. Constraints involving mandatory feature are removed from the list, constraints indicated between parent and child feature are removed, constraint between same features is removed and If a one feature of OR group is required by all other features of group then all these *requires* constraints are removed.

If the Constraints shown in Table 3.1 are added in E-shop FM without constraints causing anomalies, these constraints cause anomalies in FM as shown in Figure 3.5. Wrong extracted constraints results in 3 false optional features , one tautology caused by self-constraint and 7 redundant constraints in E-shop FM as shown in Figure 3.5 .

After removing constraints causing anomalies in FM, remaining constraints are added in to E-shop FM.

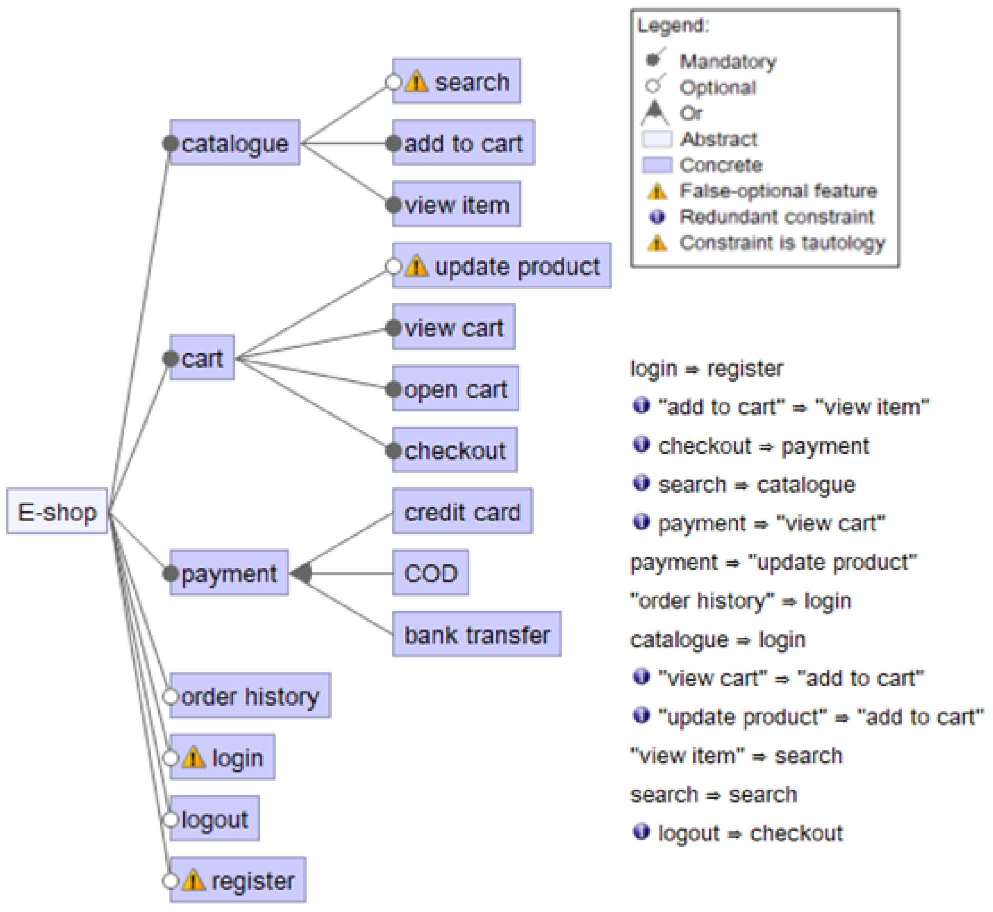


FIGURE 3.5: E-shop FM with anomalies

### 3.2.4 Inclusion of Constraints in FM

After the identification of the constraints that can be added in the FM. Next step is to include these constraints in to the FM. For the inclusion of each constraint, FM is read and both features between which the constraint is present are identified in the FM. After identification of these features in FM, requires constraint is added between these features. This process is repeated until all the constraints are added in the FM.

FM of E-shop after adding remaining constraints is given in Figure 3.6. At the end, output FM is generated that contain all the remaining constraints after removing anomalies. The output FM of E-shop example is shown in Figure 3.6.

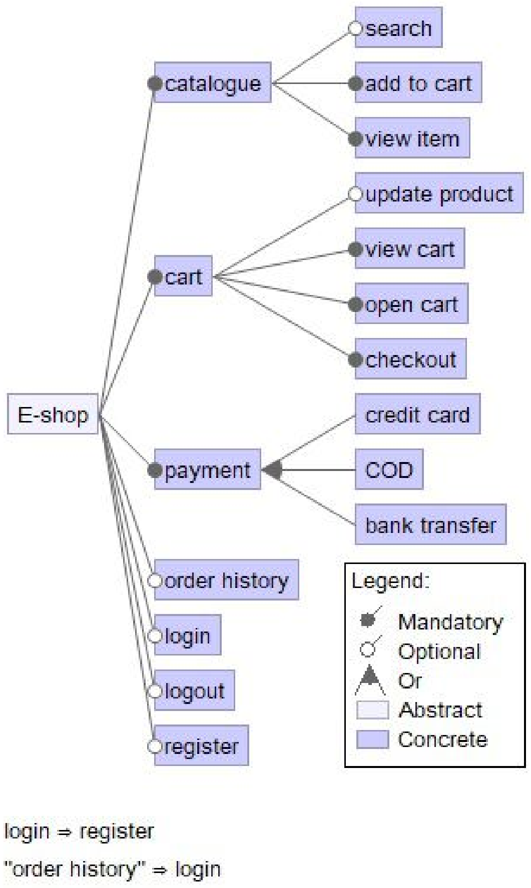


FIGURE 3.6: E-shop FM after removing anomalies

After removing anomalies two *requires* constraints are left in E-shop FM. The remaining two constraints are correct constraints and final generated FM is free from anomalies. The output FM is high quality FM as no anomaly is detected by tool for the output E-shop FM.

# Chapter 4

# Implementation

This chapter includes the implementation details of the proposed solution. Implementation of proposed solution is done using Eclipse and Java language. For FM, Eclipse plugin Feature IDE [44] is used. Our implementation comprise of four main components. First component automates the process of feature identification and displays features . Second component automates constraints identification using algorithm given in chapter 3 and displays identified constraints. Third component compares features identified by FM and features that are part of constraints and displays remaining constraints. Final component automates the process of constraints inclusion in FM and generates complete FM with constraints.

The user gives two Extensible Markup Language(XML) files as input to system. One input file is XML of FM without constraints and other is XML of activity diagram. XML of FM contains information about the features and relationship between features. XML of activity diagram contains information about the nodes and edges of activity diagram. The two input files are given as input to algorithm explained in previous chapter.

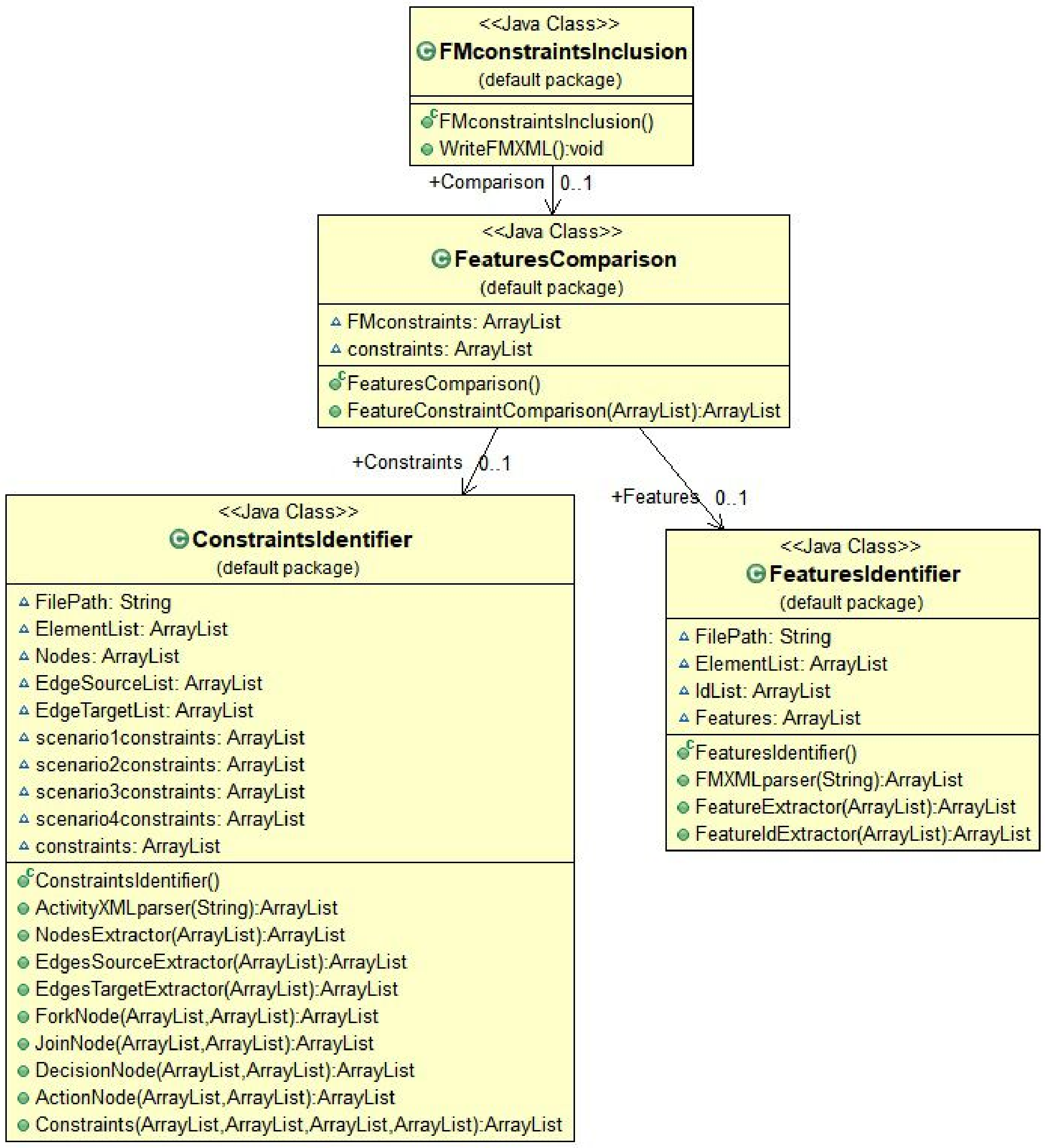
We have used star UML tool [45] for activity diagram construction and XML of activity diagram is exported using extension in star UML. Star UML is a UML modeling tool, it supports construction of almost all type of UML diagrams specified in UML 2.0. The extension used for exporting XML is XMI(XML Metadata

Interchange) extension. XMI is a standard format for representing UML diagrams. XMI is intended to help exchange of UML diagrams constructed in different tools using different languages. This extension generates XML file for activity diagram constructed in star UML and export it in XML format for further use, for feature model construction, Feature IDE plugin of eclipse is used. Feature IDE is a feature oriented software development tool available as a plugin for eclipse. When a feature model is constructed in feature IDE tool, XML file for that FM is automatically generated. We have generated XML for activity diagram using star UML and for feature model using feature IDE. These two generated XML files are given as input to our proposed algorithm.

The algorithm extracts features from FM XML file and generate list of features. Algorithm also extracts use cases from activity diagram XML file and generate list of use cases, Constraints identification is also done by algorithm by reading activity diagram XML. During identification of constraints source and target lists for constraints are maintained by algorithm. After constraints identification, algorithm compares features in feature list and features involved in constraints that are present in source and target list. After comparison, remaining constraints are added in the FM. For inclusion of constraints in FM, constraints are written in FM XML file by algorithm. In the end, algorithm generates complete FM with constraints

## 4.1 Implementation Details

This section includes the implementation details of our **Tool**. It describes all the classes and their methods used for each of the component.



<<Java Class>>
FMconstraintsInclusion
(default package)
FMconstraintsInclusion()
WriteFMXML():void
+Comparison 0..1
<<Java Class>>
FeaturesComparison
(default package)
FMconstraints: ArrayList
constraints: ArrayList
FeaturesComparison()
FeatureConstraintComparison(ArrayList):ArrayList
+Constraints 0..1
+Features 0..1
<<Java Class>>
ConstraintsIdentifier
(default package)
FilePath: String
ElementList: ArrayList
Nodes: ArrayList
EdgeSourceList: ArrayList
EdgeTargetList: ArrayList
scenario1constraints: ArrayList
scenario2constraints: ArrayList
scenario3constraints: ArrayList
scenario4constraints: ArrayList
constraints: ArrayList
ConstraintsIdentifier()
ActivityXMLparser(String):ArrayList
NodesExtractor(ArrayList):ArrayList
EdgesSourceExtractor(ArrayList):ArrayList
EdgesTargetExtractor(ArrayList):ArrayList
ForkNode(ArrayList,ArrayList):ArrayList
JoinNode(ArrayList,ArrayList):ArrayList
DecisionNode(ArrayList,ArrayList):ArrayList
ActionNode(ArrayList,ArrayList):ArrayList
Constraints(ArrayList,ArrayList,ArrayList,ArrayList):ArrayList
<<Java Class>>
FeaturesIdentifier
(default package)
FilePath: String
ElementList: ArrayList
IdList: ArrayList
Features: ArrayList
FeaturesIdentifier()
FMXMLparser(String):ArrayList
FeatureExtractor(ArrayList):ArrayList
FeatureIdExtractor(ArrayList):ArrayList

FIGURE 4.1: Class diagram of proposed solution

The class diagrams of the implementation is shown above in Figure 4.1. Figure 4.1 shows the classes and methods used for the implementation of proposed algorithm. Class diagrams also shows relationships between different classes. Each class and its methods are explained below:

### 4.1.1 FeaturesIdentifier

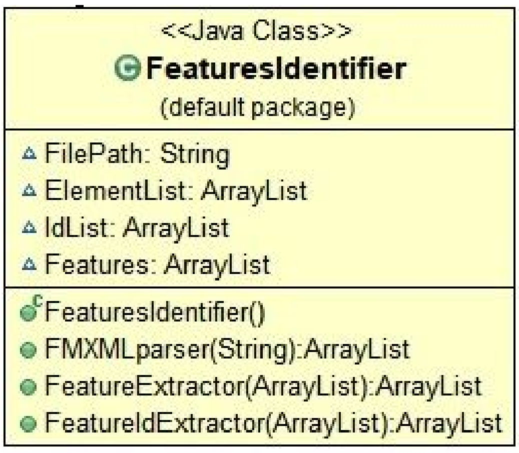
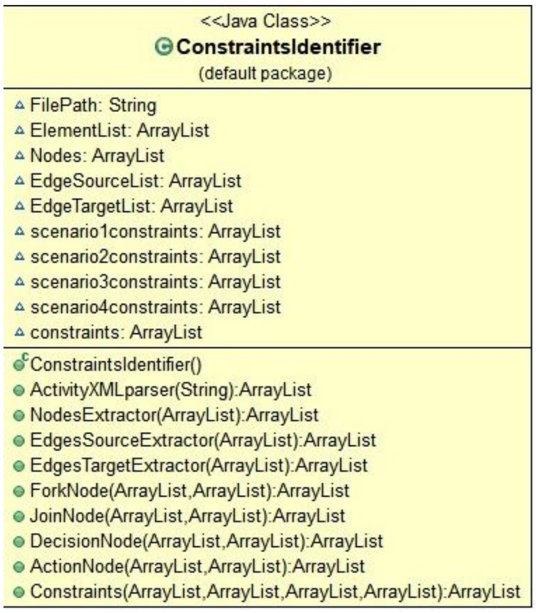


FIGURE 4.2: FeatureIdentifier class

The core functionality of this class is to read the XML file of input feature model and return the list of features extracted from it. Methods used in this class are shown in Figure 4.2. In this class, method ***FMXMLparser()*** is used to read the XML file of the input FM. After reading file this method extracts all the elements from the XML that starts with tag *feature*, *and*, *or* and *alt*. The extracted elements are stored in *ElementsList*. This method returns *ElementsList*. The method ***FeatureExtractor()*** is used for the identification of features. This method takes *ElementList* as an argument and for each element in *ElementList* it extracts name of element by reading Name tag. The names that are read by this method are stored in *features* list and are returned by this method. ***FeatureaIdExtractor()*** method is used for extracting id of each feature. This method takes *ElementList* as an argument and extract id of each element by reading tag ID and store it in*IdList*. This method returns *Idlist*.

### 4.1.2 ConstraintsIdentifier

The core functionality of this class is to read the XML file of activity diagram and return the constraints extracted from it. Variables and functions of **ConstraintsIdentifier** class is shown in Figure 4.3.



```
<<Java Class>>
ConstraintsIdentifier
(default package)

FilePath: String
ElementList: ArrayList
Nodes: ArrayList
EdgeSourceList: ArrayList
EdgeTargetList: ArrayList
scenario1constraints: ArrayList
scenario2constraints: ArrayList
scenario3constraints: ArrayList
scenario4constraints: ArrayList
constraints: ArrayList

ConstraintsIdentifier()
ActivityXMLparser(String):ArrayList
NodesExtractor(ArrayList):ArrayList
EdgesSourceExtractor(ArrayList):ArrayList
EdgesTargetExtractor(ArrayList):ArrayList
ForkNode(ArrayList,ArrayList):ArrayList
JoinNode(ArrayList,ArrayList):ArrayList
DecisionNode(ArrayList,ArrayList):ArrayList
ActionNode(ArrayList,ArrayList):ArrayList
Constraints(ArrayList,ArrayList,ArrayList,ArrayList):ArrayList
```

FIGURE 4.3: ConstraintsIdentifier class

In this class, method ***ActivityXMLparser()*** is used to read the XML of activity diagram. It extracts all the elements of XML that starts with *node* or *edge* tag and store them in *ElementList*. ***NodeExtractor()*** method is used to extract name of all the elements stored in *ElementList* by reading tag *name* of each element. It stores extracted names of elements in *Nodes* list. ***EdgeSourceExtractor()*** and ***EdgeTargetExtractor()*** methods are used to extract and store source Node and target node of edges present in activity diagram. ***EdgeSourceExtractor()*** method extracts source of all the edges present in activity diagram and store them in *EdgeSourceList.* ***EdgeTargetExtractor()*** method extracts target of all the edges present in activity diagram and store them in *EdgeTargetList.* Methods ForkNode(),JoinNode(),DecisionNode and ActionNodes takes *EdgeSourceList* and *EdgeTargetList* as argument. These methods extract constraints for each type of nodes of activity diagram and each method stores extracted constraints in List, After storing list these lists are returned by methods.

After the extraction of constraints by previous 4 methods, method ***Constraints()*** combine all 4 lists of constraints to generate final list of constraints. Method ***Constraints()*** returns final list of constraints.

### 4.1.3 FeaturesComparison

The functionality of this class is to compares the features list generated by class **featuresidentifier** and features including constraints present in constraint list generated by class **ConstraintsIdentifier**. Methods of this class are shown in Figure 4.4

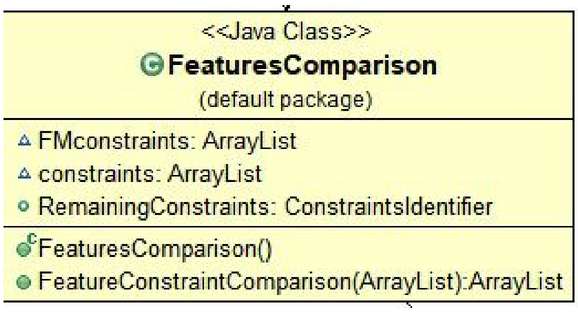


FIGURE 4.4: FeaturesComparison class

In this class, method ***FeatureConstraintComparison()*** is used to compare the features from features list and constraints list. For each constraint if both the features involved in constraint exist in features list then the feature is added in *RemainingConstraint* list. method ***FeatureConstraintComparison()*** return *RemainingConstraint* list.

### 4.1.4 FMConstraintsInclusion

The function of this class is to include extracted constraints in to FM. This class method are shown in Figure 4.5

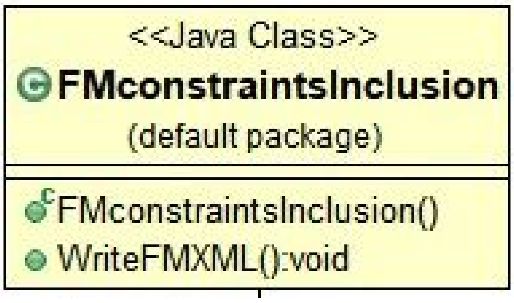
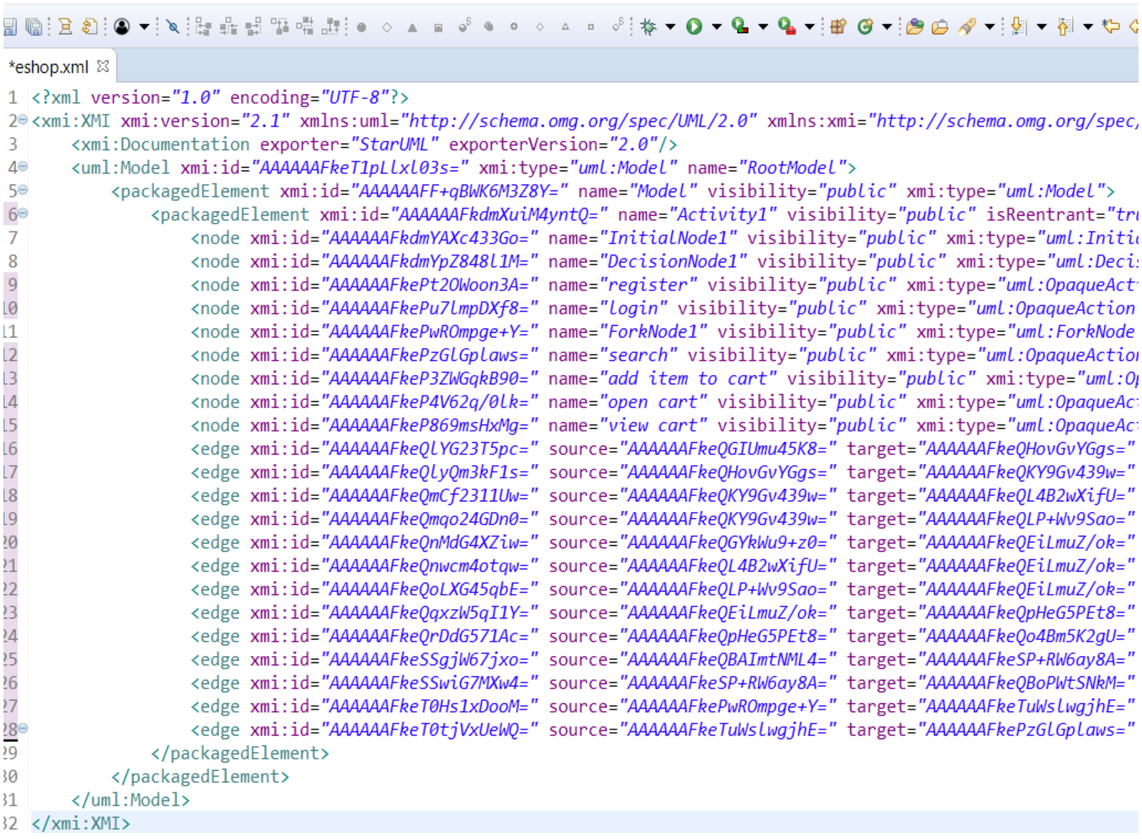


FIGURE 4.5: FMConstraintsInclusion class

In this class, method WriteFMXML() takes list of identified constraints as argument and writes identified constraints in to the FM XML file. When all the constraints are included in the XML file of FM, complete FM with constraints is generated using Feature IDE plugin of Eclipse tool.

## 4.2 Tool Usage

Implemented tool takes system activity diagram and FM without constraints as input. The activity diagram is constructed in starUML tool and exported in XML format. The XML of activity diagram is given as input to tool. The snapshot of activity diagram XML that is used by tool as input is shown in Figure 4.6

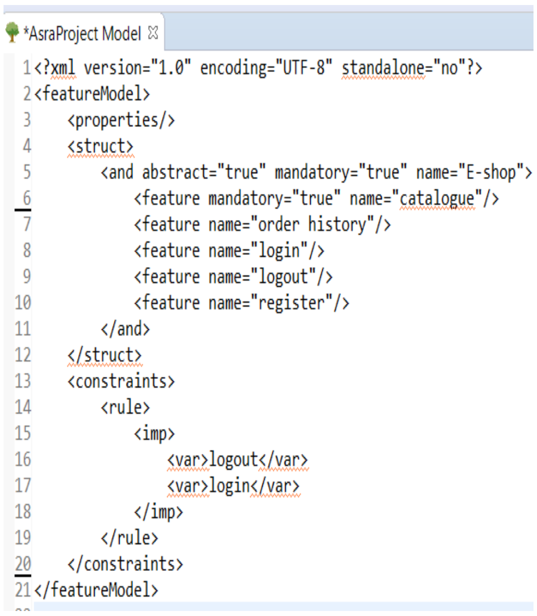


```
<?xml version="1.0" encoding="UTF-8"?>
<xmi:XMI xmi:version="2.1" xmlns:uml="http://schema.omg.org/spec/UML/2.0" xmlns:xmi="http://schema.omg.org/spec,
    <xmi:Documentation exporter="StarUML" exporterVersion="2.0"/>
    <uml:Model xmi:id="AAAAAAFkeT1pLLxl03s=" xmi:type="uml:Model" name="RootModel">
        <packagedElement xmi:id="AAAAAAFF+qBWK6M3Z8Y=" name="Model" visibility="public" xmi:type="uml:Model">
            <packagedElement xmi:id="AAAAAAFkdmXuiM4yntQ=" name="Activity1" visibility="public" isReentrant="tr
                <node xmi:id="AAAAAAFkdmYAXc433Go=" name="InitialNode1" visibility="public" xmi:type="uml:Initi
                <node xmi:id="AAAAAAFkdmYpZ848l1M=" name="DecisionNode1" visibility="public" xmi:type="uml:Deci
                <node xmi:id="AAAAAAFkePt2OWoon3A=" name="register" visibility="public" xmi:type="uml:OpaqueAct
                <node xmi:id="AAAAAAFkePu7lmpDXf8=" name="login" visibility="public" xmi:type="uml:OpaqueAction
                <node xmi:id="AAAAAAFkePwROmpge+Y=" name="ForkNode1" visibility="public" xmi:type="uml:ForkNode
                <node xmi:id="AAAAAAFkePzGlGplaws=" name="search" visibility="public" xmi:type="uml:OpaqueActio
                <node xmi:id="AAAAAAFkeP3ZWGqkB90=" name="add item to cart" visibility="public" xmi:type="uml:O
                <node xmi:id="AAAAAAFkeP4V62q/0lk=" name="open cart" visibility="public" xmi:type="uml:OpaqueAc
                <node xmi:id="AAAAAAFkeP869msHxMg=" name="view cart" visibility="public" xmi:type="uml:OpaqueAc
                <edge xmi:id="AAAAAAFkeQlYG23T5pc=" source="AAAAAAFkeQGIUmu45K8=" target="AAAAAAFkeQHovGvYGgs="
                <edge xmi:id="AAAAAAFkeQlyQm3kF1s=" source="AAAAAAFkeQHovGvYGgs=" target="AAAAAAFkeQKY9Gv439w="
                <edge xmi:id="AAAAAAFkeQmCf2311Uw=" source="AAAAAAFkeQKY9Gv439w=" target="AAAAAAFkeQL4B2wXifU="
                <edge xmi:id="AAAAAAFkeQmqo24GDn0=" source="AAAAAAFkeQKY9Gv439w=" target="AAAAAAFkeQLP+Wv9Sao="
                <edge xmi:id="AAAAAAFkeQnMdG4XZiw=" source="AAAAAAFkeQGYkWu9+z0=" target="AAAAAAFkeQEiLmuZ/ok="
                <edge xmi:id="AAAAAAFkeQnwcm4otqw=" source="AAAAAAFkeQL4B2wXifU=" target="AAAAAAFkeQEiLmuZ/ok="
                <edge xmi:id="AAAAAAFkeQoLXG45qbE=" source="AAAAAAFkeQLP+Wv9Sao=" target="AAAAAAFkeQEiLmuZ/ok="
                <edge xmi:id="AAAAAAFkeQqxzW5qI1Y=" source="AAAAAAFkeQEiLmuZ/ok=" target="AAAAAAFkeQpHeG5PEt8="
                <edge xmi:id="AAAAAAFkeQrDdG571Ac=" source="AAAAAAFkeQpHeG5PEt8=" target="AAAAAAFkeQo4Bm5K2gU="
                <edge xmi:id="AAAAAAFkeSSgjW67jxo=" source="AAAAAAFkeQBAImtNML4=" target="AAAAAAFkeSP+RW6ay8A="
                <edge xmi:id="AAAAAAFkeSSwiG7MXw4=" source="AAAAAAFkeSP+RW6ay8A=" target="AAAAAAFkeQBoPWtSNkM="
                <edge xmi:id="AAAAAAFkeT0Hs1xDooM=" source="AAAAAAFkePwROmpge+Y=" target="AAAAAAFkeTuWslwgjhE="
                <edge xmi:id="AAAAAAFkeT0tjVxUeWQ=" source="AAAAAAFkeTuWslwgjhE=" target="AAAAAAFkePzGlGplaws="
            </packagedElement>
        </packagedElement>
    </uml:Model>
</xmi:XMI>
```

FIGURE 4.6: Activity diagram XML file

The XML of activity diagram shown in Figure 4.6 is parsed by the tool. From XML file, tag *node* is used to identify names of use cases. Elements *source* and *target* of tag *edge* are used to extract dependencies between use cases. The input

FM is constructed in Eclipse using Feature IDE. The XML of constructed FM is generated by the Feature IDE. The XML of FM is shown in Figure 4.7



```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<featureModel>
    <properties/>
    <struct>
        <and abstract="true" mandatory="true" name="E-shop">
            <feature mandatory="true" name="catalogue"/>
            <feature name="order history"/>
            <feature name="login"/>
            <feature name="logout"/>
            <feature name="register"/>
        </and>
    </struct>
    <constraints>
        <rule>
            <imp>
                <var>logout</var>
                <var>login</var>
            </imp>
        </rule>
    </constraints>
</featureModel>
```

FIGURE 4.7: XML file of FM

The XML file of FM is parsed by tool and features are identified from it. Tool writes identified constraints in to the XML file of FM. Tag *imp* in XML of FM represents *requires* constraints. After adding all the constraints in XML of FM, the example output FM generated by tool is shown in 4.8

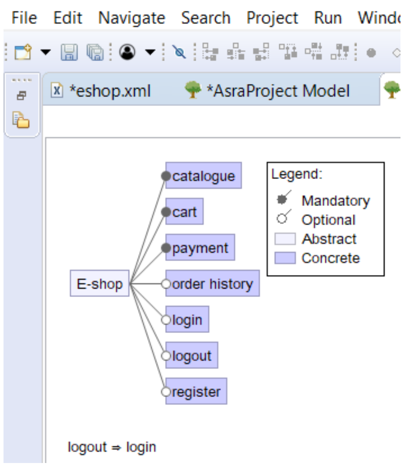


FIGURE 4.8: Output FM generated by tool

The output FM generated by tool represents *requires* constraint between features using ⇒ symbol.

# Chapter 5

# Results and Discussion

In this chapter, we have discussed results of experiments which we have performed on different case studies. By using system activity diagram, we extract constraints for FM. After extraction of constraints, constraints causing anomalies in FM are removed. The existing technique that we have used for comparison is use case based technique as it is considered as the strongest technique. Both activity based and use case based techniques are compared using FM analysis.

For evaluation of our technique, we have used three case studies from different domains. The constraints for FM of case studies are extracted using both techniques. Since our approach requires FM of SPL as input we have looked for reasonable size FM from on-line repository SPLOT [32].

FM of SPL and system activity diagram is given as input to tool described in Chapter 4. Tool identifies constraints between features of FM using activity diagram of the system.Extracted constraints are then checked for anomalies and constraints causing anomalies are removed using proposed rules. The remaining constraints after removing anomalies are added in to the input FM. Tool gives FM with remaining constraints as output.

## 5.1 Case Studies

We have used Social networks SPL, E-shop SPL and mobile media SPL as subject case studies for evaluating our approach. These case studies are developed by experts. Two case studies: E-shop and social network applications are available in SPLOT repository [32]. Third case study: mobile media is taken from [33]. A brief description of each case study is given below:

### 5.1.1 E-shop SPL

E-shop SPL, represents E-shop applications that are used for online shopping. E-shop SPL case study includes FM of E-shop that represents E-shop applications domain. FM of this case study has feature related to E-shop domain and relationships between them. This FM has total twenty two features.

### 5.1.2 Mobile Media SPL

Mobile media SPL, represents applications that manipulate photo, music, and video on mobile devices, such as mobile phones. Mobile media case study includes FM representing domain of mobile media. Mobile media FM contains features related to mobile media SPL and relationship between them. This FM has total twenty features.

### 5.1.3 Social Networks SPL

Social networking SPL, represents social networking applications. Social networks SPL case study includes FM of Social networks SPL. This FM represents social network applications domain. A social networking application FM has features related to social network domain and relationship between them. This FM has total thirty one features.

Table 5.1 shows properties of the FM's used in the case studies. In Table 5.1 NF represents number of Features, LF represent leaf features, MF represents mandatory features and OF represents optional feature.

Table 5.1: Properties of FMs

| **FM** | **NF** | **LF** | **MF** | **OF** |
|---|---|---|---|---|
| **E-shop** | 22 | 16 | 10 | 8 |
| **Mobile media** | 20 | 14 | 5 | 10 |
| **Social network** | 31 | 22 | 8 | 12 |

## 5.2 Evaluation and Comparison

Feature model of all three case studies are given as input to both proposed approach and use cased based approach. Constraints for each feature model are extracted by both approaches and included in the FM. Output of both approaches is FM with constraints.

### 5.2.1 E-shop SPL

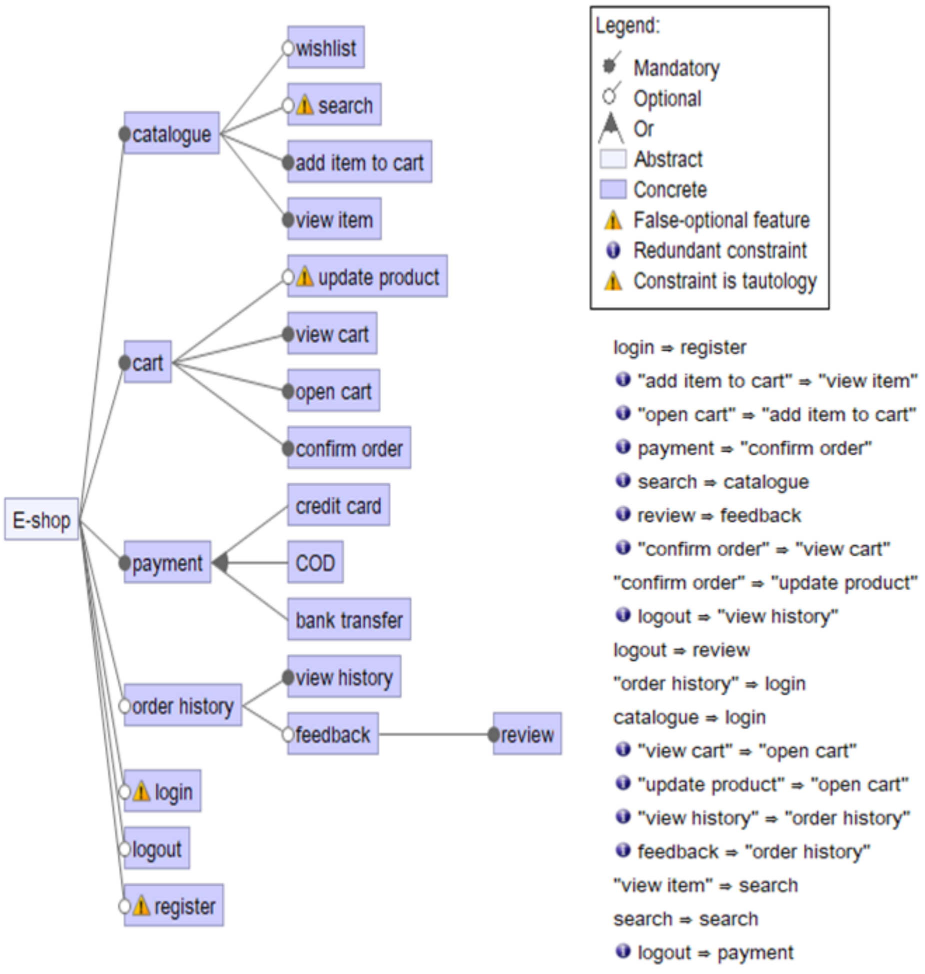
FM of E-shop is given as input to both approaches. Both approaches extract constraints of FM and output is complete E-shop FM with constraints.

#### 5.2.1.1 Activity Based Constraints Extraction

Activity diagram and SPL of E-shop without constraints is given as input. Constraints between features of FM are extracted by identifying sequential dependencies between features from activity diagram. After the identification of constraints from activity diagram, set of extracted constraints are compared with features of

FM. Constraints that involve features which are not present in FM are removed. E-shop FM after including extracted constraints is given in Figure 5.1.

The *requires* constraint in FM is represented by implies symbols for example A ⇒ B means feature A requires features B. *Excludes* constraint is represented by ⇔ symbol. After adding extracted constraints, there are four false optional features, twelve redundant constraints and one constraint tautology in E-shop FM as shown in Figure 5.1



login ⇒ register
● "add item to cart" ⇒ "view item"
● "open cart" ⇒ "add item to cart"
● payment ⇒ "confirm order"
● search ⇒ catalogue
● review ⇒ feedback
● "confirm order" ⇒ "view cart"
"confirm order" ⇒ "update product"
● logout ⇒ "view history"
logout ⇒ review
"order history" ⇒ login
catalogue ⇒ login
● "view cart" ⇒ "open cart"
● "update product" ⇒ "open cart"
● "view history" ⇒ "order history"
● feedback ⇒ "order history"
"view item" ⇒ search
search ⇒ search
● logout ⇒ payment

FIGURE 5.1: E-shop activity diagram based FM

After extraction of constraints from activity diagram, the constraints extracted from the activity diagram are checked for anomalies. For removing constraints causing anomalies four proposed rules are used. This step helps in dealing with dead features, false optional features and redundant constraints of FM.

Table 5.2 shows number of constraints removed by each rule. Twelve *requires* constraints that involves mandatory features are removed using rule 1. Four *requires* constraints are present between child and parent feature and are removed using rule 2. One self *requires* constraint search *requires* search is removed using rule 3.

TABLE 5.2: E-shop constraints removed by rules

| **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
|---|---|---|---|
| 12 | 4 | 1 | 0 |

The remaining constraints after removing anomalies are then added in to the FM to generate the complete FM. The output E-shop FM with constraints after removing anomalies is shown in Figure 5.2

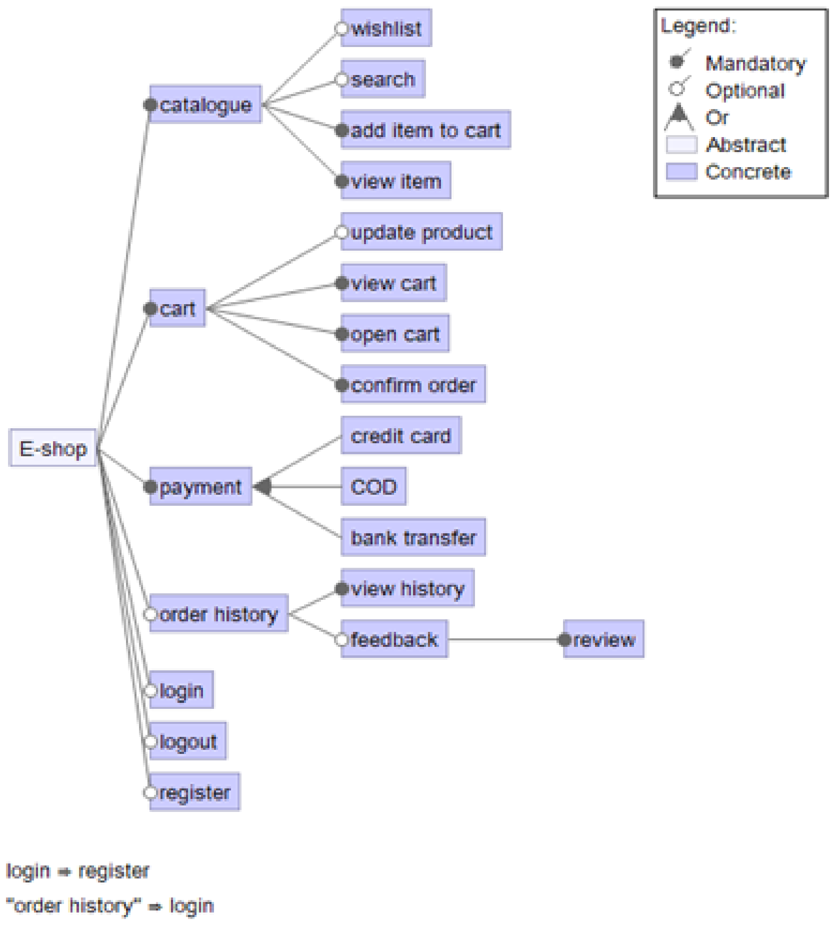


FIGURE 5.2: E-shop FM after removing anomalies

### 5.2.1.2 Use Case Based Constraints Extraction

In use case based approach [33] constraints between features of FM are identified by using use case relationships from use case diagrams. The input for this approach is E-shop FM and use case diagrams of products belonging to E-shop SPL. *Requires* constraints are identified through *include* relationship of use cases.

In E-shop SPL, there was two include relations: add item to cart *include* view item and confirm order *include* payment. Using these *include* relations two *requires* constraints are identified for E-shop FM. Two identified *requires* constraints are added in input E-shop FM. The output E-shop FM generated by use case based approach is shown in Figure 5.3.

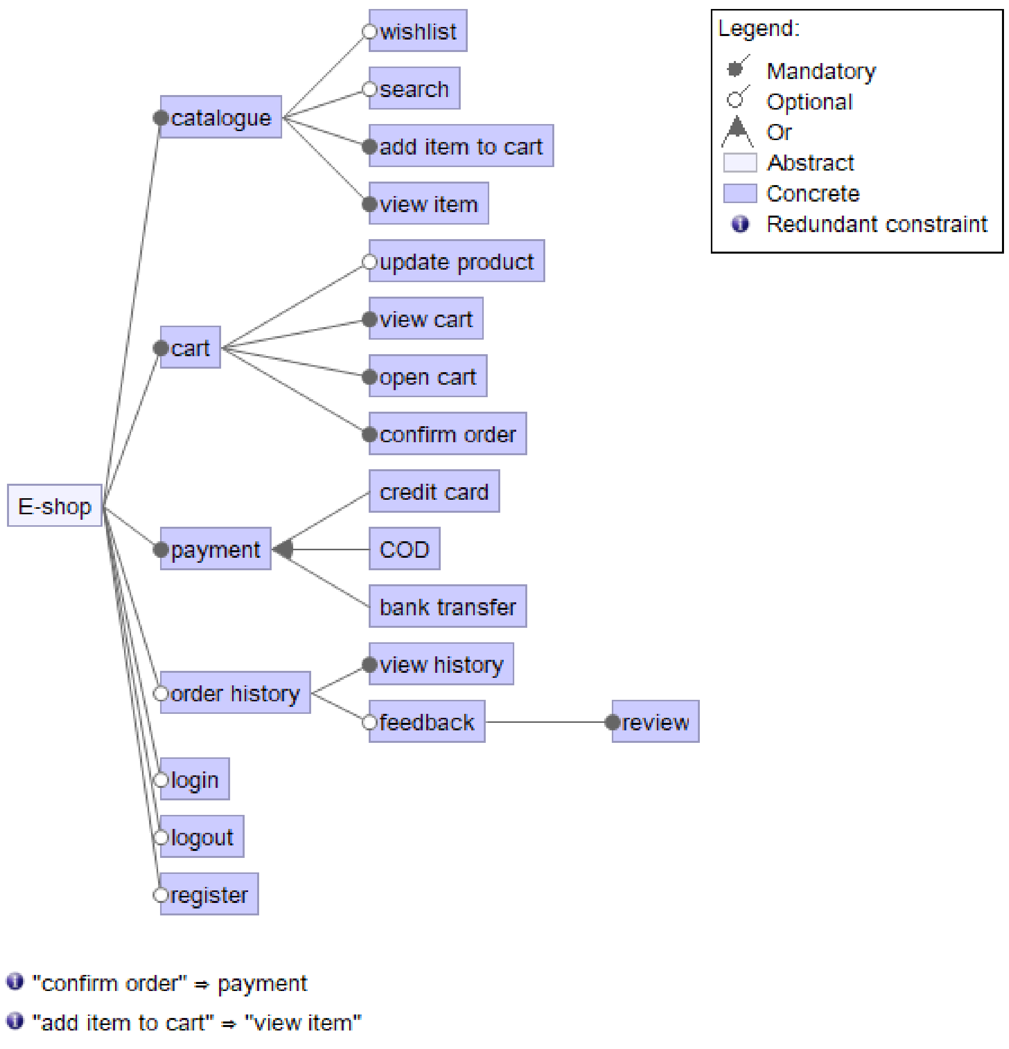


FIGURE 5.3: E-shop use case based FM

As evident from 5.3 both constraints extracted by use case based approach are indicated as redundant constraints by tool. For E-shop FM both techniques extracted two *requires* constraints. The constraints extracted by use case based approach are redundant whereas constraints extracted by activity based approach are good quality constraints.

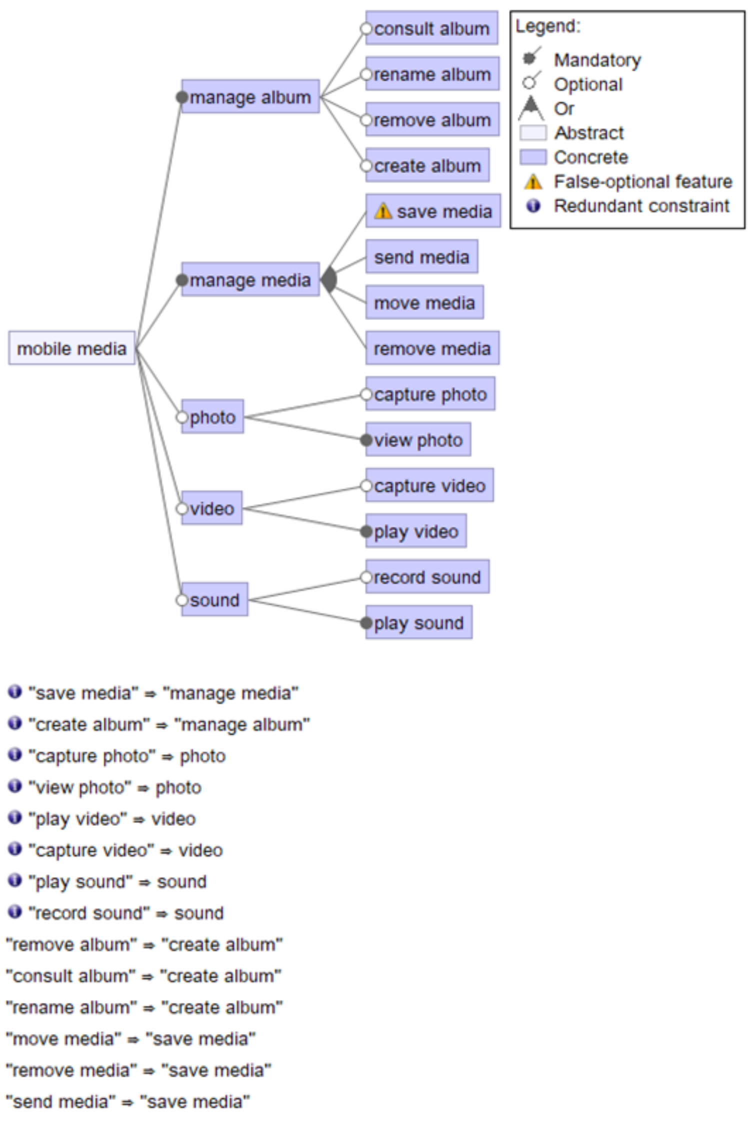
## 5.2.2 Mobile Media SPL

FM of mobile media is given as input to both approaches. Constraints are extracted for mobile media FM using both approaches and included in the input mobile media FM.

#### 5.2.2.1 Activity Based Constraints Extraction

System activity diagram of mobile media SPL and FM of that mobile media SPL is given as input. Constraints are extracted using system activity diagram. The extracted constraints are then compared with features. Remaining constraints are then added in to input mobile media FM. Mobile media after adding extracted constraints is given in Figure 5.4.

The *requires* constraint in FM is represented by implies symbols for example in FM A $\Rightarrow$ B means feature A *requires* features B. *Excludes* constraint is represented by $\Leftrightarrow$ symbol. Adding all the extracted constraints to FM results in generation of anomalies in FM. FM in Figure 5.4 has one false optional feature “save media” and eight redundant constraints that are caused due to addition of wrong constraints in mobile media FM.



● "save media" ⇒ "manage media"
● "create album" ⇒ "manage album"
● "capture photo" ⇒ photo
● "view photo" ⇒ photo
● "play video" ⇒ video
● "capture video" ⇒ video
● "play sound" ⇒ sound
● "record sound" ⇒ sound
"remove album" ⇒ "create album"
"consult album" ⇒ "create album"
"rename album" ⇒ "create album"
"move media" ⇒ "save media"
"remove media" ⇒ "save media"
"send media" ⇒ "save media"

FIGURE 5.4: Mobile media activity diagram based FM

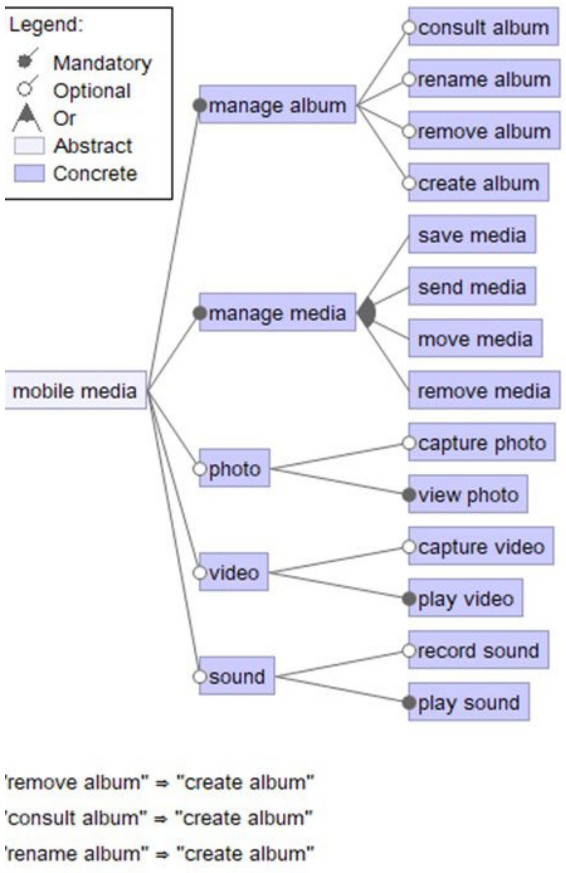
After the extraction of the constraints from the activity diagram, constraints are added in to the FM and checked for anomalies. If extracted features cause anomalies in the FM then these features are removed using four anomaly exclusion rules used by our technique. Table 5.3 shows the number of constraints removed by each rule.

TABLE 5.3: Mobile media constraints removed by rules

| **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
|---|---|---|---|
| 0 | 8 | 0 | 3 |

As shown in Table 5.3 eight *requires* constraints are present between parent and child feature and are removed by rule 2. In mobile media FM shown in 5.4 feature *save media* is required by other three features present in *or* group. These three *requires* constraints are removed using rule 4.

After removing constraints causing anomalies, remaining constraints are added in to the FM for generating complete FM. Figure 5.5 shows the complete Mobile media FM with constraints after removing anomalies.



'remove album" ⇒ "create album"
'consult album" ⇒ "create album"
'rename album" ⇒ "create album"

FIGURE 5.5: Mobile media FM after removing anomalies

#### 5.2.2.2 Use Case Based Constraints Extraction

In use case based approach [33] constraints between features of FM are identified by using use case relationships from use case diagrams. The input for this approach is mobile media FM and use case diagrams of products belonging to mobile media SPL. *Requires* constraints are identified through *include* relation of use cases.

In Mobile media SPL, there is one *include* relation: Manage media *include* manage album. Using identified *include* relation one *requires* constraint is identified for mobile media FM. The output Mobile media FM generated by use case based approach is shown in Figure 5.6.

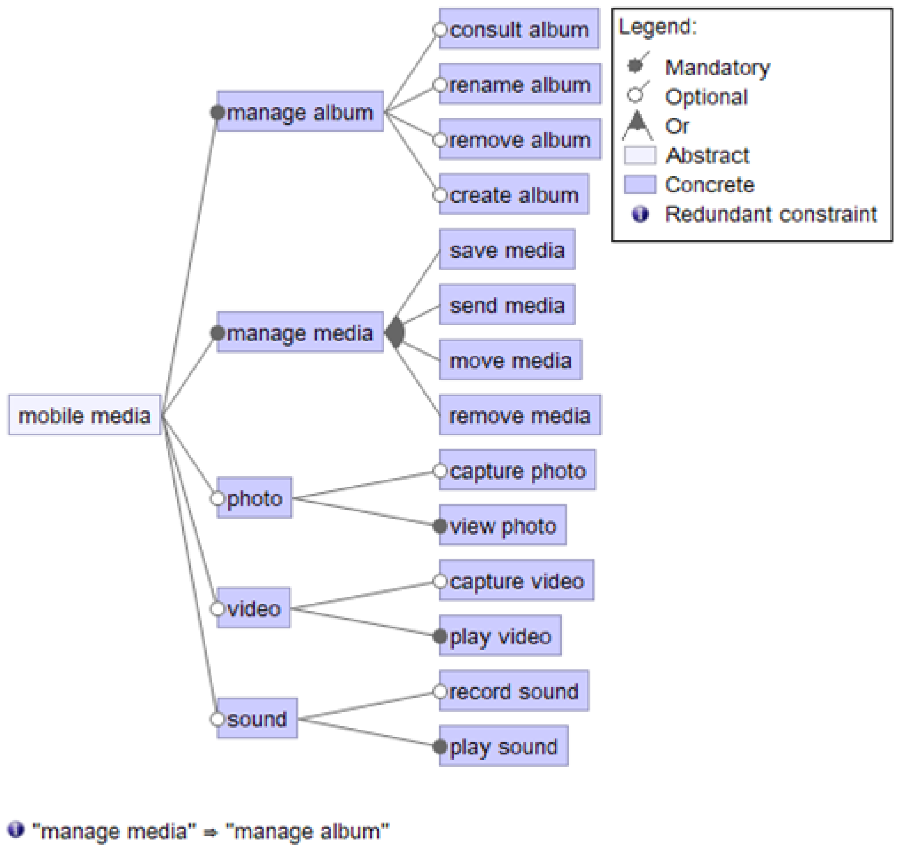


FIGURE 5.6: Mobile media use case based FM

Clear from FM shown in Figure 5.6 one *requires* constraint extracted by use case based approach is identified as redundant constraint by tool. For mobile media FM activity based approach is able to extract three *requires* constrains whereas only one *requires* constraint is extracted by use case based approach. In addition to that, constraint extracted by use case based approach is redundant constraint.

### 5.2.3 Social Networks SPL

FM of social network is given as input to both approaches. Both approaches identify constraints between features of FM. Identified constraints are then added in to the input FM. The output of both approaches is FM with constraints.

#### 5.2.3.1 Activity Based Constraints Extraction

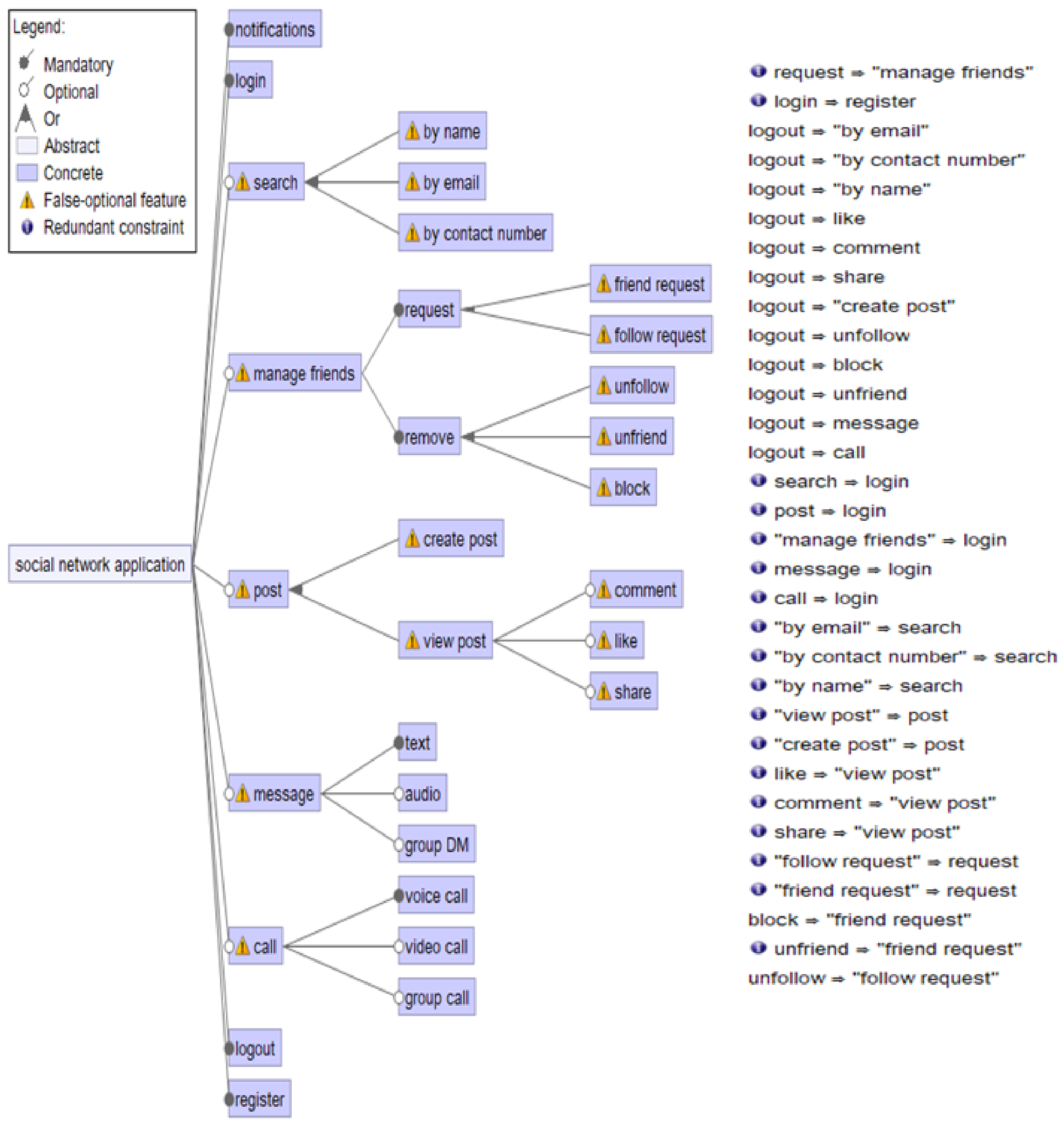
Activity diagram of the system and social network FM without constraints is given as input. Constraints extracted from the activity diagram for Social network FM are added in to the FM after comparison with FM features. Extracted constraints from activity diagram can result in identification of wrong constraints that can cause anomalies in FM. As clear from the Social network FM given in Figure 5.7.

The *requires* constraint in FM is represented by implies symbols for example A $\Rightarrow$ B means feature A requires features B. *Excludes* constraint is represented by $\Leftrightarrow$ symbol. Addition of extracted constraints without removing constraint causing anomalies in FM results in generation of false optional features and redundant constraints. There are eighteen false optional features and eighteen redundant constraints in social network FM shown in Figure 5.7 due to addition of all constraints.

After the extraction of constraints using activity diagram, constraints causing anomalies in the FM are removed using proposed rules. Table 5.4 shows the number of constraints removed by each rule.

Table 5.4: Social network constraints removed by rules

| **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
|---|---|---|---|
| 18 | 11 | 0 | 0 |



request ⇒ "manage friends"
login ⇒ register
logout ⇒ "by email"
logout ⇒ "by contact number"
logout ⇒ "by name"
logout ⇒ like
logout ⇒ comment
logout ⇒ share
logout ⇒ "create post"
logout ⇒ unfollow
logout ⇒ block
logout ⇒ unfriend
logout ⇒ message
logout ⇒ call
search ⇒ login
post ⇒ login
"manage friends" ⇒ login
message ⇒ login
call ⇒ login
"by email" ⇒ search
"by contact number" ⇒ search
"by name" ⇒ search
"view post" ⇒ post
"create post" ⇒ post
like ⇒ "view post"
comment ⇒ "view post"
share ⇒ "view post"
"follow request" ⇒ request
"friend request" ⇒ request
block ⇒ "friend request"
unfriend ⇒ "friend request"
unfollow ⇒ "follow request"

FIGURE 5.7: Social network activity diagram based FM

As shown in Table 5.4 eighteen *requires* constraints extracted for social network FM involves mandatory feature and are removed by rule 1. Eleven *requires* constraints are identified between parent and child feature and are removed using rule 2.

After removing anomalies, remaining constraints are added in to FM for generating complete FM. Figure 5.8 shows the complete Social network FM with constraints after removing anomalies.

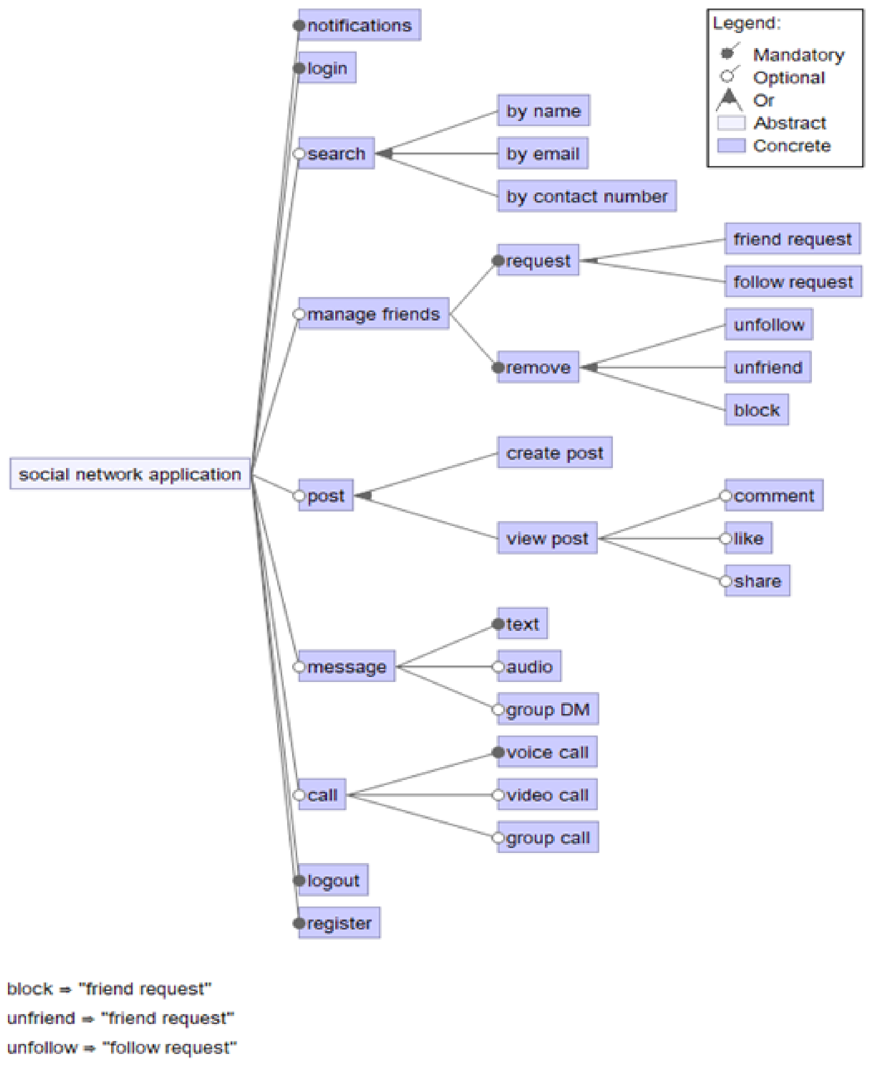


FIGURE 5.8: Social network FM after removing anomalies

#### 5.2.3.2 Use Case Based Cnstraints Extraction

In use case based approach [33] constraints between features of FM are identified by using use case relationships from use case diagram. The input for this approach is social network FM and use case diagrams of products belonging to social network SPL. *Requires* constraints are identified through *include* relationship of use cases. In social network use case diagrams, no *include* relation is present. As include relation is not present in use case diagrams of products, no constraint is extracted using use case based approach. The use case based approach failed to extract constraints for social network FM as *include* relation is not present in use case diagrams of the products of social network SPL.

## 5.3 Comparison

For evaluation purpose three case studies are used and constraints for each FM are extracted using proposed technique as well as technique proposed by Mefteh et al. [33]. Comparison of number of constraints extracted by proposed approach, existing approach and Benchmark FM is done. Benchmark FM are the FM that are downloaded from SPLOT repository. The benchmark FM without constranints are taken as input by the proposed approach. After the identification of constraints by the proposed approach, constraints extracted by proposed approach are compared with constraint present in benchmark FM to check weather the constraints present in benchmark FM are identified by proposed approach or not. Comparison of constraints identified by proposed approach, existing approach and benchmark FM for each case study are given in Figure 5.9.

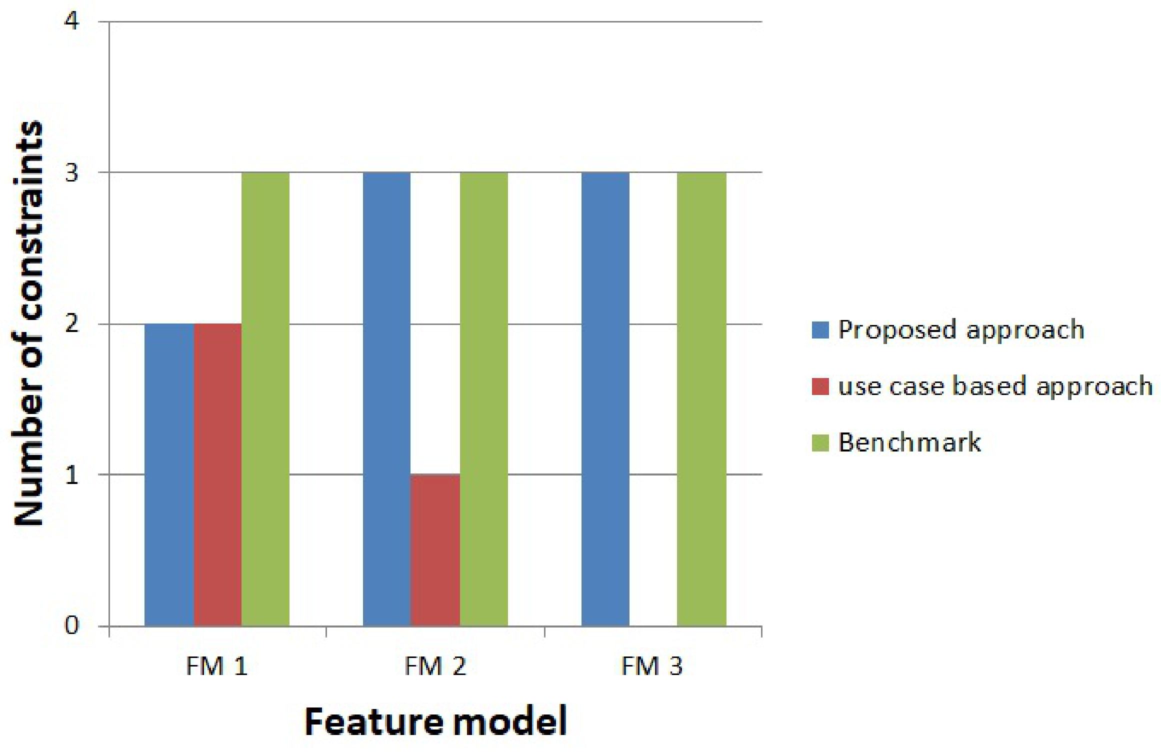


FIGURE 5.9: Number of constraints extracted for case studies

For E-shop case study two constraints are identified using proposed approach. Also using approach proposed by Mefteh et al. [33] two constraints are extracted for E-shop FM whereas three constraints are present in benchmark FM . Although the number of constraints extracted by both proposed and existing techniques are same for E-shop case study but the extracted constraints are different. As proposed approach handles constraints causing anomalies in FM, the constraint extracted by proposed approach when added in FM does not cause any anomaly which indicates that extracted constraints are correct constraints. The constraints identified by proposed approach are same as present in benchmark FM. Both constraints extracted by technique proposed by existing approach are redundant constraints as shown in Figure 5.3. In constraint **Confirm order *requires* payment** both features are mandatory features. As mandatory features are always part of every configuration, the *requires* constraint between two mandatory features give redundant information thats why constraint Confirm order *requires* payment is indicated as redundant constraint by tool as shown in Figure 5.3. In constraint **add to cart *requires* view item** again both features are mandatory and extracted constraints give redundant information. The constraint constraint add to cart *requires* view item is indicated as redundant constraint by tool as shown in Figure 5.3.

For Mobile media case study, three constraints are present in benchmark FM. Three constraints are extracted by proposed approach and all of the constraints are non-redundant and does not cause any anomaly in FM. All the constraints identified by proposed are same to the constraints present in benchmark FM. One constraint is extracted for mobile media case study by using approach proposed by Mefteh et al. [33]. The number of constraint extracted by use case based approach is less than number of constraints extracted by proposed approach; in addition to that constraint extracted by use case based approach is redundant. The constraint identified by use case based approach is **manage media *requires* manage album** as shown in Figure 5.6. Both features manage media and manage album are mandatory features. The *requires* constraint between two mandatory features gives redundant information so constraint manage media *requires* manage album is indicated as redundant constraint by tool as shown in Figure 5.6.

For social network case study three constraints are present in benchmark FM. Three constraints are extracted using proposed approach. Identified constraints are correct constraints and are same to the constraints present in benchmark

FM. No anomalies are identified by tool after adding these constraints. Existing approach failed to extract constraints for social network case study as only *Include* relationship of use cases is used by this approach for constraint extraction. In social network case study no *include* relation is present in use case diagrams so no *requires* constraint is extracted by existing approach.

The fact that existing approach is unable to extract constraints for certain scenarios indicates weakness approach. Proposed approach is able to extracted more number of constraints as well as can extract constraints for such scenarios in which existing approach failed. So over all proposed approach performed well in term of number of extracted constraints as well as quality of extracted constraints.

# Chapter 6

# Conclusion and Future Work

SPL is a promising approach for providing reusability of assets. Reusability helps in fast software production. One of the main steps of SPLE is managing variability of SPL. In literature different techniques have been proposed for managing variability of SPL but FM is one of the most commonly used techniques. FM plays an important role in managing variability and commonalities of SPL. In FM features are arranged in hierarchal structure using different relationships between features. In addition to FM relationships, CTCs are also defined between features. A lot of work has been done in the field of FM construction. Two main approaches for FM construction are: Top-down approach and bottom-up approach. Top-down approach is preferred for FM construction due to construction of FM in early stages of SDLC Different assets are used for construction of FM using top-down approach that includes, requirements, product descriptions, use case models and use case descriptions. Through the detailed literature survey and experimentation, we are able to answer our research questions described in Chapter 1. Following are the answers of our research questions:

RQ 1: Can we extract feature model constraints for certain scenarios not covered by existing techniques ?

The existing use case based FM construction approaches perform well in term of extraction of features and hierarchy. The main area that is not completely covered by proposed approaches is extraction of constraints. Most of the techniques in literature do not extracted constraints of FM. Techniques that do extract constraints are unable to extract all the constraints and also the extracted constraints are redundant in most of the cases. Already proposed techniques also do not filter constraints causing anomalies.To improve FM quality, we have used activity diagram of system for constraint extraction. Using activity diagram helps in extraction of sequential dependencies between features and constraints are extracted using these dependencies. By using activity diagram, proposed approach can extract constraints that depends on sequential dependencies of use cases which is not covered by existing techniques. In addition to that, proposed approach uses constraints avoid inclusion of wrong constraints in FM by filtering constraints causing anomalies.

RQ 2: Are constraints identified by proposed technique higher than existing techniques?

Evaluation and comparison of proposed approach is done using three case studies. The proposed approach is compared with one of the existing technique. Constraints are extracted by both techniques for all the case studies. The number of constraints identified by proposed approach, existing approach and constraints present in benchmark FM are compared. The results show that number of constraints extracted by proposed approach is higher than existing approach and are very close to number of constraints present in benchmark FM. The constraints extracted by existing approach include redundant constraints on the other hand no redundant constraints are generated by proposed approach.

## 6.1 Future Work

After the successful experimentation of proposed approach that is utilizing system activity diagram for FM construction. We can see that extraction of requires constraint for FM is improved by the proposed approach. As extraction of excludes constraint for FM is not covered by proposed approach. In future, Extraction of excludes constraints can be done along with requires constraint for further improvement in FM construction.

# Bibliography

[1] P. Clements and L. Northrop, *Software product lines: practices and patterns.* Addison-Wesley Reading, 2002, vol. 3.

[2] M. Kim, S. Park, V. Sugumaran, and H. Yang, "Managing requirements conflicts in software product lines: A goal and scenario based approach," *Data & Knowledge Engineering*, vol. 61, no. 3, pp. 417–432, 2007.

[3] S. A. Halim, D. N. A. Jawawi, and S. Deris, "Requirements identification and representation in software product line," in *Software Engineering Conference, 2009. APSEC'09. Asia-Pacific.* IEEE, 2009, pp. 340–346.

[4] G. Bockle, P. Clements, J. D. McGregor, D. Muthig, and K. Schmid, "Calculating roi for software product lines," *IEEE software*, vol. 21, no. 3, pp. 23–31, 2004.

[5] A. Metzger and K. Pohl, "Variability management in software product line engineering," in *Companion to the proceedings of the 29th International Conference on Software Engineering.* IEEE Computer Society, 2007, pp. 186–187.

[6] K. Lee, K. C. Kang, and J. Lee, "Concepts and guidelines of feature modeling for product line software engineering," in *International Conference on Software Reuse.* Springer, 2002, pp. 62–77.

[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 Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.

[8] E. Bagheri, T. D. Noia, D. Gasevic, and A. Ragone, "Formalizing interactive staged feature model configuration," *Journal of Software: Evolution and Process*, vol. 24, no. 4, pp. 375–400, 2012.

[9] D. Benavides, P. Trinidad, and A. Ruiz-Cortés, "Automated reasoning on feature models," in *International Conference on Advanced Information Systems Engineering.* Springer, 2005, pp. 491–503.

[10] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010.

[11] G. Zhang, H. Ye, and Y. Lin, "Feature model validation: A constraint propagation-based approach," in *Proceedings of the International Conference on Software Engineering Research and Practice (SERP).* Citeseer, 2011, p. 1.

[12] S. Bühne, K. Lauenroth, and K. Pohl, "Why is it not sufficient to model requirements variability with feature models," in *Proceedings of Workshop: Automotive Requirements Engineering.* Citeseer, 2004, pp. 5–12.

[13] D. Beuche, H. Papajewski, and W. Schröder-Preikschat, "Variability management with feature models," *Science of Computer Programming*, vol. 53, no. 3, pp. 333–352, 2004.

[14] M. Acher, P. Collet, P. Lahire, and R. France, "Composing feature models," in *International Conference on Software Language Engineering.* Springer, 2009, pp. 62–81.

[15] D. M. Le, H. Lee, K. C. Kang, and L. Keun, "Validating consistency between a feature model and its implementation," in *International Conference on Software Reuse.* Springer, 2013, pp. 1–16.

[16] E. Bagheri, F. Ensan, D. Gasevic, M. Boskovic *et al.*, "Modular feature models: Representation and configuration," *Journal of Research and Practice in Information Technology*, vol. 43, no. 2, p. 109, 2011.

[17] M. Riebisch, K. Böllert, D. Streitferdt, and I. Philippow, "Extending feature diagrams with uml multiplicities," in *6th World Conference on Integrated Design & Process Technology (IDPT2002)*, vol. 23, 2002.

[18] K. Czarnecki, S. Helsen, and U. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software process: Improvement and practice*, vol. 10, no. 1, pp. 7–29, 2005.

[19] K. Czarnecki, C. Hwan, P. Kim, and K. Kalleberg, "Feature models are views on ontologies," in *Software Product Line Conference, 2006 10th International.* IEEE, 2006, pp. 41–51.

[20] D. Batory, "Feature models, grammars, and propositional formulas," in *International Conference on Software Product Lines.* Springer, 2005, pp. 7–20.

[21] D. Batory, D. Benavides, and A. Ruiz-Cortes, "Automated analysis of feature models: challenges ahead," *Communications of the ACM*, vol. 49, no. 12, pp. 45–47, 2006.

[22] H. Casalánguida and J. E. Durán, "Automatic generation of feature models from uml requirement models," in *Proceedings of the 16th International Software Product Line Conference-Volume 2.* ACM, 2012, pp. 10–17.

[23] K. Berg, J. Bishop, and D. Muthig, "Tracing software product line variability: from problem to solution space," in *Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries.* South African Institute for Computer Scientists and Information Technologists, 2005, pp. 182–191.

[24] M. L. Griss, J. Favaro, and M. d'Alessandro, "Integrating feature modeling with the rseb," in *Software Reuse, 1998. Proceedings. Fifth International Conference on.* IEEE, 1998, pp. 76–85.

[25] T. Ziadi, L. Frias, M. A. A. da Silva, and M. Ziane, "Feature identification from the source code of product variants," in *Software Maintenance and*

*Reengineering (CSMR), 2012 16th European Conference on.* IEEE, 2012, pp. 417–422.

[26] C. Kästner, S. Trujillo, and S. Apel, "Visualizing software product line variabilities in source code." in *SPLC (2)*, 2008, pp. 303–312.

[27] T. von der Maßen and H. Lichter, "Deficiencies in feature models," in *workshop on software variability management for product derivation-towards tool support*, vol. 44, 2004.

[28] U. Lesta, I. Schaefer, and T. Winkelmann, "Detecting and explaining conflicts in attributed feature models," *arXiv preprint arXiv:1504.03483*, 2015.

[29] K. Czarnecki and C. H. P. Kim, "Cardinality-based feature modeling and constraints: A progress report," in *International Workshop on Software Factories.* ACM San Diego, California, USA, 2005, pp. 16–20.

[30] M. Kowal, S. Ananieva, and T. Thüm, "Explaining anomalies in feature models," in *ACM SIGPLAN Notices*, vol. 52, no. 3. ACM, 2016, pp. 132–143.

[31] A. Felfernig, D. F. Benavides Cuevas, J. Á. Galindo Duarte, and F. Reinfrank, "Towards anomaly explanation in feature models," in *ConfWS-2013: 15th International Configuration Workshop (2013), p 117-124.* CEUR-WS, 2013, pp. 117–124.

[32] M. Mendonca, M. Branco, and D. Cowan, "Splot: software product lines online tools," in *Proceedings of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications.* ACM, 2009, pp. 761–762.

[33] M. Mefteh, N. Bouassida, and H. Ben-Abdallah, "Implementation and evaluation of an approach for extracting feature models from documented uml use case diagrams," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing.* ACM, 2015, pp. 1602–1609.

[34] L. Chen, M. Ali Babar, and N. Ali, "Variability management in software product lines: a systematic review," in *Proceedings of the 13th International*

*Software Product Line Conference.* Carnegie Mellon University, 2009, pp. 81–90.

[35] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed, "On extracting feature models from sets of valid feature combinations," in *International Conference on Fundamental Approaches to Software Engineering.* Springer, 2013, pp. 53–67.

[36] A. Braganca and R. J. Machado, "Automating mappings between use case diagrams and feature models for software product lines," in *Software Product Line Conference, 2007. SPLC 2007. 11th International.* IEEE, 2007, pp. 3–12.

[37] B. Wang, W. Zhang, H. Zhao, Z. Jin, and H. Mei, "A use case based approach to feature models' construction," in *Requirements Engineering Conference, 2009. RE'09. 17th IEEE International.* IEEE, 2009, pp. 121–130.

[38] Y. Lin and X. Zhou, "A traceability approach to constructing feature model from use case models," in *Computer Science & Service System (CSSS), 2012 International Conference on.* IEEE, 2012, pp. 545–548.

[39] M. Mefteh, N. Bouassida, and H. Ben-Abdallah, "Feature model extraction from documented uml use case diagrams," *ADA USER*, vol. 35, no. 2, p. 107, 2014.

[40] H.-E. Eriksson and M. Penker, "Business modeling with uml," *New York*, pp. 1–12, 2000.

[41] R. Eshuis, "Symbolic model checking of uml activity diagrams," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 15, no. 1, pp. 1–38, 2006.

[42] P. Trinidad, D. Benavides, and A. Ruiz-Cortés, "A first step detecting inconsistencies in feature models," in *CAiSE Short Paper Proceedings*, 2006.

[43] L. Rincón, G. Giraldo, R. Mazo, C. Salinesi, and D. Diaz, "Method to identify corrections of defects on product line models," *Electronic Notes in Theoretical Computer Science*, vol. 314, pp. 61–81, 2015.

[44] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich, "Featureide: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014.

[45] "Staruml," 2011. [Online]. Available: http://staruml.sourceforge.net/en/