

FEATURE MODELING AND AUTOMATED ANALYSIS FOR AN EMBEDDED
SOFTWARE PRODUCT FAMILY

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

GÜLSEREN FEDAKAR GÖNÜL

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2013

Approval of the thesis:

**FEATURE MODELING AND AUTOMATED ANALYSIS FOR AN EMBEDDED
SOFTWARE PRODUCT FAMILY**

Submitted by **GÜLSEREN FEDAKAR GÖNÜL** in partial fulfillment of the requirements
for the degree of **Master of Science in Computer Engineering Department, Middle East
Technical University** by,

Prof. Dr. Canan Özgen
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Halit Oğuztüzün
Supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Ahmet Coşar
Computer Engineering Dept., METU

Assoc. Prof. Dr. Halit Oğuztüzün
Computer Engineering Dept., METU

Assoc. Prof. Dr. Pınar Karagöz
Computer Engineering Dept., METU

Assoc. Prof. Dr. Uluç Saranlı
Computer Engineering Dept., METU

Dr. Ahmet Serkan Karataş
K&K Teknoloji Ltd.

Date: 28.08.2013

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: Gülseren Fedakar Gönül

Signature :

ABSTRACT

FEATURE MODELING AND AUTOMATED ANALYSIS FOR AN EMBEDDED SOFTWARE PRODUCT FAMILY

Fedakar Gönül, Gülseren

M.Sc., Department of Computer Engineering

Supervisor: Assoc. Dr. Halit Oğuztüzün

August 2013, 132 pages

In the context of software product line engineering, feature models are used for modeling variability and commonality in product families. This thesis presents a basic feature model for a commercial television set product family. This work consists of three stages. First, a feature model is constructed, based on the analysis of the product family requirements. The constructed model is supplemented with a feature glossary. FeatureIDE is used as the model editor. Feature attributes, not supported by FeatureIDE, are represented in the basic feature model by using additional features. Second, the feature model in XML format is converted into the schema of the analysis tool, using a custom parser developed for this purpose. Third, the model is analyzed by well-known analysis operations. FAMA is used as the analysis tool. Performance results are obtained. Finally, lessons learned from the whole effort are discussed.

Keywords: Feature Modeling, Extended Feature Model, Software Product Lines, Commonality, Variability, Variability Management.

ÖZ

BİR GÖMÜLÜ YAZILIM AİLESİ İÇİN ÖZELLİK MODELLEMESİ VE OTOMATİK ANALİZİ

Fedakar Gönül, Gülseren

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi: Doç. Dr. Halit Oğuztüzün

Ağustos 2013, 132 sayfa

Yazılım üretim bandı mühendisliği bağlamında, özellik modelleri ürün ailelerinde değişkenlik ve ortaklıkları modellemek için kullanılmaktadır. Bu tez, ticari bir televizyon sistemi ürün ailesi için basit bir özellik modeli sunar. Bu çalışma üç aşamadan oluşur. İlk olarak, ürün ailesinin gereksinimlerinin analizine dayanan bir özellik modeli oluşturulur. Oluşturulan model, özellik sözlüğü ile desteklenir. Model editörü olarak FeatureIDE kullanılır. Ek özellikler kullanılarak, FeatureIDE’de desteklenmeyen öznitelikleri, basit özellik modelinde gösterilir. İkinci olarak, bu amaca özel geliştirilen bir çözümleyici ile XML formatındaki özellik modeli, analiz aracının şemasına dönüştürülür. Üçüncü olarak, model tanınmış analiz işlemleri üzerinden analiz edilir. Analiz aracı olarak FAMA kullanılır. Performans sonuçları elde edilir. Son olarak, bütün çalışma sonucunda öğrenilen deneyimler tartışılır.

Anahtar Kelimeler: Değişkenlik Modelleme, Genişletilmiş Özellik Modeli, Yazılım Ürün Hatları, Ortaklık, Değişkenlik, Değişkenlik Yönetimi

To My Parents

ACKNOWLEDGEMENTS

I would like to express my candid gratitude and appreciation to my supervisor, Assoc. Prof Dr. Halit Oğuztüzün, for his encouragement, guidance and support all throughout my graduate studies as well as during the preparation of this thesis. I am also thankful to Ahmet Serkan Karataş for his guidance.

I would like to thank to Arçelik Inc. especially to Özgür Say and Mustafa Uğuz for their continuous support during the implementation of the study. Without them, this work could not have been completed.

I also would like to thank to Jose Angel Galindo Duarte from the FAMA group for his support.

I am deeply grateful to my husband for his love, continuous support and encouragement which helped me in completion of this project. Also I would like to express my heartfelt thanks to my beloved parents for their blessings.

TABLE OF CONTENTS

ABSTRACT.....	iii
ÖZ	iv
ACKNOWLEDGEMENTS	vi
TABLE OF CONTENTS.....	vii
LIST OF TABLES.....	x
LIST OF FIGURES	xi
LIST OF ABBREVIATIONS.....	xiii
CHAPTERS	
1 INTRODUCTION.....	1
2 BACKGROUND AND RELATED WORK.....	5
2.1 Software Reusability and Software Product Line Engineering	5
2.1.1 Studies Related to Software Product Line Approach in Turkey.....	7
2.1.2 Software Product Line Processes	9
2.2 Software Product Line Management	10
2.2.1 Commonality and Variability in Software Product Lines	10
2.2.2 Variability Management.....	11
2.3 Feature Models and Feature Representation.....	11
2.3.1 Basic Feature Models	13
2.3.2 Cardinality Based Feature Models	14
2.3.3 Extended Feature Models	14
2.3.4 Feature Relationships and Dependencies	14
2.3.5 Feature Attributes	15
2.3.6 Feature Management	16

3	TOOL SUPPORT FOR MANAGEMENT AND AUTOMATED ANALYSIS OF FEATURE MODELS.....	17
3.1	FeatureIDE	17
3.2	FAMA	18
3.3	Other Tools.....	20
3.3.1	VARP.....	20
3.3.2	PureVariants.....	20
3.3.3	SPLOT	21
3.3.4	Gears	21
4	FEATURE MODELING OF NEWGENTV SPL.....	23
4.1	Introduction	23
4.2	Domain Analysis	23
4.3	Modeling in FeatureIDE.....	26
4.3.1	Feature Identification	27
4.3.2	Feature Model Organization	29
4.3.3	Feature Relationships and Dependencies.....	30
4.3.4	Defining Feature Attributes	32
4.3.5	Model Management in FeatureIDE.....	34
5	TRANSFORMING THE NEWGENTV FEATURE MODEL TO FAMA.....	43
5.1	Relationships	46
5.2	Constraints.....	50
5.3	Attributes.....	53
6	ANALYZING THE NEWGENTV FEATURE MODEL	55
6.1	FAMA Models	56
6.2	Input File Format.....	58
6.3	Using FAMA Framework	60
6.3.1	Working with Models	60

6.4 FAMA Operations	61
6.4.1 Validation	61
6.4.2 Products	61
6.4.3 Number of Products.....	67
6.4.4 Valid Product.....	72
6.4.5 Invalid Product Explanation	78
6.4.6 Valid Configuration.....	81
6.4.7 Error Detection	82
6.4.8 Error Explanations.....	84
6.4.9 Core Features.....	86
6.4.10 Variant Features.....	90
6.4.11 Commonality	91
6.4.12 Variability.....	91
6.4.13 Other FAMA Operations	91
7 CONCLUSION	93
REFERENCES	97
APPENDICES	
A PARSER FLOWCHARTS.....	103
B PARSER PSEUDOCODES.....	107
C FEATURE GLOSSARY	115

LIST OF TABLES

TABLES

Table 2.1 Feature Diagram Relationships	13
Table 6.1 Question Trader.....	56
Table 6.2 Products Analysis	63
Table 6.3 Number of Products Analysis	68
Table 6.4 Valid Product Analysis.....	74
Table 6.5 Core Features Analysis.....	87

LIST OF FIGURES

FIGURES

Figure 1.1 Overall Process of the Case Study	2
Figure 2.1 Common Process Structure of Software Product Line Approach [22]	9
Figure 2.2 A part of the NewGenTV feature model that shows parental relationships and constraints	15
Figure 3.1 FAMA Architecture [58]	19
Figure 3.2 Feature Diagram of the FAMA Framework	19
Figure 4.1 (A) First version of the NewGenTV feature model	25
Figure 4.2 (B) First version of the NewGenTV feature model	25
Figure 4.3 (C) First version of the NewGenTV feature model	25
Figure 4.4 (D) First version of the NewGenTV feature model	25
Figure 4.5 Constraints of first version of the NewGenTV feature model	26
Figure 4.6 A part of the NewGenTV feature model that shows some capability features	27
Figure 4.7 A part of the NewGenTV feature model that shows some operating environment features	28
Figure 4.8 A part of the NewGenTV feature model that shows some domain technology features	28
Figure 4.9 A part of the NewGenTV feature model that shows some domain technology features	29
Figure 4.10 A part of the NewGenTV feature model that shows some implementation features	29
Figure 4.11 Some Conceptual Groups of Feature Model	30
Figure 4.12 Local Dependencies in Feature Model	31
Figure 4.13 Constraint Editor	32
Figure 4.14 Attributed Model for AnalogFrontEnd	33

Figure 4.15 Attributed Model for PanelSize	33
Figure 4.16 Attributed Model for ranged values	34
Figure 4.17 Deleting features including subfeatures in FeatureIDE	35
Figure 4.18 Deleting features in FeatureIDE	36
Figure 4.19 Error of Deleting features in FeatureIDE.....	36
Figure 4.20 Feature Model Edits in FeatureIDE	37
Figure 4.21 (A) Highlighted Constraints.....	38
Figure 4.22 (B) Highlighted Constraints.....	38
Figure 4.23 Contradictions in FeatureIDE	38
Figure 4.24 Simple Configuration Editor of FeatureIDE	40
Figure 4.25 Advanced Configuration Editor of FeatureIDE	41
Figure 5.1 General Parser Process.....	44
Figure 5.2 General Representation of Transformation.....	46
Figure 5.3 Mapping of Mandatory and Optional Relationships	47
Figure 5.4 Mapping of Or-Relationships.....	48
Figure 5.5 Mapping of Alternative-Relationships.....	49
Figure 5.6 Mapping of Complex Relationships	50
Figure 5.7 Mapping of Constraints.....	51
Figure 5.8 Mapping of Complex Constraints	52
Figure 6.1 Double values in FeatureIDE.....	57
Figure 6.2 Converted double values in FeatureIDE	58
Figure 6.3 XML Schema [56]	58
Figure 8.1 Flowchart of Parser Related to Relationships Area	103
Figure 8.2 Flowchart of Parser Related to Constraints Area.....	104
Figure 8.3 Flowchart of Parser Related to Attributes Area	105

LIST OF ABBREVIATIONS

SPL	Software Product Line
SPLE	Software Product Line Engineering
FODA	Feature Oriented Domain Analysis
FORM	Feature Oriented Reuse Method
FAMA	FeAture Model Analyser
FOSD	Feature Oriented Software Development
VARP	Variability Analysis and Resolution Plugin
IDE	Integrated Development Environment
BDD	Binary Decision Diagram
SAT	Satisfiability
CSP	Constraint Satisfaction Problem
AFM	Attributed Feature Model
EFM	Extended Feature Model
DOM	Document Object Model
API	Application Programming Interface

CHAPTER 1

INTRODUCTION

The way that software products are produced has changed significantly with the time. Nowadays, there is a need to develop multiple, similar software products instead of just a single individual product. The solution to this is to increase software reuse with the Software Product Line Engineering (SPLE). SPLE models inspect commonality and variability by detecting what is common and what differs between products in the same family.

The process of analyzing commonality for software which have common basic functions besides having different capabilities/features, reduces the work required for the software development and testing thus reduces the cost while it increases consistency and quality of the software. Reusability of software is an important issue which has studied in the field of software engineering for a long time. Software Product Line is one of the approaches providing the highest level of software reusability by extracting multiple systems which have similar features on a common source code base. Software Product Line approach promises great benefits in terms of quality, productivity and software development [1].

The basis of software product line which aims to increase software reusability is based on feature modeling studies. Feature modeling study was proposed as a part of the Feature Oriented Domain Analysis (FODA) [2]. FODA defines important and distinctive system characteristics which influence user experience, as features. The purpose of feature modeling work is defining common and variable features of software which can be produced as the output of a software product line [3]. The output of feature modeling work is usually a chart which is a feature model that defines cognitive grouping of features, relationships and dependencies between features. In that model a family of products is represented as a set of features. It expresses the high-level requirements. Different kinds of features (mandatory, alternative, optional, etc.) in the domain and relationships between them are identified via feature model. After defining the software product line features and implementation in modeling phase, a subset of these features are chosen according to the need, taking into account of feature constraints and dependencies, then new software which has specific features is obtained [4].

The automated analysis of feature models is about extracting information from feature models using computer-aided automatic reasoning mechanisms. In this study, variability and commonality were inspected in an example software product line, using feature models in the context of requirement engineering, as they enable automated analysis for verifying

product line consistency and generate graphical components which help the users through the process. With this study, along with the modeling variability in software product lines which have a proactive approach, enhancement of reusability in this context is increased as well. Smart TV product line of Arçelik [65] which has a large-scale real-world system was used as case study. We named this product line NewGenTV product line in this study. The benefits of feature model are utilized using a GUI based feature modeling tool named FeatureIDE. In that tool, we used a method to define feature attributes as features. Moreover cross-tree relationships and complex constraints were defined. In order to do automated analysis, we used FAMA as it integrates some of the most commonly used solvers proposed in the literature. But before that a translation was needed as FAMA only accepts models which are in the AFM format for the attributed feature model. Therefore we developed a parser to translate this model which is in the XML format to the suitable FAMA format for the automated analyses. By modeling and analyzing in that way, we utilized both benefits of feature model, modeling tool, and model analyzer. General flow of these processes can be seen in the Figure 1.1.

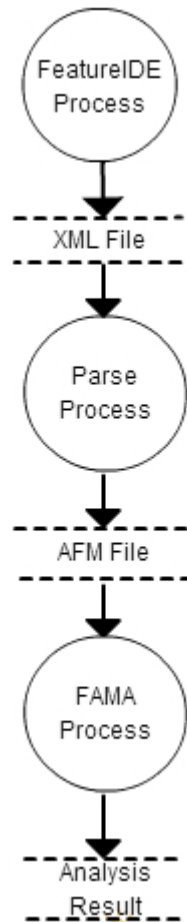


Figure 1.1: Overall Process of the Case Study

This thesis document is divided into seven chapters. Following this introductory part, there is chapter two which presents background information on the Software Product Line Engineering and Feature Modeling. Chapter three describes tool supports for management and automated analysis of feature models. In this part the tools which were used in this study along with some other tools which were not used are discussed. Chapter four describes modeling part of the study including domain analysis using NewGenTV SPL. In this part feature identification, feature model organization and model management in FeatureIDE are illustrated. Chapter five explains the translation process. In that process, a parser is presented to convert our model to the appropriate format for analyzer. After that, automated analysis of model in FAMA is inspected in sixth chapter. For conclusion, there is seventh chapter which is a summary of this thesis, including the analysis results we have drawn from our research. Following this concluding chapter, the references for the thesis are presented. Finally, there are several appendices. In Appendix A, flow chart diagrams of the parser proposed in the study are presented. In Appendix B, pseudo codes of the parser are presented. In Appendix C, Feature Glossary which constructed in domain analysis and contains information about the Smart TV domain is presented.

CHAPTER 2

BACKGROUND AND RELATED WORK

This section covers the concept of Software Product Line Engineering including software reusability and management of commonality and variability. Also studies related to software product approaches are given. In addition, feature models along with feature representation are briefly explained.

2.1 Software Reusability and Software Product Line Engineering

Nowadays, large-scale and software oriented projects are being developed for a product family instead of a single product. Thanks to this, it gets easier to change products for various customer needs and the time for producing new products is reduced. One of the methods for software development regarding the product family is software product line approach [5]. In this approach the processes of generating fundamental assets which will be used in common and generating products using these assets have been separated. The main purpose of software product line development is to satisfy variable customer/market needs and develop specified products at a lower cost and in less time. The key to achieve these is to broaden the scope of reusability and transform the procedures which enable this into a systematic way. Along with that also tools which support these procedures and context are within the scope of development of software product line.

We cannot discuss Software Product Line Engineering without mentioning the term reusability. Reusability can be defined as using assets and knowledge which derived from software development activities carried out in the past, while developing new systems [6].

The first important approach which oriented towards reuse is based on the suggestion of component notion within the framework of the industrialization of the software [7]. According to this, components are needed to be classified with regard to criteria such as accuracy, reliability and success and be presented for the customers in order to obtain efficient reuse.

Traditional reuse can be grouped in two main classes. These are source code and conceptual reuse. Source code reuse can be realized by usage of real code or more regular structures such as library. Reuse of only code can be expected to augment dependency to employee competence and knowledge level, therefore leading to inefficient use of resources.

Reusability issue has been addressed extensively by the software engineering studies. The aim of reusability is to enable suitable assets to be used many times in systems with common characteristics. The size and capacity of reused assets affect the benefits of reusability directly. While reuse of a single variable does not gain many benefits; reusability of requirements, design, source code, test assets and system features altogether enhance the meaning and effect of reusability. Reusability at this level is possible with the software product line approach.

Systematic approaches in software reusability, especially software product line engineering have been getting attention increasingly by academy and industry in order to increase the efficiency and decrease the cost of production. Organizing software reusability systematically in medium and large scaled companies, requires a challenging work.

Reusability in software related areas has been used extensively and inevitably as a way to increase software productivity. Reusability is usually encountered as opportunistic reusability. As an example we can say cut-and-paste in existing software. Although this approach works for individual programmers and small groups, it does not suitable for more crowded companies.

Alternatively, software reusability sited on a systematic basis, is effective in the realization of multi-use features. Since the beginning of 1990's, software Product Line Engineering (SPLE) attracted the attention of academia and practitioners in this direction. SPLE aims production of emerging end products along with the all similar intermediate products in the software production process, using common components within the common production framework. There are many success stories over the past twenty years in the field of SPLE [9][10]. However SPLE methodology is still considered to be young and contains some important issues which are not yet resolved such as absence of a fixed notation that has gained widespread acceptance for representing variation or challenges of physically managing all product variants. On the one hand, as recorded success stories are relatively part of a small practice category, more general usage of SPLE's principles still requires research. On the other hand, as noted by the supporters of SPLE, profit in terms of business administration of any SPL application should be primary in investment decisions. As stated by Schmid and John [11], when product development begins, there is no contract in market-driven product development -contrary to customer-driven development-. A product is planned for a market, started from a point in development time and presented to a different or no longer valid market at a later point. In this case, for the SPL, it makes sense that question of "*Should software product line development approach be adopted?*" be answered according to the comparison between investment that it returns and the benefits such as gained speed, qualifications, etc. in application development.

In broad sense, software product lines are the structures which display common and variable assets of a product line which has intensive common characteristics and enable producing new systems for the product family with the usage of these assets. The precondition for these

kinds of approaches to be successful is to define a scope which product family belongs to and to generate a skeleton which is expected to exist in all of the systems in the domain using common features. Modeling variability is the next step of product line development. Foremost productivity acquisition is obtained by developing new systems via using variability of product line assets. Known product line approaches usually defines variability between product line systems using feature models.

Strategies which can be suitable for different organizations and can support transition period of SPL have been suggested in the literature [12][13]. Although each transition strategy is beneficial within itself, it has been pointed out that most of the covered institutions are product-driven, but not customer-driven. While product-driven organizations focus to deliver the best product to the customer, customer-driven organizations focus to deliver the best solutions in the overall context to the customer [14]. Systematic reusable strategies and SPL approaches in the literature of product-driven operation have been inspected frequently [12][13]. On the other hand, there are less studies which provide insight about the question of “Where to start SPL attempt”. However, also in the customer-driven organizations, adopting SPL approach for reasons such as productivity growth, getting fast results, increasing the quality of products and solutions have been becoming more and more commonplace and inevitable.

It is observed that when looking over the examples which stand out in the literature, product-driven organizations tend towards creating product line more. There have been many examples that independently developed products in the initiatives which adopt product line methodology, constituted the initial point [13][15][5]. On the other hand, the aim of customer-driven organizations is to generate specialized solutions which driven by specific customer requirements.

In the product-driven companies, the transition starts from either existing set of products and a product line is developed, or the company starts from scratch. Software assets are developed by either domain engineering or defining common assets in the existing products. Product line architecture is used as a design tool that shows how main components are combined in each product. One of the obstacles against reusability is that sections have limited information about software assets developed in other sections. In this approach, intermediate products should be classified and indexed in order to efficient usage.

2.1.1 Studies Related to Software Product Line Approach in Turkey

Companies have started to prefer product line based approaches instead of single product engineering in order to reduce the costs and not to spend same expenses again and again while developing new products. Although these approaches are new for our country, there are some studies involving them.

The practices that are partially used Software Product Line approach are mainly done in the defense industry area in our country. Kutluca et al [16] stated that they constructed two

product lines, namely CMSCORE-PL (Combat Management System CORE Product Line) and CE-PL (Computing Environment Product Line), as a part of the GEMKOMSIS project in the MILSOFT Software Technology. However, the information about how to ensure reusability of requirements and test cases, and whether or not feature models were used, cannot be obtained from their paper. Also the information about whether or not a study was done about domain engineering and domain models is not given.

Koray et al. [17] stated that two unmanned land vehicles named İzci and Gezgin developed in the area of unmanned systems within ASELSAN, are generated according to JAUS (Joint Architecture for Unmanned Systems) reference architecture. JAUS displays domain model as well as reference architecture for unmanned systems. JAUS is a program which started in 1995 and approved by the USA Ministry of Defense [17]. JAUS reference architecture is service oriented and used in the OpenJAUS [18] framework by the ASELSAN. Koray et al. indicated that such domain model ensure the project team to reach a certain knowledge level. In this study, one can see that reference architecture and domain architecture are utilized at the usage level. In the final part of the study, they denoted that they performed two distinct studies about domain analysis and development of reference software architecture for the weapon systems in the group of ASELSAN SST.

Altıntaş et al. indicated that they developed a software product line named Aurora which is multi-layered, Web based and accelerates software development process within CYBERSOFT [19]. They express that the projects developed on Aurora are Core Banking System (CBS), Central Registry Authority and Insurance Infrastructure.

Kahraman et al. stated domain engineering studies within the weapon system projects in the ASELSAN SST group [20]. Especially it has been studied with the systems which contain firing control software. FORM approach was chosen for feature modeling and a component based approach was applied in order to form a reference architecture.

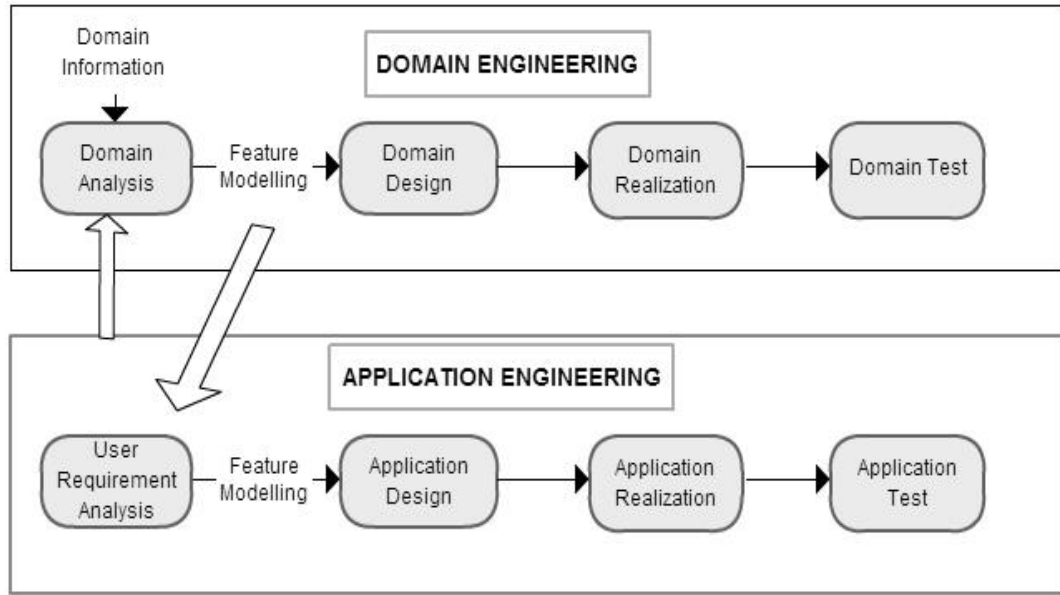


Figure 2.1: Common Process Structure of Software Product Line Approach [22]

2.1.2 Software Product Line Processes

It can be seen that product lines based software development process structure is similar when inspecting different approaches. Figure 2.1 shows this general structure which is based on approach of Pohl and et al [22].

According to Figure 2.1, software product line process consists of two sub-processes namely domain and application engineering. First domain analysis is realized with the use of domain information during domain engineering. At this point, domain scope is defined and commonality and variability are identified. An architecture which is expected to adapt to domain related systems and defines the rules, is generated as the output of the domain design. Domain implementation is the process of designing and implementing domain specific, reusable, detailed components. In the final stage, product line test infrastructure is generated.

Similar stages are also preceded by following similar stages in the application engineering. A concrete product is produced by resolving variability in the domain engineering sub process. It is important to state that these operations are not sequential. Domain engineering operations are expected to be adapted and developed according to the feedback which comes from the application engineering operations.

During the development of a product family, in domain engineering process, modeling commonality and variability as well as deciding software architect which will be created for

the product family become more of an issue. Variability is used to express different products. In domain engineering process, common products which will be used in product family and variability is identified and verified considering feature model [23].

Variability and commonalities in the product family are expressed and managed using feature models. Product family feature model is used in order to define a new product in the process of application engineering while the features of the new features are identified by using feature model.

Model based software development approach makes development process easier especially in complex system development as it has a higher level of abstraction. Automating product development which satisfies all its necessary features can be possible in the product line application engineering via modeling [24][25]. Using the model which represents and manages product line reduces the time to develop new products.

When exploring historical development, it can be seen that domain engineering approaches are suggested before product line approaches. Nonetheless, it can be seen that the focus is on the domain engineering while evaluating various product line approaches.

2.2 Software Product Line Management

2.2.1 Commonality and Variability in Software Product Lines

Commonality and variability are two main concepts of SPL. Commonality denotes the characteristic which appear in some products; whereas variability denotes the ones which appear in specific products, regarding a product line. For instance, a system characteristic which has the same features is considered as commonality; whereas a system characteristic which has different features is considered as variability. For a simpler example we can give geometric figures such as circles, triangles, etc. [35]. These figures have some commonalities such that all of these figures have an area and they are two-dimensional. For the variabilities we can say that they have different number of sides and area formulas.

When dealing with models, variabilities are represented and managed by variation points. A variation point is a point in a system which a variation occurs [36]. Variants of these points are also defined when defining variation point. These variants can be seen as the real values. For instance when dealing with “color” aspect as the variation point, the variants of this can be red, green and blue.

In [35] it is said that when analyzing a domain, inspecting commonality and variability helps one to identify the product family which will be developed in a systematic way. It also facilitates the development process by analyzing the economics of developing a product family. Analyzing commonality and variability comprehensively, enhances reusability [37].

Distinguishing common features from the variable ones can become a troublesome task. As it provides a way to graphically representing and managing the common and variable features of a product family, feature modeling ensures a simplified overview.

2.2.2 Variability Management

Variability management in product families is the set of activities and entities which are required in order to state and organize common and variable assets of a product line. There are different approaches in variability management such as feature modeling, UML based methods, techniques that model the architecture or components of the system, using natural language, formal techniques, ontology based techniques, orthogonal variability management [64]. We used feature modeling approach in this study.

Features can be used when dealing with commonality and variability in the SPLs as they are distinguishable aspects of a product family. However number of features becomes very large when the number of systems in a family is combined with the increasing size and complexity of the requirements. That complicates the detection of commonalities and variabilities in the product family. Therefore, as the complexity of a systems increases, a comprehensive tool support is needed for the automated variability management in order to deal with these systems.

2.3 Feature Models and Feature Representation

A feature can be defined as distinctive user-visible characteristic of a system [2]. Similar and variable features of various systems are congregated these systems in groups of common features or can be used to differentiate these systems.

Systems can contain thousands of features and dependencies. These complicate the job for both users and developers. For the former which instantiates a product from the product line, errors can be faced with the complicated dependencies between features during the configuration process. For the latter, adding or extracting features and dependencies entail the influence of these changes to the model. Therefore system developers solve these difficulties by providing feature models when describing their product line [51].

To manage variability of a software product line, first, the work of feature model generation is needed to be executed. A feature model is represented by a feature graph which can be consists of a directed acyclic graph or a feature tree, and constraints. Feature modeling helps to analyze commonality and variability in software products [26]. So feature modeling is used for documenting the commonality and variability of a family of systems. Feature model denotes all possible products, hierarchical features and relationships between these features which are covered by software product line.

Systems which belong to software product line are expected to have common features. At the same time, variability which differentiates these systems should be described by product line

approach. While handling the commonality and variability in software product lines; the approaches which classify systems according to their features are adopted. First approach to realize this is FODA (Feature Oriented Domain Analysis) [2]. FODA chose expressing commonality and variability in product line by the models that define features and relationships between features. It uses the architecture/design reuse and the concept of feature. FODA essentially is a domain engineering technique. Context analysis, domain modeling and architecture modeling are three main activities of FODA. In the context analysis step, context models and structure diagrams are used to define domain scope and to show relationships. During domain analysis, components relationship model, feature model, functional model are created and domain glossary is developed. In the architecture modeling phase, task interactions and components relationships should be displayed. Various different approaches were suggested after original FODA feature approach.

FORM [28] has been developed in order to improve FODA approach and to include application engineering to the software development process. The techniques developed before FORM handled feature analysis mostly on user requirement level. In FORM, using this analysis, module development has been focused. In this respect, it surpasses FODA as it also deals with the modeling reusable components after architecture modeling. Various engineering principles are identified during generation architecture and components. The information about parameterization these modules with using features and the need to develop features accordingly layered structure, lie on the bottom of these principles. Besides, application engineering sub-process is included to the FORM approach.

FODA places feature concept on the basis of domain analysis. Moreover, it establishes the relationships between assets and features which form product line for model domain-specific variability and ensuring traceability. FORM also adapts similar approach. It surpasses FODA as it realizes the parameterization of product line components, through features.

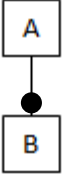
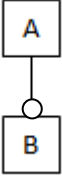
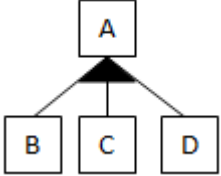
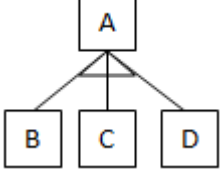
The common characteristic of product line approaches is that they both use features for modeling variability and ensuring traceability. However, as stated in the [22], managing variability of the product line depends upon the inputs of product managers, software architects, developer and testers. Additionally, in [29], it has also been stated that features in variability modeling has shortage in terms of scalability, consistency and traceability. In [30], it has been emphasized that having variability as well as commonality in the feature model complicates variability modeling; additionally, hardship of establishing traceability between various assets only using features.

Czarnecki et al. published a detailed research about feature modeling approaches in [31]. In this study, there are basically 3 feature model notations, namely basic feature models, cardinality-based feature models and extended feature models.

2.3.1 Basic Feature Models

Basic feature models are the standard ones. The elements of this feature model have some basic decompositional relationships. These are *mandatory-relationships*, *optional-relationships*, *or-relationships*, *alternative-relationships*. For example, assuming one has a parent feature A, and child features B, C and D. In the Table 2.1, we can see which relationships these children can have and meaning of these:

Table 2.1: Feature Diagram Relationships

Relationships	Meaning	Notation
Mandatory	This means a product must always have child feature (Feature B), if it has parent feature (Feature A). Feature B is a mandatory feature.	
Optional	This means a product may or may not have child feature (Feature B), if it has parent feature (Feature A). Feature B is an optional feature.	
Or	This means that at least one of the child features namely B, C, and D have to be in product.	
Alternative	This means that one and only one of the child features have to be in product.	

There are also cross-tree relationships which a basic feature model can have. These are requires and excludes. First relationship implies that whenever a feature is presented in a product, also another feature which resides in the right hand side of that constraint, has to be

presented in that product as well. For example when a model has a constraint like “Feature X requires Feature Y”, it means whenever a product produced from this model, has the Feature X, it also has to have Feature Y. For the excludes constraint, it basically works like exclusive-or. For example, constraint “Feature X excludes Feature Y” implies that Feature X and Feature Y cannot be in a product at the same time. So, if a product has Feature X, it cannot have Feature Y and vice versa.

2.3.2 Cardinality Based Feature Models

There are also cardinality based feature models. In addition to the basic feature model characteristics, these models have multiplicities similar to UML. For the feature cardinality these multiplicities indicate a lower bound and an upper bound indicating the number of instances of a feature that can be part of a product. [32]. *Mandatory-relationships* and *optional-relationships* can be examples of this cardinality. For the group cardinality, upper bound and lower bound limit the number of child features that can be part of a product when its parent feature is selected. For example, a feature that has 3 children and *or-relationships* has group cardinality $\langle 1, 3 \rangle$. According to Riebisch et al. [66] optional relations that belong to the same parent feature can be combined into a set and possible multiplicities that a set can have are: $0..1$, 1 , $0..n$, $1..n$, $m..n$, $0..*$, $1..*$, $m..*$ ($m, n \in \mathbb{N}$).

2.3.3 Extended Feature Models

Finally, there are extended feature models. They are basically attribute-included feature models. In that model, features can have attribute values. Extended feature models admit complex cross-tree constraints involving feature attributes. The further information about attributes is represented in the chapter 2.3.5.

2.3.4 Feature Relationships and Dependencies

After software features are identified, dependencies between features should be defined to complete the model. These dependencies define the conditions to make each feature to be selectable. Identifying feature dependency correctly is very important in order to ensure consistency between features. Feature dependencies in the feature model are handled in two ways: local dependencies and public dependencies.

Local dependencies are about relationships. For these dependencies one has to look at the location of the feature in the model. For a feature that has a parent feature in the model, to be selectable, its parent feature should also be selectable. For example, in the Figure 2.2 a small part which is taken from the feature model that constructed for this study is seen. According to this Figure, for InteractiveOtelTV feature is to be selectable, the OtelTV feature should also be selectable in the example model. These dependencies do not have to be identified in the model, as they are the result of feature locations in the model. As a result of this, it should be considered properly how features should be grouped in the feature tree.

Decompositional relationships appear when selectability of a feature depends on selectability of another feature. For example, according to the depicted constraints in the Figure 2.2 which is excerpted from the feature model constructed in this study, DVB_S feature is bound to SiliconTuner feature. These dependencies should be identified in course of the feature modeling work and defined in the feature model.

There are six types of feature relationships such as mandatory, optional, or, alternative, require, excludes as explained in the basic feature model. Using these relationships, the rules which restrict the selectability of each feature, can be added to the model.

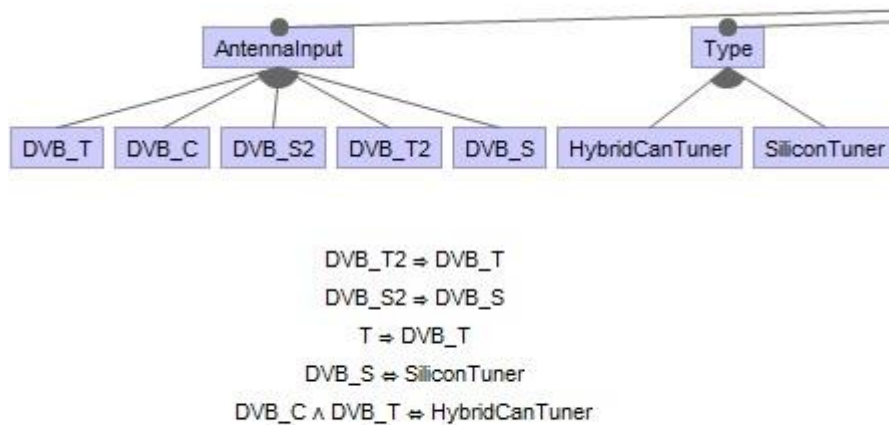


Figure 2.2: A part of the NewGenTV feature model that shows parental relationships and constraints

2.3.5 Feature Attributes

Identifying features and dependencies, provides some of the features in the software to be included or excluded according to the needs. Besides that, customization of features which correspond to functions in the software, are provided. For customization of functions, it is required to save the parameters which define behavior pattern of function, and change these when needed. For example VGA is a feature; whereas VGA_number is an attribute related to that feature.

In the modeling studies, it is resolved that it is more useful to also associate attributes that directly related to the features, to the features and add to the model. In this way, in the same location, it can be possible to access the information about attribute information and how it is customized besides features and feature dependencies, when feature model is inspected [27]. For example, in a feature model, it has been made possible to associate a feature with one or multiple values/attributes when needed. These values can represent any measurable characteristics. Examples of them are boolean, integer, decimal numbers etc.

2.3.6 Feature Management

Feature model is prepared with a language and kept in a text file. This language enable defining features, feature tree, attributes and associating these with related features, assigning accessibility constraints to features and inserting information. As feature number and people who works on the feature model are increased, it is clear that defining and updating feature model like that becomes hard and error-prone. So tools which organize feature model with their graphical user interface, have been developed. These tools can represent relationships of features by defining feature models. For example, it can be seen instantly in user interface that how other features will be affected, when a feature is disabled.

Feature modeling studies, along with feature management and feature organization tools, accelerate software development studies in feature modeling area [27]. These studies enable software products to be developed in a way that they can easily be accommodated to different circumstances meanwhile they play an important role for keeping development mistakes resulted from these flexibility at minimum. Dependencies between features are prevented to be hold messy and unorganized in the source code. This information is defined neatly in the feature model. In this way, it has been enabled to manage feature dependencies and feature related values from monitoring a single location, displaying and altering these independently of source code.

CHAPTER 3

TOOL SUPPORT FOR MANAGEMENT AND AUTOMATED ANALYSIS OF FEATURE MODELS

This section covers the main tools that support feature modeling and automated analysis. The purpose of these tools is to support management and automated analysis of feature models. Although FeatureIDE and FAMA were used in this study, other available tools were also investigated.

3.1 FeatureIDE

FeatureIDE is an open-source framework for feature oriented software development (FOSD) which is a general paradigm for the construction, customization, and program synthesis in software product lines [33]. It is based on Eclipse. Phases of software product line process namely domain and application engineering are supported in FeatureIDE. It has tool support for AHEAD, AspectJ, Munge, FeatureC++, FeatureHouse, DeltaJ, and Antenna. FeatureIDE has also support for software generating phase of SPL within the framework of FOSD. For software generating phase, FeatureIDE constructs a separate file for each feature. Codes of that feature are hold in that file. FeatureIDE has a user friendly model editor which is both graphical and text based. Using GUI helps the modeling process. When user develops model in graphical interface, XML format of this model is automatically generated. So one can easily construct feature model in XML format with this editor. As it can export feature model in the XML format, FeatureIDE can integrate with other feature modeling tools. This tool has also a constraint editor with content assist, semantic and syntax checking. For instance, it can find dead features, false optional features, conflicting rules and redundant constraints. So FeatureIDE automates tasks that previously required complex tool chains.

FeatureIDE represents feature models in tree format. Thanks to its coherent user interface, features can be organized in tree form. Model supports different kind of relationships such as parental relationships and cross-tree relationships. Features can be optional or mandatory. Moreover, abstract features are also supported.

FeatureIDE has support for developing variants for different languages such as Java, C, C++, C#, Haskell, XML. In order to generate variants, FeatureIDE has a configuration view. After defining features and relationships in the model, a selection of features is decided. These selected features are called configuration. At the moment of feature selection, the resulting

variants are automatically constructed. These variants can be tailored based on the requirements.

3.2 FAMA

FAMA (FeAture Model Analyser) is an automated analyses framework for the feature models. Actually FAMA offers two main functionalities namely visual model edition and automated model analysis [34]. However, we mainly use FAMA for automated model analysis.

It integrates some popular logic representations and solvers to optimize the analysis process. Some of the most commonly used solvers such as BDD, SAT, and CSP which are proposed in the literature, are implemented within FAMA. Actually it is the first tool which integrates different solvers for the automated analyses of feature models. The reasoners used in FAMA are Choco [38], Sat4j [39], JavaBDD [40], and JaCoP [41]. In run time, FAMA can select most efficient solver automatically for the best performance considering the operation in use; although it can be forced to use a specific solver. The mapping from a FM onto the correspondent solver is done on demand. So, FAMA automatically will select the BDD solver when the user asks for the number of possible combinations of features of an FM, as it is the most efficient known approach for this operation.

FAMA is based on Eclipse Platform. Eclipse is an open development platform and helps developing and management process with its extensible frameworks and tools. Therefore, being Eclipse based means reducing the development effort and to guarantee that software being built will be easy to access and install in any platform using Java.

FAMA takes Feature Model as an input in various formats such as XML or plain text format. Plain text formats are FM, FMF, and AFM. AFM is used for extended feature models whereas others are used for standard feature models.

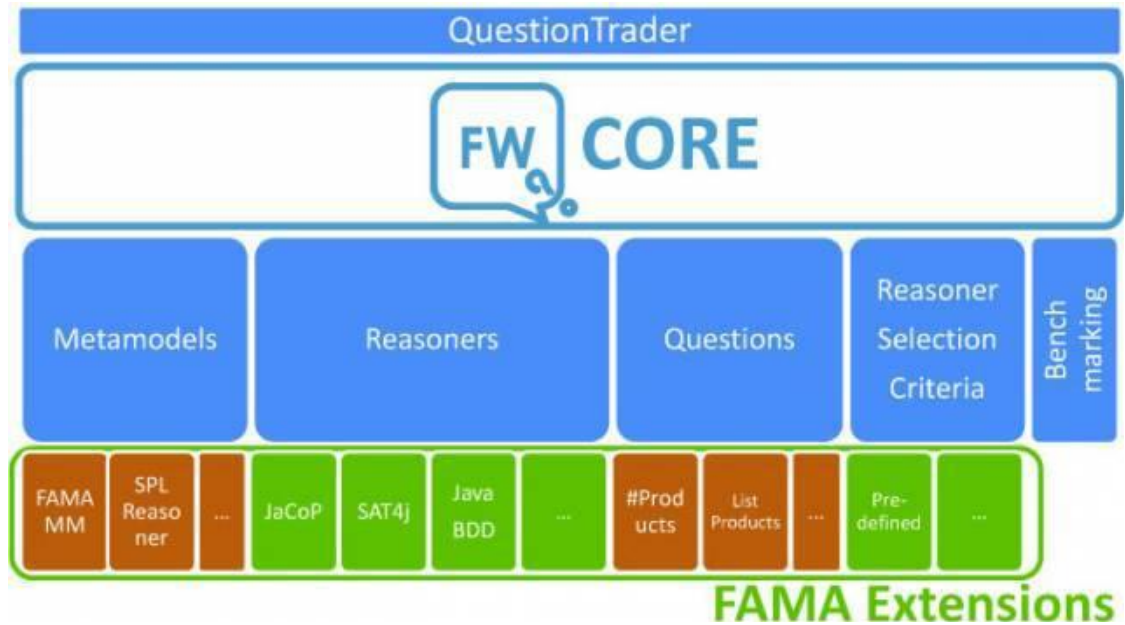


Figure 3.1: FAMA Architecture [58]

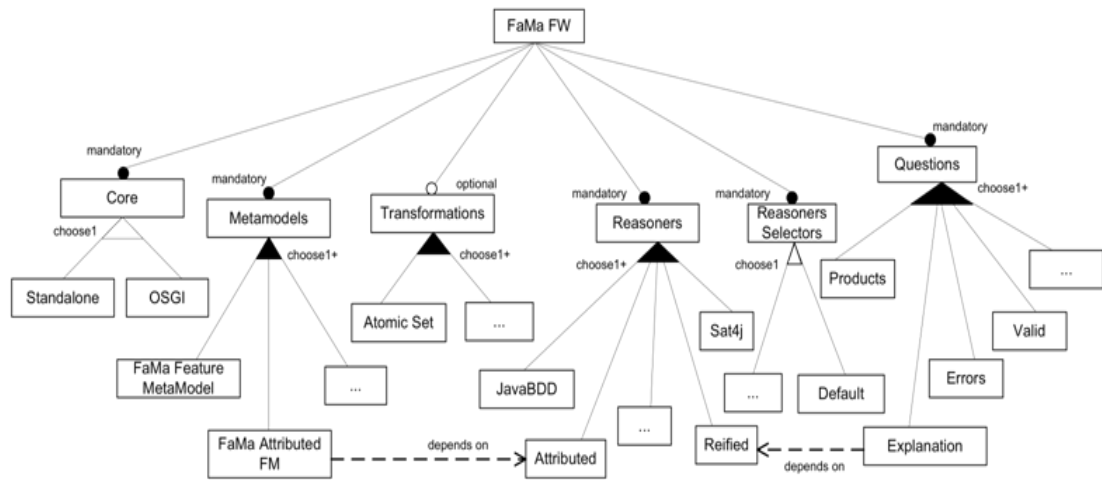


Figure 3.2: Feature Diagram of the FAMA Framework

FAMA architecture consists of some basic components which can be seen in the Figure 3.1. FAMA Questions include the question interfaces that are implemented in the reasoners. They are associated to the job requested by the user. One can use these questions to automatically analyze model. FAMA Metamodel is the specifications of the FAMA Feature Model which is the default variability model. It also contains Attributed Model. There are four types of reasoners namely JavaBDD, Sat4j, Choco, JaCoP. FAMA uses these to answer questions. It should be noted that FAMA can use only one reasoner at a time for a question as can be seen

in the Figure 3.2, which depicts the feature diagram of the FAMA, since Reasoner Selectors feature has *alternative-relationship* for its children. However, reasoner to be used is chosen at run-time or specified by the user. JavaBDD Reasoner uses Binary Decision Diagram (BDD) whereas Sat4j solves a Boolean Satisfiability Problem (SAT). Choco and JaCoP are CSP (Constraint Satisfaction Problem) reasoners. FAMA has a config.xml file. In that file, one can define reasoners, questions and criteria. FAMA can be extended or updated by the FAMA extensions with the help of its component-based architecture.

FAMA uses QuestionTrader interface in order to use reasoner questions.

3.3 Other Tools

3.3.1 VARP

VARP (Variability Analysis and Resolution Plugin) is a collection of Eclipse plugins developed by the Luca Gherardi for robotics SPLs [42]. It proposes a tool for feature model development and analysis of variability. One can easily select or deselect desired functionalities of the product line and can automatically generate products.

Before using this tool, Eclipse Modeling Tools [43] and the Graphical Modeling Tooling [44] should be installed as it is based on these two.

In general, VARP has a graphical interface tool which enables one to form feature models in the tree structure, define constraints and create instances easily by selecting necessary features. It also makes analysis regarding validation of feature model checking all constraints and relationships. For the GUI, they developed a GMF based feature model editor. Tree structured feature models are developed by that editor. Features in the model have a field indicating their type such as mandatory, optional, root and feature cardinality (minimum and maximum usage count of a feature in an instance). It also supports attribute and composite features. Composite Feature is used for showing Containment Cardinality (or, xor) ([1...*] and [1...1]). Moreover it supports constraints.

Although it is a simple and effective feature modeling tool, some statements cannot be used when defining constraints such as if, iff. Also logical operators (and, or, xor) cannot be used standalone. They have to be used with “requires” or “excludes”.

3.3.2 PureVariants

PureVariants [45] is a licensed variant management tool developed by PureSystems [46]. It is an extension to the Eclipse Integrated Development Environment (IDE) [47]. It is used for feature modeling and variant management for software product lines. It is Eclipse based and can be used as an Eclipse plug-in or a standalone application. It is mainly used as a framework for integration with other tools and types of data for the SPL design. Feature models are created and their different configurations can be derived via PureVariants using graph visualizations. Development of models is based on components such as feature,

attribute and dependency. These can be configured. Although graph visualizations eases the modeling process with expanded boxes, colored connection lines, etc.; editing options are limited considering these. Also model validation and attributed features are available in that tool. The tool is language independent. As PureVariants is a commercial tool, one can get evaluation version of the Developer Edition which has a free 30-day usage or Community Edition. Community edition has core functionalities; however the extensibility interface and support for larger models are not available in this edition.

3.3.3 SPLOT

SPLOT [59] stands for software product line online tool. It uses web based reasoning for product line systems. SPLOT targets academics and practitioners such that Software Product Lines research can be put into practice via this tool. It helps to utilize both contributions of researchers and practitioners about software product lines in the same environment. Therefore, it provides 353 generated models in its feature model repository. These can help the Software Product Line researches to share knowledge.

Feature models of the product line can be constructed and edited using the feature model editor. SPLOT also provides reasoning using SAT and BDDs [60]. The system is developed in java with HTML based engine providing simple user interfaces.

3.3.4 Gears

Gears is a commercial software product line tool that is developed by the BigLever Software Inc. [61]. It is basically a SPL life cycle framework which provides management of commonality and variability in terms of feature models. Products configuration can be enabled by selecting the desired features in the model.

According to [62], this framework supports multi-baseline in time, multi-product in the domain and multi-phase in the lifecycle. Also, it uses variation points which support implementation level variation. Gears provides feature declarations, assertions and profiles. Along with features, it also has some basic dependencies namely require and excludes. The view types of the tool for product development are syntactically and semantically well-defined text view and context-sensitive structural tree view. Although Gears is a language-independent and comprehensive tool, it is also a commercial tool and expensive for small organizations.

CHAPTER 4

FEATURE MODELING OF NEWGENTV SPL

This section covers feature modeling process of a Smart TV SPL named NewGenTV along with the domain analysis process. Modeling process was done with FeatureIDE in version 2.6.2 on an Eclipse 3.6. Feature identification, feature model organization, feature relationships and dependencies, defining feature attributes and model management in FeatureIDE were also covered in sub-sections of this chapter. All models that are shown in the figures are excerpted from the NewGenTV feature model which is constructed in this study unless stated otherwise. The properties and the hardware configurations of the PC by which this study was done are as follows: Toshiba, Windows 7 OS, Intel Core i7 CPU, 2.20 GHz, 8 GB RAM, 64 bit OS, 400 GB HDD.

4.1 Introduction

Smart TV can be expressed as the point where computer and television get very close. Similar to computers, TVs which have Smart TV characteristics have also operating system, internet, online services, software applications, etc. By means of Smart TV, plenty of internet operations can be realized.

Smart TV, whose patent right was received in 1994, is a system which integrates services to the TVs. The developers of these TVs have been competing with the other brands while adding new features to their products. These features are in a wide variety such as picture transfer via USB/DLNA from mobile phone, tablet or PC, saving a broadcast by television as a video without needing any external device, video-conference systems, connecting to the internet via TV, using social media applications, etc.

With these increasing and changing features and growing demand in the sector, variability and commonality management in the framework of SPLE are needed.

4.2 Domain Analysis

Domain analysis is the systematic study of existing systems, its domain, knowledge gained from domain experts and emerging technology within the domain. It also integrates these in a coherent model [36]. To determine commonalities and variabilities of a product family in order to understand the desired system features and capabilities, domain analysis is the first

process. It can provide a basis for understanding and communication about the problem space defining features and capabilities of the system. Therefore, it promotes reuse on SPLs, provides a common understanding of the domain and simplifies the communication. To collect related domain information and combine it to the model, domain analysis is used. The information need in this process can be gathered from various sources such as existing products, domain experts, documents, technology, etc. [36]. Main output of domain analysis is the feature model [22].

In this study, FODA method was used for performing domain analysis. In this analysis, common and variable features, grouping characteristics, decompositional relationships, variability points and constraints were inspected and determined by literature review, domain expert interviews and inspecting system.

According to Arango's survey [52], there is a common process for Domain Analysis methods. So following this process, in that study following steps was performed:

Domain characterization and project planning: Feature Modeling was chosen as the modeling method because the domain has distinctive characteristics which can be denoted as features. Also considering the size, i.e., possible numbers of features the model can have, tools were chosen. For the modeling process, FeatureIDE was used. It was chosen mainly because of being an open-source framework and working for also very large models smoothly. It has also a GUI view which is in tree format. It is easy to use and can give model output both as xml and jpg formats. Only drawback of the tool is that it does not support feature attributes. To solve that, a naming method which will be explained in further pages was generated.

Data collection: In this activity, as much information as needed were tried to be collected. As mentioned before sources of data for the domain analysis are domain experts (developers), project documents and books about Smart TVs.

Data analysis: Now, lots of information about domain and project had been gained. As they resided as a big bulk of information, they needed to be identified. At this step, reusable features were defined and some similarities and differences were identified. Feature Glossary had also been started to be developed. However, when identifying features, there are some points that needed to be considered. For example, not everything which can be a feature should be a feature. There should be an elimination. For example button size for a system is not a key distinctive property. Hence, it is unnecessary to make it a feature, although it can be expressed as a variation as there can be different sizes of this button.

Classification: Knowledge gathering in the previous steps is organized and features are grouped with the similar features. Features were abstracted according to relevant common features and according to their descriptions. Abstractions are organized into generalization hierarchies. So a primary model was constructed considering all characteristics. At this point, mainly existing system was inspected. Products Configuration Sheet, which defines

requirements, some characteristics and their values, is the main resource for the startup of developing feature model. Therefore, the first version of the model was constructed using this sheet. In that primary model, conceptual groups were presented as children of the root feature (NewGenTVSPL). 23 feature groups were identified as seen in the Figure 4.1, Figure 4.2, Figure 4.3 and Figure 4.4.

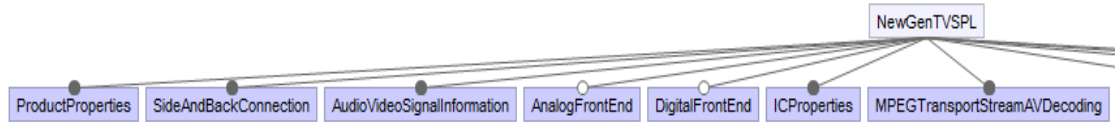


Figure 4.1: (A) First version of the NewGenTV feature model

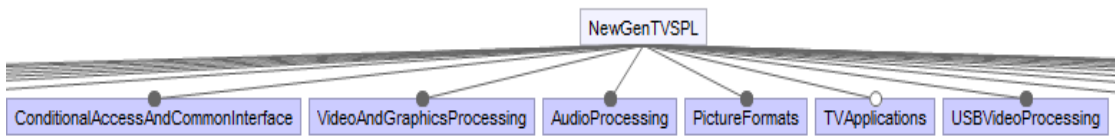


Figure 4.2: (B) First version of the NewGenTV feature model



Figure 4.3: (C) First version of the NewGenTV feature model

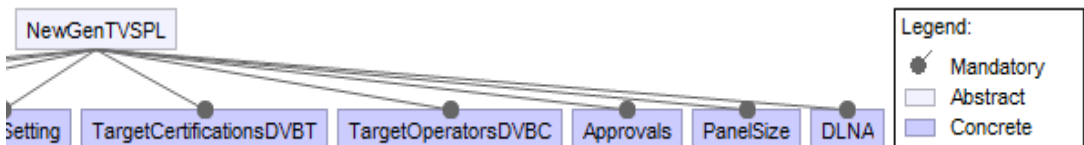


Figure 4.4: (D) First version of the NewGenTV feature model

Moreover, with the detailed study of the configuration sheet and getting help from the development team about some characteristics and confusing points, primary dependencies were defined as constraints which can be seen in the Figure 4.5. For example, the constraint DigitalProgramStorageT_C implies (DVB_T or DVB_C or DVB_T2) as seen in the Figure 4.5. As DigitalProgramStorageT_C is only applicable to terrestrial or cable broadcasts, it implies that DVB_T or DVB_C or DVB_T2 is selected.

$$\begin{aligned}
& \text{DVB_S2} \Rightarrow \text{DVB_S} \\
& \text{_3DCombFilter} \Rightarrow \text{NTSC} \vee \text{PAL} \\
& \text{DVB_T2} \Rightarrow \text{DVB_T} \\
& \text{mkv} \Rightarrow \text{divx} \\
& \text{T} \Rightarrow \text{DVB_T} \\
& \text{DVB_S} \Rightarrow \text{SiliconTuner} \\
& \text{DVB_C} \wedge \text{DVB_T} \Rightarrow \text{HybridCanTuner} \\
& \text{DVB_C} \Rightarrow \text{InputFrequencyDVBC} \wedge \text{InputLevelDVBC} \wedge \text{ModulationDVBC} \wedge \text{SymbolRateDVBC} \wedge \text{Full_Quick_NITSearch_DVBC} \\
& \text{Full_Quick_NITSearch_DVBC} \Rightarrow \text{DVB_C} \\
& \text{DigitalProgramStorageS2} \Rightarrow \text{DVB_S2} \\
& \text{DigitalProgramStorageT_C} \Rightarrow \text{DVB_T} \vee \text{DVB_C} \vee \text{DVB_T2} \\
& \text{MHEG_HD} \Rightarrow \text{DVB_T2}
\end{aligned}$$

Figure 4.5: Constraints of first version of the NewGenTV feature model

At the subsequent versions of the model, the task of determining the features to be included as common or variable is focused on. The child features of conceptual groups were positioned in appropriate places in the model. At this part, apart from consulting configuration sheet, literature about Smart TVs and concepts used are studied [48] [49] [50]. As a result of this study, Feature Glossary (see the appendix C), which explains some concept, was continued to be constructed. This Feature Glossary has repeatedly revised along with the model as the domain research continued. At this part, new constraints were also defined.

Evaluation of the domain model: In this step, developed feature model was evaluated. Errors, contradictions and deficits which were found, were corrected.

Data Collection, Data Analysis and Classification steps were followed iteratively. Therefore model was repeatedly revised. In each revision of the model, NewGenTV developers were also consulted. In case of a conflicting or confusing point, the help of developers were received again.

4.3 Modeling in FeatureIDE

In this part, the modeling process which was also mentioned in general terms at the Domain Analysis will be explicated by focusing on modeling in the FeatureIDE.

First step of generation of feature model is identification of the software features. In this study, all of the features of system are added to the model. Some features can change in the different configuration of the software. Some features that are likely to change can be anticipated. Also base features which appear in all versions of the software, can be identified.

In our work, the relationship between structures which hold variability information is represented by a feature model. As an example, we can give a part of the feature model which was formed in this study in Figure 4.6. In our model, features are represented as

rectangles. They can either be optional or mandatory. Optional ones are denoted by an empty circle whereas mandatory ones are denoted by a filled circle. But before modeling, we had to identify features. As unnecessary implementation details or all requirements should not be presented in the model as features, we focused on the system properties, characteristics, and assumptions which can differentiate the product from others in the domain.

4.3.1 Feature Identification

Identification of features involves abstracting domain knowledge obtained from domain analysis. Although features are used to distinguish between various system properties in terms of commonalities and variabilities, they do not describe functionalities in detail [53]. Features can be separated into four general categories namely; capability features, operating environment features, domain technology features and implementation features [2]. So, when defining features and feature relationships in FeatureIDE; first, features were identified. In the below, these feature categories with their examples in the NewGenTV model were explained.

Capability features are the ones with the user visible properties. They can be distinct services which are marketable units and are found in the user manual and requirements sheets, operations that are internal functions to provide services and non-functional characteristics of system.

For example, in the Figure 4.6, a part of the feature model of our system can be seen. For that model, some of the children of AnalogFrontEnd feature which is the child of the root feature can be seen. In the Figure, ChannelBandwidthForAnalog, ManualSearch and AFT features are capability features. AFT that is used in the receiver to maintain the correct oscillator frequency in the tuner (see Feature Glossary for definitions), is an operational feature. ManualSearch is a distinct service provided for user. ChannelBandwidthForAnalog is a non-functional characteristic.

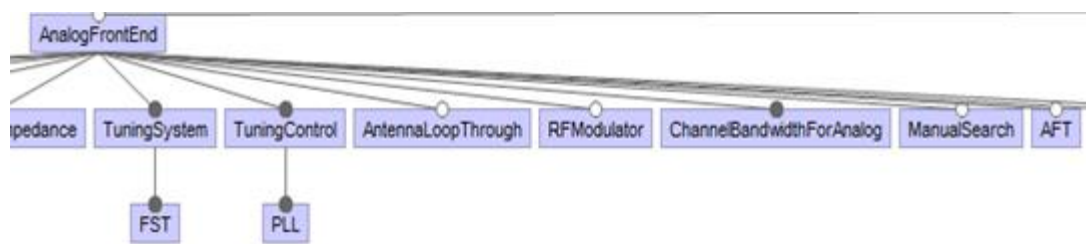


Figure 4.6: A part of the NewGenTV feature model that shows some capability features

Operating environment features are the features which are related to the environment (both hardware and software) in which the system is used and communication of this system with external systems. For example, in our model, which shows two mandatory children of

ICProperties feature whose parent is root feature, is depicted in the Figure 4.7. CPUPower and RAMPower are the operating environment features.

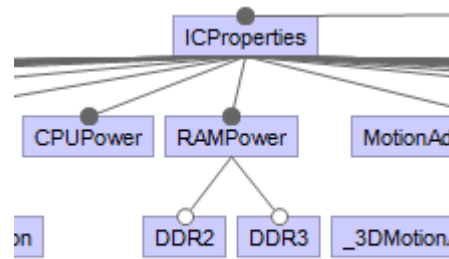


Figure 4.7: A part of the NewGenTV feature model that shows some operating environment features

Domain technology features are the features which are directly related to the domain and cannot be meaningful in other domains. They can be domain-specific theories; methods, analysis patterns and standards. They indicate the techniques of implementing operations or services. For example, in the Figure 4.8 and Figure 4.9, it can be seen that some children of the SideAndBackConnection and VideoAndGraphicProcessing whose parent are the root feature, are domain technology features. For the AntennaInput feature, which is a variation point, DVB_T, DVB_C, DVB_S2, DVB_T2, DVB_S are the domain technology features, because they are Digital Video Broadcasting standards for television (see Feature Glossary for definitions). For the ChromaDecodingStandard feature which is also a variation point, PAL, SECAM and NTSC are the domain technology features, because they are analogue television color encoding system standards (see Feature Glossary for definitions).

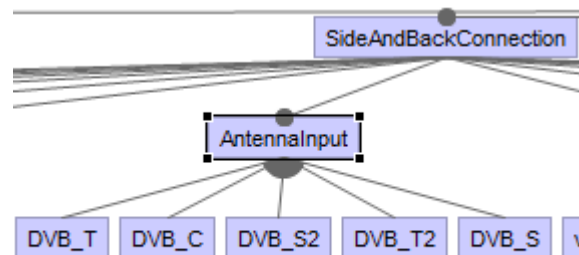


Figure 4.8: A part of the NewGenTV feature model that shows some domain technology features

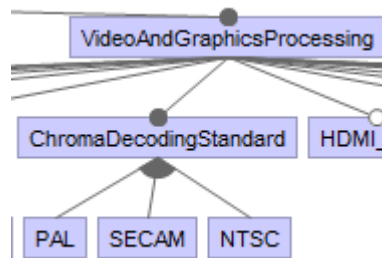


Figure 4.9 A part of the NewGenTV feature model that shows some domain technology features

Implementation features are generic functions or technologies which can be also used in other domains. Communication methods, design patterns and synchronization methods are examples of these features. They are used to implement other features. For instance, we can give LNBpowerAndPolarization_DVBS2 feature, which is the child feature of DigitalFrontEnd_DVBS2, which can be seen in Figure 4.10, as an example of implementation feature. The feature is about LNB, which is Low-noise Block Down converter. It is the receiving device on an antenna used for satellite TV reception (see Feature Glossary for definitions). So, this feature is used to implement a satellite TV reception although it can be used in other domains too. Also DiseqC12Support is an implementation feature as DiSEqC (Digital Satellite Equipment Control) is a communication protocol for using between a satellite receiver and a device such as a multi-dish switch or a small dish antenna rotor (see Feature Glossary for definitions).

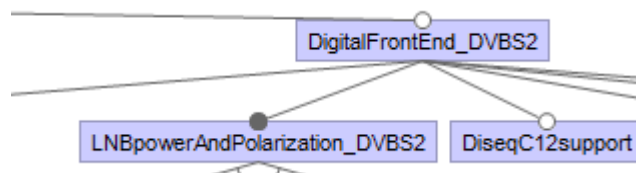


Figure 4.10: A part of the NewGenTV feature model that shows some implementation features

4.3.2 Feature Model Organization

Now, features were identified considering their types. Once the features of all applications used in the analysis are named and defined and any naming conflict is resolved, then a hierarchical model should be created by classifying and structuring features using the relationships. Whether or not each feature is mandatory, alternative or optional should be indicated in the model. Also, feature grouping was considered when defining features in FeatureIDE according to their characteristics. Understanding these groups makes commonality identification easier. Therefore, listing features and modeling becomes more efficient. Primary model mentioned in the previous section was constructed at this point.

This primary model, as seen in the figures from Figure 4.1 to Figure 4.4, has a depth of 2. After defining main feature groups in model, their features were added.

Features, as seen in the Figure 4.11, are added to the related conceptual group as the child feature. Each feature is tied under the related feature (parent feature). For example, in the Figure 4.11, it can be seen that TP4 feature is tied under the RemoteControl feature as this feature is only meaningful under the RemoteControl feature. Also some features are also grouped together according to their function. According to this part of the model, the structures whose parent is the root feature represent the conceptual group that variabilities and commonalities are belong to. There are 22 conceptual groups in our full feature model. Also, some children of these features can also be conceptual groups. They are like variability points. For example feature ProductProperties that is in the Figure 4.11, is not a real feature of the software, and rather serves as a group name for its child features. The children of “ProductProperties” namely Cabinet, TouchKey, etc, are the real values of variabilities and commonalities, so, they are the real features.

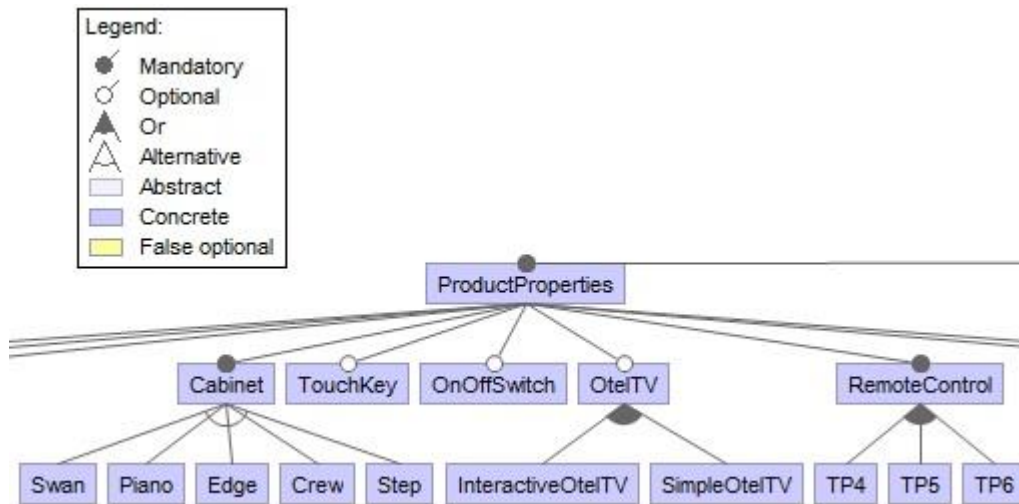


Figure 4.11: Some Conceptual Groups of Feature Model

4.3.3 Feature Relationships and Dependencies

To mention groups in feature model, we can look at the part of the feature model represented in Figure 4.11. The relationships between children of “ProductProperties” are *mandatory-relationship* and *optional-relationships*. They state common properties of parent feature. In that part, Cabinet and RemoteControl have *mandatory-relationships* with their parent and others have *optional-relationships*.

When decompositional relationships were defined in the model, the local dependencies were also handled as these types of relationships are the local dependencies. For the local dependencies one has to look at the location of the feature in the model. For a feature, which

has a parent feature in the model, to be selectable, its parent feature should also be selectable. These dependencies do not have to be identified in the model, as they are the result of the feature locations in the model. For example, in the Figure 4.12, it can be seen that the selection of the child feature InteractiveOtelTV enforces the selection of its parent feature OtelTV. On the other hand, selection of a parent feature with mandatory child feature requires the selection of that child feature. For instance, the selection of AnalogFrontEnd requires the selection of its mandatory children and vice versa; for that part of the model these are AerialInputImpedance, TuningSystem and TuningControl. However, optional child features in the Figure 4.12 such as AntennaLoopThrough and RFModulator do not need to be selected if its parent AnalogFrontEnd feature is selected.

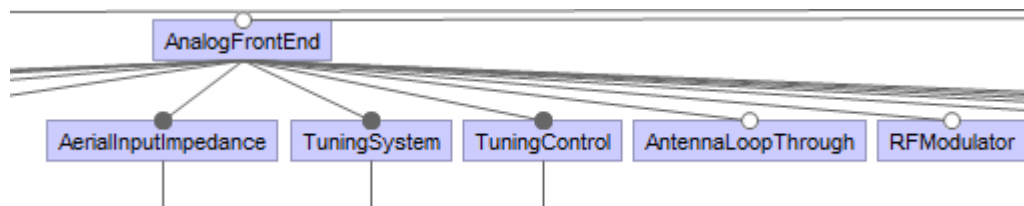


Figure 4.12: Local Dependencies in Feature Model

To represent dependencies between sibling features; features may also belong to a relationship group. According to Figure 4.11, Children of Cabinet are in a xor-group meaning that they are mutually excluding and one and only one may be selected. Therefore a product must have only one of Swan, Piano, Edge, Crew or Step. The relationship about features between children of RemoteControl is *or-relationship*. It means that at least one of the TP4, TP5 or TP6 should be chosen. Also, the children of the OtelTV feature are organized in an *or-relationship*. However there is a difference when comparing this to RemoteControl. In RemoteControl, at least one of the children of that feature always has to be chosen for a product. However, for children of OtelTV, if and only if OtelTV is chosen to be in product, at least one of its children has to be also chosen. If OtelTV is not chosen, we cannot choose its children to be in product. The reason of that OtelTV is an optional feature.

All relationships cannot be defined with the *mandatory-relationships*, *optional-relationships*, *or-relationships*, *alternative-relationships*. So, additional cross-tree constraints are needed. They can be identified by comparing the feature descriptions. The Feature Glossary constructed in the previous sections was utilized in this process.

Cross-tree constraints allow for the specification of further dependencies among hierarchically unrelated features. In our tool, we defined complex cross-tree dependencies. FeatureIDE allows defining constraints with the logical operators such as implies (\Rightarrow), iff, and, or, not. For example, in the feature model “DVB_S implies QPSK” was defined as constraint because in DVB_S, the data is modulated on the carriers with QPSK (see Feature Glossary for definitions). FeatureIDE has a constraint editor. Using that editor, constraints were defined and edited such as in the Figure 4.13. Editor can make syntactic and semantic

validity checks of constraints. It is enriched with a content assist. It can understand mismatching brackets, dead features, false optional features and redundant constraint.

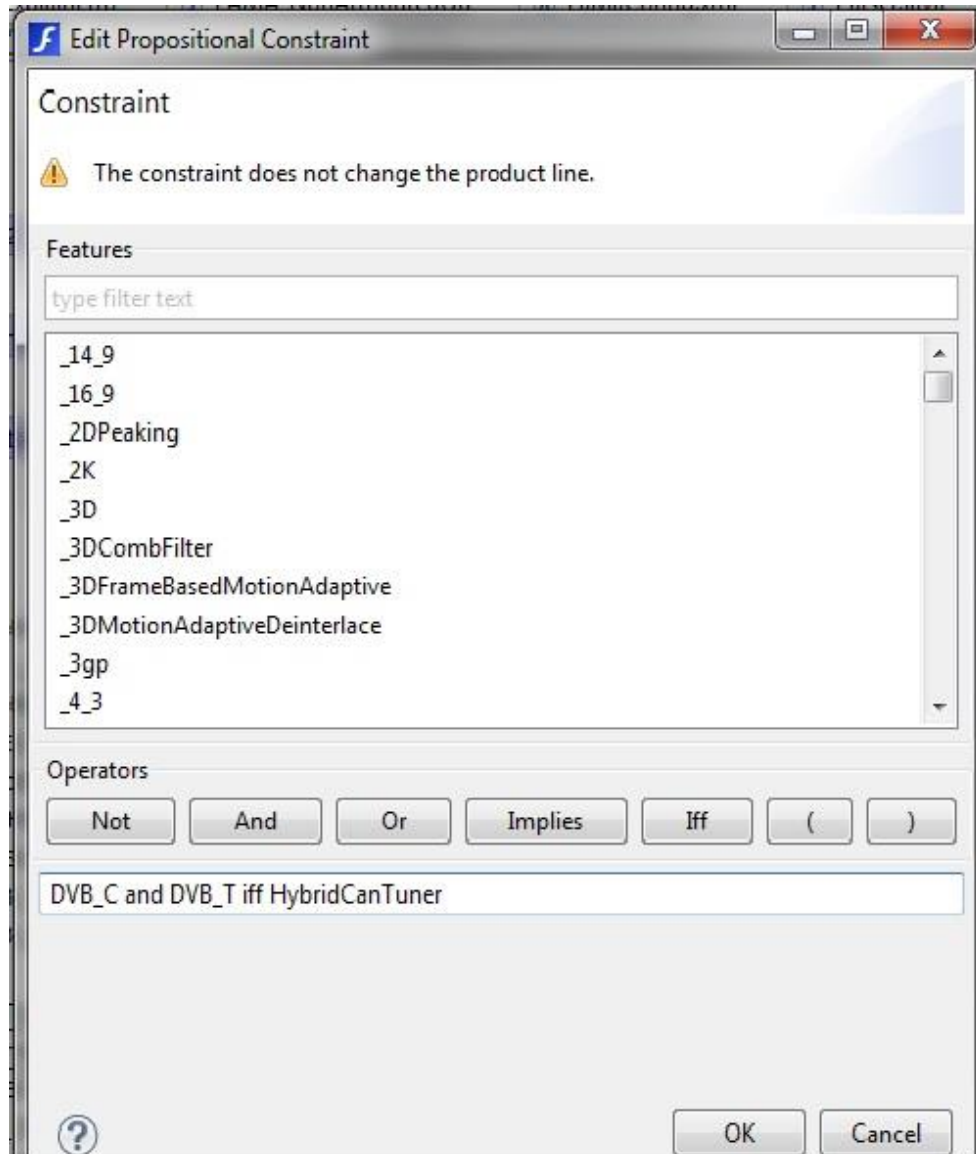


Figure 4.13: Constraint Editor

4.3.4 Defining Feature Attributes

As mentioned before, features can have attributes which are boolean, integer or decimal number parameters.

However, as mentioned before, FeatureIDE has no representation for attributed features. But in our model, we had to define also attributed features. Therefore, a naming convention was developed for this study to define them. By this naming convention, as told in the previous

section, we can detect these features as feature attributes and process them accordingly. In that way, we constructed a basic feature model based on our original SPL which has attributes and can be an extended model. This extended feature model is semantically equivalent to the basic feature model which was constructed in [67].

In FeatureIDE, characters which can be used in the feature name are limited to java identifiers. These are numbers (0 to 9), letters (a to z), underscore (_), and dollar sign (\$). However as FAMA, does not accept \$ sign, this sign cannot be used. So, for the attributed features, a child feature of that feature was defined as a variation point. To indicate that it is a variation point, the first two characters of the feature was given as “vp”. Full naming convention is like the following: “vp” + “parent feature name” + “_in” + “unit type”. The value of an attribute was defined as the children of that variation point by a naming as “parent feature name” + “__” + “value”. For example, as represented in the Figure 4.14, AerialInputImpedance has an attribute 75 Ohm and ChannelBandwidthForAnalog has an attribute with two possible values namely 7 MHz and 8 MHz. For the latter, features with attributes were denoted by *or-relationships* as a product can support at least one of these values or both of these values.

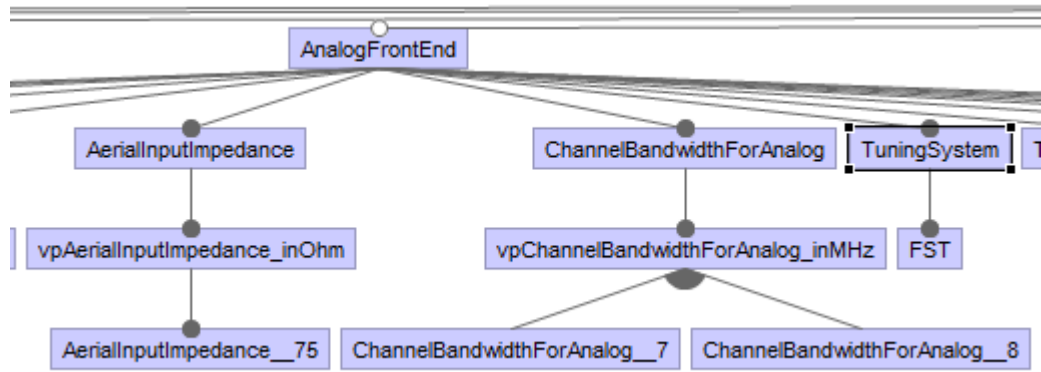


Figure 4.14: Attributed Model for AnalogFrontEnd

For the PanelSize feature, as PanelSize can have one and only one value in a product configuration, child of this variation points have *alternative-relationships* which can be seen in the Figure 4.15.

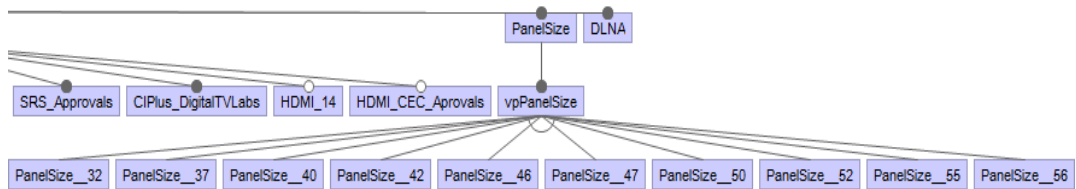


Figure 4.15: Attributed Model for PanelSize

Also, feature attributes can have range as values. These were defined using “to” between values as seen in the Figure 4.16.

After defining all feature attributes like that, we got 140 new features.

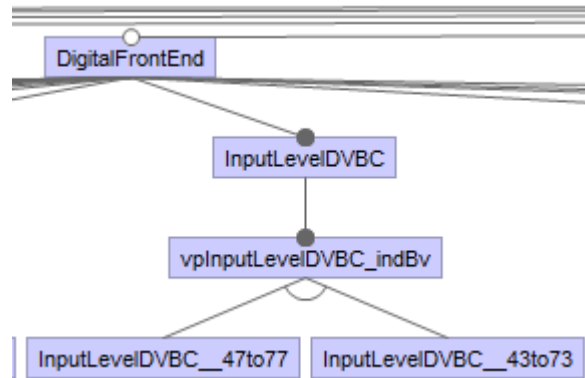


Figure 4.16: Attributed Model for ranged values

4.3.5 Model Management in FeatureIDE

At the end of modeling process, our feature model contained a single root feature named NewGenTVSPL (level-0) and 23 main feature groups (level-1). After adding all features to the model, we got a model with a depth of 5. There are 621 features in total including the root feature. There are 36 constraints. NewGenTVSPL represents the core of all derivable product configurations and 621 features each corresponding to distinct, user-visible product characteristics. Thus, each feature constitutes a configuration parameter that is either selected or deselected.

FeatureIDE is not only useful for just modeling but also model management. We can easily alter the model by adding, deleting or repositioning features or constraints. Repositioning can be done by drag and drop using mouse. Also, we can change their feature type (mandatory/optional) by double clicking them. To rename a feature, only a single click is needed. After the name of a feature is changed, FeatureIDE refactors and makes necessary changes if that feature is used in constraints. Features that have sub-features can be deleted with or without their sub-features, as seen in the Figure 4.17. Connections and mandatory property can be changed with a double click.

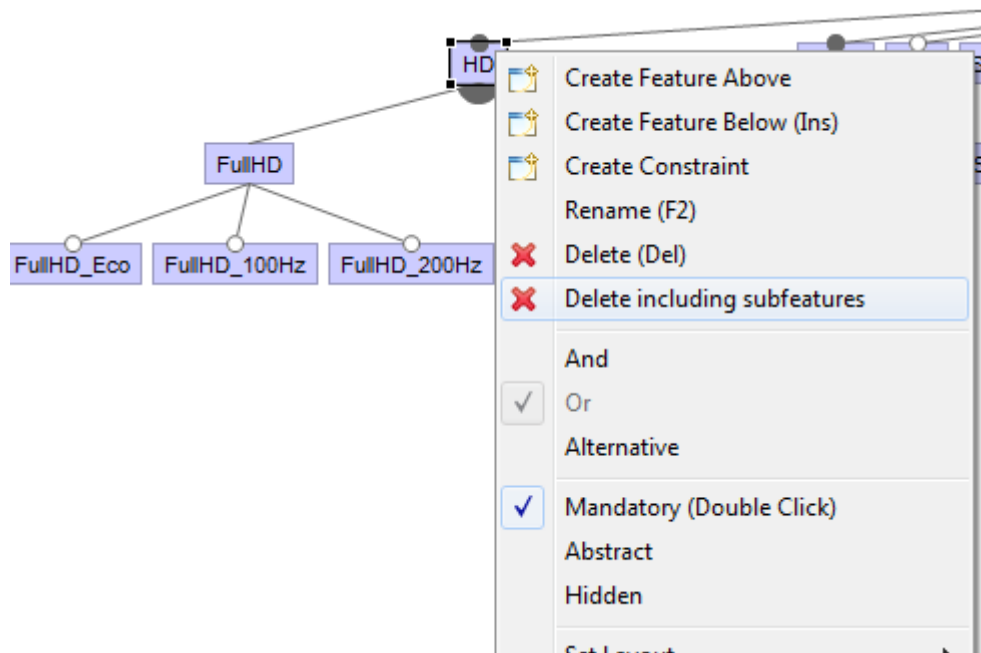


Figure 4.17: Deleting features including subfeatures in FeatureIDE

Model refactors itself after a change. For instance, when we change a feature name, if that feature is used by a constraint, the feature name in the constraint also changes automatically. If there is some problem with the refactoring, FeatureIDE tells the problem. For example, as depicted in the Figure 4.18 and Figure 4.19, when the DVB_T feature which is used in the constraints, is attempted to delete, it gives the error which showed in the Figure 4.19.

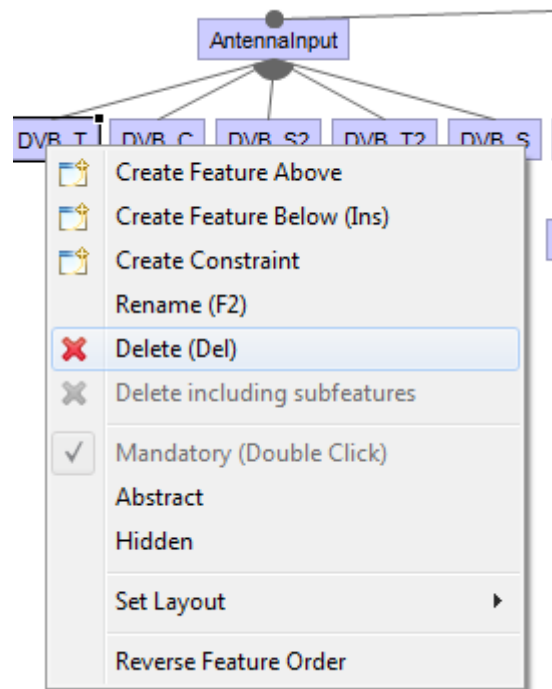


Figure 4.18: Deleting features in FeatureIDE

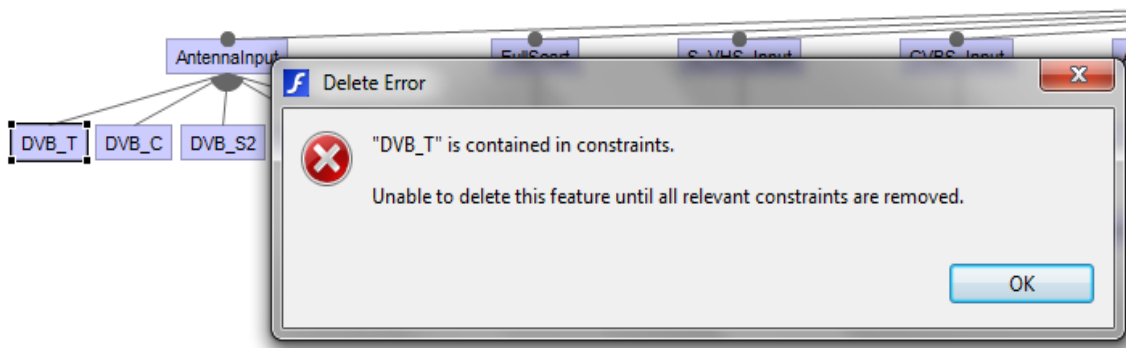


Figure 4.19: Error of Deleting features in FeatureIDE

Using FeatureIDE, we also kept track of the changes and their effects to the model as seen in the Figure 4.20. It compares the current edited model with the last saved version. It analyses the added or removed variants and gives statistics about model.

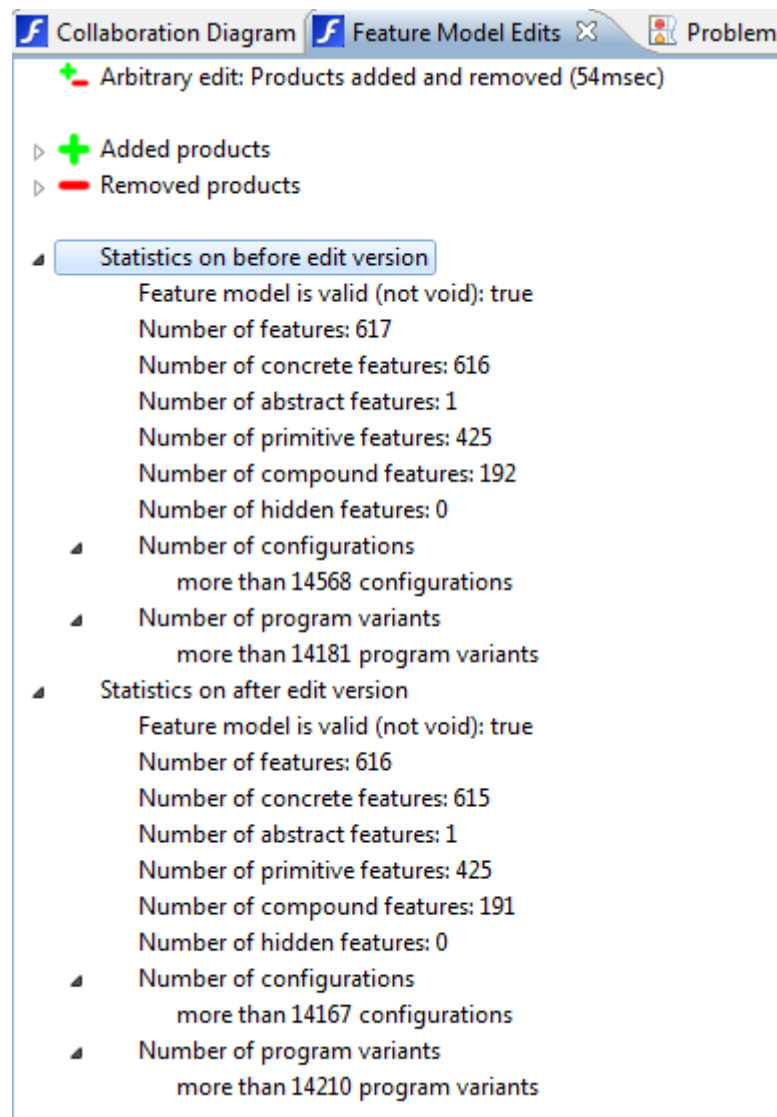


Figure 4.20: Feature Model Edits in FeatureIDE

In the feature model like NewGenTV, which has lots of features and constraint, it is hard to keep track of the features and constraints related to those features. FeatureIDE also has a nice display functionality to ease this hardship. As seen in the Figure 4.21 and Figure 4.22, whenever a feature is selected by mouse, both the feature and related constraint of that feature are shown in the black framework.

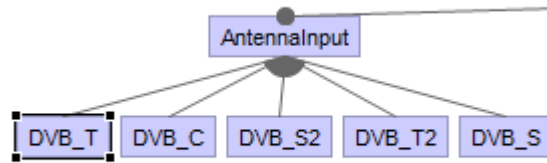


Figure 4.21: (A) Highlighted Constraints

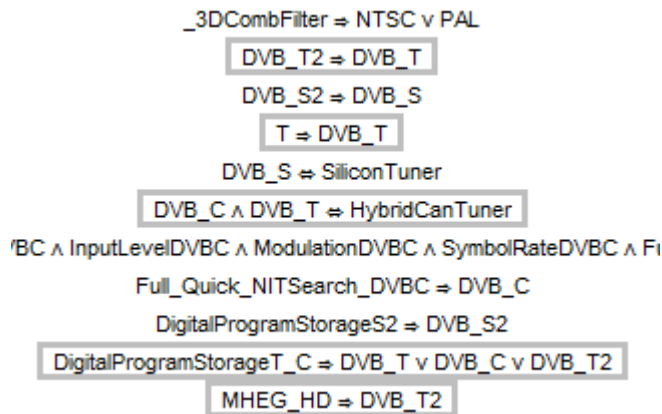


Figure 4.22: (B) Highlighted Constraints

FeatureIDE also does a simple model validation. When the model has been constructed, it shows the contradictions and defects which were found in the model. For example, as represented in the Figure 4.23, constraint $T_C_Tuner \Rightarrow SignalLevel$ is redundant and could be removed. However in a big model having lots of features, relationships and constraints, it is hard to find the reason why this constraint is redundant. But, that is not a problem as we make analysis with FAMA.

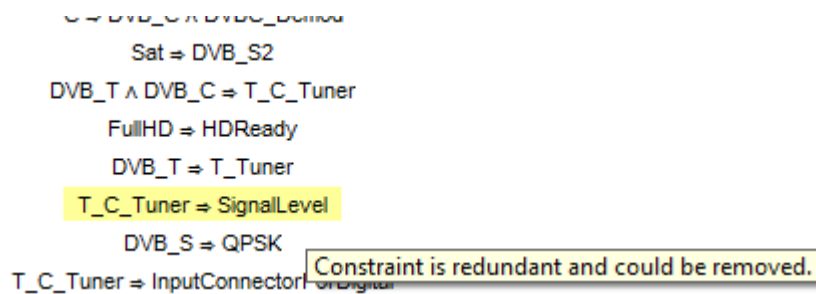


Figure 4.23: Contradictions in FeatureIDE

Configuration management is also supported in FeatureIDE by a graphical editor called configuration editor. A configuration means a selection of features. Using this editor,

possible configuration options can be inspected. To realize this, the features which are needed in the configuration are selected and saved in a configuration file. FeatureIDE checks validity of the configuration. Moreover, when creating a configuration by selecting features, the tool obeys the feature model relationships and rules. For example, as depicted in the Figure 4.24, it does not allow selection of Swan and Piano at the same configuration as these have *alternative-relationships*. Likewise, mandatory features reside in all configurations and there is no way to deselect them. In the Figure 4.24, grey colored features indicate the ones which cannot be changed in the current state. Also, green colored ones indicate that selecting them results in a valid configuration whereas red ones results in an invalid configuration. FeatureIDE also provides an advanced configuration editor as shown in the Figure 4.25. In the advanced configuration editor, one can eliminate features manually.

Actually, the configuration editor helps developers to create valid configurations. The editor indicates whether a configuration is valid in the current configuration step. But for the NewGenTV model, which has lots of features and constraints, this level of configuration analysis is not much of a help. Therefore FAMA, which is more comprehensive tool in feature model analysis, was used.

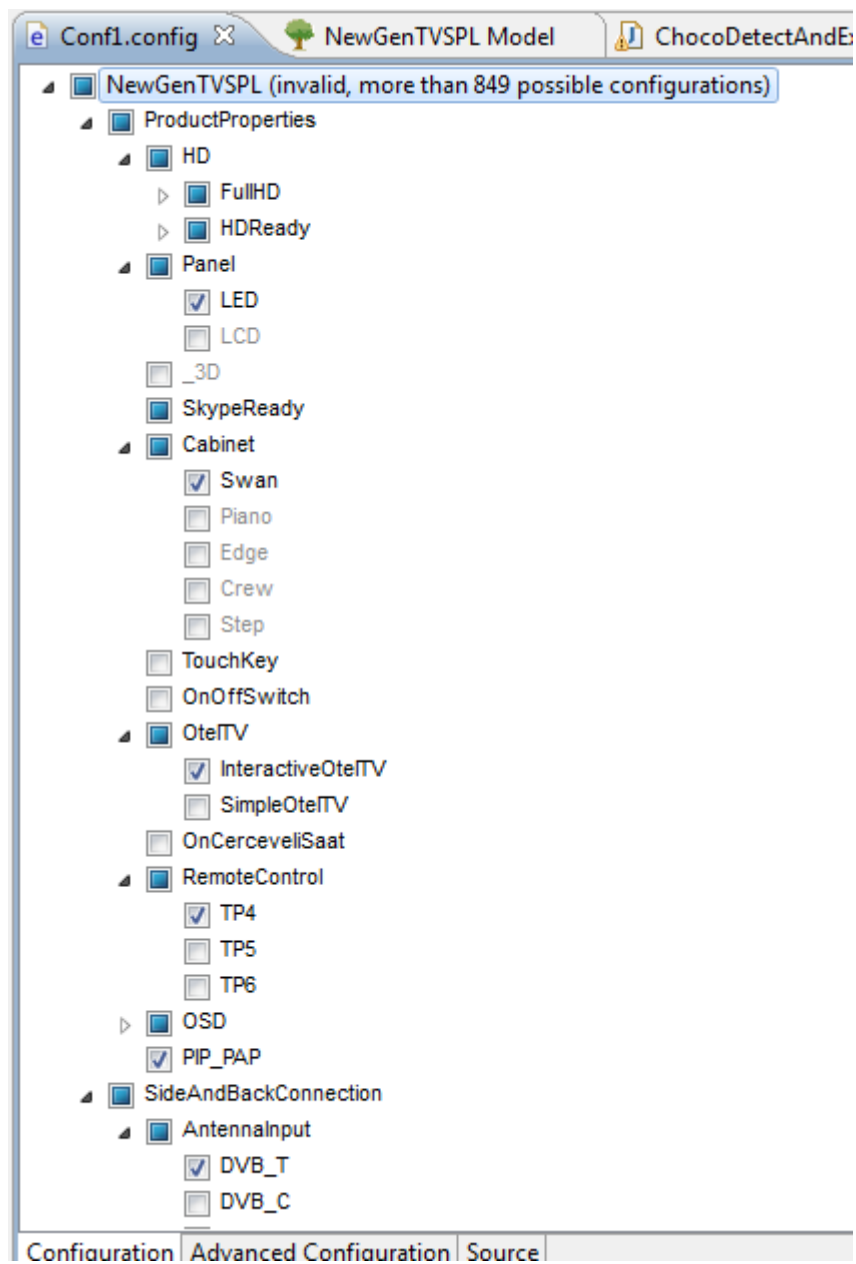


Figure 4.24: Simple Configuration Editor of FeatureIDE

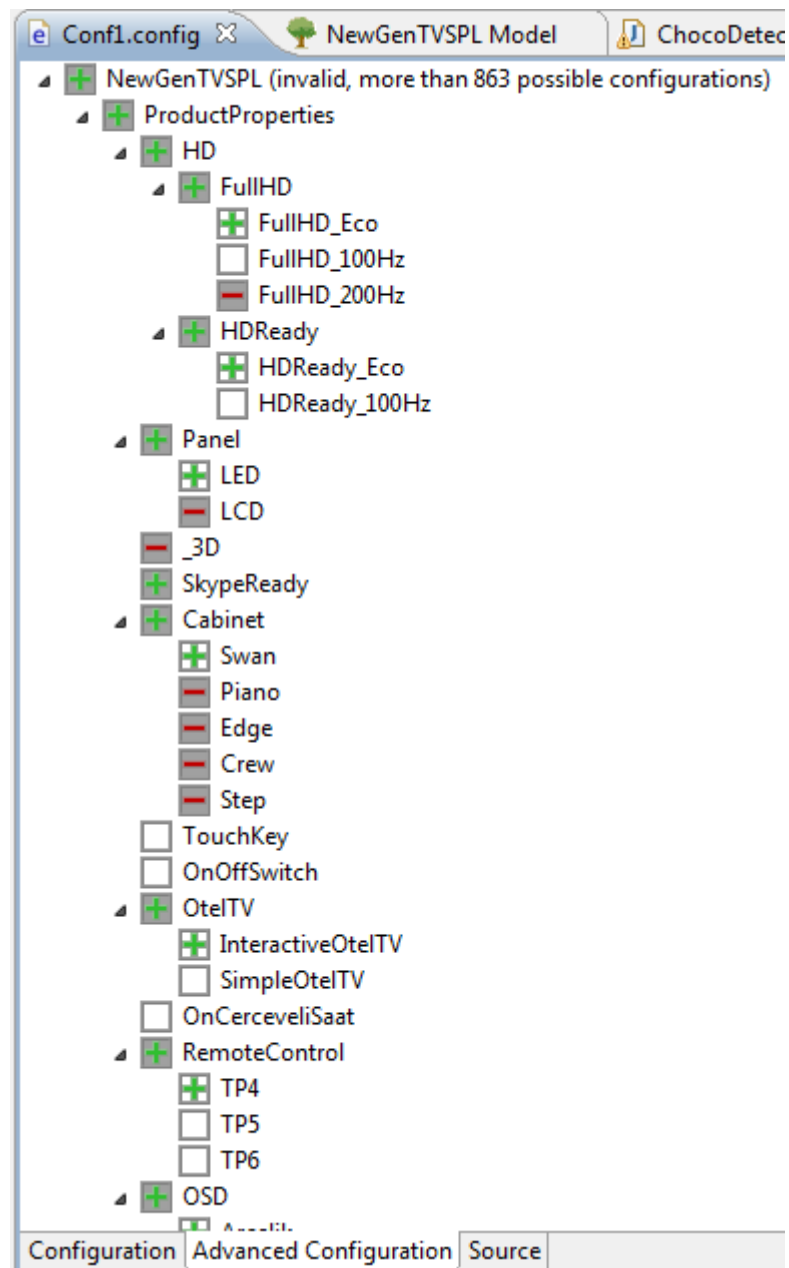


Figure 4.25: Advanced Configuration Editor of FeatureIDE

CHAPTER 5

TRANSFORMING THE NEWGENTV FEATURE MODEL TO FAMA

As mentioned before, FeatureIDE was chosen as the modeling tool because besides other things, it also can be imported easily both as a picture and as a XML file. XML [21] is Extensible Markup Language. Although FAMA which used in analysis part of the study only accepts simple text format (AFM and EFM for attributed models), having a model which is in XML format helps a lot because XML files have a structure which is easy to parse.

After we construct model with FeatureIDE, we now had a XML based feature model which was needed to be analyzed in FAMA tool. Therefore, at this point a translation was needed. To transform our feature model which is available in XML format to the accepted FAMA format (AFM), a parser was developed. Standard Java API was used because DOM (Document Object Model) can take whole document as a document model and allows flexible parsing of that model.

DOM [54] is an application programming interface (API) for well-formed XML documents. It is used to manage data. It is platform and language independent interface and allows programs and scripts to dynamically access and update the content. Using it, one can traverse the XML file and creates the corresponding DOM objects. These DOM objects are linked together in a tree structure. It is a standard set of objects in order to access and manipulate HTML and XML documents.

The parser developed for this study takes the FeatureIDE's XML document and parses it to the FAMA's AFM format as seen in the Figure 5.1. As mentioned before FAMA can take XML, FM, FMF, AFM formats as input. These formats, except XML, are plain text formats. Our feature model has attributes; therefore it is an extended feature model. So AFM which is used for attributed models was used.

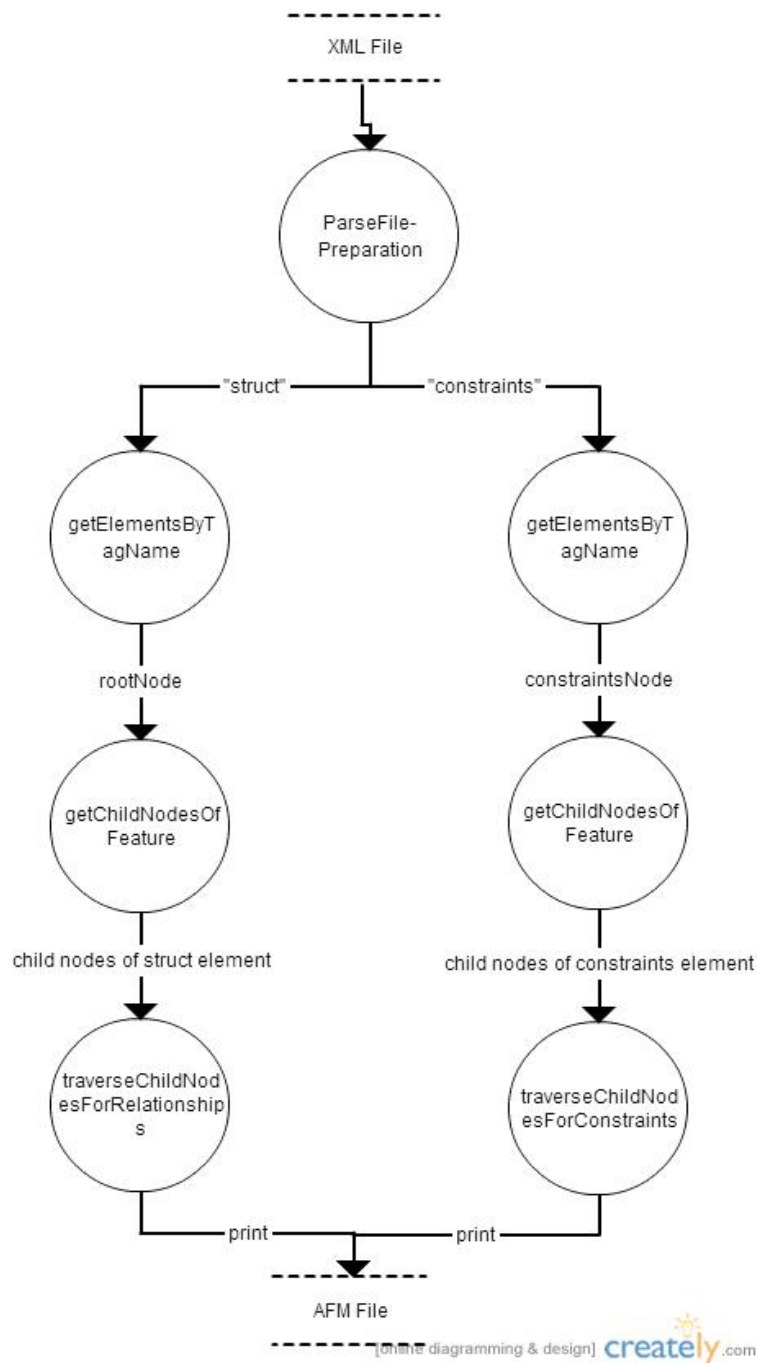


Figure 5.1: General Parser Process

We gave general information about tags using in FeatureIDE's XML format and their corresponding components in AFM File in that section. As mentioned before, DOM was used in parser. So, whole XML document was parsed as a Document Model. After that, necessary information was subtracted and transformed to the corresponding AFM components. In the Figure 5.1, general parser process is shown. As whole features and

parental relationships reside in the <struct></struct> tag, we subtracted these information using “getElementsByTagName("struct")”. Then the children of the nodeList which held these data traversed iteratively. While traversing each child, the feature type and parental relationships were identified inspecting the tag name (and, or, alt, feature) and processed accordingly. For example, as mentioned before, when feature-a is parent feature and it has two child namely b and c which has *or-relationships*, these parsed information is translated as a: [1,2] {b,c}. Also attributes were identified as they were defined as features in the FeatureIDE.

As shown in the Figure 5.1, In order to parse constraints, “getElementsByTagName("constraints")” method was used. By using it, all constraints which were in the <rule> tag were fetched. The nodeList which held constraint data traversed iteratively. While traversing each child, constraint types identified inspecting the tag name (imp, eq, conj, disj, not) and processed accordingly converting them to their AFM Format.

Below we presented some diagrams which was drawn in a drawing tool[55] in order to give a clear understanding of the FeatureIDE’s XML format and its corresponding component which is in AFM format derived using parser.

In Figure 5.2, general representation of transformation can be seen. In the left, there is FAMA’s XML format. Features with their relationships including *mandatory-relationships*, *optional-relationships* *or-relationships*, *alternative-relationships* reside between the tag <struct> and </struct>; whereas constraints reside in the tag <constraint>. Each constraint is presented with the tag <rule>. As AFM format needs to take relationships after %Relationships identifier and constraints after %Constraints identifier; these two components placed appropriate places in the AFM file via parser.

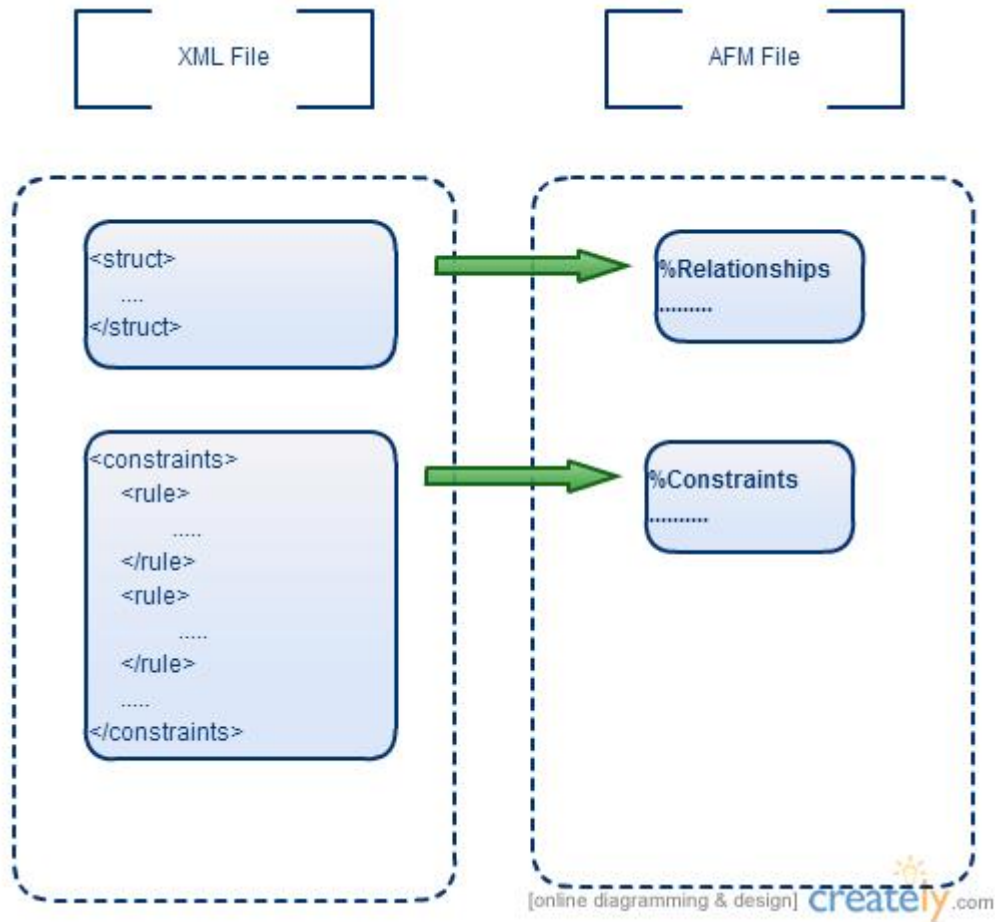


Figure 5.2: General Representation of Transformation

5.1 Relationships

In order to enter into details, *mandatory-relationships* and *optional-relationships* are shown in Figure 5.3. As seen in the Figure 5.3, at the left there is a variety of example trees which shows these relationships. Their corresponding XML formats in FeatureIDE are shown in the middle section which is denoted by [XML File] in FeatureIDE's XML format. Their translated versions (via parser), are shown in the right. As one can see, "a" is parent feature; whereas "b", "c", and "d" are child features and also leaf nodes of the tree. *Mandatory-relationships* and *optional-relationships* are shown with the `<and>` tag. The relationship tag also has an element ("name") which shows the name of the parent feature. Leaf nodes in the tree are denoted by `<feature>` tag in the XML File. The parent can have as much child feature as needed. This tag also has the element "name" in order to shows the name of the leaf node feature. There is also feature type which is mandatory or optional. To mandatory feature, a black cycle is used in the tree; whereas to optional one an empty circle is used. In XML File, it is denoted with the "mandatory" element. The translated versions of these

components are shown at the right. As seen, optional features are demonstrated between square brackets.

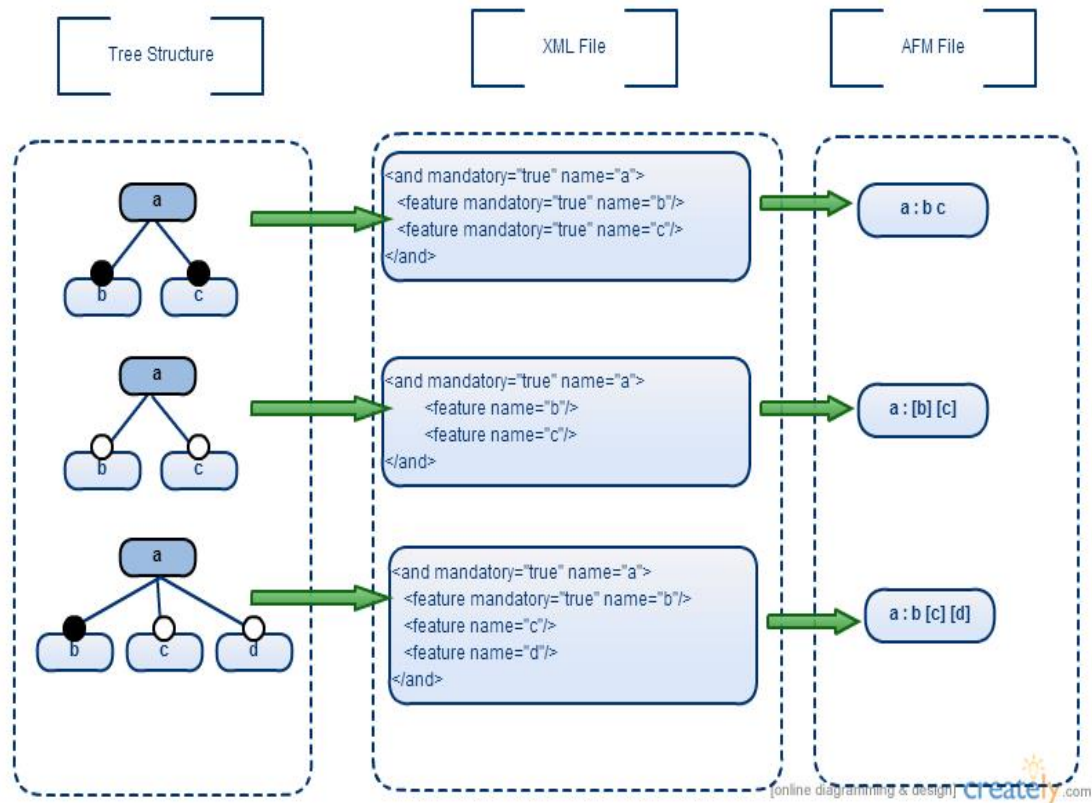


Figure 5.3: Mapping of Mandatory and Optional Relationships

Figure 5.4 depicts the *or-relationships*. As seen in this Figure, at the left there are example trees with different child sizes. The key point of detecting this tree has an *or-relationship* is the black circle-like figure which is below parent feature. Their corresponding XML formats in FeatureIDE are again shown in the middle section which is denoted by [XML File] in FeatureIDE's XML format. Their translated versions (via parser), are shown in the right. In the FeatureIDE *or-relationships* are shown with the `<or>` tag. The relationship tag also has an element ("name") which shows the name of the parent feature. Leaf nodes in the tree are denoted by `<feature>` tag in the XML File. Again parent in this tree can has as much children as needed. This tag also has the element "name" in order to shows the name of the leaf node feature. This part is same as the *mandatory-relationship* and *optional-relationships* explained earlier. However there is one point which is different in the *or-relationships*. There is no circle which shows mandatory or optional feature, in the children. The reason for this situation is that FeatureIDE always regards children in the *or-relationships* as mandatory features. As seen in the XML File, they have mandatory tag whose value is true.

The translated versions of these *or-relationships* components have a different presentation. Firstly, children have to be in curly brackets. Secondly before curly brackets are opened, [1,x] should be written. The x in here represents the number of children of parent feature.

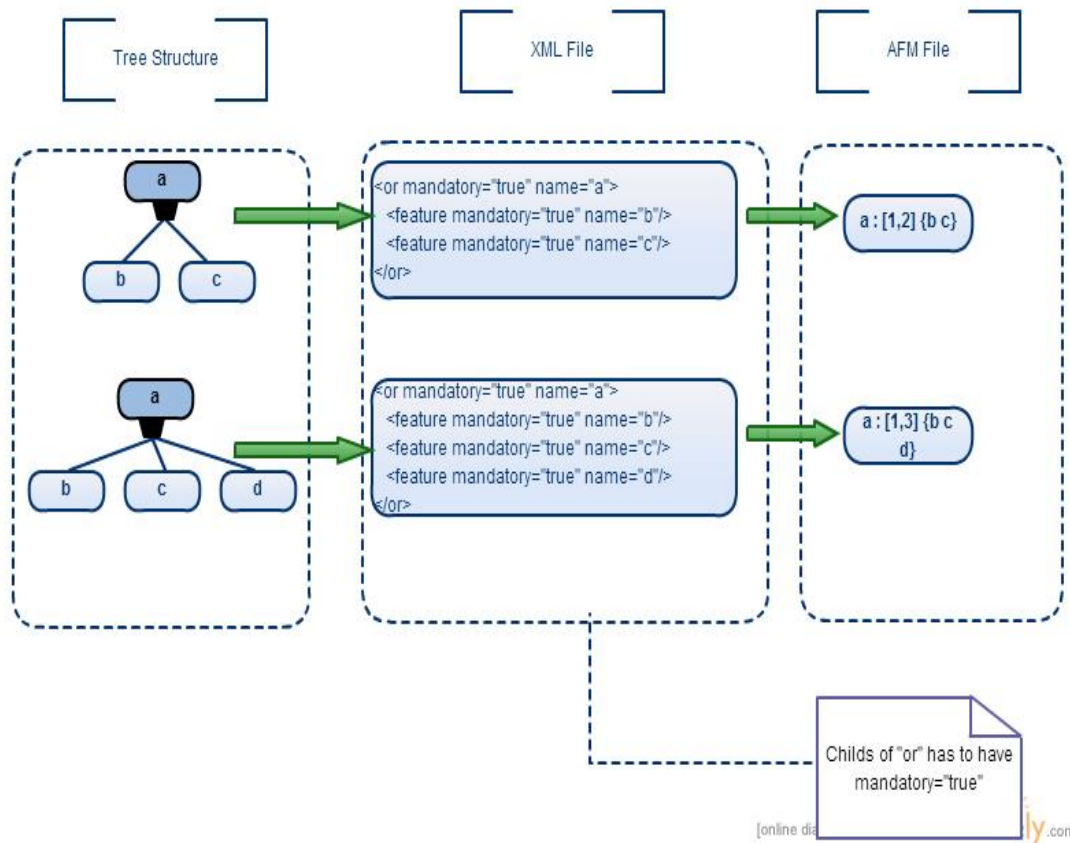


Figure 5.4: Mapping of Or-Relationships

The components which have *alternative-relationships* are shown in the Figure 5.5. They are similar to *or-relationships* except for a few differences. For example, the circle-like figure which is in the below of parent feature is empty (hollow) for these kind of relationships. Also the XML tag which denotes it is `<alt>`. Lastly, for [1,x] indicator which resides before curly brackets, x is always 1 in here as in *alternative-relationships* parent can have only one children at a time.

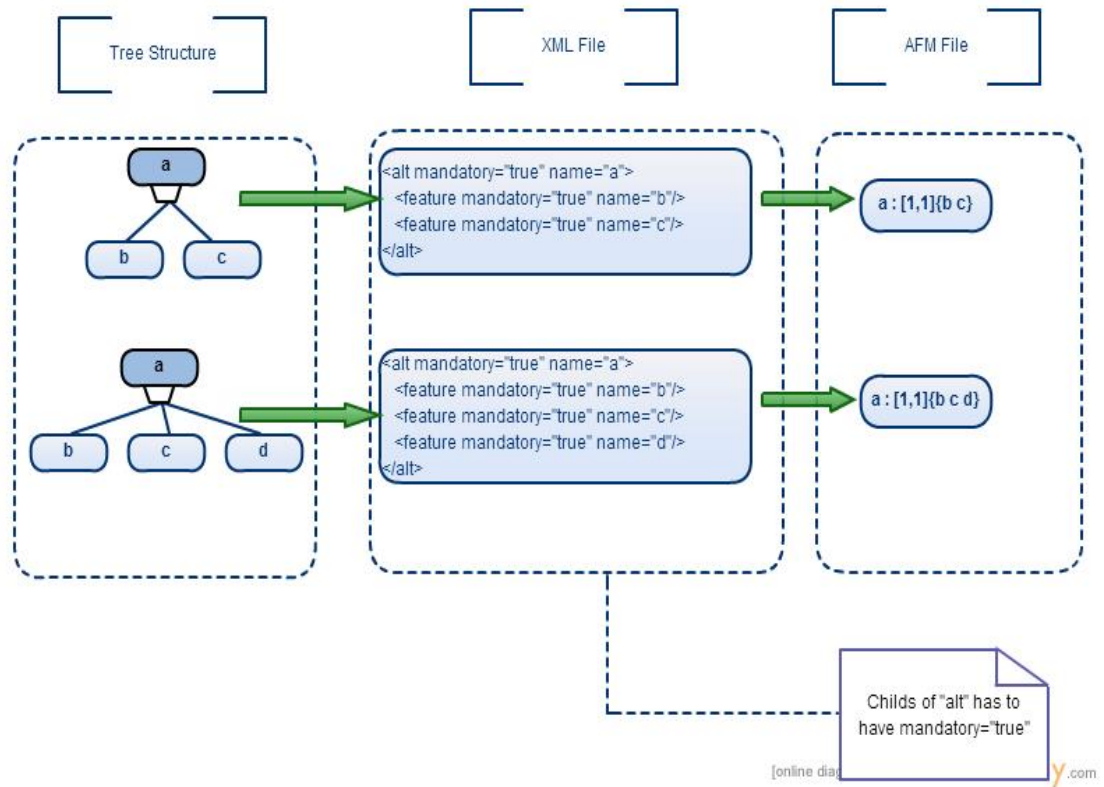


Figure 5.5: Mapping of Alternative-Relationships

At this point, simple decompositional relationships were shown. However in our model, all of these relationships are presented together. Therefore in the Figure 5.6, an example tree structure with complex relationships is presented to understand translation better. All relationships which mentioned before are shown together. In the example model, “a” is the root feature. Child features of “a” have *optional-relationships* except feature-b. Feature-b is mandatory feature and feature-c, feature-d, feature-e are optional ones. Although feature-c is a child feature of feature-a, it is also a parent whose child features are “c1”, “c2” and “c3”. Therefore it is denoted by `<or name="c">` tag. One of these child features, namely “c1” is also a parent. Its children have *alternative-relationships*. As mentioned before, leaf features, namely “b”, “d”, “e”, “c2”, “c3”, “c11”, “c12”, “c13”, are denoted by `<feature>` tag. Via parser, they translate accordingly. As seen in the right side of the Figure 5.6, in AFM File, each parent-child feature relationships are written in a new line.

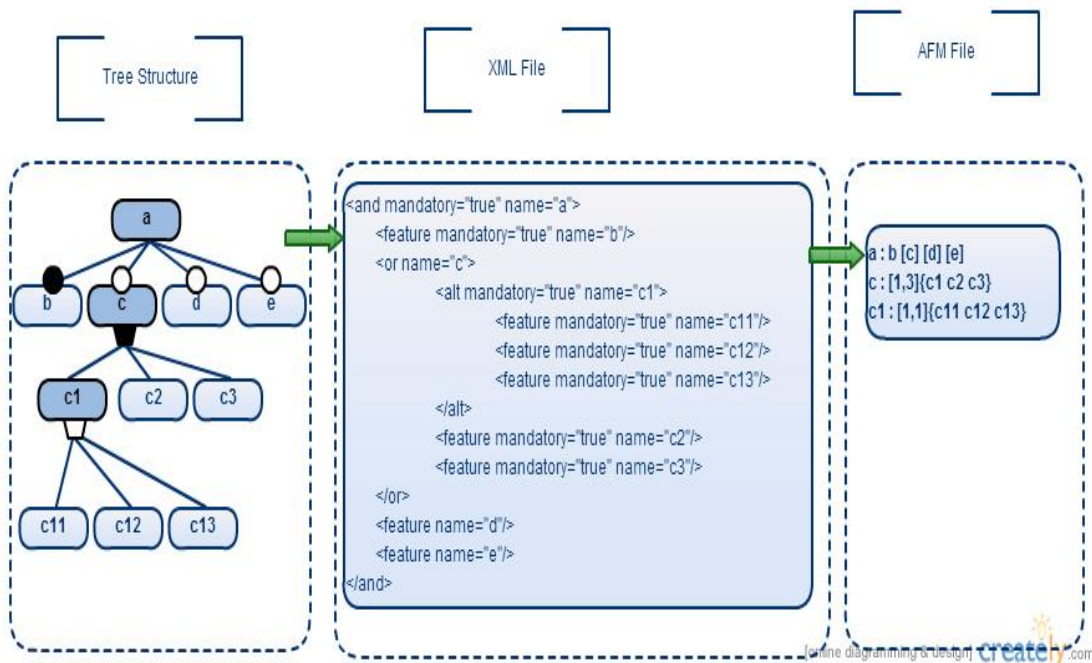


Figure 5.6: Mapping of Complex Relationships

5.2 Constraints

Feature Models have also constraints which define cross-tree dependencies. As mentioned before, each constraint is written between `<rule>` `</rule>` tag. However there are also tags which show the type of constraint. These are `<eq>` and `<imp>` tags. The features which constrain uses, there is `<var>` tag. As presented in the Figure 5.7, first one is translated as IFF. This means that a product in that family line can has feature-a if and only if feature-b is also presented in that product. For the rule `<imp>`, it translates as IMPLIES rule meaning, if feature-a exists to be true in a product, feature-b is also exists.

Although they do not count as constraints alone, there are some conditional operators which can be used as a part of a constraint. These are conjunction, disjunction and not. For FAMA, these are translated as and, or, not respectively.

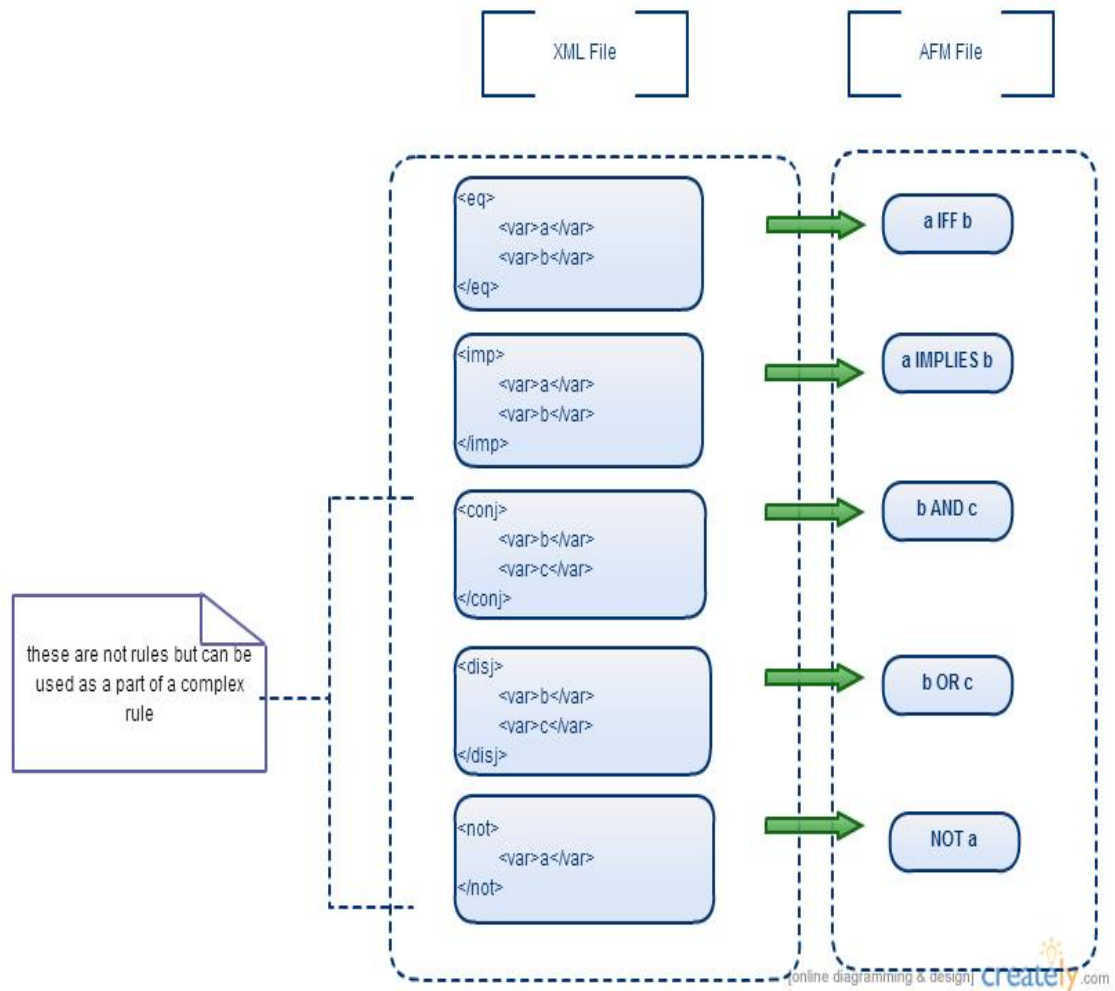


Figure 5.7: Mapping of Constraints

Using conditional operators mentioned above and rules, complex constraints can be defined as seen in the Figure 5.8. Constraints are always in the form of “x IMPLIES y” or “x IFF y”. However x and y can contain as much operators as needed. For example x be ((a OR b) OR b) and y can be ((a AND b) AND (NOT c)).

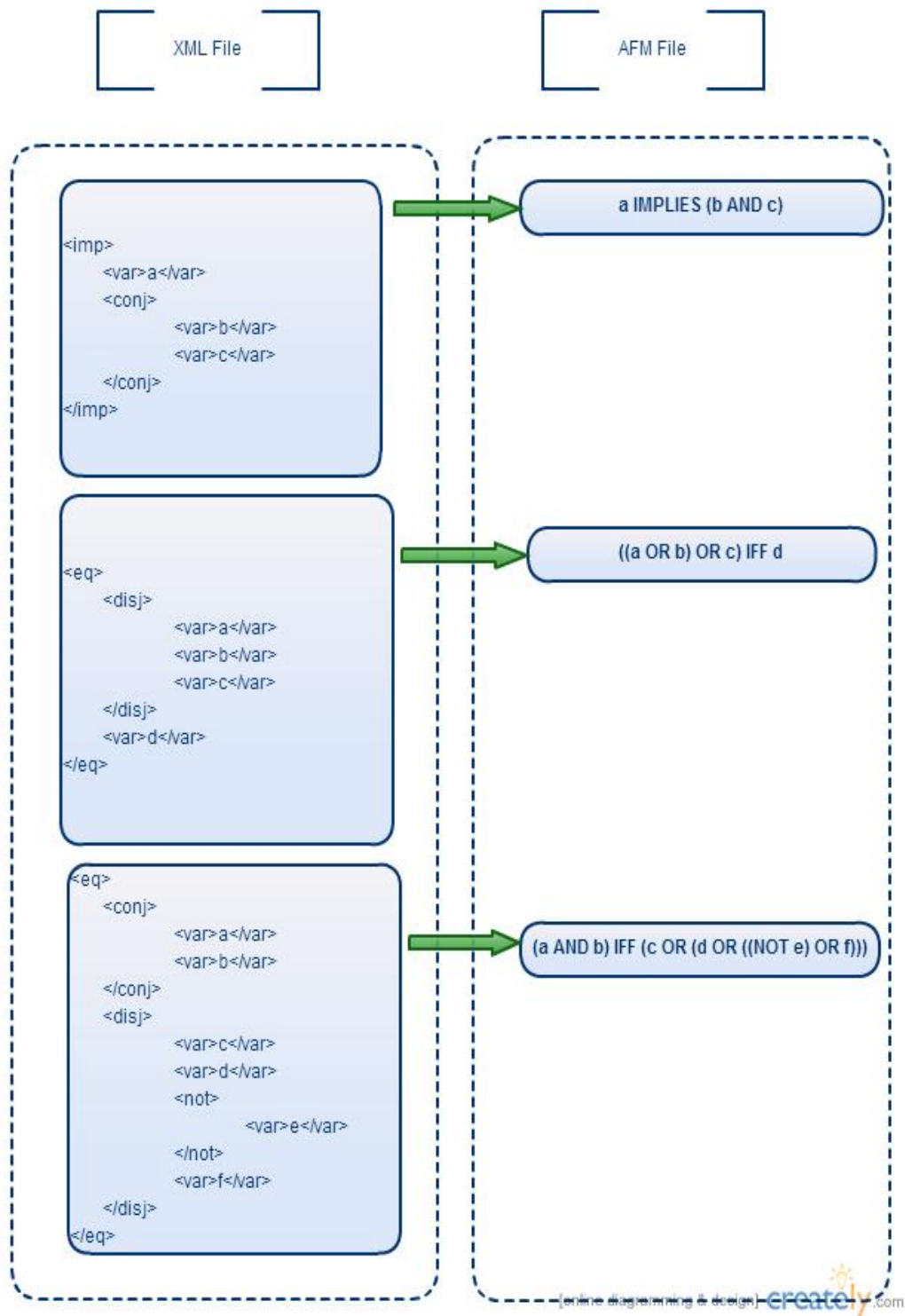


Figure 5.8: Mapping of Complex Constraints

5.3 Attributes

For the last part, feature attributes were identified. As explained before, a child feature of attributed feature was defined as a variation point. According to our naming convention, the name of this child features start with “vp”. Actually the reason of this was to indicate that its child is attributed feature. Therefore when parsing, this naming convention was utilized. The value of the attribute was subtracted taking the number after the double underscore character. The attribute value was written after the line “%Attributes”. According to FAMA’s format, in %Attributes, feature’s attributes are defined with their type, domain, default value and null value which is the value when feature is not selected. It can have more than one domain. Syntax is as follows:

FeatName.AttName: Type[min1 to max1],..., [minN to maxN], defaultVal, nullVal;

The parser also holds the parent and grandparent of the attributed feature. For the default value, if feature has more than one value, parser takes the first one as default. If the domain is a range, parser calculates default value by taking average value of that range.

To more detailed information about how the translation was done, pseudo codes of parser and their flow chart diagrams are presented in the Appendix-A and Appendix-B.

CHAPTER 6

ANALYZING THE NEWGENTV FEATURE MODEL

A tool is needed to analyze Feature Models when dealing with Software Product Lines. Model can be get too big and complicated when dealing with SPL and features in SPL is based on the principle of mass customization. Feature models which used in real-life SPLs usually have hundreds of features and become complicated with lots of cross-tree relations. As it is too hard or sometimes not possible to do analyses with such big data, automated reasoning is desirable.

At this point, using FAMA Framework helps to make some complex analysis in those models. For instance, it can answer questions like “is this a valid car configuration” for a car SPL for the company. Also customers can benefits from those analyses when they want to know what configuration is the cheapest. Doing these tasks manually is takes too much time and error-prone. Therefore, we need FAMA to automate this process. Using FAMA one can only need a feature model and apply the necessary operations for it. Besides its comprehensive usage and various questions, FAMA analyses a system which is modeled with features with or without attributes.

FAMA can be used in four different ways namely, Shell, Web Service, OSGi, and standalone. We used FAMA standalone distribution and version 1.1.2 which is the latest one. It is simply a Java library which can be easily integrated in Java Projects. FAMA is a distributed framework that can be used as a Java library. Therefore it can be easily integrated in existing or new Java Projects. It has QuestionTrader interface which is simple and stable front-end Java interface, implementing a query-based interaction. In the Table 6.1, QuestionTrader methods and their jobs were presented.

Table 6.1: Question Trader

Method syntax	Comments
<code>addStagedConfiguration(Configuration c): void</code>	Fixes a configuration
<code>ask(Question q): PerformanceResult</code>	Executes an operation
<code>createQuestion(String s): Question</code>	Creates a new operation from its id
<code>createTransform(String s): IVariabilityModelTransform</code>	Creates a new transformation from its id
<code>getAvailableReasoners(String questionId): Collection<Reasoner></code>	Returns a set of reasoners that implement a chosen question
<code>getCriteriaSelector(String s): CriteriaSelector</code>	Returns a CriteriaSelector named as a passed string
<code>getCriteriaSelectorNames(): Iterator<String></code>	Returns an iterator with available criteria selectors
<code>getHeuristics(String): Map<String, Object></code>	Returns a set of available heuristics on FaMa
<code>getQuestionById(String id): Question</code>	Returns a question from its id
<code>getQuestionsId(): Iterator<String></code>	Returns all question ids
<code>getReasonerById(String id): Reasoner</code>	Returns a reasoned from its id
<code>getReasonersId(): Iterator<String></code>	Returns all reasoned ids
<code>getSelectedReasoner(): Reasoner</code>	Returns the fixed reasoned
<code>getVariabilityModel(): VariabilityModel</code>	Returns the current variability model
<code>openFile(String path): VariabilityModel</code>	Loads a model from a file
<code>setCriteriaSelector(CriteriaSelector sel): Boolean</code>	Fixes a CriteriaSelector on FaMa
<code>setSelectedReasoner(Reasoner r): void</code>	Fixes a reasoned on FaMa
<code>setHeuristics(String s): void</code>	Fixes a heuristic on FaMa

Features model can be analyzed via FAMA framework by extracting relevant information to support decision making in the course of SPL development. For example, with an analysis operation, core features of the SPL can be identified or can learn validity of a product or a non-finished product. It can validate feature selections and help the end-user by providing product configuration and suggesting corrections for invalid configurations.

Actually FAMA is not a reasoner but a framework that contains different Variability Management analysis tools. In run time, FAMA can select most efficient solver automatically considering the operation in use for the best performance.

6.1 FAMA Models

FAMA deals with Feature Models which are basically data structures in tree format with relationships and dependencies. There are two types of feature model that FAMA uses namely standard feature models and extended feature models.

Standard feature model is the simple feature model with features, parental and cross-tree relationships and dependencies. Extended feature models have all of the functionalities of a standard model and also have attributes. An attribute is like a value which is related to a feature. Therefore it should have a type and a domain which is basically a set of candidate values. In FAMA, integer values are supported in the attributes. Moreover attributes can be used in constraints. As our NewGenTV feature model has attributes, it is an extended feature model. However, when I tried to use this model in FAMA framework, I saw that doubles are not supported by FAMA. Contacting with the FAMA developers, it was confirmed. The reason of this none of the solvers can properly reason over them (infinite values between 0.1 and 0.2). Therefore, double values were converted to integer ones as a solution. As an example, we can give InputFreq_VHF and InputFreq_UHF features. The attribute value of InputFreq_VHF is a range of 48,25 - 463,25MHz. As featureIDE's naming convention do not accept "." or ","; i used "_" instead of these in order to show floating point as shown in the Figure 6.1: Double values in FeatureIDE. Now I do not need that as floating points are not allowed in FAMA. Therefore these values needed to be converted to kHz as 48250 - 463250 as shown in the Figure 6.2: Converted double values in FeatureIDE.

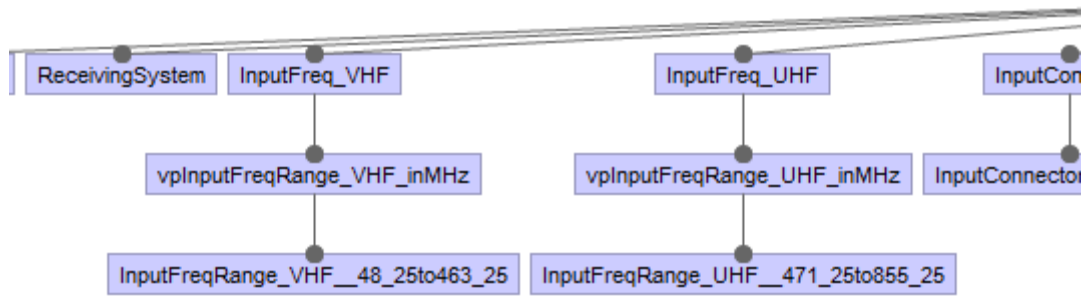


Figure 6.1: Double values in FeatureIDE

At this point, FeatureIDE's user friendly GUI was utilized. Because changes like this can be hard and provokes other modifications as it is a huge model and those components which needed to be changed can be used by other components like constraints. However, with FeatureIDE, I only changed feature name as I defined attributes as features such that "parent feature name" + "_" + "value". Then I run the parser once more to convert updated model's XML file to AFM file. After that I have the update feature model in AFM File without spending much time and money.

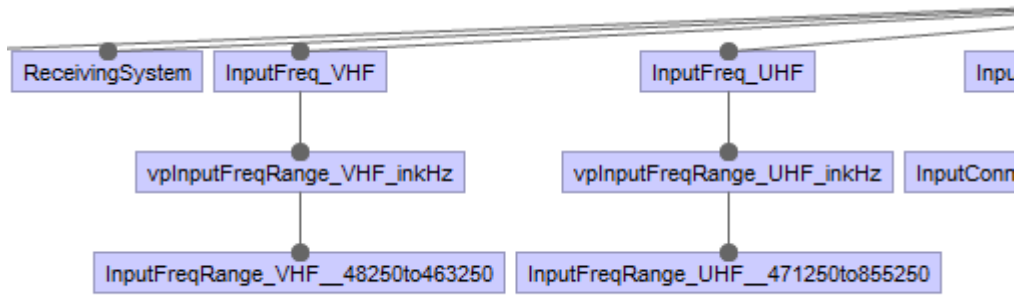


Figure 6.2: Converted double values in FeatureIDE

6.2 Input File Format

FAMA accepts two kind of input formats namely, XML and plain text.

XML format is the primitive format for FAMA. FAMA distributions [57] have a XML Schema named feature-model-schema.xsd that is attached to them. Extensions of this format can be .xml or .fama. Standard feature models can be defined with this format. As an example, we can give the model in XML Format in Figure 6.3 which was adapted from [56]:

```

<?xml version="1.0" encoding="UTF-8"?>
<feature-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="http://www.tdg-
seville.info/benavides/featuremodelling/feature-model.xsd">
<feature name="rootFeature">
<binaryRelation name="Option01">
<cardinality max="1" min="0" />
<solitaryFeature name="childFeature1" />
</binaryRelation>

<binaryRelation name="Option02">
<cardinality max="1" min="0" />
<solitaryFeature name="childFeature2" />
</binaryRelation>
</feature>
</feature-model>
  
```

Figure 6.3: XML Schema [56]

For the plain text format, we have .fm, .fmf, .afm extensions. Standard or extended feature models can be defined with this format. .fm and .fmf file extensions are used for the standard models whereas .afm file extensions are used for the extended models which are models with attributed features. As our feature model is extended feature model, we used .afm format. When defining feature models in .afm format, there are 3 different sections which needed to be considered. These are Relationships, Attributes and Constraints.

Tree structure, features and their relationships with their types are defined after the %Relationships identifier. First defined feature in the model is root feature. Each parent feature should be written in a new line obeying the rules. Children of parent feature are written after the parent name and “:” character. If child feature is not mandatory feature, it is written in brackets. As an example, a part of the FAMA model which contains root feature and its children can be given:

```
NewGenTVSPL: ProductProperties SideAndBackConnection AudioVideoSignalInformation
[AnalogFrontEnd] [DigitalFrontEnd] ICProperties MPEGTransportStreamAVDecoding
ConditionalAccessAndCommonInterface VideoAndGraphicsProcessing AudioProcessing
PictureFormats [TVApplications] USBVideoProcessing USBApplications
USBSupportedFiles MenuLanguages SupportedDisplayTypes PowerSetting
TargetCertificationsDVBT Approvals TargetOperatorsDVBC PanelSize DLNA;
```

The children of the root feature as shown above has *and -relationship* which are the default relationships. *Or-relationships* and *alternative-relationships* are defined with cardinality and a set of features such that [n,m]{Feature01 Feature02...}. For *alternative-relationships* n and m are 1 whereas for *or-relationships* n is 1 and m is the child number of parent feature.

Naming convention of FAMA for features are same as the FeatureIDE such that characters which can be used in the feature name are limited to java identifiers in FAMA.

%Attributes identifier is for the feature attributes. In that section of the afm file, feature's attribute, its type, domain, default value, and null value are written. Null value is the value of the attribute when related feature is not selected whereas default value is the one when the feature is selected. Domain is basically the values the attribute can take. If the domain consists of discrete number, its syntax is as follows:

```
FeatureName. AttributeName: [value1, value2,..., valueN], defaultValue, nullValue;
```

However when the domain is range, the syntax:

```
FeatureName. AttributeName: Type[min1 to max1],..., [minN to maxN], defaultValue,
nullValue. However as this syntax is given in FAMA user manual as Type[min1 to max1,...,
minN to maxN], model did not worked in FAMA at first. This problem was solved by
communicating with the FAMA developers.
```

As an example to attribute, we can give attribute information of PanelSize feature in NewGenTV feature model: PanelSize.vpPanelSize: [32,37,40,42,46,47,50,52,55,56], 32, 0;

Cross-tree constraints and dependencies are defined after the %Constraints identifier. In that section, we mainly used IFF and IMPLIES operators with and, or, not operators. For example:

```
DVB_S2 IFF (TransportStreamDVBS2 AND (ProfileLevelDVBS2
AND (AudioDecodingDVBS2 AND (AudioModeDVBS2 AND(SamplingFrequencyDVBS2
AND (DataRatesDVBS2 AND (DigitalFrontEnd_DVBS2 AND (Demodulation_DVBS2
AND SignalLevel_DVBS2)))))))));
```

6.3 Using FAMA Framework

There are three main classes which FAMA Framework uses for analysis. These are QuestionTrader, GenericFeatureModel and Question.

QuestionTrader as seen in the Table 2.1, is the main interface of the tool which has basic FAMA functionalities such as ask(Question q) and createQuestion(String s). GenericFeatureModel, on the other hand, is an abstraction for our model. It can also be created in FAMA, but we already had our feature model extracted from FeatureIDE. For the feature model, GenericAttributedFeatureModel which was a version of GenericFeatureModel for the extended models was used in our work as we had attributed features. And finally, Question class is the class which holds all analyses operations.

6.3.1 Working with Models

The first step of using FAMA Framework in order to do analysis is loading a model. Before that we need some preparation.

First thing is to create QuestionTrader object which will be needed for main FAMA functionalities. It was created such that:

```
QuestionTrader qt = new QuestionTrader();
```

After that our model which was generated using FeatureIDE and converted to the afm format was loaded. When loading file, openFile(String s) which is QuestionTrader's method, was used. It returns a VariabilityModel which has subclasses GenericFeatureModel and GenericAttributedFeatureModel. GenericAttributedFeatureModel was used as our model is attributed feature model.

```
GenericAttributedFeatureModel afm = (GenericAttributedFeatureModel)
```

```
qt.openFile("C:/Users/gulseren/Desktop/model.afm");
```

```
qt.setVariabilityModel(afm);
```

After setting the variability model to QuestionTrader, we asked questions. However before that, createQuestion method was used with the necessary identifier which was found in FAMAconfig.xml and returned desired question. Finally question was asked.

```
ValidQuestion vq = (ValidQuestion) qt.createQuestion("Valid");  
  
qt.ask(vq);  
  
vq.isValid();
```

6.4 FAMA Operations

FAMA makes analysis on feature models using its operations. Each operations has different specialty.

6.4.1 Validation

This operation determines if the model is valid. “It checks if a model is not empty, or in other words, it has at least one product” [56]. It is called “*void feature model*” in [63].

Methods of the operation:

- isValid(): It does not take any input parameter and returns boolean as the output parameter. This boolean represents the validity such that if it is true, model is valid and otherwise it is not valid.

Supported reasoners for that operation are Choco, ChocoAttributed, Sat4j, JavaBDD and JaCoP. However, only Choco provides both standard and extended model support for that operation. Others support just standard models.

This is the simplest analysis operation. It uses “ValidQuestion” interface. Therefore, in order to check validity of the model, simply “ValidQuestion” was asked in our analysis such that:

```
ValidQuestion vq = (ValidQuestion) qt.createQuestion("Valid");  
  
qt.ask(vq);  
  
System.out.println(vq.isValid());
```

As our feature model is not empty, in other words it has at least one product, vq.isValid() returns true. By doing this, we learned that the model is valid.

6.4.2 Products

This operation “calculates all valid products of a feature model” [56]. It enables accessing to the set of products of the feature model. This operation is called “*all products*” in [63].

Methods of the operation:

- `getNumberOfProducts()`: It returns the total number of products for the model.
- `getAllProducts()`: It returns a collection which has all the possible products. A product have a list of features that one can access with `getFeature(int index)` method.

Output parameter for that operation is the list of valid products. It does not take any input parameter. Supported reasoners for that operation are Choco, Sat4j, JavaBDD and JaCoP. All of them support just standard models. This operation uses “ProductsQuestion” interface.

The usage of this Question is:

```
ProductsQuestion pq = (ProductsQuestion) qt.createQuestion("Products");
```

```
qt.ask(pq);
```

```
long imax = pq.getNumberOfProducts();
```

```
Iterator<? extends GenericProduct> it= pq.getAllProducts().iterator();
```

However, as told earlier, this question is not applicable to our model as it is an extended feature model. But products and number of products are still can be analyzed. Although it is not documented in the FAMA User Manual [56], Choco Reasoner has a question named “ChocoProductsQuestion” for attributed feature models. Therefore a transformation was needed. At this point, `transformTo` method was used. So instead of using `QuestionTrader()`, we used `ChocoReasoner()` and asked `ChocoProductsQuestion` like that:

```
ChocoReasoner r = new ChocoReasoner();
```

```
fm.transformTo(r);
```

```
ChocoProductsQuestion pq= new ChocoProductsQuestion();
```

```
r.ask(pq);
```

This question has also `getNumberOfProducts()` and `getAllProducts()` methods.

```
long imax = pq.getNumberOfProducts();
```

```
Iterator<? extends GenericProduct> it = pq.getAllProducts().iterator();
```

At this point, first using `ChocoProductsQuestion` interface, question was asked. Then using `getNumberOfProducts()` method, maximum product number that our model can have be calculated. In the same way, `getAllProducts()` was used to extract those products as a Java

Collection. At this point, we can get each product by a loop and can make other analyses. For instance, we can check validity of each product by ValidProduct Question. But this part will be explained in the Valid ProductQuestion section.

However, for this question, we were faced with a problem. As number of features in our model is too high, all possible products that can be extracted from the model can be tremendous. Actually it is impossible with a simple PC. To demonstrate this, there are analyses results of this question using our model with reduced number of features that presented in the Table 6.2 and Figure 6.4. In Figure 6.4, y-axis shows number of products.

Table 6.2: Products Analysis

Input file	# of features	# of core features	# of variant features	Time spent in seconds	# of products
Out24.afm	24	21	3	1 seconds	8
Out25.afm	25	22	3	1 seconds	8
Out27.afm	27	22	5	1 seconds	24
Out27v2.afm	27	22	5	1 seconds	16
Out27v3.afm	27	22	5	1 seconds	32
Out35.afm	35	30	5	1 seconds	24
Out40.afm	40	30	10	1 seconds	768
Out45.afm	45	35	10	1 seconds	768
Out50.afm	50	35	15	2 seconds	23808
Out55.afm	55	40	15	3 seconds	23808
Out60.afm	60	40	20	5 seconds	119040
Out61.afm	61	41	20	6 seconds	119040
Out62.afm	62	41	21	9 seconds	238080
Out63.afm	63	42	21	10 seconds	238080
Out64.afm	64	42	22	15 seconds	396800
Out64v2.afm	64	42	22	18 seconds	476160
Out65.afm	65	43	22	16 seconds	396800

Table 6.2 (Continued)

Out66.afm	66	43	23	28 seconds	714240
Out67.afm	67	44	23	29 seconds	714240
Out68.afm	68	44	24	57 seconds	1349120
Out69.afm	69	44	25	88 seconds \cong 1 minutes	2063360
Out70.afm	70	44	26	11 minutes	3491840
Out71.afm	71	44	27	More than 6 hours	

Out24.afm is consists of 21 mandatory features and 3 optional features. Adding one more core feature, Out25.afm was constructed. For the Out25.afm input, number of products remained the same.

Out27.afm was constructed by adding 2 variant features which has *or-relationships* to the Out25.afm. For this input, number of products became 24.

Out27v2.afm was constructed by adding 2 variant features which has *alternative-relationships* to the Out25.afm. For this input, number of products became 16.

Out27v3.afm was constructed by adding 2 variant (optional) features which have *optional-relationships* to the Out25.afm. For this input, number of products became 32.

Out35.afm was constructed by adding 8 core feature to the Out27.afm. Number of products remained the same.

Out40.afm was constructed by adding 5 optional features which has *optional-relationships* to the Out35.afm. For this input, number of products became 768.

Out45.afm was constructed by adding 5 core feature to the Out40.afm. Number of products remained the same.

Out50.afm was constructed by adding 5 variant features which has *or-relationships* to the Out45.afm. For this input, number of products became 23808.

Out55.afm was constructed by adding 5 core feature to the Out45.afm. Number of products remained the same.

Out60.afm was constructed by adding 5 variant features which has *alternative-relationships* to the Out55.afm. For this input, number of products became 119040.

Out61.afm was constructed by adding 1 core feature to the Out60.afm. Number of products remained the same.

Out62.afm was constructed by adding 1 optional feature which has *optional-relationships* to the Out61.afm. For this input, number of products became 238080.

Out63.afm was constructed by adding 1 core feature to the Out62.afm. For this input, number of products became 238080.

Out64.afm was constructed by adding 1 optional feature which has *optional-relationships* (and whose parent is also optional feature) to the Out63.afm. For this input, number of products became 396800.

Out64v2.afm was constructed by adding 1 optional feature which has *optional-relationships* (and whose parent is also mandatory feature) to the Out63.afm. For this input, number of products became 476160.

Out65.afm was constructed by adding 1 core feature to the Out64.afm. For this input, number of products became 396800.

Out66.afm was constructed by adding 1 optional feature which has *optional-relationships* to the Out65.afm. For this input, number of products became 714240.

Out67.afm was constructed by adding 1 core feature to the Out66.afm. For this input, number of products remained the same.

Out68.afm was constructed by adding 1 optional feature which has *optional-relationships* to the Out67.afm. For this input, number of products became 1349120.

Out69.afm was constructed by adding 1 optional feature which has *optional-relationships* to the Out68.afm. For this input, number of products became 2063360.

Out70.afm was constructed by adding 1 optional feature which has *optional-relationships* to the Out69.afm. For this input, number of products became 3491840.

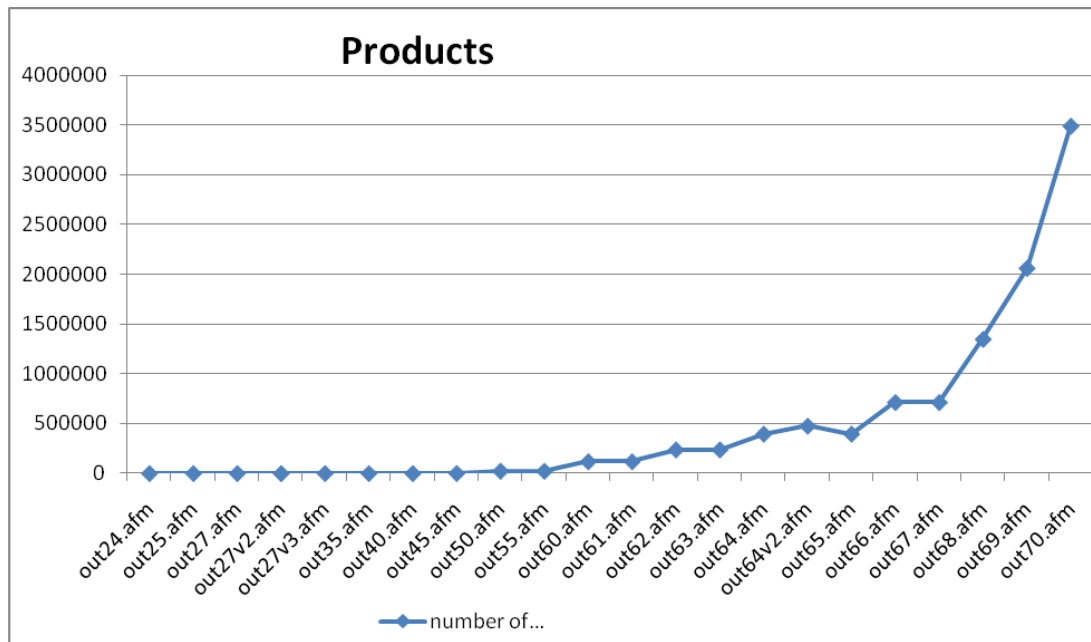


Figure 6.4: Products Analysis Diagram

In this table, first a very reduced version of our feature model was used as input. This version just contains root feature and its children. As this model has 3 variant feature (optional feature) and those variant features can be existed in the model (or not existed at all) in 8 different combinations, number of different products that the model can have is 8. This analysis can also be done manually. However when the model gets a bit more complicated, as seen in the table, # of products increases exponentially.

Another thing that can be deduced from this analysis is that when we just increased variant feature –added a variant feature to the model-, both time spent and number of products were increased whereas when we increased core feature number, time spent was increased but number of products did not change.

Moreover variant feature can be a child with *or/and/alternative-relationships*. As an example, we can give out27.afm. As told earlier, we added two variant features to the out25.afm. When those variant features have *or-relationships*, number of products became 24. For out27v2.afm, when those variant features have *alternative-relationships*, number of products became 16. And for out27v3.afm, when those variant features have *optional-relationships*, number of products became 32. Therefore it can be deduced that adding variants which have *optional-relationships* increases product number most whereas adding variants which have *alternative-relationships* increases product number least.

Also parent of that added variant feature can also be variant feature or a mandatory feature. Each has different effect on product number. When one adds a feature whose parent is an optional feature, the increment in the number of products is less than adding a feature whose

parent is mandatory feature. As an example we can give Out64.afm and Out64v2.afm. First one was constructed adding a variant feature whose parent is optional feature and number of products for that is 396800 whereas second one was constructed adding a variant feature whose parent is mandatory feature and number of products for that is 476160 which is more than first one.

This question is not only gives number of products for a model but also calculates these possible products. So it is not possible to do these analyses manually. As you can see, for out70.afm model, it calculated all the 3491840 products. This analysis is impossible without using an automated analysis tool like FAMA.

As one see, even with the model of 70 features, our run time increased so much. When the model which has just one more feature than the previous one (out71.afm) was run, we saw that it takes more than 6 hours. Therefore we ended run. So it is not efficient to analyze with this question to the more crowded models.

6.4.3 Number of Products

This operation “calculates the number of products” [56]. It is also called as “*variation degree*” in the literature as [63].

Methods of the operation:

- `getNumberOfProducts()`: It returns the number of products for the model.

Output parameter for that operation is the number of valid products in long type. It does not take any input parameter. Supported reasoners for that operation are Choco, Sat4j, JavaBDD and JaCoP. All of them support standard models. This operation uses “`NumberOfProductsQuestion`” interface.

The usage of this Question is similar to the Products Question:

```
ChocoReasoner r = new ChocoReasoner();
```

```
fm.transformTo(r);
```

```
ChocoNumberOfProductsQuestion pq = new ChocoNumberOfProductsQuestion ();
```

```
r.ask(pq);
```

```
double imax = pq.getNumberOfProducts();
```

Like Products, Choco Reasoner has a question named “`ChocoNumberOfProductsQuestion`” for attributed feature models. So, we used Choco version of number of products question because only that supports extended feature models. Again a transformation was needed. After that `ChocoNumberOfProductsQuestion` was asked.

For the analysis, inputs created for “Products” question was also used.

Table 6.3: Number of Products Analysis

Input file	# of features	# of core features	# of variant features	Time spent in seconds	# of products
Out24.afm	24	21	3	1 seconds	8
Out25.afm	25	22	3	1 seconds	8
Out27.afm	27	22	5	1 seconds	24
Out27v2.afm	27	22	5	1 seconds	16
Out27v3.afm	27	22	5	1 seconds	32
Out35.afm	35	30	5	1 seconds	24
Out40.afm	40	30	10	1 seconds	768
Out45.afm	45	35	10	1 seconds	768
Out50.afm	50	35	15	1 seconds	23808
Out55.afm	55	40	15	1 seconds	23808
Out60.afm	60	40	20	3 seconds	119040
Out61.afm	61	41	20	3 seconds	119040
Out62.afm	62	41	21	5 seconds	238080
Out63.afm	63	42	21	5 seconds	238080
Out64.afm	64	42	22	7seconds	396800

Table 6.3 (Continued)

Out64v2.afm	64	42	22	9 seconds	476160
Out65.afm	65	43	22	8 seconds	396800
Out66.afm	66	43	23	13 seconds	714240
Out67.afm	67	44	23	13 seconds	714240
Out68.afm	68	44	24	24 seconds	1349120
Out69.afm	69	44	25	37 seconds	2063360
Out70.afm	70	44	26	62 seconds \cong 1 minutes	3491840
Out71.afm	71	44	27	123 seconds \cong 2 minutes	6983680
Out72.afm	72	44	28	255 seconds \cong 4 minutes	1.396736E7
Out73.afm	73	44	29	489 seconds \cong 8 minutes	2.793472E7
Out74.afm	74	44	30	744 seconds \cong 12 minutes	4.190208E7
Out75.afm	75	44	31	1289 seconds \cong 21 min	6.98368E7

Table 6.3 (Continued)

Out76.afm	76	44	32	2241 seconds \cong 37 minutes	1.2570624E8
Out77.afm	77	44	33	4341 seconds \cong 72 minutes \cong 1 hours	2.3744512E8
Out78.afm	78	44	34	9220 seconds \cong 153 minutes \cong 2 hours	4.6092288E8
Out79.afm	79	44	35	17718 seconds \cong minutes 295 \cong 5 hours	9.2184576E8
Out80.afm	80	44	36	More than 6 hours	

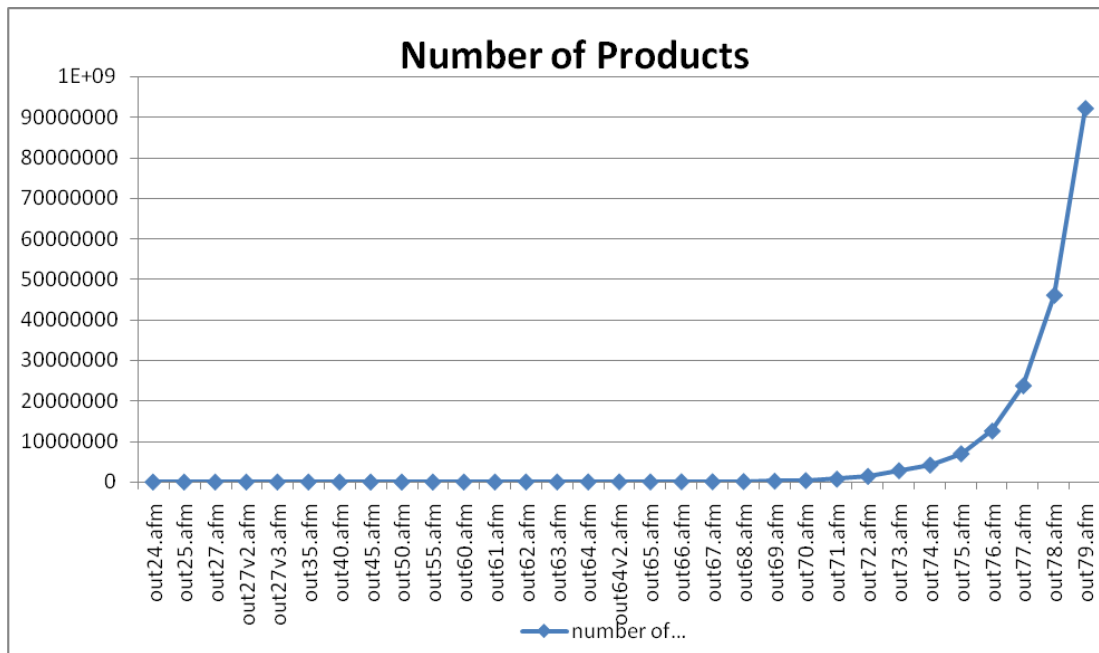


Figure 6.5: Number of Products Analysis Diagram

Results are exactly the same as you can see in the Table 6.3. So we can say, functionality of `getNumberOfProducts()` which is a method of `ChocoProductsQuestion` interface is the same as the method of `getNumberOfProducts()` method of `ChocoNumberOfProductsQuestion`. However the only difference is in the time spent as seen in the Figure 6.6, as calculating all possible products takes much more time than just calculating number of those products. Therefore analysis of number of products can be done with the models with more features as depicted in the Figure 6.5. However, when the variant feature number is increased at each input, time spent for run is also increased. As seen in the Table 6.3, for the `Out80.afm`, when it was run, it took more than 6 hours. So it is not efficient to analyze with this question to the more crowded models.

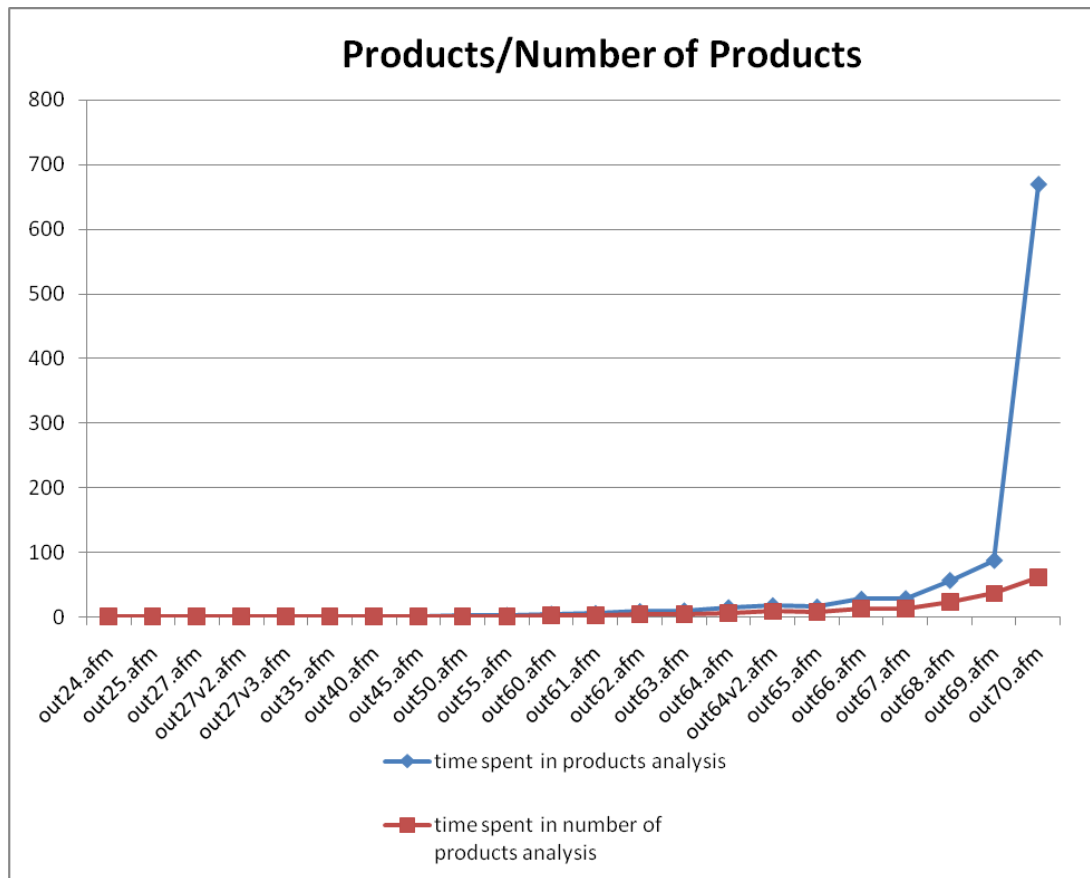


Figure 6.6: Products/Number of Products Run Time Analysis Diagram

6.4.4 Valid Product

This operation “determines if a product is valid for a given model” [56]. It “returns a value that determines whether the product belongs to the set of products represented by the feature model or not” [63].

Methods of the operation:

- `setProduct(Product p)`: It sets the product whose validity is to be calculated.
- `isValid()`: It returns a Boolean value such that true for valid product and false otherwise for that feature model. However before calling this method, it is necessary to call `setProduct(Product p)` method in order to specify product which we want to know its validity.

Output parameter for that operation is the product validity in boolean type. It takes a product whose validity is to be calculated as the input parameter. Supported reasoners for that operation are Choco, ChocoAttributed, Sat4j, JavaBDD and JaCoP. However, only Choco provides both standard and extended model support for that operation. Others support just standard models. It uses “ValidProductQuestion” interface.

The usage of this Question is:

```
Collection<GenericFeature> features = new ArrayList<GenericFeature>();

GenericAttributedFeature f = afm.searchFeatureByName("NewGenTVSPL ");

features.add(f);

f = afm.searchFeatureByName("ProductProperties");

features.add(f);

.....

Product p = new Product();

p.addAllFeatures(features);

ValidProductQuestion vcq = (ValidProductQuestion)qt.createQuestion("ValidProduct");

vcq.setProduct(p);

qt.ask(vcq);

vcq.isValid();
```

As shown in the code above, first, features that will construct the product to be analyzed were created. In that part, `searchFeatureByName(String featureName)` was used to extract the feature whose type is `GenericAttributedFeature`. After that a product object was created and previously created feature array was added to that product object. Now we had a product. However, we did not yet know that weather is a valid product or not. Therefore `ValidProductQuestion` object was created with the "ValidProduct" identifier and after setting our product to that question object, question was asked. `isValid()` method returns true if a product is possible with given features for the model and false for otherwise.

It is clear that to make this analysis to our model without using FAMA is error prone and time consuming as our model has lots of features and constraints. It is very hard to inspect whether or not a product is valid manually.

As an example, giving following features in the Table 6.4 as an input, we can analyze whether or not a product which contains these is valid. The input which shown in the Table 6.4 is organized in two columns namely Selected Parent Features and Selected Child Features in order to present the input in a more arranged way.

In the table, a feature can be both a child and a parent. For example, ProductProperties feature which is in the Selected Parent Features column in the Table 6.4 is also in the Selected Child Features columns for the NewGenTVSPL feature because ProductProperties is one of the children of NewGenTVSPL.

Table 6.4: Valid Product Analysis

Selected Parent Features	Selected Child Features
NewGenTVSPL	ProductProperties SideAndBackConnection AudioVideoSignalInformation DigitalFrontEnd ICProperties MPEGTransportStreamAVDecoding ConditionalAccessAndCommonInterface VideoAndGraphicsProcessing AudioProcessing PictureFormats USBVideoProcessing USBApplications USBSupportedFiles MenuLanguages SupportedDisplayTypes PowerSetting TargetCertificationsDVBT TargetOperatorsDVBC Approvals PanelSize DLNA
ProductProperties	HD Panel SkypeReady Cabinet RemoteControl OSD TouchKey OnOffSwitch OtelTV
SideAndBackConnection	AntennaInput FullScart S_VHS_Input

Table 6.4 (Continued)

	CVBS_Input AudioInput ComponentInput PC_AudioInput VGA HDMI_Input USB_Input Network_RJ45_Connection_Ethernet SPDIF HeadphoneOutput LineoutOutput CI_Connectors
AudioVideoSignalInformation	AnalogRF_Signal DigitalRF_Signal Y_C_signal RGB_FBsignal RGB_HsVs_Signal ProgressiveSignal HDMI_Signal CVBS_SignalOutput Stereo_L_R_AudioSignalInput Stereo_L_R_AudioSignalOutput SPDIFSignalOut AudioOutputPower Speakers Type
ICProperties	RAMPower CPUPower VideoPictureEnhancers AudioDecoder DSPClockRate SW_BASE VIF_SIF SECAM_Decoder Pre_SNR DNR Mosquito_Noise_Reduction De_BlockingNoiseReduction GaussianNoiseReduction SpikeNoiseReduction MotionAdaptiveDeinterlace EdgeOrientedDeinterlace FilmModeDetection CadenceFilmDetection_Blackline _2DPeaking ICC IHC IBC

Table 6.4 (Continued)

	YC_Delay CTI xvYCC_Support ScalingTaps ScalingDeRinging MPEGEncoder PreFiltering
MPEGTransportStreamAVDecoding	TransportStream ProfileLevel AspectRatio VideoResolution AudioDecoding
ConditionalAccessAndCommonInterface	CommonInterface
VideoAndGraphicsProcessing	MainPicture_Deinterlacing ChromaDecoding ChromaDecodingStandard HDMI_Resolution TxtPages Standart
AudioProcessing	StereoDecoding GermanA2_Nicam SRS
USBApplications	SupportedMenuLanguages USBConnection20
PowerSetting	PSUType InputRange StandByPowerConsumption
Approvals	SRS_Approvals CIPlus_DigitalTVLabs
VideoPictureEnhancers	ProACEAlgorithm
AudioDecoder	DualDSP_Structure
SW_BASE	Ecos_On_RISC Linux_On_MIPS
SECAM_Decoder	SECAM_Decoder_ImprovedThenS1Plus
DNR	DNR_Temporal_Noise_Reduction
GaussianNoiseReduction	SpatialGaussianNoiseReduction
SpikeNoiseReduction	SpatialSpikeNoiseReduction
MotionAdaptiveDeinterlace	_3DMotionAdaptiveDeinterlace
EdgeOrientedDeinterlace	AdaptiveEdgeOriented_2D_Deinterlace
MPEGEncoder	MPEG4Encoder
MainPicture_Deinterlacing	_3DFrameBasedMotionAdaptive
GaussianNoiseReduction	SpatialGaussianNoiseReduction
Standart	BG DK L I
RemoteControl	TP5

Table 6.4 (Continued)

OSD	Arcelik
Cabinet	Swan
Type	HybridCanTuner
Panel	LCD
DigitalFrontEnd	HierarchicalModes_DVBT TransmissionMode_DVBT Constellations_DVBT GuardInterval_DVBT InputFrequency_DVBT ChannelBandwidth_DVBT ConvolutionalCodeRatesDVBT InputLevelDVBC AerialInputImpedanceForDigital SignalLevel InputFrequency_IntermediateFrequency
HierarchicalModes_DVBT	Hierarchical
HD	FullHD HDReady
AntennaInput	DVB_T DVB_S
DigitalRF_Signal	S_Tuner T_Tuner
ChromaDecodingStandard	NTSC SECAM
ChromaDecoding	Digitized
PSUType	EXT
ProfileLevel	ProfileLevel_MPEG2
De_BlockingNoiseReduction	DynamicBlockingArtifactRemoval
TransmissionMode_DVBT	_8K
SupportedDisplayTypes	SingleTTLsupport
MenuLanguages	English Turkish
	FEC_MODE InputSymbolRate LNBpowerAndPolarization_DVBS2 SignalLevel_DVBS2

At this point, it is worth pointing out that when constructing a product as an input, FeatureIDE's configuration editor was used. Of course this task could be done manually, but using a graphical editor helps a lot. However as told earlier, FeatureIDE is not sufficient enough to create a valid product as there are also complex constraints and attributes that needed to be considered.

When the ValidProduct analysis was executed with the product stated above, valid question returned false value. The run time of the analysis is 1 seconds. By doing this analysis, we learned this product is not valid. However we have no idea what is wrong. Maybe some features needed to be added or removed from the product to make it valid. But it is not really efficient to analyze further the validity result manually. FAMA also provides a question for this job. With Explain Invalid Product Question, one can get help to fix this product. This will be explained in the Explain Invalid Product Explanation section.

6.4.5 Invalid Product Explanation

This operation “provides options to repair an invalid product for a given model” [56].

Methods of the operation:

- `setInvalidProduct(Product p)`: It sets an invalid product.
- `getSelectedFeatures()`: It returns features which are to be selected to repair invalid product along with `getDeselectedFeatures()` method.
- `getDeselectedFeatures()`: It returns features which are to be deselected to repair invalid product along with `getSelectedFeatures()` method.
- `getFixedProduct()`: It returns a repaired product. Actually this is the product which can be formed selecting features which return from `getSelectedFeatures()` method and deselecting features which return from `getDeselectedFeatures()` method.

Output parameter for that operation is a set of selection and deselections of features. It takes an invalid product as the input parameter. Supported reasoners for that operation are only Choco. It provides standard model support for that operation. It uses “`ExplainInvalidProductQuestion`” interface.

The usage of this Question for extended feature models:

```
Product p = new Product();
```

```
Collection<GenericFeature> features = new ArrayList<GenericFeature>();
```

```
GenericAttributedFeature f = afm.searchFeatureByName("NewGenTVSPL ");
```

```
features.add(f);
```

```
f = afm.searchFeatureByName("ProductProperties");
```

```
features.add(f);
```

.....

```
p.addAllFeatures(features);
```

This part is the same as Valid Product Question. After that “ChocoExplainInvalid AttProductQuestion” which is the attributed model version of ExplainInvalidProductQuestion interface was used.

```
ChocoExplainInvalidAttProductQuestion q=new ChocoExplainInvalidAttProductQuestion();
```

```
q.setInvalidProduct(p);
```

```
ChocoReasoner r = new ChocoReasoner();
```

```
afm.transformTo(r);
```

```
r.ask(q);
```

```
for(ExtendedConfigurationExplaining e: q.getProductExplaining()){
```

```
out.write("Initial Product " + e.getInitialProduct() + " ");
```

```
out.write("Selections " + e.getSelections() + " ");
```

```
out.write("Deselections " + e.getDeselections() + " ");
```

```
out.write("Fixed Product " + e.getFixedProduct() + " ");
```

```
}
```

q.getProductExplaining() gives the suggestion to fix the model. These are Initial Product, Selections, Deselections and Fixed Products.

As an example, fixing input which mentioned in valid product analysis, in the Table 6.4, can be given. From the valid product analysis, we know that this product is invalid. This time, with invalid product explanations, FAMA will give us the suggestions to make this product valid.

In this analysis, there is basically 4 different results namely, initial product, selections, deselections and fixed product. This is a suggestion. There can be more than one suggestion for an invalid product. In our example, I realized that produced suggestions were not unique. Therefore I traversed suggestions and made each suggestion unique. After that 6 different suggestions were produced. Our selections and deselections were as follows:

1. Selections [QPSK, SiliconTuner, DigitalFrontEnd_DVBS2, Demodulation_DVBS2]
Deselections [OtelTV, HybridCanTuner, NTSC]

2. Selections [QPSK, _3DCombFilter, SiliconTuner, DigitalFrontEnd_DVBS2, Demodulation_DVBS2]
Deselections [OtelTV, HybridCanTuner]
3. Selections [QPSK, SiliconTuner, DigitalFrontEnd_DVBS2, InteractiveOtelTV, Demodulation_DVBS2]
Deselections [HybridCanTuner, NTSC]
4. Selections [QPSK, _3DCombFilter, SiliconTuner, DigitalFrontEnd_DVBS2, InteractiveOtelTV, Demodulation_DVBS2]
Deselections [HybridCanTuner]
5. Selections [QPSK, SimpleOtelTV, SiliconTuner, DigitalFrontEnd_DVBS2, Demodulation_DVBS2]
Deselections [HybridCanTuner, NTSC]
6. Selections [QPSK, _3DCombFilter, SimpleOtelTV, SiliconTuner, DigitalFrontEnd_DVBS2, Demodulation_DVBS2]
Deselections [HybridCanTuner]

When first suggestion is inspected, following result can be driven:

DigitalFrontEnd_DVBS2 features should be added because children of this feature are in the input product and if a child is in the product, parent also has to be existed in the product. In our model these children are SignalLevel_DVBS2, LNBpowerAndPolarization_DVBS2, InputSymbolRate, FEC_MODE and their children and children of children if available.

After adding DigitalFrontEnd_DVBS2 to the model, Demodulation_DVBS2 feature should also be added because of the tree relationships such that if a parent feature is in the product, mandatory children also have to be existed in the product. Demodulation_DVBS2 also has children which have *or-relationships* between them. Therefore at least one of the children should be added to the model. In the first suggestion, this was QPSK.

As a last addition, Silicon_Tuner feature was added to the model because of the constraint “DVB_S IFF Silicon_Tuner”. Therefore if DVB_S is existed in the product, Silicon_Tuner should also be existed and vice versa.

There is also deselections. First deselection is OtelTV. In our invalid product, we have OtelTV however we do not have either of its children which are InteractiveOtelTV and SimpleOtelTV. So we remove OtelTV. We could also add children of the OtelTV to the model but this is a different suggestion.

HybridCanTuner should also be deselected as the product does not have the DVB_C feature and it has the constraint “(DVB_C AND DVB_T) IFF HybridCanTuner”.

Lastly, NTSC should be deselected because of the constraint “(PAL OR NTSC) IMPLIES _3DCombFilter”. _3DCombFilter is not existed in the products, so this constraint should not be true if it is false in this constraint.

As seen, after the features which are in the Selections are added to the initial product which is the product to be analyzed and the features which are in the Deselections are removed from the initial product, fixed product was formed. With this analysis invalid products can be easily fixed, using the suggestions of this question in a short time. Run time of the analysis with our feature model and with this invalid product is 10 seconds.

6.4.6 Valid Configuration

This operation is similar to the Valid Product Method. The only difference is that Valid Configuration can evaluate both products and non-finished products. It is called “valid partial configuration” in literature and “returns a value informing whether the configuration is valid or not, i.e. a partial configuration is valid if it does not include any contradiction” [63].

Methods of the operation:

- `setConfiguration(Configuration p)`: It sets the configuration whose validity is to be calculated.
- `isValid()`: It returns a Boolean value such that true for valid configuration and false otherwise for that feature model. However before calling this method, it is necessary to call `setConfiguration(Configuration p)` method in order to specify configuration which we want to know its validity.

Output parameter for that operation is the configuration validity in boolean type. It takes a configuration whose validity is to be calculated as the input parameter. Supported reasoners for that operation are Choco, ChocoAttributed, Sat4j, JavaBDD and JaCoP. However, only Choco provides both standard and extended model support for that operation. Others support just standard models. It uses “ValidConfigurationQuestion” interface.

The usage of this Question is:

```
Configuration p = new Configuration();
```

```
GenericAttributedFeature f = afm.searchFeatureByName("NewGenTVSPL ");
```

```
p.addElement(f,1);
```

```
f = afm.searchFeatureByName("ProductProperties");
```

```

p.addElement(f,2);

.....

ValidConfigurationQuestion vcq =

(ValidConfigurationQuestion) qt.createQuestion("ValidConfiguration");

vcq.setConfiguration(p);

qt.ask(vcq);

vcq.isValid();

```

As shown in the above, first, a configuration object was created. After that, features that will construct the configuration to be analyzed were created and added to that configuration object. In that part, `searchFeatureByName(String featureName)` was used to extract the feature whose type is `GenericAttributedFeature`. Now we had a configuration. However, we did not yet know that weather is a valid configuration or not. Therefore `ValidConfigurationQuestion` object was created with the "ValidConfiguration" identifier and after setting our configuration to that question object, question was asked. `isValid()` method returns true if a configuration is possible with given features for the model and false for otherwise.

For the valid configuration analysis, like the valid product analysis, it is also very hard and time consuming to make this analysis to our model without using FAMA as the model has lots of features and constraints.

As an example, following features can be given as a product configuration. We can analyze whether or not a configuration of NewGenTV which contains these is valid obeying all constraints and relations:

NewGenTVSPL, ProductProperties, SideAndBackConnection, Panel, PanelSize, TouchKey, OnOffSwitch, OtelTV, AntennaInput, TP5, OneTouchRecording, Hierarchical.

FAMA returns true for the valid configuration question. The run time of the analysis is 1 seconds.

6.4.7 Error Detection

This operation “looks for errors on a feature model.” [56]. It is called “*anomalies detection*” [63] in the literature.

Methods of the operation:

- `setObservations(Collection<Observation> observations)`: It sets the observations.
- `getErrors()`: It returns the errors if the model has them.

Output parameter for that operation is set of errors. It takes a set of observations obtained from the `getObservations()` method as input parameter. Supported reasoners for that operation are Choco, ChocoAttributed, Sat4j, and JaCoP. However, only Choco provides both standard and extended model support for that operation. Others support just standard models. It uses “DetectErrorsQuestion” interface.

The usage of the Question is:

```
ChocoDetectErrorsQuestion eq= new ChocoDetectErrorsQuestion();
```

```
ChocoReasoner r = new ChocoReasoner();
```

```
afm.transformTo(r);
```

```
eq.setObservations(afm.getObservations());
```

```
qt.ask(eq);
```

“ChocoDetectErrorsQuestion” which is the attributed model version of ExplainInvalidProductQuestion interface was used.

ChocoDetectErrorsQuestion interface used for detecting FAMA errors if the model has any. In order to analyze model, first, a transform method was used then model’s observations were set. After that question was asked, errors can be accessed via `getErrors()` in Collection Error type if the model has any FAMA errors.

At this point, brief information about the FAMA errors is needed. There are 4 types of errors which FAMA Framework can detect. These are dead feature, false mandatory, wrong cardinality and void fm. Dead feature is a feature which does not appear in any valid product of the feature model. False mandatory occurs when a feature is modeled as optional but it is actually mandatory for any product. Wrong cardinality is happens when the maximum number of features for cardinality is actually not selectable. For example, when 3 features named A, B and C has *or-relationships*, maximum number of features that can be selected from these children is 3. However some constraints like “A excludes B can prevent feature A and B existed in a product at the same time. Therefore maximum number of feature cannot be implemented at the same time in any possible product. Last error, void FM means that not a single product can be represented by the feature model.

When the feature model of NewGenTV was analyzed with the detect error question like above; it returned dead feature, false mandatory and wrong cardinality errors. It did not return any void feature model error. This was expected as the feature model was constructed

after detailed analysis. Also, using a tool like FeatureIDE helped us to see the model in GUI and prevented too many contradictions in modeling phase. Therefore our feature model is not void, in other words it has at least one product as expected.

6.4.8 Error Explanations

This operation “looks for explanations in terms of relationships for the errors, when a model has errors.” [56]. It tells how to correct an error. It is called “*Explanations*” and returns “information (so-called explanations) about the reasons of why or why not the corresponding response of the operation” [63].

Methods of the operation:

- `setErrors(Collection<Error> colErrors)`: It sets the errors.
- `getErrors()`: It returns the errors with explanations.

Output parameter for that operation is set of model errors with explanations. It takes a set of model errors obtained from the previous detect error operation as input parameter. Supported reasoners for that operation are Choco, Sat4j, and JaCoP. However, only Choco provides both standard and extended model support for that operation. Others support just standard models. It uses “`ExplainErrorsQuestion`” interface.

The usage of the Question is:

```
ChocoReasoner r = new ChocoReasoner();

afm.transformTo(r);

ChocoExplainErrorsQuestion qe = new ChocoExplainErrorsQuestion();

qe.setErrors(errors);

qt.ask(qe);

errors = qe.getErrors();
```

In this analysis, `ChocoExplainErrorsQuestion` interface was used for extended feature model. Before this question was asked, we had to have model’s FAMA errors as we were to set the errors to `ChocoExplainErrorsQuestion`. Therefore first detect error question was asked and errors were calculating. After that that error were set to `ChocoExplainErrorsQuestion`, and only after that question was asked. Using the `getErrors()` method of `ChocoExplainErrorsQuestion`, now we had not just have errors but errors with explanations. Although both explain error question and detect error question have method named `getErrors()`, they are different. After that explanations and relations were extracted.

Each error has explanation whereas each explanation has relations related to that error explanation. These were accessed with `getExplanations()` along with its `getRelations()` methods.

Explanations help to find the source of error. If we do not have explanations, debugging an SPL with hundreds of features and constraints manually is either not possible or takes too much time. For a feature model error, an explanation gives the relations which were needed to be removed or modified in order to fix the error.

In the previous analysis, Error Detection, we learned that our feature model has dead feature, false mandatory and wrong cardinality errors. Now, after analyzing it with Error Explanations Question, we got following results:

- 1-Explanations for error False-mandatory Feature: HDReady
Relation: HD -> FullHD HDReady
Relation: HD -> HDReady
Relation: FullHD IMPLIES HDReady
- 2-Explanations for error Dead Feature: MHEG_HD
Relation: TVApplications -> MHEG5_1_06
Relation: TVApplications -> MHEG_HD
Relation: MHEG5_1_06 IMPLIES NOT MHEG_HD
- 3-Explanations for error Wrong cardinality {3} in relation rel-330
Relation: NTSC_CCS IFF NTSC;
Relation: ChromaDecodingStandard -> PAL
Relation: NTSC_CCS IFF (NOT PAL)
Relation: ChromaDecodingStandard -> NTSC

As seen in the results, first error is False--mandatory Feature: HDReady. This analysis not only tells that HDReady is a False-mandatory feature, but also gives information about reason of this error. The relations seen in the outputs are the explanations for the errors. They help us to detect the error reason and fix the error.

For the False-mandatory error, the analysis gave us three relations. When we inspected first two relations, we saw that HD is the parent of two features namely FullHD, HDReady and these children have *or-relationships*. Therefore they are not mandatory. However, when the third relation (FullHD IMPLIES HDReady) was inspected, it can be deduced that HDReady should be selected in every possible product. Actually there are three possible product configurations related to those features. As at least one of the children of HD should be selected, for the first configuration, we chose FullHD. Because of the constraint (FullHD IMPLIES HDReady), HDReady should also be selected. For the second configuration, we chose HDReady. Lastly, for the third configuration both of these children were selected. Therefore, in all there possible configurations, HDReady was selected. So it is false-

mandatory, in other words it is a mandatory feature, although it was defined as optional in feature model.

With the light of this error explanation, some changes had been made to the model. As told earlier, HDReady should be mandatory. Therefore *or-relationships* had been converted to *mandatory-relationship* and *optional relationship*. HDReady had been made a mandatory feature and other child feature had been made an optional feature.

For the Dead feature error, the analysis gave us three relations. When we inspected first two relations, we saw that MHEG5_1_06 and MHEG_HD are the child features of the parent TVApplications feature. Third relation is a constraint and indicates that if MHEG5_1_06 is selected for a product, MHEG_HD cannot be selected. However MHEG5_1_06 is a mandatory feature of the TVApplications. Therefore it is always selected. So MHEG_HD is a dead feature. In other words, it does not appear in any valid product. To fix this, first, definition of the MHEG5_1_06 was inspected. According to Feature Glossary which is in the Appendix C, MHEG5_1_06 is for SD (Standard Definitions) whereas MHEG_HD is for HD. Making MHEG5_1_06 is an optional feature fixes the dead-feature error and does not contradict with the definitions. Therefore, MHEG5_1_06 was made optional.

For the Wrong cardinality error, the analysis gave us four relations. When we inspected ChromaDecodingStandard -> PAL and ChromaDecodingStandard ->NTSC relations, we learned that PAL and NTSC are child features of ChromaDecodingStandard. Actually these child features has *or-relationships*, and has [1,3] cardinality. However, inspecting constraints that output as relations (NTSC_CCS IFF (NOT PAL), NTSC_CCS IFF NTSC), we saw that PAL and NTSC cannot be selected for the same product. Therefore maximum number of cardinality (3), cannot be achieved for ChromaDecodingStandard feature. To fix this error, the model can be modified such that children of ChromaDecodingStandard would have *optional-relationships*.

6.4.9 Core Features

This operation “calculates features that are present on every product” [56].

Methods of the operation:

- getCoreFeats(): It returns core features which are the features that are present on every product.

Output parameter for that operation is a set of features. It does not take any input parameter. The supported reasoner for that operation is only Choco and it supports standard model. It uses “CoreFeaturesQuestion” interface.

The usage of this Question is:

```
CoreFeaturesQuestionpq = (CoreFeaturesQuestion) qt.createQuestion("Core");
```

```
qt.ask(pq);
```

```
q.getCoreFeats();
```

However, as told earlier, this question is not applicable to our model as it is an extended feature model. Therefore transformTo() method and Choco Reasoner's question named "ChocoCoreFeaturesQuestion" were used such that:

```
ChocoCoreFeaturesQuestion q = new ChocoCoreFeaturesQuestion();
```

```
ChocoReasoner r = new ChocoReasoner();
```

```
afm.transformTo(r);
```

```
r.ask(q);
```

```
q.getCoreFeats();
```

The core features which are the features that exist every product for our feature model was calculated using this analysis and it had been detected that 122 of 652 features are core features. Output of the analysis can be seen in the Table 6.5. It was arranged according to decompositional relationships because when a child feature is a core feature, its parent is also a core feature. Therefore, arranging them in two columns namely parent and child, helps to see these relationships better. In the table, a feature can be both a child and a parent. For example, ICProperties feature which is in the Parent Core Features column in the Table 6.5, is also in the Child Core Features columns for the NewGenTVSPL parent feature because ICProperties is one of the children of NewGenTVSPL.

Table 6.5: Core Features Analysis

Parent Core Features	Child Core Features
NewGenTVSPL	ProductProperties SideAndBackConnection AudioVideoSignalInformation ICProperties MPEGTransportStreamAVDecoding ConditionalAccessAndCommonInterface VideoAndGraphicsProcessing AudioProcessing PictureFormats USBVideoProcessing USBApplications

Table 6.5 (Continued)

	USBSupportedFiles MenuLanguages SupportedDisplayTypes PowerSetting TargetCertificationsDVBT TargetOperatorsDVBC Approvals PanelSize DLNA
ProductProperties	HD Panel SkypeReady Cabinet RemoteControl OSD
SideAndBackConnection	AntennaInput FullScart S_VHS_Input CVBS_Input AudioInput ComponentInput PC_AudioInput VGA HDMI_Input USB_Input Network_RJ45_Connection_Ethernet SPDIF HeadphoneOutput LineoutOutput CI_Connectors
AudioVideoSignalInformation	AnalogRF_Signal Y_C_signal RGB_FBsignal RGB_HsVs_Signal ProgressiveSignal HDMI_Signal CVBS_SignalOutput Stereo_L_R_AudioSignalInput Stereo_L_R_AudioSignalOutput SPDIFSignalOut AudioOutputPower Speakers Type
ICProperties	RAMPower CPUPower VideoPictureEnhancers AudioDecoder

Table 6.5 (Continued)

	DSPClockRate SW_BASE VIF_SIF SECAM_Decoder Pre_SNR DNR Mosquito_Noise_Reduction De_BlockingNoiseReduction GaussianNoiseReduction SpikeNoiseReduction MotionAdaptiveDeinterlace EdgeOrientedDeinterlace FilmModeDetection CadenceFilmDetection_Blackline _2DPeaking ICC IHC IBC YC_Delay CTI xvYCC_Support ScalingTaps ScalingDeRinging MPEGEncoder PreFiltering
MPEGTransportStreamAVDecoding	TransportStream ProfileLevel AspectRatio VideoResolution AudioDecoding
ConditionalAccessAndCommonInterface	CommonInterface
VideoAndGraphicsProcessing	MainPicture_Deinterlacing ChromaDecoding ChromaDecodingStandard HDMI_Resolution TxtPages Standart
AudioProcessing	StereoDecoding GermanA2_Nicam SRS
USBApplications	SupportedMenuLanguages USBConnection20
PowerSetting	PSUType InputRange StandByPowerConsumption
Approvals	SRS_Approvals CIPlus_DigitalTVLabs

Table 6.5 (Continued)

VideoPictureEnhancers	ProACEAlgorithm
AudioDecoder	DualDSP_Structure
SW_BASE	Ecos_On_RISC Linux_On_MIPS
SECAM_Decoder	SECAM_Decoder_ImprovedThenS1Plus
DNR	DNR_Temporal_Noise_Reduction
GaussianNoiseReduction	SpatialGaussianNoiseReduction
SpikeNoiseReduction	SpatialSpikeNoiseReduction
MotionAdaptiveDeinterlace	_3DMotionAdaptiveDeinterlace
EdgeOrientedDeinterlace	AdaptiveEdgeOriented_2D_Deinterlace
MPEGEncoder	MPEG4Encoder
MainPicture_Deinterlacing	_3DFrameBasedMotionAdaptive
Standart	BG DK L I

Detecting core features help to decrease the errors when creating products and affects product decisions in planning and development.

6.4.10 Variant Features

This operation “calculates features that are not present on every product” [56].

Methods of the operation:

- getVariantFeats(): It returns variant features which are the features that are not present on every product.

Output parameter for that operation is a set of features. It does not take any input parameter. It uses “VariantFeaturesQuestion” interface. The supported reasoner for that operation is only Choco and it supports standard model. Therefore, Variant Features analysis cannot be done in FAMA for our model as there is no interface method which supports extended models in any reasoner. However we can detect variant features by stating that features other

than core ones are variants. As our model has 652 features and 185 of them are core features, we can say that number of variant features are 467.

6.4.11 Commonality

This operation “calculates appearances of a given feature into the list of products” [56]. In other words, it calculates the commonality which is the number of products where a feature appears.

Methods of the operation:

- `setFeature(GenericFeature f)`: It sets the feature that we want to know its commonality.
- `getCommonality()`: It returns the commonality value of the feature in long type. However before calling this method, it is necessary to call `setFeature(GenericFeature f)` method in order to specify feature which we want to know its commonality.

Output parameter for that operation is commonality in long type. It does a feature as an input. It uses “CommonalityQuestion” interface. Supported reasoners for that operation are Choco, Sat4j, JavaBDD and JaCoP. All of them support just standard models. It does not have support for extended models. Therefore Commonality analysis cannot be done in FAMA for our model as there is no interface method which supports extended models in any reasoned.

6.4.12 Variability

This operation calculates variability factor. It was defined as “the ratio between the number of products and 2^n where n is the number of features considered. In particular, 2^n is the potential number of products represented by a feature model assuming that any combination of features is allowed” [63].

Methods of the operation:

- `getVariability()`: It returns the variability factor of the feature model in float type.

Output parameter for that operation is variability in float type. It does not take any input parameter. It uses “VariabilityQuestion” interface. Supported reasoners for that operation are Choco, Sat4j, JavaBDD and JaCoP. All of them support just standard models. It does not have support for extended models. Therefore Variability analysis cannot be done in FAMA for our model as there is no interface method which supports extended models in any reasoner.

6.4.13 Other FAMA Operations

FAMA has also some operations and methods in order to create, modify or set some values of the feature model. For example with `FilterQuestion`, one can assign a filter to the model.

Using `addValue(VariabilityElement feature, int value)` and `removeValue(VariabilityElement feature)` methods of this question, features, relations, or values can be added or removed from the model. However in the study, this model modification was not used as FAMA was only used for model analysis in the study whereas model creation and modification were done via FeatureIDE. But FilterQuestion can also be used for specified a value to an attribute. For example:

```
SetQuestion sq = (SetQuestion) qt.createQuestion("Set");

FilterQuestion fq = (FilterQuestion) qt.createQuestion("Filter");

ValidQuestion vq = (ValidQuestion) qt.createQuestion("Valid");

fq.addValue(afm.searchFeatureByName("CVBS_Input").searchAttributeByName("vpCVBS_InputNumber"), 1);

sq.addQuestion(fq);

sq.addQuestion(vq);

qt.ask(sq);

vq.valid();
```

In this example, along with the FilterQuestion, SetQuestion was also used because two questions were needed to be asked together. Questions can be added to the SetQuestion using the `addQuestion(Question p)` method. The orders of the questions are not important.

Using FilterQuestion as in the example, a filter on the feature model was specified. A value (2) was given to the attribute of feature CVBS_Input were given to the model. According to the question, we found out that, a model whose CVBS_Input feature's attribute is 2, cannot be a valid model. When we looked at the model, we saw that there is no 2 as the value in the domain. However when we use `addValue(Question q)` method with the same attribute but has the value 2, it returns true as this value is in the attribute domain.

However FilterQuestion is deprecated in the last version of FAMA (version 1.1.2). Therefore it was not used.

CHAPTER 7

CONCLUSION

In our work, a feature model with various analyses was proposed for dealing with the commonality and variability in terms of feature models and making automated analysis.

Feature modeling is commonly used in order to represent and manage commonality and variability in software product line engineering. By constructing feature model, elements of SPL families can be identified better for reusing and developers and customers can understand the SPL better whereas making analyses on that model can help to improve the model by finding errors and contradictions if there is any and can reduce the time and money as information about products can be gained in the early stages of development.

This work consists of three stages, namely modeling, transformation, and analyses. Although our SPL domain is applicable to an extended feature model as it has attributes, in the modeling part we used basic feature model. However, in the translation part, this model was converted back to the extended feature model and analysis operations were used in this extended feature model.

In the modeling part of the study, FODA, which is a SPL approach based on feature diagrams, was used. The information gained from domain analysis was utilized in the modeling phase using a GUI based modeling tool, namely FeatureIDE. Feature modeling takes advantage of this tool such that it is easy to understand and to edit the SPL with the tree-structured visual model. Although graphical modeling tools seem to be more convenient to also non-technical people such as customers, they are not efficient and sometimes not fully functional while working with large feature diagrams and complex constraints. Besides, the capability of FeatureIDE is limited such that expressing attributed feature models and constructing complex cross-tree constraints are not possible. There is also an unfavorable aspect of FeatureIDE with large feature models. As the model gets larger, using constraint editor gets harder. Sometimes it slows down so much that using constraint editor would become impossible at a point. However, even with the really large feature models, managing constraints from the XML file of the model in FeatureIDE is still possible. Therefore FeatureIDE is still a good tool for feature modeling.

A powerful automated analysis tool is needed for the analysis part. Therefore the approach suggested in this thesis, proposes using FAMA framework which supports some widely used reasoners such that Choco, JavaBDD, JaCoP, Sat4j. However the feature model that is

constructed in this study was extracted in XML format. Therefore before starting to analyze, a transformation was done to convert the model to the FAMA's AFM format, which is the format for the attributed feature model. A parser was written, in Java, for this purpose. After that FAMA analyses for the attributed feature models which were executed and they were inspected with the examples. These analyses improved the model, proposed solutions to fix the errors, and helped developers and customers to understand model better. Therefore money and time can be saved as a result of the analyses in the early stages of development.

Moreover we saw that, as the modeling process progresses, maintaining feature model has become difficult, although visual modeling tool helps in this stage. Therefore, even with the models which constructed as a result of detailed domain analysis, developers can face some model errors which are hard to see as it is difficult to inspect the model manually. In our study, we inspected these errors by doing FAMA analysis. Also all the FAMA operations for extended feature models were used in the study. These FAMA operations are Validation, Products, Number of Products, Valid Product, Invalid Product Explanation, Valid Configuration, Error Detection, Error Explanation, and Core Features. We saw that, using some analyses operations such as Products and Number of Products, takes too much time when feature number of the model reaches to a certain threshold, although these analyses are said to be used with extended feature models. FAMA solvers should be improved in order to decrease run time of those operations as feature number of realistic models can easily exceed that threshold as seen in the model constructed in this study. However, FAMA is still the best choice for feature model analysis as it is the most competent and comprehensive tool in the field.

The main contribution of this work is the modeling experience gained by working on a realistic product family. As there are lots of concepts, dependencies and constraints in realistic models, feature identification and grouping them according to characteristics are crucial. This study converted Product Configuration Sheet which gives the technical information about the products, along with some domain terms gained from domain analysis to the feature model. These documents and terms were mainly a bunch of vague information for the people other than domain experts before we chose appropriate ones and extracted them as features.

Also editing and maintaining feature model was made very easy and fast with this approach. In analysis part, when the results of analyses suggest some modifications of the model, for example to fix some errors (in the Error Explanation Analysis), change some feature relationships or delete/add features (in the Invalid Product Explanation Analysis), it can be easily realized by following proposed steps. At these steps, first we utilized the FeatureIDE's modeling GUI which is easy to use and making changes. Then using the parser, the model is converted to the AFM file. After that FAMA framework is used for automated analysis.

The main challenge of the modeling approach is its dependency on the user's experience and domain knowledge. Therefore it can be suggested that modeling process should be done by a

separate group of people which are experts of the modeling domain. Also the analysis operations of FAMA are limited. Variability, commonality and variant features operations cannot be used for the extended feature models and as told earlier some operations take too much time when running with large models. Apart from these, FAMA framework is fast and effective for feature model analysis. Moreover FAMA developers are constantly working to improve operations. So in the future, more operations can support extended feature models in less run time.

REFERENCES

- [1] J. Estublier and G. Vega. Reuse and variability in large software applications. *ACM SIGSOFT Software Engineering Notes*, 30(5): 316 – 325, 2005.
- [2] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature-oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90TR-21, Software Engineering Institute, Carnegie Mellon University, 1 – 52, 1990.
- [3] D. Dhungana, P. Grünbacher and R. Rabiser. Domain-specific Adaptations of Product Line Variability Modeling. *Situational Method Engineering: Fundamentals and Experiences*, Springer, 244: 238 – 251, 2007.
- [4] L. Daizhong and D. Shanhui. Feature dependency modeling for software product line. *International Conference on Computer Engineering and Technology (ICCET'09)*, vol. 2, 256 – 260, 2009.
- [5] Clements, P., Northrop, L. *Software Product Lines: Practice and Patterns*. Addison Wesley (2002).
- [6] W. B. Frakes and K. Kang. Software reuse research: status and future. *IEEE Transactions on Software Engineering*, Vol. 31, No. 7, July 2005.
- [7] M. D. McIlroy. Mass produced software components. *In proceedings of NATO Software Engineering Conference*, pp. 138-155, Garmisch, Germany, 1968.
- [8] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1994.
- [9] H. P. Jepsen, J. G. Dall, and D. Beuche. Minimally invasive migration to software product lines. *11th International Software Product Line Conference (SPLC 2007)*, pp. 203-211, September 2007.
- [10] C. W. Krueger. Introduction to the emerging practice of software product line development. *Methods and Tools*, vol. 14, no. 3, pp. 3-15, 2006.
- [11] K. Schmid and I. John. Product line development as a rational, strategic decision. In *Proceedings of the International Workshop on Product Line Engineering in Early Steps: Planning, Modeling, and Managing (PLEES'01)*, 2001, pp. 1-6.
- [12] G. Böckle et al. Adopting and institutionalizing a product line culture. *Lecture Notes In Computer Science Vol 2379*, pp. 49-59, 2002.
- [13] C. W. Krueger. Easing the transition to software mass customization. *PFE 01 Revised Papers from the 4th International Workshop on Software Product Family Engineering*, vol. 1, no. 512, pp. 282-293, 2002.

- [14] J. Galbraith. Designing the Customer-Centric Organization. A guide to strategy, structure, and process. Wiley & Sons, Inc., 2005.
- [15] D. Muthig. A light-weight approach facilitating an evolutionary transition towards software product lines. Fraunhofer IRB Verlag, 2002.
- [16] H. Kutluca,, İ.E. Çetin, U. Çakır, M. Kılıç. GEMKOMSSIS, development of battle management software system as R&D project, the common infrastructure that it offers to marine environments and application of coastguard and search and rescue ship. *Software Quality and Software Development Tools Symposium*, 9-10 October 2008, İstanbul, pp. 3-11.
- [17] T. Koray, C.T. Yurdakul, İ. Yakın. Domain model in command control systems and usage of reference architecture. *In proceedings of National Software Conference*, 11-12 September 2008, İzmir, pp. 99-106.
- [18] OpenJAUS, <http://www.openjaus.com>. (last accessed on 01/08/2013).
- [19] N.İ. Altıntaş, M. Surav, O. Keskin, S. Çetin. Aurora software product line. *In proceedings of Second Software Engineering Symposium*, 22-24 September 2005, Ankara, pp. 109-118.
- [20] E. Kahraman, T. İpek, B. İyidir, C.F. Bazlamaçcı, S. Bilgen. Domain Engineering studies intended for developing component based software product line. *In proceedings of Fourth National Software Engineering Symposium*, 8-10 October 2009, İstanbul, pp. 283-287.
- [21] XML, <http://www.w3.org/XML/>. (last accessed on 01/08/2013).
- [22] K. Pohl, G. Böckle and F. van der Linden. Software Product Line Engineering: Foundations, Principals and Techniques, Springer, Berlin Heidelberg New York, 2005.
- [23] J. Lee, K. C. Kang, and S. Kim. A feature-based approach to product line production planning, LNCS, Vol. 3154/2004, pp. 183-196, 2004.
- [24] W. C. Krueger, New methods in software product line development. 10th International Software Product Line Conference, 21-24 August 2006, Baltimore, Maryland, USA.
- [25] I. Groher, H. Papajewski, M. Voelter. Integrating model-driven development and software product line engineering. Eclipse Summit Europe 2007, Ludwigsburg, Germany.
- [26] O. Spinczyk and D. Beuche. Modeling and building software product lines with eclipse. *In Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages and applications (OOPSLA'04)*, 18-19, 2004.

- [27] D. Benavides, P. Trinidad and A. Ruiz-Cortés. Automated reasoning on feature models. 17th Conference on Advanced Information Systems Engineering (CAiSE'05), 2005.
- [28] K. C. Kang, S. Kim, J. Lee, K. Kim, G. J. Kim, and E. Shin. FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, *Annals of Software Engineering*, 1998.
- [29] K. Berg, J. Bishop, and D. Muthig. Tracing software product line variability – from solution to problem space. *In proceedings of SAICSIT*, 2005.
- [30] A. Metzger, P. Heymans. Comparing feature diagram examples found in the research literature. Technical report TR-2007-01, Univ. of Duisburg-Essen, 2007.
- [31] K. Czarnecki, S. Helsen ve U. Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process Improvement and Practice*, 10(2):143–169, 2005.
- [32] K. Czarnecki, S. Helsen, and U. Eisenecker. Staged configuration using feature models. In *proceedings of the Third International Conference on Software Product Lines*, (SPLC '04), volume 3154 of *Lecture Notes in Computer Science*. Springer Berlin/Heidelberg, August 2004.
- [33] 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 (SCP)*, 2012.
- [34] D. Benavides, S. Segura, P. Trinidad, and A. R. Cortés. FAMA: Tooling a framework for the automated analysis of feature models. *VaMoS 2007*: 129-134.
- [35] J. Coplien, D. Hoffman and D. Weiss. Commonality and variability in software engineering. *IEEE Software*, Volume 15, No. 6. Nov/Dec 1998. pp. 37-45.
- [36] K. Czarnecki and U. W. Eisenecker. *Generative Programming – Methods, Tools and Applications*. Addison-Wesley. 2000. ISBN 0-201-30977-7.
- [37] K. C. Kang, J. Lee and K. Lee. *Feature Oriented Product Line Software Engineering: Principles and Guidelines from. Domain Oriented Systems Development: Perspectives and Practices*. Taylor & Francis. UK. 2003. ISBN 0-415-30450-4.
- [38] Choco-Solver: <http://choco-solver.net>. (last accessed on 20/07/2013).
- [39] Sat4j: <http://www.sat4j.org>. (last accessed on 16/07/2013).
- [40] JavaBDD: <http://javabdd.sourceforge.net>. (last accessed on 03/08/2013).
- [41] JaCoP-Solver: <http://www.jacop.eu>. (last accessed on 03/08/2013).

- [42] L. Gherardi, D. Brugali. An eclipse-based feature models tool chain. *In Proc. of the 6th Workshop of the Italian Eclipse Community (Eclipse-IT 2011)*, September 22-23, 2011, Milano, Italy.
- [43] Eclipse Modeling Tools: <http://www.eclipse.org/downloads/packages/eclipse-modeling-tools/junosr1>. (last accessed on 01/08/2013).
- [44] Graphical Modeling Tooling: <http://www.eclipse.org/modeling/gmp/>. (last accessed on 03/08/2013).
- [45] Pure-systems GmbH (2008). pure::variants User's Guide: Version 3.0 for pure::variants 3.0. Available from: <http://www.puresystems.com/fileadmin/downloads/pure-variants/doc/pv-user-manual.pdf>. (last accessed on 20/07/2013).
- [46] Pure-systems GmbH: <http://www.pure-systems.com/>. (last accessed on 01/08/2013).
- [47] J. Garvin. Users' Choice: 2009 Software development platforms - a comprehensive user satisfaction survey of over 1200 software developers. Technical report, Evans Data Corporation, 2009.
- [48] M. S. De Alencar. Digital Television Systems. Cambridge University press, 2009.
- [49] C. Poynton. Digital Video and HDTV: Algorithms and Interfaces. Morgan Kaufmann. Publishers, 2003.
- [50] H. Benoit. Digital Television MPEG-1, MPEG-2 and Principles of the DVB System. Wiley & Sons Inc (Arnold), New York 1997.
- [51] T. Berger, S. She, R. Lotufo, A. Wasowski, and K. Czarnecki. Variability modeling in the real: A perspective from the operating systems domain. In ASE, 2010.
- [52] G. Arango. Software Reusability, Chapter 2. Domain analysis methods. pp. 17-49. Workshops M.E. Horwood, London 1994.
- [53] M. L. Griss, J. Favaro and M. d'Alessandro. Integrating feature modeling with RSEB. *In Proceedings of the International Conference on Software Reuse*. Victoria, Canada. June 1998.
- [54] DOM Tutorial, <http://www.w3schools.com/dom/>. (last accessed on 02/08/2013).
- [55] Creatly, <http://creately.com/>. (last accessed on 29/07/2013).
- [56] FAMA User Manual, <http://www.isa.us.es/fama/?Documentation>. (last accessed on 01/08/2013).
- [57] FAMA Distributions, <https://code.google.com/p/famats/>. (last accessed on 10/08/2013).

- [58] FAMA, http://www.isa.us.es/fama/?FaMa_Framework. (last accessed on 10/08/2013).
- [59] SPLOTS, <http://www.splot-research.org/>. (last accessed on 20/07/2013).
- [60] M. Mendonca, M. Branco, and D. Cowan. S.P.L.O.T. - Software product lines online tools. *In Companion to the 24th ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, OOPSLA 2009, October 2009, Orlando, Florida, USA.
- [61] The Systems and Software Product Line Engineering Lifecycle Framework BigLever Software technical report #200805071r4, www.biglever.com/learn/whitepapers.html, January 2013.
- [62] Gears, <http://www.biglever.com/>. (last accessed on 20/07/2013).
- [63] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Information Systems* 35(6) pp. 615-636, 2010.
- [64] Chen, L., Ali Babar, M., A Systematic Review of Evaluation of Variability Management Approaches in Software Product Lines, *Information and Software Technology*, 53(4), pp. 344-362, 2011.
- [65] Arçelik, <http://www.arcelikas.com/>. (last accessed on 28/09/2013).
- [66] M. Riebisch, K. Bollert, D. Streitferdt, I. Philippow, Extending Feature Diagrams With UML Multiplicities, 6th Conference on Integrated Design & Process Technology (IDPT 2002), Pasadena, California, USA, 2002.
- [67] A.S. Karataş, H. Oğuztüzün, A. Doğru, Transforming cross-tree relations involving attributes into basic constraints in feature models, in: *Proceedings of the Fifth International Conference on Application of Information and Communication Technologies (AICT 2011)*, pp. 157-161, (2011).

APPENDIX A

PARSER FLOWCHARTS

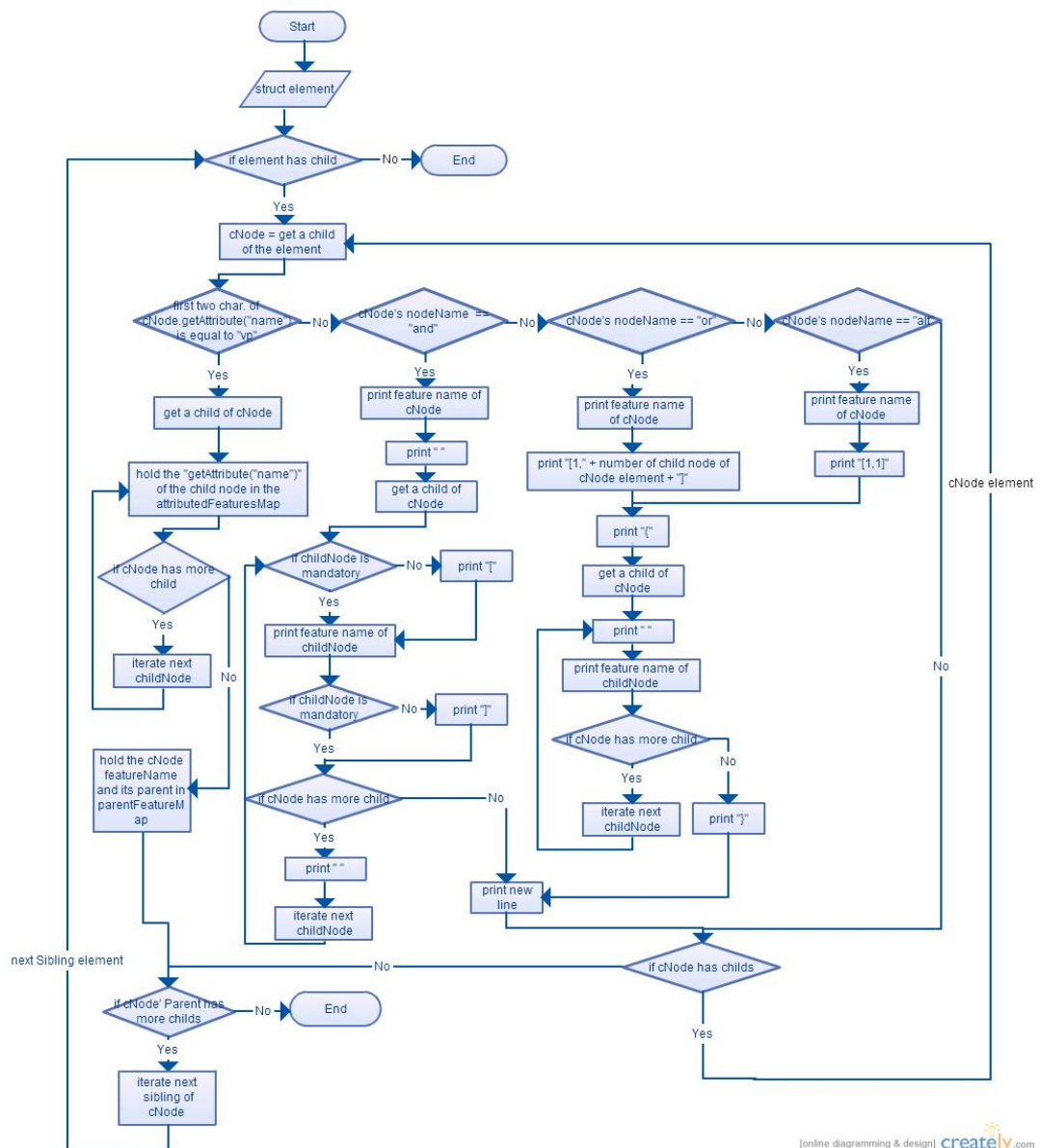


Figure 8.1: Flowchart of Parser Related to Relationships Area

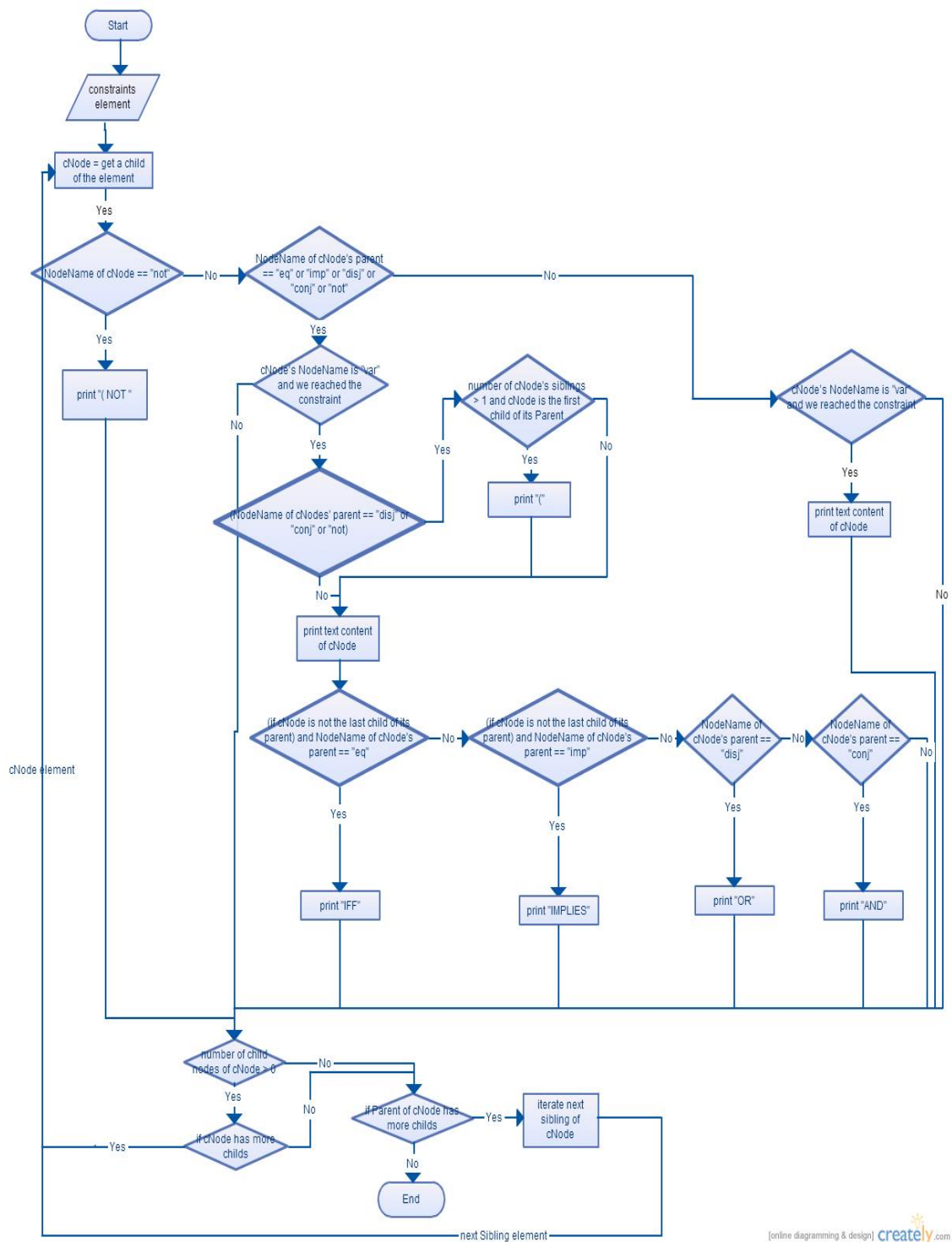


Figure 8.2: Flowchart of Parser Related to Constraints Area

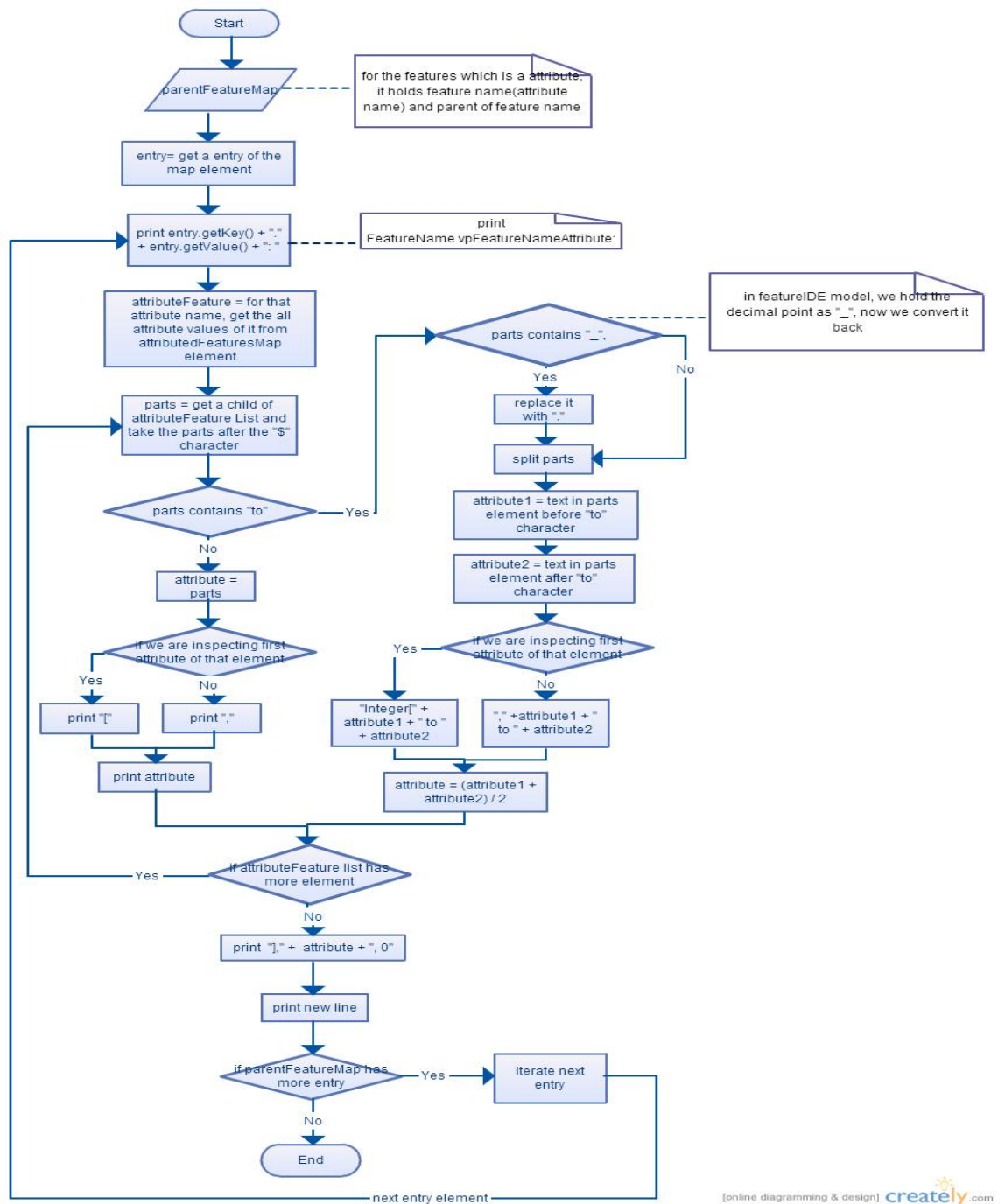


Figure 8.3: Flowchart of Parser Related to Attributes Area

APPENDIX B

PARSER PSEUDOCODES

pseudoCodeForTraverseRelationshipsAndAttributes:

```
function printChildFeaturesForAnd(List element):  
begin  
for every child node of List element  
    if child node's type is ELEMENT_NODE  
        print " "  
        if child node.getAttribute("mandatory") != true  
            print "["  
        end if  
        // print feature name  
        print child node.getAttribute("name")  
        if child node.getAttribute("mandatory") != true  
            print "]"  
        end if  
    end if  
end for  
print new line  
end printChildFeaturesForAnd
```

```
function printChildFeaturesForOrXor(List element):  
begin  
print "{ "  
for every child node of List element  
    if child node's type is ELEMENT_NODE  
        print " "  
        // print feature name  
        print child node.getAttribute("name")  
    end if  
end for  
print "}"  
print new line  
end printChildFeaturesForOrXor
```

```
function traverseChildNodesForRelationships(List element):  
begin
```

```

for i = 0 to number of child node of input element
    cNode = ith Child Node of element
    if cNode's type is ELEMENT_NODE
        // hold the attributed and the feature which that attributes belong in
        the Map<> format.
        if first two char. of cNode.getAttribute("name") is "vp"
            for every child node of cNode element
                hold the "getAttribute("name")" of the child node in the
                attributedFeaturesMap
            end for
            hold the cNode featureName and its parent in
            parentFeatureMap

        else if cNode's nodeName is "and"
            // print feature name
            print "cNode.getAttribute("name") : "
            call printChildFeaturesForAnd(List of child nodes of cNode
            element)

        else if cNode's nodeName is "or"
            // print feature name
            print "cNode.getAttribute("name") : "
            print "[1," + number of child node of cNode element + "]"
            call printChildFeaturesForOrXor(List of child nodes of cNode
            element)

        else if cNode's nodeName is "alt"
            // print feature name
            print "cNode.getAttribute("name") : "
            print "[1,1]"
            call printChildFeaturesForOrXor(List of child nodes of cNode
            element)

        else if cNode's nodeName is "feature"
            end if

        if number of child nodes of cNode element > 0
            //recursion
            call traverseChildNodesForRelationships(List of child nodes of
            cNode element)
        end if
    end if
end for
end traverseChildNodesForRelationships

function inspectAndWriteAttributes(Map<String,List<String>>
attributedFeaturesMap, Map<String,String> parentFeatureMap):
begin

```

```

    for every element of parentFeatureMap //it holds the feature names and parents of
    that features which have attributes
        //print the attributes in FAMA's format
        print Parent Feature Name + "." + vpFeatureName
        //List<String> attributeFeature =
    attributedFeaturesMap.get(entry.getValue())
        for every attribute of that vpFeatureName
            parts = take the parts which comes from "$" char. of attribute
            if parts contains "to" // it is a range, like 0 to 10
                split parts as range1 and range2
                //since in featureIDE, it is not allowed to use ".", i used "_"
                for decimal value. not i convert it back to "."
                if range1 or range2 has "_" convert it to "."
                end if
                if it is first attribute of that vpFeatureName
                    print "Integer[" + range1 + " to " + range2
                else
                    print "," + range1 + " to " + range2
                end if
                int attribute = (range1 + range2) / 2;
            else
                int attribute = parts;
            end if
            if it is first attribute of that vpFeatureName
                print "[" + attribute
            else
                print "," + attribute
            end if
        end for
    print "], "
    print attribute + ", 0"
    print new line
end for
end inspectAndWriteAttributes

```

```

function main
begin
set input file model.xml as doc
set output file out.afm
print "%Relationships"
print new line
call traverseChildNodesForRelationsShips(List of child nodes of element whose tag
name is "struct");
print new line
print "%Attributes"
print new line
call inspectAndWriteAttributes(attributedFeaturesMap, parentFeatureMap);
print new line

```

```

print "%Constraints"
print new line
call traverseChildNodesForConstraints(List of child nodes of element whose tag
name is "constraints");
close output file
end main

```

pseudoCodeForTraverseConstraints:

```

function traverseChildNodesForConstraints(List element):
begin
int childIndex = 0;
for i = 0 to number of child node of input element
    boolean recursion = false;
    cNode = ith Child Node of element
    if cNode's type is ELEMENT_NODE
        if cNode's NodeName is "not"
            print "(NOT "
        else if "ParentNode of cNode"'s NodeName is "eq"
            if cNode's NodeName is "var" and cNode has
            exactly 1 child // this means we reached the
            constraint <var>rule</var>
            // print rule
            print TextContent of cNode + " "
            childIndex++;

            if childIndex is not equal to number of cNode's
siblings
                // if cNode is not the last child of its parent.
                print "IFF "
                end if
            else
                if number of child nodes of cNode > 0
                    call traverseChildNodesForConstraints(List of child
nodes of cNode element)
                if childIndex is 0
                    recursion = true;
                    childIndex++;
                    print "IFF "
                end if
            end if
        end if
    else if "ParentNode of cNode"'s NodeName is "imp"
        if cNode's NodeName is "var" and cNode has exactly 1 child
            // this means we reached the constraint
            <var>rule</var>
            // print rule
            print TextContent of cNode + " "

```

```

        childIndex++;

        if childIndex is not equal to number of cNode's siblings
            // if cNode is not the last child of its parent.
            print "IMPLIES "
        end if
    else
        if number of child nodes of cNode > 0
            call traverseChildNodesForConstraints(List of child nodes of
cNode element)
            if childIndex is 0
                recursion = true;
                childIndex++;
                print "IMPLIES "
            end if
        end if
    end if
else if "ParentNode of cNode"'s NodeName is "disj"
    if cNode's NodeName is "var" and cNode has exactly 1 child
        // this means we reached the constraint <var>rule</var>
        if number of cNode's siblings > 1 and childIndex == 0
            print "("
        end if

        // print rule
        print TextContent of cNode + " "
        childIndex++;
        print "OR "
    else
        if number of child nodes of cNode > 0
            call traverseChildNodesForConstraints(List of child nodes of
cNode element)
            if childIndex is 0
                recursion = true;
                childIndex++;
                print "OR "
            end if
        end if
    end if
else if "ParentNode of cNode"'s NodeName is "conj"
    if cNode's NodeName is "var" and cNode has exactly 1 child
        // this means we reached the constraint <var>rule</var>
        if number of cNode's siblings > 1 and childIndex == 0
            print "("
        end if

        // print rule
        print TextContent of cNode + " "
        childIndex++;

```

```

        print "AND "
    else
        if number of child nodes of cNode > 0
            call traverseChildNodesForConstraints(List of child nodes of
cNode element)
            if childIndex is 0
                recursion = true;
                childIndex++;
                print "AND "
            end if
        end if
    end if
else if "ParentNode of cNode"'s NodeName is "not"
    if cNode's NodeName is "var" and cNode has exactly 1 child
        // this means we reached the constraint <var>rule</var>
        if number of cNode's siblings > 1 and childIndex == 0
            print "("
        end if

        // print rule
        print TextContent of cNode
    else
        if number of child nodes of cNode > 0
            call traverseChildNodesForConstraints(List of child nodes of
cNode element)
        end if
        recursion = true;
    end if
    childIndex++;
else if cNode's NodeName is "var" and cNode has exactly 1 child
    print TextContent of cNode
    childIndex++;
else
    childIndex++;
end if

if recursion is false
    if number of child nodes of cNode element > 0
        //recursion
        call traverseChildNodesForConstraints(List of child nodes of
cNode element)
    end if
end if

if lastProcessedTag is "not"
    print ")"
end if

if number of cNode's siblings equal to childIndex

```



```

        if "Parent of cNode"'s NodeName is "conj" or "disj"
            print ")"
        end if
    end if

    if cNode's NodeName is "rule"
        print new line
    end if
end if
end for
end traverseChildNodesForRelationsShips

```


APPENDIX C

FEATURE GLOSSARY

HD Ready: Any display that is capable of accepting and displaying a high-definition signal using a component video or digital input, but does not have a built-in HD-capable tuner is HD Ready. But if transmitted at 1080, it will reduce the signal to the 720p resolution.

Full HD: TVs that are made to receive and display the 1080 standard without having to downgrade the signal.

1080i: An interlaced scanning standard for HDTV, having an image structure of 1920x1080, and a frame rate (written after the i) of either 29.97 Hz or 30.00 Hz.

1080p: A progressive scanning standard for HDTV, having an image structure of 1920x1080, and a frame rate (written after the p) of 23.976, 24, 29.97, 30.00, 59.94 or 60.00 Hz.

Cabinet: The outer case of a television.

Touch key: It is used on a touch screen. It means the device has touch sensitive buttons for activation.

Otel TV: It is the in-suite television content presented in hotel-rooms provides services such as TV channels, information portal, games, weather, shopping, bill viewing, wake-up system (interactive otelTV).

OSD: On screen display. It is the control panel on a television screen. Its main job is to allow selecting viewing options and/or adjust components of the display.

Ambient sensor: It helps to save power by compensating for the bright light and gives a more comfortable view for the human eye. It detects light and adjusts the brightness of a monitor according to the brightness of the surrounding.

LED on IR (Infrared) Board: There is LED on television. It lights when it receives the IR code that is sent from remote controller by an IR light emitting diode in the remote control.

Self-Diagnostic: It can help to specify the cause of some failures. When there is an error, the standby LED starts blinking. The number of times that the LED blinks gives information about the source of the problem.

Screen Saver: When the TV is not used, it displays a picture.

E-pop: It removes or makes ads, icons, and logos appear in the corner of the TV.

Photo Frame: It displays the picture with clock and calendar by USB

PIP: Picture-In-Picture. In the corner of the main screen, a small screen can also be displayed. The images come from two different sources. There should be two-tuner to allow that. Sound just comes to speakers for the main picture.

PAP: Picture-and-Picture. It is like PIP. In that, sound for one picture comes from speakers and sound from the other one comes through headphones.

FullSCART: SCART is a standard connector which connects audio-visual equipment together such as televisions, VCRs, DVD players. For RGB and data transfer, a fully wired scart which has all pins connected is used. The signals carried by SCART include both composite and RGB (with composite synchronization) video, stereo audio input/output and digital signaling.

VHS: The Video Home System is a consumer-level analog video cassette recording standard.

S-VHS input: (Super-VHS) An improved version of VHS which used higher quality cassette. It increases resolution from 240 to 400 lines. As brightness and color information are separated, it improves picture quality. S-VHS introduced the S-video interface, which separated the luminance from the color.

S-video: Super Video. Y/C. It is an analog video transmission (no audio) that carries standard definition video. Video information is encoded on two channels: luma (luminance (black and white), intensity, "Y") and chroma (color, "C"). This separation is in contrast with slightly lower quality composite video (1 channel) and higher quality component video (3 channels). It's often referred to by JVC (Victor Company of Japan, Ltd) as both an S-VHS connector and as Super Video.

CVBS: "Color, Video, Blanking, and Sync." Composite Video (1 channel). It is an analog video transmission connector type that color and brightness signals exist together. It does not contain audio signal. Although higher quality S-video (2 channel) and even higher quality component video (3 channels) can be used for video information encoding; in composite video, one channel is used.

Component Input (YPBPR): It is the analog video signal carried by component video cable. It has 3 colored cables for 3 separated signals (brightness and two color signals). Y is carried by the green cable, PB is carried by the blue cable and PR is carried by the red cable (RGB). As brightness and color components of signal are separated, it represents higher quality.

VGA: A Video Graphics Array is an analog display standard. VGA connector has 15 pins in three-row. The images on a computer can be displayed on a TV by a VGA connector. It has 3 RGB signals which are analog and carry the pixels; and horizontal synchronization, vertical synchronization signals which provide the timing information necessary for the monitor to correctly display the pixel data.

Synchronized signals: hs/vs

hs: Horizontal Synchronization, HSYNC is a horizontal synchronization signal. When this signal is given, monitor stops drawing the current horizontal line, and start drawing the next line. The time is measured in Hertz (Hz).

vs: Vertical Synchronization, VSYNC is a vertical synchronization signal. It sets a value to monitor about the time to draw the next vertical line. The time is measured in Hertz (Hz).

HDMI input: High Definition Multimedia Interface. It transfers uncompressed digital audio or video data from an HDMI compliant device to a digital television. It contains a single cable which has 19 wires wrapped in it. It provides fast transfer speeds and sends audio and video over the same cable.

DVB: Digital Video Broadcasting. Open standards for digital television.

DVB-S: Digital Video Broadcasting - Satellite. It is the original Digital Video Broadcasting standard used for forward error correction and demodulation for Satellite Television. It is used via satellites serving every continent of the world.

DVB-S2: Digital Video Broadcast Satellite Second Generation. It is based on DVB-S. It is also based the Digital Satellite News Gathering standard. Mobile units use this for sending sounds and images from remote locations world-wide back to their home television stations. DVB-S2 receiver also uses services via HDTV, interactive services, Internet. VCM (Variable Coding and Modulation) and ACM (Adaptive Coding and Modulation) are the modes of DVB-S2. They allow optimizing bandwidth utilization by dynamically changing transmission parameters. It has also four modulation modes, namely QPSK, 8PSK, 16APSK and 32APSK. Broadcast applications uses QPSK and 8PSK. On the other hand professional, semi-linear applications use 16APSK and 32APSK. However 16APSK and 32APSK can also be used for broadcasting in some conditions.

DVB-C: Digital Video Broadcasting - Cable. It is the DVB standard for the broadcast transmission of digital television over cable. It uses QAM modulation. MPEG-2 or MPEG-4 digital audio/digital video stream uses DVB-C to transmission.

DVB-T: Digital Video Broadcasting - Terrestrial. It is the DVB standard for the broadcast transmission of digital terrestrial television. Compressed digital audio, digital video and other data in MPEG use it to transmission. It uses coded orthogonal frequency-division multiplexing (COFDM or OFDM) modulation.

DVB-T2: Digital Video Broadcasting - Second Generation Terrestrial. It is the extension of the television standard DVB-T. It constructed for the broadcast transmission of digital terrestrial television. It uses OFDM modulation with concatenated channel coding. Compressed digital audio, video, and other data in "physical layer pipes" (PLPs) are transmitted via this system. Increase in offered bit rate in DVB-T, makes a system more suitable for carrying HDTV signals.

RJ45 connection: It is the common name for a 8 position 8 conductor (8P8C) connector. It is commonly used for Ethernet.

S/PDIF: Sony/Philips Digital Interface Format. It is a standard audio transfer file format. It allows transmission of audio signals from one device to another in digital format over reasonably short distances. In that transmission, it does not need conversion to and from an analog format, so signal quality is not degraded. The connector cable can be coaxial (RCA connector) or optical fiber.

Line-out Output: It is a port that connects external speakers, headphones or other output devices to the TV or computer. By these sound-out connectors, sounds that the tv/computer is generated can be heard.

Common Interface (CI): It is a technology that has been used to decode a pay TV service. It separates conditional access functionality from a digital TV receiver-decoder into a removable conditional access module. Manufacturer just build CI slot and leave the CI card up to the customer to pay TV provider.

CI+ (CI Plus): It is a feature to prevent taking a copy of the broadcasting to watch the program/movie later or share it with other people.

RF: Radio Frequency. It creates an electromagnetic field for wireless broadcasting and communications when an RF current is supplied to an antenna.

FB Signal: Fast blanking is the signal on pin 16 of the SCART. It instructs the TV to select external video to be superimposed as an on screen display (OSD). It is used to select either external CVBS information or external RGB information. It has to do with the synchronization of the background video and the superimposed display source by imposing an on screen display presentation (RGB) upon a CVBS background.

YPbPr: It is a color space used in video electronics, in particular in reference to component video cables. The "Y," "Pb" and "Pr" are sets of three inputs or outputs on better video equipment and TVs. The three cables used in a YPbPr connection represent higher quality than the single-wire composite cable commonly used to hook up video equipment, because the brightness and color components of the signal are maintained separately. The YPbPr signals are derived from the red, green and blue (RGB) colors captured by a scanner or digital camera, and RGB is converted into brightness and two color difference signals (B-Y and R-Y) for TV/video.

YPbPr component video is the analog counterpart of digital component video, which is YCbCr. Whereas YPbPr uses three cables between video equipment, YCbCr uses a single cable.

HDMI (High-Definition Multimedia Interface) is an interface standard used for transferring uncompressed digital audio/video data from an HDMI-compliant device ("the source device") to a compatible digital audio device, computer monitor, video projector, or digital television. HDMI is a digital replacement for existing analog video standards.

RCA Connector: It is a type of electrical connector commonly used to carry audio and video signals. It refers to a cable with three color-coded plugs, which connect to three correspondingly-colored jacks; yellow for composite video, red for the right channel, and white or black for the left channel of stereo audio. (Composite video is analog, or non-digital, and carries all the video data in one signal. Because analog video is made up of three separate signals to begin with, squeezing them into one signal reduces the quality.) RCA cables transmit analog, or non-digital, signals. Because of this, they cannot be plugged directly into a computer or other digital device.

TV Tuner: It converts radio frequency transmission into audio and video signals allowing one to receive signals on his or her television, and therefore receive different channels. The oldest type of tuner was used to change channels and allow the television to pick up different analog signals broadcast over the air(analog). Sometimes, in order to get certain cable channels, a TV needs a special box purchased or rented from a cable company.

Some new TVs are required to include a digital tuner to allow the set to pick up digital television signals by antenna

Hybrid tuner: It is a tuner that can be configured to act as an analog tuner or a digital tuner.

Silicon Tuner: It has advantages over traditional can tuners. The signal quality of a silicon tuner surpasses that of a CAN tuner, has small size, require no special handling or materials, handle worldwide broadcast standards, simplifying inventory management and -costs

Multi System: PAL, NTSC, SECAM

PAL: Phase Alternating Line. It is an analogue television color encoding system used in broadcast television systems in some parts of Europe, Australia, Middle East and Africa. It has 625 horizontal scan lines and 25 frames per second image.

NTSC: National Television System Committee. It is the analog television system that is used in most of North America, parts of South America. It has 30 frames per second image.

SECAM: Sequential Color with Memory. It is an analog color television system used in France, Russia and other countries which do not use NTSC or PAL. It has 625 horizontal scanning lines, and displays 25 frames per second picture. It can make NTSC videos to be displayed in compatible PAL TVs. It is the first European color television standard.

VHF: Very High Frequency. It is the ITU-designated range of radio frequency electromagnetic waves from 30 MHz to 300 MHz

UHF: Ultra High Frequency. It designates the ITU radio frequency range of electromagnetic waves between 300 MHz and 3 GHz (3,000 MHz)

IEC 169-2 connector (Belling-Lee connector or TV aerial plug or PAL connector). It is the traditional antenna connector for European TV sets and FM / DAB-radio receivers. It is the oldest coaxial RF connector still commonly used in consumer devices today. It connects a receiver to a terrestrial VHF/UHF roof antenna, antenna amplifier, or CATV network, via a coaxial cable.

Antenna (aerial) impedance: It is a measure of the resistance to an electrical signal to the antenna

Tuning Systems: It is a system used for tuning the television so that it can detect the channels sent through the antenna or cable line. Once tuned, the television will memorize the channels available in that area.

PLL: Phase-Locked Loop. It is an electronic circuit with a voltage or current-driven oscillator that is constantly adjusted to match in phase the frequency of an input signal. In addition to stabilizing a particular communications channel, a PLL can be used to generate a signal, modulate or demodulate a signal, reconstitute a signal with less noise, or multiply or divide a frequency.

FST: Frequency Synthesized Tuner.

Frequency Synthesizer: It is an electronic system for generating any of a range of frequencies from a single fixed time base or oscillator.

Antenna Loop-Through: Loop Through connections are usually for TV's that have loop through connectors on them. It was a manufacturer's way of supplying some extra connectors to be able to add additional equipment so that you wouldn't have to pass your connections through one machine to another for the connections.

RF Modulator: It takes a baseband input signal and outputs a radio frequency modulated signal. It converts the video (and/or audio) output of devices such as media players, VCRs and game consoles into a format that is compatible with a TV's cable or antenna input. If you have an older Television that only has a cable/antenna connection, you will need an RF Modulator in order to connect a DVD player or DVD recorder to the Television.

Channel Bandwidth: The bandwidth of a system is the difference between the highest and lowest frequencies which the system can carry.

AFT: Auto Fine Tuning. A circuit used in a color television receiver to maintain the correct oscillator frequency in the tuner for best color picture by compensating for drift and incorrect tuning

Demodulation: It is the act of extracting the original information bearing signal from a modulated carrier wave. A demodulator is an electronic circuit that is used to recover the information content from the modulated carrier wave.

Hierarchical Modes: The hierarchical modes of the DVB-T specification provide a means by which the MPEG-2 bit stream can be divided into two parts. One stream, the high priority (HP) stream, is heavily protected against noise and interference, and whereas the second, low priority (LP), stream is much less well protected. The HP stream often carries data at a lower bit-rate than the LP stream.

DVB-T Transmission: It is based on Coded Orthogonal Frequency Division Multiplex (COFDM). COFDM uses a large number of carriers. Each of these carriers is used to transmit only a portion of the total amount of data. The data is modulated on the carriers with

QPSK or QAM. COFDM has the advantage that it is very robust against multipath reception and frequency selective fading. This robustness against multipath reception is obtained through the use of a 'guard interval'. This is a proportion of the time there is no data transmitted. This guard interval reduces the transmission capacity. There are two COFDM transmission modes possible in the DVB-T system. A 2k mode which uses 1705 carriers and a 8k mode which uses 6817 carriers. The 2k mode is suitable for single transmitter operation and for relatively small single frequency networks with limited transmission power. The 8k mode can be used both for single transmitter operation and for large area single frequency networks. The guard interval is selectable.

Guard Interval: It is used to ensure that distinct transmissions do not interfere with one another. It allows the receiver to cope with strong multipath situations. Guard Interval length can be chosen according to a trade-off between data rate and SFN (Single Frequency Network) capability : the longer the guard interval, the larger the potential SFN area.

dBm: the power ratio in decibels of the measured power referenced to one mill watt. Signal strength is traditionally measured in either percentile or dBm. Higher dBm is means better strength.

Convolutional Code: It is a type of error-correcting code in which each m-bit information symbol (each m-bit string) to be encoded is transformed into an n-bit symbol, where m/n is the code rate ($n \geq m$) and the transformation is a function of the last k information symbols, where k is the constraint length of the code. Convolutional codes are used extensively to achieve reliable data transfer. There are five valid coding rates: 1/2, 2/3, 3/4, 5/6, and 7/8.

QAM: Quadrature Amplitude Modulation, the format by which digital cable channels are encoded and transmitted via cable television providers. QAM tuners can be likened to the cable equivalent of an ATSC tuner which is required to receive over-the-air (OTA) digital channels broadcast by local television stations; many new cable-ready digital televisions support both of these standards.

Modulation: It is the transmission of a signal by using it to vary a carrier wave; changing the carrier's amplitude or frequency or phase. There are 5 allowed modulation modes: 16-QAM, 32-QAM, 64-QAM, 128-QAM, 256-QAM.

Symbol Rate: It is the number of symbol changes (waveform changes or signaling events) made to the transmission medium per second using a digitally modulated signal.

dB: Decibel. It is a logarithmic unit that indicates the ratio of a physical quantity (usually power or intensity) relative to a specified or implied reference level.

dB μ V = It is the voltage relative to 1 microvolt. It is used in television and aerial amplifier specifications

LNB: Low-noise Block Down converter. It is the receiving device on an antenna, for example a satellite dish commonly used for satellite TV reception.

PSK: Phase-shift keying. It is a digital modulation scheme that conveys data by changing, or modulating, the phase of a reference signal (the carrier wave).

BPSK. Binary Phase Shift Keying. It modulates the signal at 1 bit per symbol (a symbol is basically one oscillation of a sine wave). Because it is binary, it can only represent two states, 0 and 1. These states are represented by shifting the phase of the signal 180 degrees.

QPSK. Quadrature phase-shift keying. It modulates the signal at 2 bits per symbol (a symbol is basically one oscillation of a sine wave) It can only represent four states, 00, 01, 10 and 11. These states are represented by shifting the phase of the signal 90 degrees.

8PSK. 8 Phase Shift Keying. It modulates the signal at 3 bits per symbol (a symbol is basically one oscillation of a sine wave) It can only represent eight states, 000, 001, 010, 011, 100, 101, 110, and 111. These states are represented by shifting the phase of the signal 45 degrees.

DiSEqC. Digital Satellite Equipment Control. It is a special communication protocol for use between a satellite receiver and a device such as a multi-dish switch or a small dish antenna rotor.

ACE PRO: Advanced Contrast Enhancer PRO. It features Sony's own algorithm for local dimming, improves contrast and dynamic range by controlling the LED backlight level by area so that detail is maintained in the dark areas, while other areas are driven near peak brightness. The technology reduces unnecessary light emission resulting in true, deep blacks and reduced power consumption compared to conventional LED backlit models.

ADC: Analog-to-Digital Converter

DAC: Digital-to-Analog Converter

DSP: Digital Signal Processing. It manipulates analog information, such as sound or photographs that has been converted into a digital form.

If an incoming signal is analog (like a standard television broadcast station), the signal is first converted to digital form by an ADC. As the resulting digital signal has two or more levels which are not always at the standard values, DSP circuit adjust levels and eliminates noise. Then signal is converted back to analog by a DAC.

If a received signal is digital (like computer data), then the ADC and DAC are not necessary. The DSP acts directly on the incoming signal and eliminates irregularities caused by noise.

HE AAC: High-Efficiency Advanced Audio Coding. It is a lossy data compression scheme for digital audio. It is optimized for use with streaming audio applications where low bit rates are required such as Internet Radio, streaming music services, etc.

Dolby Digital (AC-3): It is a digital audio coding technique that reduces the amount of data needed to produce high quality sound. It reduces, eliminates or masks the noise of an audio signal. Doing that, the amount of data is reduced to one tenth of the data on a compact disk. Dolby Digital is used with digital versatile discs (DVDs), high definition television (HDTV), and digital cable and satellite transmissions. It has been selected as the audio standard for digital television (DTV) because of its popularity with film producers and consumers, its ability to use a single audio streaming video because of the down mixing feature, and its high

quality sound. The European DVB standard does not use Dolby Digital for audio, but instead uses MPEG standard technology for both the audio and video signals.

The U.S. cable television industry has also adopted Dolby Digital for DTV applications. Most television facilities are not equipped to produce 5.1 channel sound. For this reason, many DTV programs use two-channel sound. The 5.1 channel sound is used primarily for theatrical films on pay-per-view channels and at theaters. (<http://searchcio-midmarket.techtarget.com/definition/Dolby-Digital>)

DD+: Dolby Digital Plus, E-AC-3. It is a digital audio compression scheme. Compared with Dolby Digital (AC-3), it has a support for a wider range of data rates (32kbit/s to 6144kbit/s), increased channel count and multi-program support. It supports up to 15 full-bandwidth audio channels at a maximum bitrate of 6.144 Mbit/s.

DDCO: Dolby Digital Compatible Output.

IF: Intermediate Frequency Amplifier. It is a frequency to which a carrier frequency is shifted as an intermediate step in transmission or reception. The intermediate frequency is created by mixing the carrier signal with a local oscillator signal in a process called heterodyning, resulting in a signal at the difference or beat frequency. Intermediate frequencies are used in superheterodyne radio receivers, in which an incoming signal is shifted to an IF for amplification before final detection is done. It improves frequency selectivity.

VIF: Video Intermediate Frequency.

SIF: Sound Intermediate Frequency.

3D Comb Filter: It is an electronic filter that separates the luminance signal from the chrominance signal three horizontal scan lines of video at a time. The results eliminate or reduce video noise (cross-color interference or color bleeding) and dot crawl (dot interference) and gives the viewer, a clearer, sharper video picture.

SNR: Signal to Noise Ratio. It is the ratio of the video signal level to the amount of noise. The higher the signal-to-noise ratio, the higher the camera's image quality. This ratio is usually expressed in decibels but it can also be represented as a normal ratio

DNR: Dynamic Noise Reduction. It is the process of removing noise from a signal. As it does not require encoded source material, it can be used to remove background noise from any audio signal.

Temporal Noise Reduction: It compares one frame to the next and removes tiny specks that are not the same in each frame.

Cross Color Suppression: It helps to keep cross color and cross luminance to a minimum by eliminating frequency components that may result in such artifacts being generated prior to the signal output. These frequency components are virtually eliminated from the Y/R-Y/B-Y signals within the camera head through sophisticated digital three-line (NTSC)/five-line

(PAL) comb filtering, resulting in a great reduction of the cross color and dot crawl normally seen on picture monitors fed with a composite video signal.

Mosquito Noise: It is a distortion that appears near crisp edges of objects in MPEG and other video frames that are compressed with the discrete cosine transform (DCT). It occurs at decompression when the decoding engine has to approximate the discarded data by inverting the transform model.

Mosquito Noise Reduction: It recognizes object edges and removes mosquito noise to produce crisp, clear images.

De-blocking Noise Reduction: It smoothes images and decreases unwanted edges.

Blocking Artifact: It appears in compressed video material as abnormally large pixel blocks. Video uses lossy compression, and the higher the compression rate, the more content is removed. At decompression, the output of certain decoded blocks makes surrounding pixels appear averaged together and look like larger blocks. As TVs get larger, blocking and other artifacts become more noticeable.

Edge Detection: It identifies points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities.

Gaussian Noise: A random distribution of artifacts in analog video images that makes everything look soft and slightly blurry. On close inspection, one can see tiny specks in random patterns. Found on films shot with older cameras as well as films and videotapes that have been archived for a long time, dynamic noise reduction (DNR) circuits can eliminate much of the Gaussian noise when the analog material is converted to digital.

Spike Noise: An image containing spike noise will have dark pixels in bright regions and bright pixels in dark regions. This type of noise can be caused by analog-to-digital converter errors, bit errors in transmission.

Interlacing: It is a technique of doubling the perceived frame rate introduced with the signal used with analog television without consuming extra bandwidth. Since the interlaced signal contains the two fields of a video frame captured at two different times, it enhances motion perception to the viewer and reduces flicker by taking advantage of the persistence of vision effect

Deinterlacing: It is the process taking a stream of interlaced frames and converting it to a stream of progressive frames.

Motion Adaptive Deinterlace: It is an algorithm for deinterlacing pure video content. It detects which portions of the image are still, and which portions are in motion, and then applies different processing to each scenario.

Edge Oriented Deinterlacing: It detects edges in images performing jagged edge correction, restores vertical sharpness and resolution.

Film Mode Detector: It detects film mode of a series of fields of video by comparing pixels in a field adjacent the current field, with corresponding pixels directly above and directly

below the pixels in an adjacent field. The number of pixels in the adjacent in time to the current field having (or not having) a value approximately between values of the pixels above and below in the current field is assessed. Film mode for a current field may be detected by monitoring the assessment from field to field.

Original Design Manufacturer (ODM): It is a company which designs and manufactures products for another company for sale.

Stereo L/R Audio: L corresponds to left signal and R to the right. In stereo mode, one channel can have a higher bit rate than the other. The encoder can "decide" a higher bitrate would improve the quality in one channel and a lower bitrate would suffice for the other channel at any given time.

Progressive Scan Signal: It is one of the techniques which used displaying video, especially in the newer televisions. To create the image on the TV, lines which creates the image, are drawn in a standard fashion. All lines are drawn at one pass sequentially.

Interlaced Scan Signal: It is one of the techniques which used displaying video, especially in the older televisions. To create the image on the TV, lines which creates the image, are drawn in an alternating fashion. First odd numbered lines are drawn from top to bottom, and then even numbered ones are drawn. For example 1080i HDTV will draw 540 odd, then 540 even lines of resolution. It can be seen as 1080 lines of resolution by combining them. However still 1080p is slightly better than 1080i.

Subwoofer: An electronic speaker that produces low-pitched audio frequencies.

Cadence Film Detection: Also called Reverse 3:2 Pulldown. As films are recorded at 24 fps, movies have to be restored into original frame sequence when released for the television broadcast. It is a part of deinterlacing as it determines if the original movie was shot with a film or video camera.

Peaking filter: It enhance the sharpness impression in images. It can reduce image quality.

Black White Level Extension: It helps to increase the contrast levels in both dark and light images. It makes dark images more clearer and light images more vibrant. It helps to create high contrast image.

Dynamic Contrast Ratio: Contrast ratio is calculated by taking differences between the blackest black and the whitest white which the TV can display. To give a better viewing, the back lighting is adjusted dynamically for darker scenes

Local Dynamic Contrast Adjust: It gives dynamic contrast which is adapted to the local conditions at a pixel. It gives better visibility and effect for contrast enhancement based on the local information at the pixel neighborhood

YC Delay: It is when the colors in the image are not matched up with the monochrome picture in the horizontal plane. This is occurs because of the S-Video (C of Y/C) or the component video's color difference (Pb, Pr) is delayed from the luminance signal (Y).

CTI: Color Transient Improvement. It enables more sharp and clear image by reducing judder and blend effect that can be occur in color transitions.

DCTI: Digital Color Transient Improvement.

LTI: Luminance Transient Improvement. It achieves sharpness improvement by processing on the luminance.

DLTI: Digital Luminance Transient Improvement.

xvYCC Support: xvYCC means extended gamut YCbCr color space for video electronics. It supports a gamut that is 1.8 times as large as that of the sRGB color space. xvYCC helps to view all of the full range of colors that can be viewed by the human eye.

BlueStretch: It makes white colors appear brighter by increasing components of white and near white colors and decreasing red components.

MPEG: Moving Picture Experts Group. It develops standards for digital video and digital audio compression. It was formed by ISO and IEC.

MPEG Encoder: Encoding translates data between formats. MPEG Encoding captures/converts video or audio to one of the MPEG video or audio standards for distribution or for archiving.

MPEG-1: It is a standard for lossy compression of video and audio and published as ISO/IEC 11172. It is used by VCD's. Its quality is similar to VHS video and used for digital audio, TV broadcasting and creation of video CDs. Although MPEG-2 and MPEG-4 are cheaper and powerful, they do not have better compression and quality.

MPEG-2 Systems: It is formally known as ISO/IEC 13818-1 and as ITU-T Rec. H.222.0. It is used for digital TV broadcast, DVD, HD-DVD, Blu-ray discs and in most digital video broadcasting and cable distribution systems.

MPEG-4: It is a compression method (for video and audio). It is created by Moving Pictures Experts Group (MPEG) and used for low-bandwidth encoding.

MPEG-4 ASP (Advanced Simple Profile): It is defined in MPEG-4 Part 2. Most popular encoders are DivX and XviD. A MPEG-4 ASP encoder can play any video which encoded with any MPEG-4 ASP. It has coding tools like B-Frames (B-VOPS), QuarterPixel Motion Estimation (QPEL), Global Motion Compensation (GMC) and MPEG/Custom Quantization.

MPEG Transport Stream (TS): It is a standard format for audio, video and data communication transmission. It makes data join into a single transmission bit stream by multiplexing of the digital video and audio. Broadcast systems like DVB, IPTV use TS.

H.264/MPEG-4 Part 10 or AVC (Advanced Video Coding). It is a lossy compression algorithm standard for video compression and distribution of high definition video and posted by the ITU-T Video Coding Experts Group (VCEG) and the ISO/IEC Moving Picture Experts Group (MPEG). It is based on MPEG-1 and MPEG-2 and designed to transmit video

and images over a narrower bandwidth and can mix video with text, graphics and 2-D and 3-D animation layers.

Profile Level: Profiles shows the outline of certain encoding and decoding capabilities of a given video compression standard. Levels are a means of ensuring that a hardware decoder can properly decode a stream.

AspectRatio: It is the ratio of width to height of a screen or image frame. The aspect ratio for widescreen displays are 16:9 and used for DVDs, high-definition TV programming. The ratio for traditional TVs is 4:3. When a widescreen film is displayed in a non-widescreen TV, two approaches, namely Pan and Scan and Letterbox, can be used.

Pan and Scan: To convert a film to widescreen, a part of the film, named action area, is extracted and the other parts are lost.

Letterbox: To convert a non-widescreen film to widescreen, the full image is displayed. As it is not fitted to the screen, you see black bars at the left and right of the image (window boxing). Also when a widescreen film is displayed on the widescreen, you see black bars at the top and bottom of the image.

AC3: Audio Coding 3. Same as Dolby Digital. DD. It is a digital audio compression scheme. It is used for DVD encoding to match Dolby Digital's specification, but is not specifically licensed by Dolby. It can contain up to 6 channels of sound and used on DVD, Blu-ray and other digital video formats.

AC3Plus: Same as Dolby Digital Plus. DD+ or E-AC-3 (Enhanced AC-3). It is a digital compression scheme that is based on the AC3. It reduces the faults in the compression by improvement in coding techniques. It is used by Blu-ray Disc and HD-DVD Technologies and compatible with the audio portion of the HDMI interface.

AFD: Active Format Description. It is standard four bits of binary code that defines the active image area which is the area that needs to be shown. It can be sent in the MPEG Stream and SD HD video stream and holds information about their aspect ratio. It is used in the broadcast during format conversion (4:3 and 16:9) to optimally present images sent in either format.

Sampling Frequency: It is the number of samples per second of a continuously varying signal.

Data Rates: It is used to describe how quickly information can be exchanged between electronic devices.

Conditional Access (CA): It is used to protect access to digital television services by encrypting the transmitted programming. By doing this only authorized users can Access this content. It also can be used digital data broadcasts and interactive services.

To Up TV: It is a paid service which provides additional content and services through encrypted TV channels. It can be used on the DVB-T Television Broadcasting Platform.

SOG (Sync On Green): It is an RGB signal format. It combines composite sync with the green signal in RGB. So the signals which sent are red, green with Sync and blue signals.

Gamma Correction: It adjusts the image for color distortions by doing code and decode luminance values. It helps to distinguish details better in the dark areas of a picture.

HDMI CEC (Consumer Electronics Control): The devices which has HDMI-CEC feature, can be controlled by a single remote control without a programming or setup when connected with HDMI cable. Multiple devices(up to 10) can be operated with just one remote control.

HDMI 1.4: New generation HDMI standard used new 3D HD TVs. It enables faster(100Mbps) Ethernet connection without a separate Ethernet cable. It supports dual-stream 1080p 3D image, audio return channel, and more color standard.

HDCP: High-bandwidth Digital Content Protection. It is a digital content protection management mechanism which prevents data sending without encrypting in systems which used DVI and HDMI.

Fastext: It is a feature in some TVs to enable displaying teletext pages instantly when requested by storing them in advance. It is used by colored buttons in remote controller.

Toptext: It is a readjusted version of conventional teletext system used in Germany. It enables access to free info services which are provided by special Top-Text decoders.

VPS (Video Programming System), PDC (Program Delivery Control): VPS and PDC are used to automatically correct the recording time on the recorder, when you arrange a timer to record a program and it run late.

CNI (Country and Network Identification): It is the country and program code of the TV program in hexa-decimal form. CNI is transmitted with the VPS data so that it can be understood which program is to be recorded.

WSS (Widescreen Signaling): It is a digital stream which embedded in the TV signal. It handles aspect ratios of the image. It allows a WSS enhanced widescreen (16:9) TV to display programs in their correct aspect ratio.

BG, DK, L\L', I: It's the variation of the PAL system that refers to differences between the sound carrier and the vision carrier and also country. B/G is used for most of Europe and Australia, I is used for UK and South Africa, L is used for France, and D/K is used for Russia, Poland and Romania.

Nicam (Instantaneous Companded Audio Multiplex): It is a lossy compression for digital (stereo) audio developed in 1970s. It is used when broadcasting for transmissions of stereo TV sound to the public. It offers one digital stereo sound channel, two mono sound channels and 704Kbit/sec data.

German, A2 (Zweikanalton, two channel sound): It a TV sound transmission system. It used in countries that use PAL-B/G like Germany. It has two separate Frequency Modulation

carriers. So it can be used for bilingual broadcasts as well as stereo. Unlike NICAM standard, A2 is an analog system.

Dynamic Bass: It means using more than a single power supply which is found in classic bass amps.

Equalizer: It is an audio device which is used for altering the frequencies of sound recordings

Dual I = Mono (1 channel).

Dual II = 2 channel stereo (Left + Right).

SRS (sound retrieval system): It is 3D audio processing technology which uses head related transfer functions (studies how a sound from a point will arrive at ear). It widens the center point between two speakers where one hears the audio in full quality and creates an immersive 3D soundfield by a more spacious sense of ambience.

Spatial Audio: It is a technology when two speakers' sound are heard as if each one placed anywhere in space.

Panorama: It is an image captured in a wider format which is much more than eyes can see naturally.

4:3 (four by three): It is aspect ratio of video picture which used in standard TV's for decades.

16:9 (sixteen by nine): It is aspect ratio of the new HDTV standard. It is a standard for HDTV, digital television and analog widescreen TVs (EDTV) PALPlus.

14:9 (fourteen by nine): It is a compromise aspect ratio. Both 4:3 and 16:9 can use it. Its material is derived from 16:9 or 4:3 shot.

OAD (On-Air Download): It is a method of upgrading TV automatically.

EPG (Electronic Program Guide): It is a TV application to provide a list of programs for each channel, scheduled information and short information about each program.

MHEG: Multimedia and Hypermedia Experts Group

MHEG5 (ISO/IEC 13522-5): It is an open standard about presentation of multimedia information and used as a language to describe interactive television services. It is published by MHEG. It provides interactive services beyond broadcast video.

MHEG5 1.06: It is UK Profile of MHEG-5. As it is a stricter standard, writing MHEG code which works on all receivers is easier. It also can scale quarter-screen video up to full size and vice versa. It is Standard Definition (SD) whereas MHEG5 2.0 is for High Definition.

DISEQC (Digital Satellite Equipment Control): It is a control protocol to communication used between satellites and external equipment. It is developed by EUTELSAT to control

equipment such as rotor, LNB, switch via receiver. It enables to control different external units with a single coaxial cable by receiver.

Digital Program Storage: It is the storage and insertion of extra, pre-recorded broadcast programming like local content, advertising or VOD (video-on-demand).

AVL (Automatic Volume Level): It is a feature which limits the maximum volume level and enables maintaining an equal volume level automatically when the channel is changed.

Swap: It is the feature which enables user to return the previously viewed channel.

Zapp: It is a function for smart channel search.

NIT (Network Information Table) Search: It conveys information about satellite, the characteristics of the network, channel list from the satellite.

Child Lock: It is a feature to control TV watching time and the channels for children. One can lock certain channels or lock the TV at certain hours of the day.

Panel Lock: It is a feature which allows saving (locking) specific settings on TV. It also helps to prevent children from inappropriate programs.

Picture Format Switching: It enables the Picture to be in the right format by changing between 16:9 and 4:3. The information which used for picture format switching can be come from WSS data, scart pin 8. Pin 8 -the switch signal pin- carries a signal. So between 4.5 V and 7 V means a 16:9 signal whereas; signal between 9.5 V and 12 V means 4:3 signal.

Auto RGB Detect Through Scart1 (Pin16)

Pin 16, the blanking signal pin, carries a signal. By that one can understand if the signal RGB or composite. So a signal between 0 V and 0,4 V means composite whereas; signal between 1 V and 3 V means RGB.

DDC (Display Data Channel) Support: It is a communication protocol which works like a “Plug and Play” compatibility. It is used between display and graphics adapter transmission for automatic configuration of the hardware like display modes, brightness.

Timer: It is a feature which adjusts the time to turn the power on and off.

Eco Standard: It is a feature to make the TV power off, if no action is detected.

Encoding: It is used for efficient transmission or storage by putting a sequence of characters into a specialized format. It can be used to analog to digital conversion and vice versa.

Decoding: It is the process of the transforming the encoded format back to the original format.

DV (Digital Video): It is a standard which image and sound are recorded in moving picture cameras. It can make analog or digital recording in 8mm with VHS; whereas it can make digital recording in Mini DV. The recording in video CD and DVD is only digital.

MJPEG (Motion JPEG): It is a video format which compresses the video as individual images with jpg conversion. It is used devices like by digital cameras, webcams, non-linear video editing systems.

MP3 (MPEG-1 or MPEG-2 Audio Layer III): It is an encoding format for digital audio for compression of audio signals. It increases the frequency resolution and shrinks the original sound data from a CD without sacrificing sound quality.

AAC (Advanced Audio Coding): It is a technique for compressing digital audio files. Officially part of the MPEG-4 standard. It is the successor to MP3. It is like MP3. Both are use lossy algorithm, so the original digital audio cannot be recreated from the compressed bits alone. It offers same or higher quality than MP3 using smaller sized files. It is most widely used to create small digital audio files.

WMA (Windows Media Audio): It is a technology used for audio data compression. It is both a file format and audio code. It is used for encoding digital audio files which is like MP3. It claims to compress files at a higher rate than MP3.

WMV (Windows Media Video): It is a video compression format which is usually packed into Advanced Systems Format and developed by Microsoft. It helps to compress large files at a higher quality.

WAV (Waveform Audio File Format): It is a lossless PC audio format standard. It could be used as a medium for other platforms.

Time Shift: It is a technology where a live TV can be paused and then resumed to watch later.

Fast Picture Search: It is a technology where one can scan at high speed through the video by disjointed pictures on the screen.

Picture by picture playback: The replaying of previously recorded moving images.

A-B Repeat: It is a technology which enables one to select start and end spots in a track. Then one can listen everything between these two points in loop.

USB (Universal Serial Bus): It is a serial connection type which can enable devices to connect the computer. The output value of a standard USB 2.0 is 4.00 volt, 500 ma. It can use enable devices to work instantly when plugged to computer –plug and play-.

M4A (MPEG 4 Audio): It is an audio file extension. It is similar to MP3, but it enables higher quality at smaller size.

JPE: It is a file extension which used for JPEG files.

BMP (BitMaP): It is an image file extension which used no compression. It used on on Microsoft Windows platform. As there is no compression, it holds more space than PNG and JPEG.

TIFF (Tagged Image File Format): It is a file extension used for storing high quality images which are not compressed. It is used in photography and printing.

3GP: It is a video container format developed for the third generation (3G) mobile phones. It is a simplified version of the MP4 format and designed to watch videos on mobile phones by reducing storage and bandwidths.

MOV: It is a multimedia file extension, developed by Apple. It is the default video format of QuickTime. As it has high quality picture even in small sized files, it is used by movie industry to distribute trailers, advertising videos and online television channels. Its popularity is decreased after the use of FLV and DivX.

DivX: It is a video compression format which uses a higher compression method and reduces quality loss. Its compression method is compatible with the MPEG-4 part 2 “Advances Simple” profile.