

AN XML-BASED FEATURE MODELING LANGUAGE

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

BY

LEILI NABDEL

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

SEPTEMBER 2011

Approval of the thesis:

AN XML-BASED FEATURE MODELING LANGUAGE

Submitted by **LEILI NABDEL** 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 Eng. Dept., METU**

Dr. Ahmet Serkan Karataş
Co-Supervisor, **Computer Eng. Dept., METU**

Examining Committee Members:

Assoc. Prof. Dr. Ferda Nur Alpaslan
Computer Engineering Dept., METU

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

Assoc. Prof. Dr. Cem Bozşahin
Computer Engineering Dept., METU

Assist. Prof. Dr. Pınar Şenkul
Computer Engineering Dept., METU

Dr. Bülent Mehmet Adak
Senior Engineer, ASELSAN

Date: 16. 09.2011

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 : Leili N Abdel

Signature :

ABSTRACT

AN XML-BASED FEATURE MODELING LANGUAGE

Nabdel, Leili

M.Sc., Department of Computer Engineering

Supervisor : Assoc. Prof. Dr. Halit Oğuztüzün

Co-Supervisor: Dr. Ahmet Serkan Karataş

September 2011, 136 pages

Feature modeling is a common way of representing commonality and variability in Software Product Lines. There are alternative notations reported in the literature to represent feature models. Compared to the graphical notations, the text-based notations are more amenable to automated processing and tool interoperability. This study presents an XML-based feature modeling language to represent extended feature models that can include complex relationships involving attributes. We first provide a Context Free Grammar for the extended feature model definitions including such complex relationships. Then we build the XML Schema Definitions and present a number of XML instances in accordance with the defined schema. In addition, we discuss a validation process for the validation of the XML instances against the defined schema, which also includes additional tasks such as well-formedness checking for the XML instances.

Keywords: Feature Modeling Language, Extended Feature Model, Complex Constraint, XML

ÖZ

XML TABANLI ÖZELLİK MODELLEME DİLİ

Nabdel, Leili

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Yöneticisi : Doç. Dr. Halit Oğuztüzün

Ortak Tez Yöneticisi: Dr. Ahmet Serkan Karataş

Eylül 2011, 136 sayfa

Özellik modelleme, yazılım ürün hatlarında ortaklık ve değişkenliğin gösterilmesinde yaygın olarak kullanılan bir yoldur. Literatürde özellik modellerin ifadesi için farklı gösterimler bulunmaktadır. Metin tabanlı gösterimler, grafik gösterimlere nazaran, otomatik işleme ve araçlar arası uyumluluk gibi kıstaslar göz önüne alındığında daha elverişli çözümler sağlamaktadır. Bu çalışmada özneliklerin yer aldığı karmaşık ilişkiler içerebilen genişletilmiş özellik modellerinin gösterimi için XML tabanlı bir özellik modelleme dili sunulmaktadır. İlk olarak, karmaşık ilişkiler içerebilen genişletilmiş özellik modelleri için bir bağlamdan-bağımsız gramer sunulmaktadır. Daha sonra XML şema tanımları kurulmakta ve bu şemaya uygun birtakım XML örnekleri verilmektedir. Ayrıca, XML örneklerinin tanımlanmış şemaya uygunluğunun gösterilebilmesi için bir doğrulama süreci de tartışılmaktadır. Bu süreç XML örneklerinin biçimsel olarak düzgünlüğünün kontrolü gibi ek görevler de içermektedir.

Anahtar kelimeler: Özellik Modelleme Dili, Genişletilmiş Özellik Modeli, Karmaşık İlişki, XML

“To My Family”

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to Assoc. Prof. Dr. Ali Dođru for his guidance, advice, and encouragement throughout the research.

TABLE OF CONTENTS

ABSTRACT	iv
ÖZ	v
ACKNOWLEDGEMENTS	vii
TABLE OF CONTENTS	viii
CHAPTER	
1. INTRODUCTION.....	1
2. BACKGROUND AND RELATED WORK	6
2.1 Feature Models and Feature Representations	6
2.1.1 Feature Models.....	6
2.1.2 Feature Relationships	7
2.1.2.1 Mandatory Relations	7
2.1.2.2 Optional Relations.....	7
2.1.2.3 Alternative Relations.....	8
2.1.2.4 Or Relations	8
2.1.2.5 Requires Relations	8
2.1.2.6 Excludes Relations	8
2.2 Graphical Notations for Feature Diagrams	10
2.2.1 Feature Oriented Domain Analysis (FODA)	10
2.2.2 FODA Subsequent Proposals.....	11
2.3 Extended Feature Models.....	11
2.3.1 Cardinality-based Feature Models	11
2.3.2 Attribute Included Feature Models	13
2.4 Textual Notations for Feature Diagrams.....	14
2.4.1 Feature Description Language (FDL)	15
2.4.2 Text-based Variability Language (TVL)	15
2.4.3 XML-based Feature Modeling Language	17
2.4.3.1 XML-based Feature Modeling (XFeature)	17
2.4.3.2 Feature Modeling Markup Language (FeatureML).....	18

2.4.3.3 The Feature Modeling Plug-In for Eclipse (FeaturePlugin)	19
2.4.3.4: Tooling a Framework for the Automated Analysis of Feature Models: (FAMA)	19
2.4.3.5 A Tool Framework for Feature Oriented Software Development: FeatureIDE	20
2.5 Automated Reasoning on Feature Models	20
2.5.1 Automated Support on the Automated Analysis of Feature Models	21
2.5.1.1 Propositional Logic-based Analysis	21
2.5.1.2 Description Logic-based Analysis	22
2.5.1.3 Constraint Programming-based Analysis	22
2.5.1.4 Other Proposals	22
3. CONTEXT FREE GRAMMAR EXPRESSING EXTENDED FEATURE MODELS	23
3.1 Context Free Grammar (CFG)	23
3.1.1 Backus-Naur Form (BNF)	24
3.1.2 Extended BNF (EBNF)	24
3.2 CFG for Extended Feature Models	25
3.2.1 A CFG for Expressing Features	26
3.2.2 A CFG for Expressing Decomposition Relationships	27
3.2.2.1 A CFG for Expressing Cardinality-based Decomposition Relationships	29
3.2.3 A CFG for Expressing Complex Cross-Tree Relationships	31
3.2.3.1 A CFG for Expressing Condition Relationship	31
3.2.3.2 A CFG for Expressing Excludes Relationship	34
3.2.3.3 A CFG for Expressing Requires Relationship	35
3.2.3.4 A CFG for Expressing Complex Constraint Relationship	37
3.2.3.5 A CFG for Expressing Guarded Constraint Relationship	39
3.2.3.5.1 A CFG for Expressing Guarded Constraint Combination Relationship	40
4. XML SCHEMA DEFINITION FOR EXPRESSING EXTENDED FEATURE MODELS	42
4.1 XML Schema Definition (XSD)	42
4.1.1 XSD <schema> Element	43

4.1.2 XSD Simple Elements	44
4.1.2.1 XSD Restrictions.....	44
4.1.3 XSD Complex Elements	45
4.1.3.1 XSD Indicators.....	45
4.2 XSD for Extended Feature Models	45
4.2.1 XSD for Expressing Features	46
4.2.2 XSD for Expressing Decomposition Relationships	48
4.2.3 XSD for Expressing Complex Cross-Tree Relationships	55
4.2.3.1 XSD for Condition Relationship	56
4.2.3.2 XSD for Expressing Excludes Relationship	60
4.2.3.3 XSD for Expressing Requires Relationship.....	61
4.2.3.4 XSD for Expressing Complex Constraint Relationship.....	64
4.2.3.5 XSD for Expressing Guarded Constraint Relationship.....	65
4.2.3.5.1 XSD for Expressing Guarded Constraint Combination Relationship.	67
5. EXTENSIBLE MARKUP LANGUAGE INSTANCES EXPRESSING EXTENDED FEATURE MODELS.....	69
5.1 eXtensible Markup Language (XML).....	69
5.1.1 XML Elements.....	70
5.1.2 XML Attributes.....	71
5.1.3 XML Tree.....	71
5.2 XML Instances for Extended Feature Models	71
5.2.1 XML Instances for Expressing Features.....	72
5.2.2 XML Instances for Expressing Decomposition Relationships	75
5.2.3 XML Instances for Expressing Complex Cross-Tree Relationships	78
5.2.3.1 XML Instances for Expressing Condition Relationships.....	78
5.2.3.2 XML Instances for Expressing Excludes Relationship.....	80
5.2.3.3 XML Instances for Expressing Requires Relationship	81
5.2.3.4 XML Instances for Expressing Complex Constraint Relationship	84
5.2.3.5 XML Instances for Expressing Guarded Constraint Relationship.....	88
5.2.3.5.1 XML Instances for Expressing Guarded Constraint Combination	93
6. FEATURE MODEL MARKUP LANGUAGE VALIDATION	97
6.1 XSD Validation.....	98

6.1.1 Valid and Invalid Instances.....	99
6.2 XML Well-Formedness Validation.....	103
6.2.1 Valid and Invalid Instances.....	106
7. CONCLUSION	114
REFERENCES.....	116
APPENDIX	
A. GRAPHICAL XML SCHEMA STRUCTURE FOR THE EXPRESSIONS DEFINED IN CHAPTER 4	120

LIST OF FIGURES

FIGURES

Figure 1.1 The envisioned tool structure [8].....	4
Figure 2.1 An example feature model [13].....	7
Figure 3.1 Sample BNF grammar	24
Figure 3.2 Sample feature with attributes	27
Figure 3.3 Mandatory relationship.....	28
Figure 3.4 Optional relationship	28
Figure 3.5 Alternative relationship	29
Figure 3.6 Or relationship	29
Figure 3.7 Group cardinality.....	30
Figure 3.8 Solitary cardinality.....	31
Figure 3.9 A sample feature model.....	33
Figure 3.10 A sample feature model expressing Excludes relationship	34
Figure 3.11 A sample feature model.....	36
Figure 3.12 A sample feature model.....	38
Figure 3.13 A sample feature model.....	40
Figure 3.14 A sample feature model.....	41
Figure 4.1 A simple schema structure.....	44
Figure 4.2 A sample of restrictions from [10]	44
Figure 4.3 XSD representation for feature.....	47
Figure 4.4 XSD representation for feature attributes.....	47
Figure 4.5 XSD representation for all of the features	48
Figure 4.6 XSD representing Decomposition relationship	49
Figure 4.7 XSD expressing Mandatory relation	50
Figure 4.8 XSD expressing Optional relation	50
Figure 4.9 XSD expressing Alternative relation.....	51
Figure 4.10 XSD expressing Or relation.....	51
Figure 4.11 XSD expressing Feature Cardinality	52
Figure 4.12 XSD expressing Solitary Cardinality	53
Figure 4.13 XSD expressing Group Cardinality.....	53
Figure 4.14 XSD expressing Cardinality	54
Figure 4.15 XSD expressing ChildSet	54
Figure 4.16 XSD expressing cross-tree relationships	55
Figure 4.17 XSD expressing Condition relationship	56
Figure 4.18 XSD expressing ExpressionRelOpExpression	57
Figure 4.19 XSD expressing the structure of Expression	58
Figure 4.20 XSD expressing BoolAttributeDesequaltoTruthValue.....	59
Figure 4.21 XSD expressing NumericAttributeDes.....	60
Figure 4.22 XSD expressing Excludes relationship.....	61
Figure 4.23 XSD expressing Requires relationship	62
Figure 4.24 XSD expressing the structure of Boolean Formula	62
Figure 4.25 XSD expressing BooleanFormulaConBooleanFormula.....	63
Figure 4.26 XSD expressing Propositional Logic Connectives	63

Figure 4.27 XSD expressing ComplexConstraint relationship	64
Figure 4.28 XSD expressing Complex Part (CP)	65
Figure 4.29 XSD expressing GuardedConstraint	66
Figure 4.30 XSD expressing the structure of Guard	66
Figure 4.31 XSD expressing GuardedConstraintCombination	67
Figure 4.32 XSD expressing NegationGuardedConstraint	68
Figure 5.1 XML instance expressing Attribute value	71
Figure 5.2 XML tree structure [6]	71
Figure 5.3 A sample feature model for a Computer	72
Figure 5.4 XML instance expressing the Feature Model	73
Figure 5.5 XML instance for expressing Mandatory relation	75
Figure 5.6 XML instance for expressing Optional relation	76
Figure 5.7 XML instance for expressing Alternative relation	76
Figure 5.8 XML instance for expressing Or relation	77
Figure 5.9 XML instance for expressing SolitaryCardinality	77
Figure 5.10 XML instance for expressing GroupedCardinality	78
Figure 5.11 XML instance expressing Condition relationship including ExpressionRelOpExpression	79
Figure 5.12 XML instance expressing Condition relationship including BoolAttributeDesequaltoTruthValue	80
Figure 5.13 XML instance expressing Excludes relationship	81
Figure 5.14 XML instance expressing Requires relationship	82
Figure 5.15 XML instance expressing Requires relationship	83
Figure 5.16 XML instance expressing Complex Constraint relationship	85
Figure 5.17 XML instance expressing Complex Constraint relationship	86
Figure 5.18 XML instance expressing Guarded Constraint relationship	89
Figure 5.19 XML instance expressing Guarded Constraint relationship	91
Figure 5.20 XML instance expressing Guarded Constraint Combination relationship	93
Figure 6.1 Validating processes	98
Figure 6.2 Mandatory relation	99
Figure 6.3 XML instance expressing Mandatory relation according to Figure 6.2 and XSD illustrated in Figure 4.7	100
Figure 6.4 Valid XML instance expressing Mandatory relation	100
Figure 6.5 XML instance expressing Mandatory relation according to Figure 6.2 and XSD illustrated in Figure 4.7	101
Figure 6.6 Unacceptable XML instance expressing Mandatory relation	101
Figure 6.7 A Sample feature including Attribute	102
Figure 6.8 XML instance expressing feature including Attribute according to Figure 6.7 and XSD illustrated in Figure 4.4	102
Figure 6.9 Unacceptable XML instance expressing feature including attribute	103
Figure 6.10 Feature model sample	104
Figure 6.11 XML validating process	104
Figure 6.12 XML instance illustrating Mandatory relationship according to Figure 5.3 and XSD illustrated in Figure 4.7	106
Figure 6.13 Valid XML instance expressing relationship among features	107

Figure 6.14 XML instance illustrating Mandatory and Optional relationships according to Figure 5.3 and XSD illustrated in Figure 4.7 and 4.8	107
Figure 6.15 Unacceptable XML instance expressing the relationships among features	108
Figure 6.16 XML instance expressing Grouped Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.13	109
Figure 6.17 Acceptable XML instance expressing the Feature Cardinality	110
Figure 6.18 XML instance expressing Feature Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.11	110
Figure 6.19 Unacceptable XML instance expressing Feature Cardinality	111
Figure 6.20 XML instance expressing Feature Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.11	112
Figure 6.21 Unacceptable XML instance expressing Feature Cardinality	113

LIST OF TABLES

TABLES

Table 2.1 Symbols used for designing a feature model.....	9
Table 2.2 Most common multiplicities	12
Table 2.3 Cardinality notations with explanations.....	13
Table 2.4 Symbols for designing a feature model including Attributes	14
Table 3.1 EBNF symbols	25
Table 5.1 Entity references [6].....	70
Table 6.1 Java Libraries applied for validating XML instances according to XSD.....	99
Table 6.2 Java Libraries applied for validating XML instances	105

CHAPTER 1

INTRODUCTION

As the use of software systems started to increase rapidly in a more complex way and software business become the bottom line for many organizations, it became more difficult to handle the new problems of software systems with the traditional methods in terms of quality, cost and time. As a result, software reuse becomes a significant concept employed by many applications. By reusing existing assets, development costs will be reduced and a faster development will be possible. Consequently, efficiency and productivity will be improved. A number of new approaches for software reuse have been provided over the past years. The introduction to Object Oriented Programming paradigm (OOP) [1] is one the significant approaches to software reuse. OOP supports software reusability introducing concepts such as polymorphism, encapsulation and inheritance [2].

Over the past few years, a new approach to software reuse has gained considerable attention both by industry and academia. It is known as Software Product Line development [3] which refers to a set of software-intensive systems that share a common set of features in order to resolve the specific needs of a market organization or specific mission. The product line concept has been used by the manufacturing industry for a long time in order to decrease human effort, cost and time consumption by exploiting commonalities between products. As an example, Boeing, Ford and McDonalds are some companies that apply product lines for a long time. However, product line concept for software industry is relatively new. Although software product line development might seem just like traditional software reuse, it is lot more complex. Software Product Line Engineering is "a

paradigm to develop software applications (software-intensive systems and software products) using platforms and mass customization" [3].

Managing commonality and variability in software product lines is a key concept. Variability defines the scope of product family via predicting which family members may change over the lifetime of family. Commonality specifies the common products in a product family. Feature models establish an important foundation for product line development and play an important role in product lines success. The commonalities and variabilities of the product line are presented in the problem space [3]. Whereas solution space, describes the required platform elements and additional application parts. Problem space can be defined using a feature model or a Domain Specific Language (DSL) [4]. In order to illustrate the solution space, different options are applicable such as DSL compilers and component libraries.

Feature modeling (FM) is a key activity for managing commonality and differences in Software Product Lines [3] and feature models are hierarchical models specifying the common and different parts of a product line. In most researches they have been quoted as one of the most important contributions to Software Product Line (SPL) modeling. Using feature modeling, the domain of the feature model will be specified in order to compare with other domains. Generally, a domain is an abstract space where common requirements, functionality and terminology of related software systems can be shared. Commonalities specified in the model indicate obviously where reuse opportunities are and complex interactions between features become apparent. As an example, suppose we want to buy a new car. In order to buy a new car, we can configure some features such as color, product year, power, etc. A feature model is a way represents all these features in a single model. It illustrates the information of all available products of a software product line [5]. Furthermore, it represents the possible products with features and relationships among them.

In order to represent feature models different approaches are presented so far. Some of these approaches are graphical; on the other hand others are textual. Compared with graphical notations, text-based notations can be more amenable to automated

processing and tool interoperability. Both of the approaches are going to be described in detail in the subsequent chapter.

In this work, we propose a text-based feature modeling language. The eXtensible Markup Language (XML) [6] is utilized to encode feature models through a language. XML is a universally accepted standard way of structuring data and is recommended by World Wide Web Consortium (W3C) since February 10, 1998. XML is much like HTML. But, it is designed for different aims; XML is designed in order to transport and store data. On the other hand, displaying the data is the most significant goal for designing HTML. The marketplace supports XML with a wide selection of free or inexpensive tools, and all modern browsers have a built-in XML parser that works straightforward over the Internet. It is an open standard that no single vendor can make changes on it. In addition, according to the textual encoding of it, any source code illustrated in XML can be interchanged among different computer systems. Domain-specific XML-based markup languages have proved to be a convenient means of exchanging models among tools for developers. As a result, we believe that XML provides a flexible and extensible framework to our feature modeling language.

In this work we propose an XML-based feature modeling language to represent the structure of feature models. It accounts for the features including attributes. By inclusion of attributes more information about features can be provided. In addition, the relationships among the features can be defined which consists of decomposition and cross-tree relationships covered in next chapter. Furthermore, complex cross-tree constraints which may include feature-feature, feature-attribute and attribute-attribute relationships can be presented with the language presented in this work. Some parts of the thesis work appears in [7] as an introduction to an XML-based feature modeling language. Figure 1.1 illustrates the envisioned tool structure accessed from the [8] and this thesis work covers the FMML [7] part in order to be the tool language.

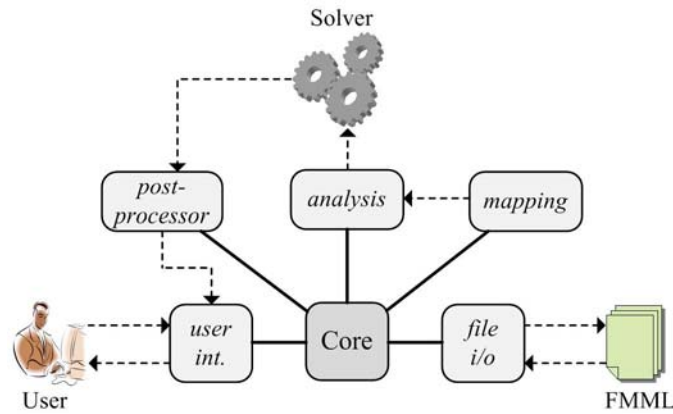


Figure 1.1 The envisioned tool structure [8]

In this Figure the *file i/o* component retrieve or store the feature modeling language the tool can process it. The *user int.* component has the capability of processing the user language which should be comprehensive language for the user. It can be a combination of both textual and graphical notations. The *mapping* component performs the translations to the required notation, and the *analysis* component applies an off-the shelf constraint solver for the analysis operations. The *post-processor* component illustrates the results provided by the solver in a more user friendly and smooth way.

This thesis document is divided into seven chapters. The organization of this paper is as follows: second chapter will provide necessary knowledge in order to understand the concepts discussed in this thesis work. It will discuss feature models with related examples and the relationships among them. We will also describe the important extensions for feature models in detail. It will also cover the different approaches for presenting feature models and point to the related works. In third chapter, a Context Free Grammar (CFG) [9] for feature models and relationships among them is illustrated. The grammar also covers the complex cross-tree constraints according to the definitions specified in detail. Fourth section covers the structure of the XML Schema Definition (XSD) [10] built on the definitions and related CFG presented in the third section. Fifth chapter contains a case study in order to illustrate how the proposed approach is utilized. The Sixth chapter will present a validation process in order to check the validity of the XML instances according to the XSD and do extra

well-formedness controls. The last chapter provides an analysis of the proposed approach, concludes the document, and points to future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This section covers the concept of feature models and related extensions. In addition, different approaches for presenting feature models will be illustrated. The section will also provide a brief explanation of the works which are related to our research study.

2.1 Feature Models and Feature Representations

2.1.1 Feature Models

A feature is a distinct property or aspect of a software system such as components related to same stakeholder of the system. A feature model is a hierarchical structure of system requirements of a given problem domain [11]. Feature diagrams are an important part of a feature model which is used to illustrate common and variable features and their relations. The feature diagram is illustrated with a graph including nodes and edges in a 2-D space. It starts with a node representing features at the root position and at the next level beneath the features will follow hierarchically. A high-level node may illustrate a concept or a feature in a specific domain. Feature models are commonly used in order to provide a compact and comprehensive view of all the products of an SPL in terms of features. They support variability management in SPLs [12]. An example of feature model from [13] is provided below.

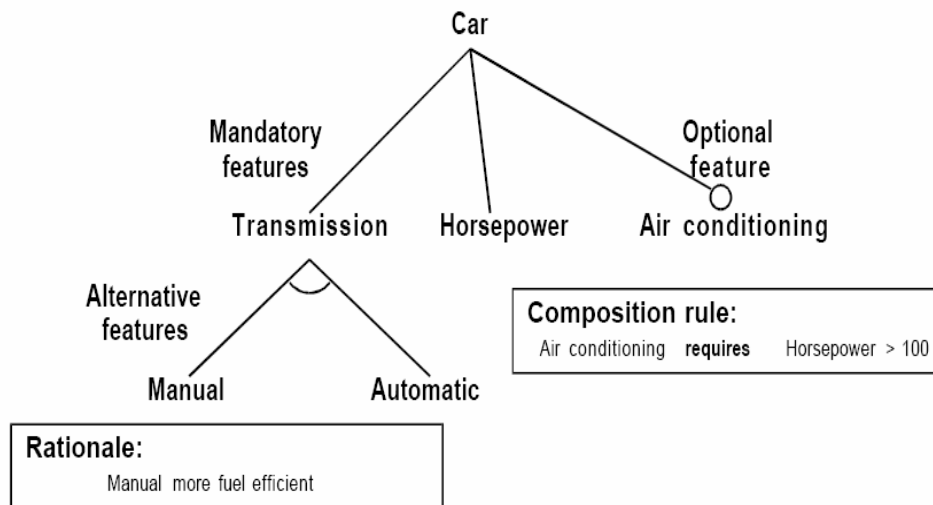


Figure 2.1 An example feature model [13]

2.1.2 Feature Relationships

There are two kinds of relationships among features in basic feature models: decomposition relationships and cross-tree relationships. Decomposition relationships determine hierarchically arranged set of features. The relationship is between a parent and its child features (or sub features) and includes four relations which are written as follows:

2.1.2.1 Mandatory Relations

A child feature has a *mandatory* relationship with its parent when the child is included in which its parent feature is included in a model as well. A mandatory relation is illustrated with a simple edge between parent and child feature with a filled circle on its connection point to the child feature.

2.1.2.2 Optional Relations

A child feature has an *optional* relationship with its parent when the child can be optionally included in which its parent feature is included in a model. An optional

relation is shown with a simple edge between parent feature and child feature with an empty circle on its connection point to the child feature.

2.1.2.3 Alternative Relations

A set of child features have an *alternative* relationship with their parent when only one feature of the child features must be included when its parent feature is included in a model. An alternative set is provided by an empty arc connecting the parent feature to the child set of features.

2.1.2.4 Or Relations

A set of child features have an *Or* relationship with their parent when one or more feature of the child features can be included when its parent feature is included in a model.

Note that inclusion of the child feature is possible only when its parent feature is included in the model of a product. Furthermore, the root feature is a part of all the model of the product line.

In addition to the parental relationships between features in a basic feature models, a feature model may include cross-tree constraints between features which are written and explained below:

2.1.2.5 Requires Relations


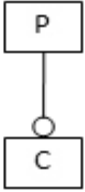
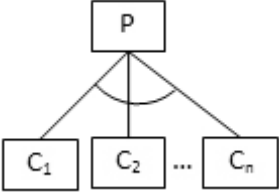
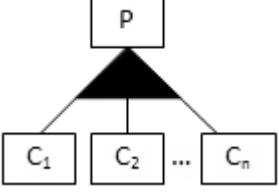

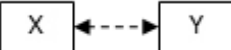
If a feature A requires feature B, the inclusion of A in a model of a product implies the inclusion of B as well in that model.

2.1.2.6 Excludes Relations

If a feature A excludes a feature B, the inclusion of A in a model of a product implies the exclusion of B in such a model.

The notations of these explanations are shown in table 2.1 which is similar to the one proposed by Benavides *et al.* in [5].

Table 2.1 Symbols used for designing a feature model

Symbol	Name
	Mandatory Relation
	Optional Relation
	Alternative Relation
	Or Relation
	Requires Relation
	Excludes Relation

In addition to the represented relationships among features, the proposed language in this work accounts for the features with attributes and more complex cross tree constraints which will be discussed in subsequent chapters in detail.

2.2 Graphical Notations for Feature Diagrams

Human designers may be more convenient using graphical notations since graphical models are more intuitive and editing the graphical notations may be much easier. This section will cover the feature modeling languages using graphical notation proposed so far.

2.2.1 Feature Oriented Domain Analysis (FODA)

Feature diagrams were introduced by Kang *et al.* as a part of Feature Oriented Domain Analysis (FODA) [13] back in 1990. It is a domain analysis method developed at the Software Engineering Institute (SEI). The FODA method provides software reusability in both functional and architectural levels. Domain analysis provides the scope of the domain product according to the domain requirements and represents the common functionality and architecture of applications in a domain. FODA analysis the domain in three steps written as below:

- Context analysis in order to understand the scope
- Domain modeling in order to illustrate the problem and system requirements. They provide features of the domain, a standard vocabulary of domain experts, documentation of entities and generic software requirements.
- Architecture modeling in order to establish the solution space; the presentations provide developers with architectural models. The models may also provide the libraries of existing components.

The domain analysis gained considerable success applying in many mature domains. In general, there are some criteria results the domain analysis become successful which is explained below:

- The scope of the domain should be suitable; for instance the size of the domain for analysis is feasible.
- A complete and comprehensive abstraction of requirements should exist from the application level to the problem level.
- Providing a documentation of the problem abstraction in order to specify the requirements in detail is needed.

2.2.2 FODA Subsequent Proposals

FODA gained a large popularity due to number of unique characteristics. One of them is including a comprehensive and straightforward scheme in order to classify the domain knowledge in order to make it understandable for human. This leads to the development of several extended FODA model over the years. One of the most important extensions for FODA is FeaturSEB [14] which is an integration of feature modeling with the Reuse driven Software Engineering Business (RSEB). RSEB is a use-case driven reuse process in which the architecture of the system is described using use-cases and subsequently transformed into object models. FODA and FeaturSEB are both model-driven approach with a domain knowledge captured from different models with their specific requirements.

In addition to FaeturSEB which is an extension for FODA concept, there are other extensions which will be explained in next section in detail.

2.3 Extended Feature Models

2.3.1 Cardinality-based Feature Models

After FODA, several extensions to feature models were proposed. One particular extension to feature models is UML-like cardinalities [11] which are used in decomposition relations. Feature diagrams have multiplicities that are used when a set of features are going to be applied. The most common multiplicities are shown in table 2.2.

Table 2.2 Most common multiplicities

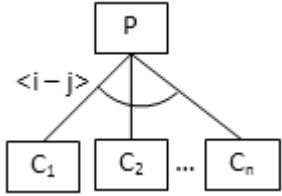
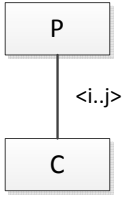
Multiplicity	Explanation
0..1	At most one feature should be chosen from the set
1	Exactly one feature should be chosen from the set
0..*	Zero or more feature(s) should be chosen from the set
1..*	At least one feature should be chosen from the set

The proposed work illustrates multiplicities of features in an understandable way. In addition, it unifies the notation of multiplicity in feature modeling and UML. Note that other multiplicities beside the common ones can be applied such as 0..3, 1..3 or simply 3. In [15] Riebisch *et al.* has been introduced the use of UML multiplicities as group cardinalities in order to illustrate multiplicities in the form of $\langle m..n \rangle$.

Beside the proposed extension, in [16, 17] there is another cardinality-based extension including feature cardinalities as well as group cardinalities. Czarnecki *et al.* proposes feature cardinality as an interval of the form $\langle m..n \rangle$ where $m \in \mathbb{Z} \cup n \in \mathbb{Z} \cup \{\infty\}$ and $0 \leq m \leq (m \leq n \cup n = \infty)$. In the cases that any feature of the feature model has feature cardinality, the hierarchy of these features will be called cardinality-based feature models. Notice that the Kleene star * in the formula, shows the possibility of selection of the feature in unbounded number of times. In this case the feature with the cardinality [1..1] is equal to *mandatory* and [0..1] is equal to *optional* relation concept in feature modeling language.

Table 2.3 illustrates the notation of cardinalities with their related meaning.

Table 2.3 Cardinality notations with explanations

Symbol	Name	Explanation
	Group Cardinality	P is a parent feature of features ($C_1, C_2, \dots,$ and C_n) with a group cardinality $\langle i..j \rangle$. If P is included in the model, then at least i and at last j of the child features must be included.
	Solitary Cardinality	P is a parent feature and C is the solitary child feature. If P is included with solitary cardinality, then at least i and at last j number of times C may be included.

2.3.2 Attribute Included Feature Models

Another extension to basic feature models is the introduction of the attributes of features, which provides more information about features. An attribute of a feature, is any observable characteristic of the feature. Every attribute belongs to a domain which is a space of possible values where the attribute takes its values. The domain of an attribute may be discrete such as integers or continuous such as real [5].

The relationships between features and feature attributes are illustrated by Kang *et al.* in [13]. The author also has mentioned non-functional features related to feature attributes in [18] which is the introduction of Feature Oriented Reuse Method (FORM). FORM is an extension of FODA with a specific phase for software design and illustrates how the feature model is used to develop domain architectures and components for reuse. It is a systematic method finding the variability and commonalities in a domain in terms of features and making use of analysis results in order to enhance the system. The main purpose of this extension is managing and reusing the features of a specific domain which characterize each variant product in

that domain. Czarnecki *et al.* in [19] introduces the use of attributes in feature models. Later a notation for extended feature models was proposed by Benavides *et al.* in [5]. Table 2.4 illustrates the feature attribute notation in a feature diagram.

Table 2.4 Symbols for designing a feature model including Attributes

Symbol	Name
<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: auto;"> <p style="text-align: center;">F attr : Domain</p> </div>	Feature Attribute (F.attr)

2.4 Textual Notations for Feature Diagrams

Although graphical representations are supposed to be more accessible to non-technical stakeholders, working with large size of feature models has some disadvantages listed below:

- Without a tool support, creating a large industry-size feature diagram needs extraordinary human effort in order to create an understandable and accurate graphical syntax.
- Navigating and interpreting the feature model is hard as the distances between the features in large size models may be too much.
- There are not any appropriate graphical notations in order to represent concepts such as attributes and constraints.

An advantage of text-based languages is that there are many accepted applications supporting this kind of modeling such as text editors, source control systems and etc. In addition a text-based model is a convenient way in order to specify the input for driving the code generator and implement in modeling tool in a more convenient way.

Textual representations should not only contain all the information illustrated via diagrams, but also they should support automatic processing.

This part of the thesis work will provide an overview of related text-based works presenting feature models.

2.4.1 Feature Description Language (FDL)

Feature Description Language (FDL) [20] is the first text-based language supporting a feature diagram algebra provides a syntax for writing a program specification. An FDL definition defines the feature model with a feature name followed by ":" and consequently feature expression. The most important notations in FDL for expressing feature models are described as following:

- An atomic or a composite feature; the definition of the composite feature is given in another place of the definition.
- An optional feature; a feature expression followed by "?" notation.
- Mandatory and alternative features terminated with "*all ()*" and "*one-of ()*" notations in order.

However, FDL does not support attributes, cardinality-based decomposition, complex cross-tree constraints, DAGs or duplicate feature names.

2.4.2 Text-based Variability Language (TVL)

A notable feature modeling language is Text-based Variability Language (TVL) proposed in [21]. The main goal of proposing TVL was to provide a language with high readability for users with a comprehensive syntax to make modeling easy and convenient. It also avoids ambiguity with formal semantics. It has a C-like syntax described by an LALR grammar. It accounts for cardinality-based decompositions and feature attributes which is not proposed in most existing feature modeling languages. In addition, complex cross-tree constraints can be expressed in this language. As TVL is text-based, many applications such as text editors supports the

modeling as well. Besides that, it can be used in combination with graphical notations. The language consists of five major parts which are defined as follows:

- Features: the features and the relationships, including decomposition and cross-tree relationships, among them can be represented. The hierarchically presentation of the feature model is practical applying keywords such as *root*, *group*, *allof* and etc.
- Attributes: TVL supports four different attribute types including integer, real, Boolean and enumeration. Attributes are illustrated by type and name of them in TVL.
- Expressions: expressions are used in order to determine the value of an attribute. Basic expressions such as *integer* can be combined by applying operators such as +, &&, > etc. in addition, keywords illustrating cross-tree constraints (*requires* and *excludes*) can be used as Boolean expressions.
- Constraints: *ifIn* and *ifOut* are the keywords that used in order to perform constraints. *ifIn* means that it is only applicable if the containing feature is selected. On the other hand, *ifOut* is applicable only when the containing feature is not selected.
- Modularization mechanisms: this concept is one of the most important goals for designing the TVL to make the system organizable. As a result there are some methods and keywords offered by TVL in order modularize models. For instance the *include* keyword is used to take the path of the file as a parameter and processes directives.

DAG structures can also represented by TVL. The *shared* keyword is used in order to illustrate this concept and means that the shared feature has several parents. As TVL is a text-based modeling language, it has the advantages of being text-based. In addition, due to the C-like syntax of TVL, less learning effort is required for engineers. Apart from FDL, TVL is the only language for which a formal semantics exists.

2.4.3 XML-based Feature Modeling Languages

The availability of a wide range of free and commercial XML tools facilitates the processing of XML-based representations. Furthermore, all modern browsers have a built-in XML parser straightforwardly usable over the Internet. In addition, due to the application independent nature of XML and the facility to define the XML Schema, the interoperability between XML documents is convenient. Generally XML is a meta-language for creating markup languages. A Schema specifies which structures and attribute values are allowed to be explained. Creating a schema has the following advantages:

- The schema specifies the permissible documents for XML.
- Computer documents can specify whether the XML document is acceptable or not by applying the schema.
- Creating application programs is possible by benefiting from the schema.

As a result XML-Schemas are an important factor for the development of XML-based applications. XSD is one of the most important and acceptable schema languages in order to specify and manage the XML document precisely.

All the mentioned advantages cause a wide range of using XML-based representations during the passage of time. In this section the most important XML-based feature modeling languages are going to be explained.

2.4.3.1 XML-based Feature Modeling (XFeature)

XFeature is an XM-based feature modeling tool, provided as a plug-in for the Eclipse platform and graphical editor based on the Graphical Editing Framework (GEF) from Eclipse foundation [22]. It uses XML languages to express the feature models and XML schemas to represent the meta-model. The graphical elements used in the GUI editor for the feature model are similar to the ones in FODA feature diagrams with cardinality support.

XFeature provides two kinds of constraints: local constraints and global constraints. Local constraints are referred to the combination of sub-features which are children of the same feature. For instance consider the constraint in such a way that a parent feature must contain one of the four child features. Local constraints are similar to group relationships defined by FODA. Global constraints can apply to any feature in any vertical or parallel structure. This kind of constraint is explained in a separate XML Schema document as a constraint meta-model. There are four types of global constraints: Require, exclude, if-then and custom constraint. If-then constraint means that if constraint A is included then constraint B must be applied as well. Arbitrary constraint relationships are defined with custom constraints. However, attributes in cross-tree constraints and expressions for complex cross-tree constraints are not covered by XFeature. This tool does not support the automated analysis of feature models either.

2.4.3.2 Feature Modeling Markup Language (FeatureML)

FeatureML [23] is another XML based feature modeling language which consists of feature elements in order to construct the hierarchical feature diagram and relationships between features similar to those found in FODA and FORM. The language syntax is written as an XML Schema. Feature models are defined by XML document conforming to the XML Schema.

FeatureML defines static and dynamic entity structures and dependency relationships. It specifies features into two categories: behavior features and data features. Data features define the data content and property structures whereas behavior features specifies the dynamic aspects of the features. A behavior feature is influenced by feature variation points. The more the number of feature variation points, the more possible design choices for a behavior feature. In addition to the definitions related to the feature model structure and design choices, a feature model also includes additional elements such as code generation instructions and traceability information among the design choices and code generation. Code generation instructions have different types corresponding to a specific generation

pattern. FeatureML supports cardinality-based feature modeling, specialization of feature diagrams, and configuration based on feature diagrams. It has an XML schema definition that does not support attributed feature models and complex cross-tree constraints.

2.4.3.3 The Feature Modeling Plug-In for Eclipse (FeaturePlugin)

FeaturePlugin [24] is another XML-based approach. It is an extension of FODA providing feature and group cardinalities, feature attributes, feature diagram references and user-defined annotations. FeaturePlugin is generated from an Eclipse Modeling Framework (EMF) and some additional customization codes. The constraints such as implies and excludes are expressed using XPath [25] with a tree representation of the XML document to navigate through elements and attributes in an XML document.

2.4.3.4 Tooling a Framework for the Automated Analysis of Feature Models: (FAMA)

The FAMA framework [26] also uses an XML-based file format and is used for editing and automated analysis of feature models. FAMA describes two main functionalities: visual model edition/creation and automated model analysis. In general FAMA utilizes four different operations listed below:

- Validity control of the feature model
- Calculating the number of products resulted from the feature model
- List the possible products of a feature model
- Calculating the commonality of a feature

FAMA optimizes the analysis process by integrating different logic representations and solvers. The current version of FAMA integrates three logic representations of automated analysis on feature models named Constraint Satisfaction Problem (CSP)

[27], Boolean Satisfiability problem (SAT) [28] and Binary Decision Diagrams (BDD) [29]. More solvers may be added if required according to the requirements.

FAMA framework has a simple Java interface, implementing a query-base interaction; as a result it facilitates the integration. In addition extending or updating the existing product is feasible in FAMA architecture. The framework has an XML schema that supports decomposition relationships, but not complex cross-tree constraints.

2.4.3.5 A Tool Framework for Feature Oriented Software Development: FeatureIDE

FeatureIDE [30] is an open source framework of an Integrated Development Environment (IDE) for SPL engineering based on Feature Oriented Software Development (FOSD). Benefiting from FOSD designing and implementing applications based on features is achievable. FeatureIDE supports FOSD in many languages such as Java, C++, C#, XML, and etc. It supports the whole life cycle of a product line which starts with domain analysis, feature modeling, design, implementation and maintenance at last. However, cardinality-based feature diagrams and features including attributes in a feature diagram are not supported. In addition complex cross-tree constraints cannot be explained with the tool in order to construct complex feature-feature, feature-attribute and attribute-attribute relationships.

2.5 Automated Reasoning on Feature Models

Automated analysis for feature models was defined in FODA by Kang *et al.* [13] and still it is an ongoing research area. There are four main proposals for automated reasoning on feature models which will be explained in subsequent sub-section. However, among the proposals, only some of them are capable of handling extended feature models.

2.5.1 Automated Support on the Automated Analysis of Feature Models

Automated analysis of feature models can be divided into four main groups explained by Benavides *et al.* in [31], although it is an ongoing research area and the mapping among feature models and automated platforms are not fully specified.

This work proposed an XML-based feature modeling language in order to represent extended feature models with complex relationships. Although graphic-based feature modeling makes the editing and managing more convenient, from a code generation point of view textual feature model is the only way as input specification to drive the code generator. The language represented in this work do the automated processing in addition to the basic editing processes utilizing XML-based modeling.

2.5.1.1 Propositional Logic-based Analysis

There are some proposals in literature that represents the translation of basic feature models into propositional formulas. The first representation was proposed by Mannion [32] where the feature models were utilized as requirements in SPLs. In order to map the models into propositional formulas, specific rules were provided. Later Zhang *et al.* [33] proposed an extension to the previous work by making use of an automated tool support based on Software Variability Management (SVM) [34]. It also includes an explanation of a process function that can be followed on the automated analysis of feature models. Consequently, by Batory's [35] proposal, a connection between feature models, grammars and propositional formula was established. This approach supports the basic feature models and does not support cardinality-based and extended feature models.

2.5.1.2 Description Logic-based Analysis

The analysis proposed a translation of feature models into Ontology Web Language (OWL) Description Language (DL) ontology. The OWL is a family of knowledge representation languages in order to describe an ontology. Basic feature models are supported by this approach.

2.5.1.3 Constraint Programming-based Analysis

Constraint programming is a programming paradigm in which relations between variables are represented as constraints. Constraint Programming techniques can be utilized in order to automate the feature models. It is the only approach in which extended and cardinality-based feature models are supported beside the basic feature models proposed by Benavides *et al.*

2.5.1.4 Other Proposals

There are some other related works [20, 36] which are not strongly proposed so far. The existing proposals support only the basic feature models but not the extended feature models and the cardinality-based feature models.

CHAPTER 3

CONTEXT FREE GRAMMAR EXPRESSING EXTENDED FEATURE MODELS

This section will cover the grammar we proposed for the feature modeling language. The grammar will cover the explanation of basic feature models structure including features and decomposition relationships among them. The decomposition relationships are listed as mandatory, optional, alternative, and or which is defined in the previous section. In addition it will cover the grammar of extended feature models including feature attributes and cardinality-based feature models. An attribute of the feature is used in order to expose more information about the feature. Furthermore, the proposed grammar will cover the definitions explained in [37] in order to construct more complex feature models which may consist of feature-feature, feature-attribute, and attribute-attribute relationships.

In this thesis work we use Context Free Grammar (CFG) with Extended Backus-Naur Form (EBNF) notations in order to explain the language we defined. This section will be covered by an overview of the concept of the CFG grammar and its extended productions. The CFG grammar for our markup language will be defined in detail consequently.

3.1 Context Free Grammar (CFG)

A CFG [9] is a formal grammar in which every production rule is in the form of $V \rightarrow w$ where V is a single nonterminal symbol and w is a string of terminals and/or nonterminals. "Nonterminals are syntactic variables that denote set of strings". They are used to define the language generated by the grammar and structure of the

language hierarchically. Terminals are the basic symbols from which strings are formed. The word *token* is a synonym for terminal when we are talking about grammars for programming languages. A context free language is the language generated by context free grammars. The CFG is developed in the middle of 1950s by Naom Chamsky.

3.1.1 Backus-Naur Form (BNF)

BNF is an accepted notation for CFGs and is used for describing the syntax of languages [38] developed by John Backus (and possibly Peter Naur as well). A sample of BNF grammar is as following.

$ \begin{aligned} S &:= '-' FN FN \\ FN &:= DL DL '.' DL \\ DL &:= D D DL \\ D &:= '0' '1' '2' '3' '4' '5' '6' '7' '8' '9' \end{aligned} $

Figure 3.1 Sample BNF grammar

Where, S demonstrates the Start symbol, FN produces Fractional Number, DL is a Digit List and D is a Digit. The start symbol is FN which may be followed by more FNs. FN may be negative or not. It will continue with Digit List. In order to product a fractional number, the '.' is used between two DLs. According to the grammar given here, all the numbers, possibly fractional and negative ones are valid sentences such as 3.14 or -3.14.

3.1.2 Extended BNF (EBNF)

There are many extensions to BNF notation such as Extended BNF (EBNF) [39] introduced by Niklaus Wirth. EBNF is a notation for representing the grammar of a language. EBNF has some advantages over BNF described below:

- Options and repetitions could be directly expressed in EBNF.
- Terminals are expressed in quotation marks ("..." or '...') in EBNF facilitates the use of characters in the language.

- In EBNF a terminating character, the semicolon ";", applied in order to illustrate a rule is terminated. However, a rule can be expressed in one line using BNF syntax.

Table 3.1 demonstrates the standards defined for EBNF.

Table 3.1 EBNF symbols

Usage	Notation
definition	=
concatenation	,
termination	;
alternation	
option	[...]
repetition	{ ... }
grouping	(...)
terminal string	" ... "
terminal string	' ... '
comment	(* ... *)
special sequence	? ... ?
exception	-

Note that EBNF is more convenient than BNF defining the language. However, power of the two approaches is the same for defining languages and any EBNF production can be translated to a BNF production.

3.2 CFG for Extended Feature Models

This part will cover the CFG defined in order to explain the extended feature models. It will cover the definition of both basic and extended feature models which may include complex cross-tree relationships among them proposed in [37].

3.2.1 A CFG for Expressing Features

A feature is a distinguishable characteristic of a software item. A feature diagram contains hierarchically set of features and relationships defined among them. A feature is defined with its specific name and it may include an attribute or not. An attribute of a feature is any characteristic of a feature that can be measured [5]. A feature is defined by its name.

Definition 3.2.1:

A feature diagram consists of features with a specific relation among them. In order to illustrate the features, the name of the features will be utilized. The parent name of any feature may be demonstrated in order to validate the hierarchically design of the feature. If a feature is represented as a root feature, it will not get any value for its parent name. The right-hand side rewriting rule of the feature name will be the name of the features included in the feature model.

Numeric attribute designator and Boolean attribute designator are non terminals that are used in order to define the feature with its attribute [8]. They are defined with feature name followed with the attribute name as illustrated in rule 3.2.1.

Rule 3.2.1:

$$\textit{NumericAttributeDes} ::= \textit{FeatureName}.\textit{AttributeName} ;$$
$$\textit{BooleanAttributeDes} ::= \textit{FeatureName}.\textit{AttributeName} ;$$

The attributes of the Numeric Attribute Designator are used in order to express the infinite set of numeric digits belonging to the feature with the name *FeatureName*. The attributes belonging to the Boolean Attribute Designator demonstrate the attributes ranging over a Boolean domain with the name *FeatureName*.

Example 3.2.1:

For instance, consider the given feature model. The feature is illustrating the hard disk with two attributes. The Capacity attribute with the integer value [1.5..3] TB and the Type attribute with the string value which may be Maxtor or Hitachi.

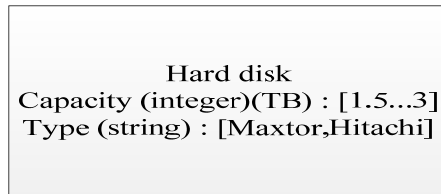


Figure 3.2 Sample feature with attribute

3.2.2 A CFG for Expressing Decomposition Relationships

Decomposition relationships among the features are declared using four types of relationships: *Mandatory*, *Optional*, *Alternative*, and *Or* between the parent feature and child feature(s).

Definition 3.2.2:

Mandatory relations and *Optional* relations are placed exactly between a parent and one child. On the other hand, *alternative* and *or* relations are placed among the parent feature and a set of child features. The child set may include one or more children. The type of the parent and child are both features declared by a specific name.

Rule 3.2.2:

The CFG rules for deriving the decomposition relationships are as follows:

MandatoryRel ::= Parent "Mandatory" Child ;

OptionalRel ::= Parent "Optional" Child ;

OrRel ::= Parent "Or" Children ;

AlternativeRel ::= Parent "Alternative" Children ;

Parent ::= FeatureName;

Child ::= FeatureName;

Children ::= "(" FeatureName Siblings ")"

Siblings ::= "," FeatureName | "," FeatureName Siblings ;

Example 3.2.2:

For instance consider the following Figures demonstrating the discussed relationships:

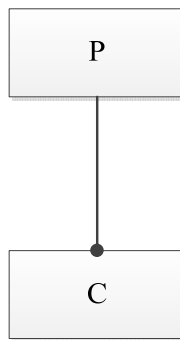


Figure 3.3 Mandatory relationship

Figure 3.3 illustrates the Mandatory relation among P and C. "P Mandatory C" is a valid Mandatory relationship as the relationship among the parent and child features is *mandatory*. Similarly, Figure 3.4 illustrates the optional relationship among the parent feature "P" and child feature "C".

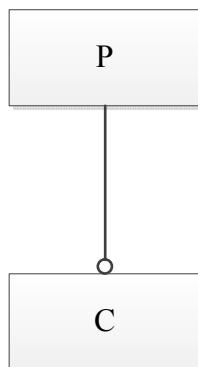


Figure 3.4 Optional relationship

As another example consider "P Alternative C1, C2, C3" in which exactly one child from the child set should be selected from the child set. The relationship is illustrated in Figure 3.5.

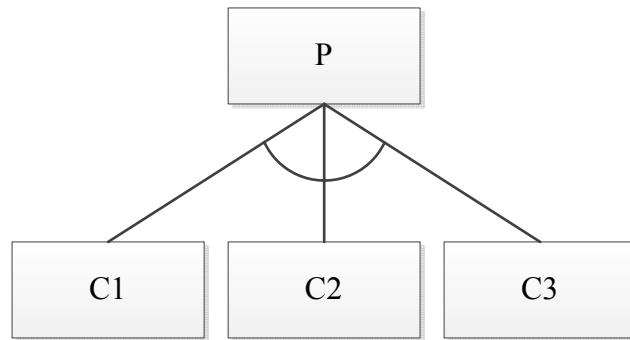


Figure 3.5 Alternative relationship

The "Or" relation exists among the parent feature and a set of child features. For instance consider the Figure 3.6 illustrating the relationship.

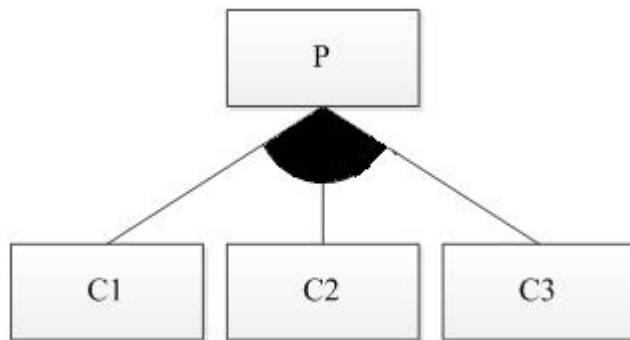


Figure 3.6 Or relationship

3.2.2.1 A CFG for Expressing Cardinality-Based Decomposition Relationships

There are two kinds of cardinality-based relationships illustrated in feature models, Group cardinality and Solitary cardinality. The Group cardinality relationship is expressed among the parent and set of children in which the inclusion of the parent feature causes the inclusion of child set according to lower and upper bound of the cardinality. The Solitary cardinality relationship is expressed as a relationship among the parent feature and child feature in which the inclusion of parent feature causes

the inclusion of child feature in many number of times which is specified by the lower and upper bound of the cardinality.

The CFG rules for generating cardinality-based decomposition relationships are as follows:

Rule 3.2.2.1:

SolitaryCardinality ::= Parent "<" LowerBound ".." UpperBound ">" Child ;

GroupCardinality ::= Parent "<" LowerBound ".." UpperBound ">" Children ;

LowerBound ::= "0" | "1" | "2" | ...

UpperBound ::= "0" | "1" | "2" | ...

Parent ::= FeatureName;

Child ::= FeatureName;

Children ::= "(" FeatureName Siblings ")"

Siblings ::= "," FeatureName | "," FeatureName Siblings ;

Example 3.2.2.1

Figure 3.7 illustrates *Group Cardinality* in which the lower bound is equal to one and the upper bound is equal to three means that the inclusion of the parent feature "P" implies at least one and at most three of the child features "C1, C2, C3".

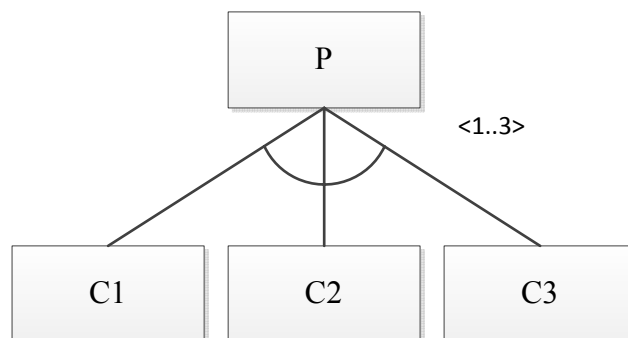


Figure 3.7 Group cardinality

The following Figure also is represented in order to demonstrate the *Solitary Cardinality* among the parent and child feature. The lower bound is equal to one and the upper bound is equal to three means that the inclusion of the parent feature "P" implies at least once and at last three times of occurrence of the child feature C.

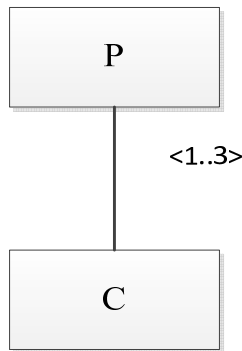


Figure 3.8 Solitary cardinality

3.2.3 A CFG for Expressing Complex Cross-Tree Relationships

This section will cover the basic cross-tree relationships among the features in a feature model. In addition complex cross-tree constraints for constructing complex feature-feature, feature-attribute and attribute-attribute relationships will be covered. An abstract syntax proposed in [37] describes all the possible complex cross-tree constraints among features. The CFG for expressing the definitions will be expressed in this work.

3.2.3.1 A CFG for Expressing Condition Relationship

Definition 3.2.3.1

A *condition* is a Boolean expression either in the form $Expression_1 \text{ relop } Expression_2$, or $Feature.attribute = truth-value$. A $\text{relop} \in \{=, \neq, <, \leq, >, \geq\}$ designates common relational operators where the domain of the operands are subsets of integers and possibly infinite.

Domain of *Feature.attribute* is {true, false}, and domains of *Expression1* and *Expression2* are compatible.

An expression is:

- An integer constant, or
- Value of an attribute which will be described as attribute designator in this work and is in the form of *Feature.attribute*, or
- Any well-formed formula constructed by combining integer constants and/or attributes' values with the common integer arithmetic operators {+, -, *, div, mod}.

Rule 3.2.3.1

The CFG rule in order to represent the condition relation is written below:

$$\begin{aligned} \textit{ConditionRel} & ::= "(" \textit{Expression} \textit{RelOp} \textit{Expression} ")" \mid \\ & \quad \textit{BooleanAttributeDesignator} "=" \textit{TruthValue} ; \\ \textit{Expression} & ::= \textit{IntegerConstant} \mid \\ & \quad \textit{NumericAttributeDesignator} \mid \\ & \quad "(" \textit{Expression} \textit{IntOp} \textit{Expression} ")" ; \\ \textit{NumericAttributeDesignator} & ::= \textit{FeatureName} "." \textit{AttributeName} ; \\ \textit{RelOp} & ::= "=" \mid "\neq" \mid "<" \mid "\leq" \mid ">" \mid "\geq" ; \\ \textit{IntOp} & ::= "+" \mid "-" \mid "*" \mid "div" \mid "mod" ; \end{aligned}$$

Example 3.2.3.1

As an example of valid condition, consider the following condition relationships belongs to the feature model illustrated in Figure 3.9. P is the parent feature. The

Mandatory relation connects the "P" to "X" and "Z" and the Optional relationship is among the "P" as a parent feature and "Y" as its child.

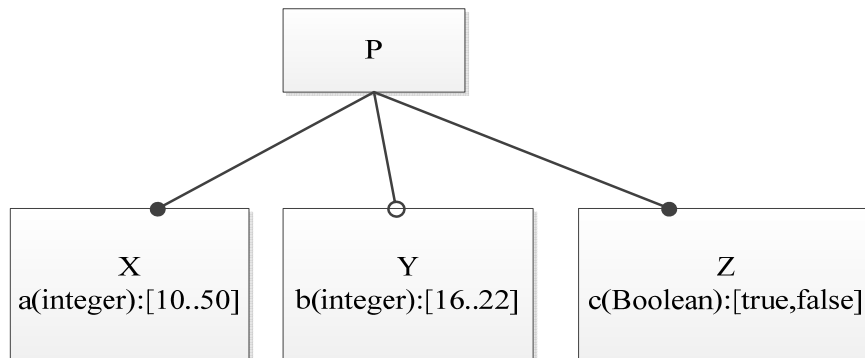


Figure 3.9 A sample feature model

$$(1) X.a \geq Y.b \text{ div } 2$$

It is in the form of $Expression_1 \text{ relop } Expression_2$. In this example, X.a and Y.b are called attribute numeric designators where X and Y illustrates the features belongs to the feature model demonstrated with their names. a is the attribute of feature X and b is the attribute of feature Y. The relop operator between $Expression_1$ and $Expression_2$ is defined as \geq . (Y.b div 2) illustrates the formula combining attribute value with an integer constant and the *div* operator is defined as an arithmetic operator.

As another valid example consider the following:

$$(2) Z.c = \text{true}$$

The expression is in the form of $Feature.attribute = truth-value$. The left hand side of the equation is an attribute designator which is illustrated as a feature name followed by "." and attribute name. The feature name is Z with an attribute named c.

3.2.3.2 A CFG for Expressing Excludes Relationship

Definition 3.2.3.2

An *excludes* is a relationship in the form of $P \text{ excludes } Q$ among two features. P and Q are both features.

Rule 3.2.3.2

The CFG rule for deriving an *excludes* relation is illustrated below:

$$\text{ExcludesRel} ::= \text{FeatureName "excludes" FeatureName};$$

Example 3.2.3.2

For instance, consider a feature model as shown in Figure 3.10. The relationship among the feature X and feature Y is defined as an *excludes* relationship, which can be explained as following. It means that the inclusion of X in the feature model implies the exclusion of Y in that model. It is a valid sample demonstrating *exclude* relationship.

(1) X excludes Y

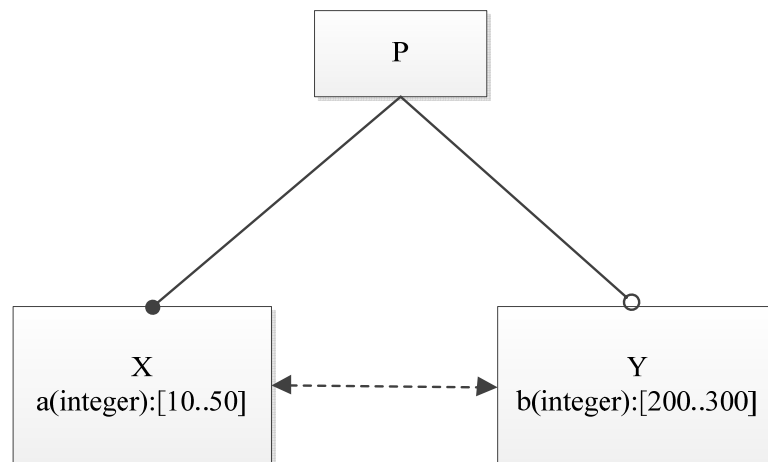


Figure 3.10 A sample feature model expressing Excludes relationship

3.2.3.3 A CFG for Expressing Requires Relationship

Definition 3.2.3.3

A *requires* is a relationship in the form of *P requires Q*, where P is a feature and Q may be:

- A feature, or
- A *condition*, which is defined in definition 1, or
- Any well formed formula constructed by combining features and/or conditions with the propositional logic connectives {and, or, not, conditional, biconditional}

Rule 3.2.3.3

The CFG derived from the *requires* relationship is written as following:

$$\textit{RequiresRel} ::= \textit{FeatureName} \textit{"requires"} \textit{Q};$$
$$\textit{Q} ::= \textit{FeatureName} |$$
$$\textit{ConditionRel} |$$
$$\textit{"(} \textit{Q} \textit{ BinaryConnective} \textit{Q} \textit{)"} |$$
$$\textit{Negation} \textit{"(} \textit{Q} \textit{)"};$$
$$\textit{BinaryConnective} ::= \textit{"and"} | \textit{"or"} | \textit{"conditional"} | \textit{"biconditional"};$$
$$\textit{Negation} ::= \textit{"not"};$$

Example 3.2.3.3

For instance consider the following feature model with the *requires* relationship in Figure 3.11.

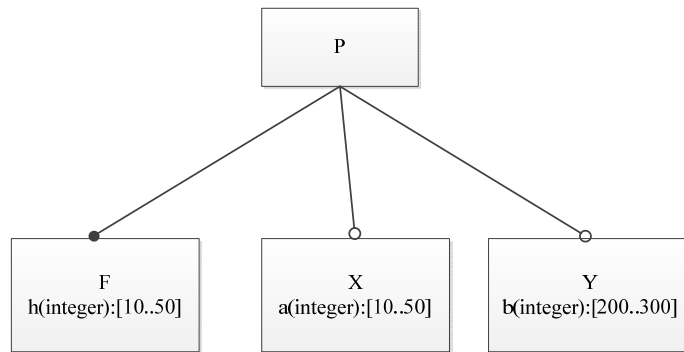


Figure 3.11 A sample feature model

The *requires* relationship below is a valid one expressed in expression (1):

(1) F requires (X.a > 10)

In this example, F is a feature existing in the feature model. (X.a > 10) is in the form of *condition* relation (*Expression₁ relop Expression₂*). The *relop* is > between two expressions. The left hand side *expression* is an attribute designator in the form of feature name followed by "." and attribute name consequently. The right hand side *expression* is an integer constant.

As another example consider the following example expressed in expression (2) as a valid one:

(2) F requires (X and not Y)

In this example, the left hand side of the *requires* relationship is a feature named F existing in the feature model. The right hand side of the relationship is in the form of "(*Q BinaryConnective Q*)", where the binary connective is expressed as "and". X and Y are the names of a features existing in the feature model. The negation notation is represented using "not" keyword.

3.2.3.4 A CFG for Expressing Complex Constraint Relationship

Definition 3.2.3.4

A *complex constraint* is a *requires/excludes* relationship, or any well-formed formula constructed by combining features and/or *requires/excludes* relationships with the propositional logic connectives.

Rule 3.2.3.4

Rule 3.2.3.4 illustrates the CFG rule deriving the *Complex Constraint* relationship which is written below:

$$\begin{aligned} \text{ComplexConstraintRel} &::= \text{RequiresRel} \mid \\ &\quad \text{ExcludesRel} \mid \\ &\quad "(" \text{ CP BinaryConnective CP } ")" \mid \\ &\quad \text{Negation "(" CP ")"} ; \\ \\ \text{CP} &::= \text{RequiresRel} \mid \\ &\quad \text{ExcludesRel} \mid \\ &\quad \text{FeatureName} \mid \\ &\quad \text{Negation "(" CP ")"} \mid \\ &\quad "(" \text{ CP BinaryConnective CP } ")" ; \\ \\ \text{BinaryConnective} &::= \text{"and"} \mid \text{"or"} \mid \text{"conditional"} \mid \text{"biconditional"} ; \\ \\ \text{Negation} &::= \text{"not"} ; \end{aligned}$$

Example 3.2.3.4

In order to make a valid example, consider the feature model illustrated in Figure 3.12. The root feature "P" contains three children named A, B and C. C is the parent of X.

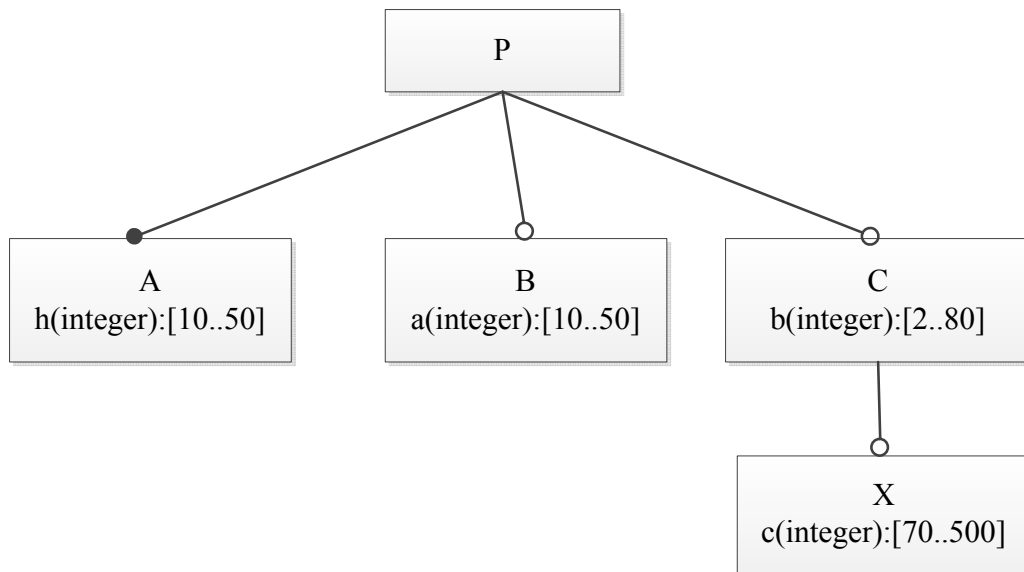


Figure 3.12 A sample feature model

The complex constraint relationships written below are some valid samples:

(1) A and B implies not C

Expression (1) is a well-formed combination of features with propositional logic connectives in the form of *"(" ComplexConstraintRel BinaryConnective ComplexConstraintRel ")"*. A, B, and C demonstrates feature names which is defined as a choice for a *complex constraint* relationship. *and* and *implies* are binary connectives and *not* is described as negation formula.

As another example consider the one written as follows:

(2) A requires (B.a > C.b or X.c > 100)

In expression (2) the *complex constraint* relationship is in the form of *RequiresRel*. Left hand side of the *RequiresRel* demonstrate a feature named A. The right hand side of the relationship is in the form of *"(" Q BinaryConnective Q ")"* which one of the right hand side choices defined in *RequiresRel*. The *BinaryConnective* is defined with "or" and both sides of the *BinaryConnective* are defined as a *ConditionRel*. According to the rule defined for *ConditionRel*, they are in the form of *"(" Expression RelOp Expression ")"*.

3.2.3.5 A CFG for Expressing Guarded Constraint Relationship

Definition 3.2.3.5

In some cases the use of conditional constraints may be inevitable. It is in the form of "if (some feature-related condition holds) then (some feature-related constraint must hold)".

The conditional constraint is provided as a separate type of constraint for convenience and it is often found in practice in different domains and purposes. The constraint is defined as *Guarded Constraint* which is defined as follows:

A *Guarded Constraint* is a relationship in the form: if *Guard* then *Complex Constraint*. Guard is any Boolean combination of conditions.

Rule 3.2.3.5

$$\textit{GuardedConstraintRel} ::= \textit{"If" Guard "Then" ComplexConstraintRel} ;$$
$$\textit{Guard} ::= \textit{"(" Guard BinaryConnective Guard ")"} |$$
$$\textit{Negation " (" Guard ")"} |$$
$$\textit{ConditionRel} ;$$
$$\textit{BinaryConnective} ::= \textit{"and"} | \textit{"or"} | \textit{"conditional"} | \textit{"biconditional"} ;$$
$$\textit{Negation} ::= \textit{"not"} ;$$

Example 3.2.3.5

As an example, consider the feature model demonstrated in Figure 3.13. The root feature P has two sub features named A and B with their own attributes.

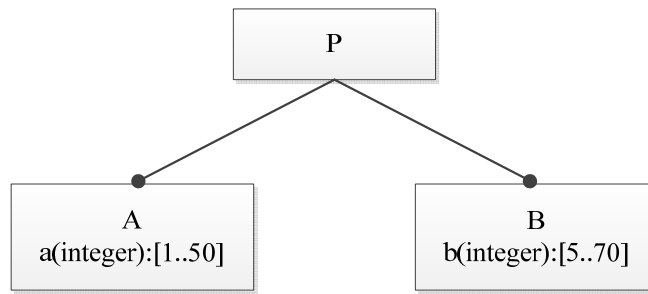


Figure 3.13 A sample feature model

There may be a conditional constraint between features such as the following one:

- (1) If the value of attribute of feature A is greater than 4, then feature A requires $B.b \geq 10$. It will be illustrated in the form: if $A.a > 4$ then A requires $B.b \geq 10$ which is a valid conditional constraint according to the rule. In this sample, *Guard* is in the form of *ConditionRel*. It is defined in the form of "(*Expression RelOp Expression*)". The *ComplexConstraintRel* part is in the form of *RequiresRel* which is defined in Rule 3.2.3.4.

3.2.3.5.1 A CFG for Expressing Guarded Constraint Combination Relationship

Definition 3.2.3.5.1

The *guarded constraints* can be combined with the propositional logic connectives in order to establish more complex and complicated constraints.

Rule 3.2.3.5.1

GuardedConstraintRelCombination ::=

"(*GuardedConstraintRel BinaryConnective GuardedConstraintRel*)" |

GuardedConstraintRel |

Negation "(*GuardedConstraintRel*)";

BinaryConnective ::= "and" | "or" | "conditional" | "biconditional";

Negation ::= "not";

Example 3.2.3.5.1

As an example consider the feature model given in Figure 3.14. X, Y and Z are some of the existing features in the feature model.

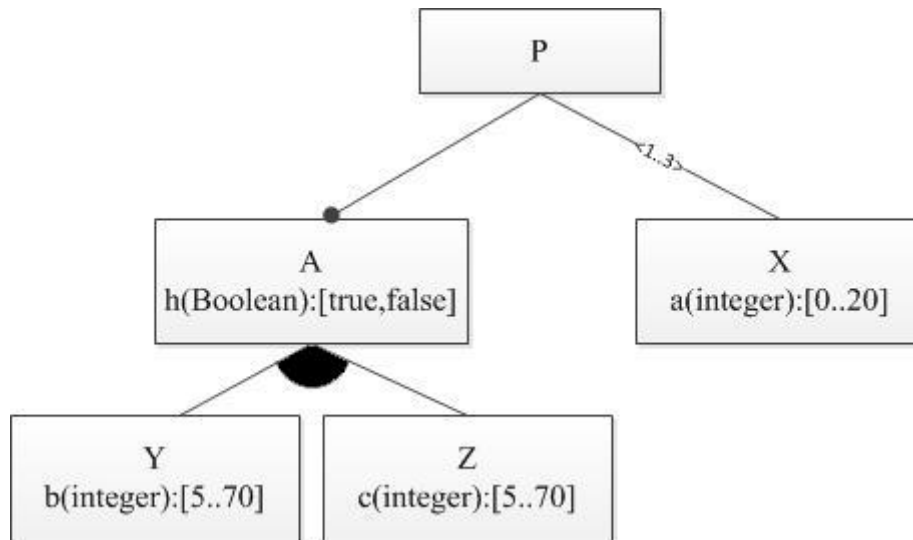


Figure 3.14 A sample feature model

Now, consider an example of combined guarded constraint which is a valid one and written below:

- (1) If $X.a < 10$ and $X.a > 2$ then X excludes Y or if $X.a = Y.b$ then X requires Z.

This sample is in the form of *"(" GuardedConstraintRel BinaryConnective GuardedConstraintRel ")"*. The *BinaryConnective* is illustrated with "or". The left hand side of the "or" connective is a *GuardedConstraintRel* in which the *Guard* is *ConditionRel* in the form of *"(" Expression RelOp Expression ")"* and the *ComplexConstraintRel* is in the form of *ExcludesRel*. The right hand side of the "or" connective is a *GuardedConstraintRel* as well. The *Guard* is *ConditionRel* in the form of *"(" Expression RelOp Expression ")"* and the *ComplexConstraintRel* is in the form of *RequiresRel*.

CHAPTER 4

XML SCHEMA DEFINITION FOR EXPRESSING EXTENDED FEATURE MODELS

In this section, we present the XML Schema Definition (XSD) that has been built on the definitions discussed in the previous chapter in order to illustrate feature models. Feature models are used to illustrate SPL in terms of features and relationships among them. Via extending feature models, demonstrating extended feature models with attributes and cardinality-based feature modeling become achievable as well. The main contribution of this work is the proposal of an XML-based feature modeling language. In addition to the capability of expressing the structure of basic feature models with the relationships among them, the language also provides the modeling language in order to express features with complex cross-tree constraints involving attributes. Benefiting from an XML-based language as a meta-model, the use of rich selection of off-the-shelf XML parsers which offers advantages that are not readily available with traditional text encoding, will be convenient.

The language we present in this work uses XML languages to express the feature models and XML Schemas to express the meta-model. According to the relationship among a model and a meta-model is then expressed by saying that the XML-based language must be validated by using XML Schema which represents its meta-model.

4.1 XML Schema Definition (XSD)

An XML Schema is published as a W3C recommendation in May 2001. In general, schema is an abstract collection of meta-data including schema components such as complex types, simple types, declarations for elements and attributes and etc. [10]

Similar to most of the schema languages, XSD can be used in order to represent a set of rules to which an XML document must conform considering the validity of the XML document according to that schema.

An XSD describes the structure of an XML document and in spite of wide range of use of Document Type Definition (DTDs) it is a replacement for DTDs. DTD is a set of markup declarations defining a document type. It was a precursor to XML Schema with similar functionalities but different capabilities. XML Schema approach has the following advantages:

- Schemas support rich set of data types. Thus, describing document content and validating the correctness of data become more convenient.
- Schemas have the same syntax as XML. As a result, there is no need for an extra learning effort, and special tools in order to edit and parse.
- Schemas are extensible as they are written in XML. Due to this property, one Schema can be reused in another Schema and multiple Schemas may be referred in the same document.
- Creating new data types is possible.
- They provide enhanced control over markup and secure data communication. Thus make it readable and understandable for both sender and receiver of the document.

4.1.1 XSD <schema> Element

Every XML Schema starts with a root element <schema> [10] and may contain a reference element to an XML document. A simple structure of a schema is illustrated in Figure 4.1.

```
<?xml version="1.0"?>
<xs:schema>
...
...
</xs:schema>
```

Figure 4.1 A simple schema structure

4.1.2 XSD Simple Elements

A simple element is used to define an XML element containing only texts. The text can be defined as one of the types included in XSD such as integer, string and etc. in addition it can be custom type defined by the user. Simple elements may also have default or fixed values. At the time that no value is specified to an element, a default value is automatically assigned to the element. In the case of applying the fixed values, specifying another value automatically is impossible.

4.1.2.1 XSD Restrictions

In order to define acceptable values for an XML element restrictions are used called facets. As an example, Figure 4.2 illustrates a restriction applied on a set of values. It defines an element called "car" with a restriction. The only acceptable values for "car" are "Golf", "Audi" and "BMW".

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Figure 4.2 A sample of restrictions from [10]

4.1.3 XSD Complex Elements

A complex element may contain other elements and/or attributes [10]. Four main types of complex elements are as written in the following:

- (1) empty elements
- (2) elements including only other elements
- (3) elements includes only text
- (4) elements that contain both other elements and text

These types of complex elements may include attributes as well.

4.1.3.1 XSD Indicators

Indicators are used in order to control the behavior of the elements in a document.

There are three main types representing indicators which are listed below:

- (1) Order indicators: in order to define the order of the elements these kinds of indicators are used. `<all>`, `<choice>` and `<sequence>` indicators are listed as order indicators. Applying `<all>` indicator means that each child element must occur exactly once. They can apply in any order. Applying `<choice>` indicator means that you can choose the elements to be occurred. `<sequence>` indicator is used where the order of the elements is significant.
- (2) Occurrence indicators: consist of two main types: `minOccurs` and `maxOccurs`. In order to specify the minimum number of times that an element should be occurred, `minOccurs` is used. On the other hand while specifying the maximum number of times that an element can be occurred, `maxOccurs` indicator is used.
- (3) Group indicators: they are used to define the elements related to each other.

4.2 XSD for Extended Feature Models

This part will cover the XSD provided in order to explain the extended feature models by both graphical and textual XML Schema. Graphical and textual modeling

languages serve different audience. Working with graphical modeling elements may be more convenient for human designers because of the ease of editing and managing graphs compared to the text including a complex syntax. On the other hand, from a code generation point of view, a textual feature model is the only way as input specification to drive the code generator. Editing and parsing a feature model via a graphical model is difficult utilizing computer programs.

The XSD represented in this section will cover the textual definition of both basic and extended feature models which may include complex cross-tree relationships among them. The graphical XML schema definitions are shown in Appendix A. The XSD explanations are provided according to the CFG rules illustrated in previous chapter.

4.2.1 XSD for Expressing Features

A feature model consists of hierarchically set of features starts with a root feature and continues with sub feature(s) with feature relationships among them. Features are identified by their specific names. They also may include attributes or not which is an extension for basic feature models [5]. Generally an attribute of a feature is illustrated by its name, domain and value. If a feature including its attribute is mentioned in a relationship among features, it would be shown as an *attribute designator*. An *attribute designator* is represented as a feature name followed by its attribute name with "." among them. Another main extension to the concept is including the feature cardinalities to basic ones [15]. There are two kinds of cardinalities: feature cardinality and group cardinality. The solitary cardinality between a parent feature and its child feature with <i..j> cardinality means that when the parent feature is included, the child feature will be included at least i and at most j number of times. Group cardinality between a parent feature and a set of child features with a <i..j> cardinality, the inclusion of a parent feature causes the inclusion of at least i and at most j child features. The i is less than or equal to j, and j is the maximum number of child features.

The schema definition in order to express the features is illustrated as follows. Appendix A also demonstrates the graphical representation for schema.

```
<xsd:complexType name="Feature">
  <xsd:sequence>
    <xsd:element ref="FeatureAttribute" minOccurs="0"    maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="Name" type="xsd:string" use="required"/>
  <xsd:attribute name="ParentName" type="xsd:string" />
</xsd:complexType>
```

Figure 4.3 XSD representation for feature

```
<xsd:element name="FeatureAttribute">
  <xsd:complexType>
    <xsd:attribute name="Name" type="xsd:string" use="required"/>
    <xsd:attribute name="Domain" type="xsd:string" use="required"/>
    <xsd:attribute name="Value" type="xsd:string"/>
  </xsd:complexType>
</xsd:element>
```

Figure 4.4 XSD representation for feature attributes

```

<xsd:element name="AllFeatures">
  <xsd:complexType>
    <xsd:sequence minOccurs="0" maxOccurs="unbounded" >
      <xsd:element name="Features" type="Feature" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.5 XSD representation for all of the features

In Figure 4.3, an XSD is represented demonstrating the structure of a basic feature which is identified by its name. It is defined as a complex type due to the use of feature type in other definitions with different name but similar structure. The `<sequence>` indicator is used in order to express the definitions in a specific order. By making use of the Occurrence indicators, the number of including an each element in the definition can be managed. The default number of occurrences is identified as one by default which may be changed by the programmer. The ParentName of any feature may be expressed while defining the feature. The structure of the "FeatureAttribute" and "AllFeatures" is expressed in a separate part in Figure 4.4 and Figure 4.5. As a result, referencing the elements with number of occurrences will be appropriate in other parts. "FeatureAttribute" and "AllFeatures" are both defined as an element in order to make the referring process convenient.

4.2.2 XSD for Expressing Decomposition Relationships

This section is covered by the XSD definitions for decomposition relationships among features. There are four kinds of decomposition relationships among features: Mandatory, Optional, Alternative and Or. The definition of these four relationships is grouped into two different expressions. Mandatory and Optional Relations are similar as the number of child features is only one. In contrast, the number of children features applying Alternative and Or relations is more than one. In addition,

the decomposition relationship may include feature cardinalities. There are two kinds of cardinalities; solitary cardinality and grouped cardinality. The "FeatureCardinality" element is defined as complex type including a <choice> indicator in order to choose the elements to be occurred in the definition. The element consists of two choices, "SolitaryCardinality" and "GroupedCardinality". The definitions are illustrated in the consequent Figures.

The XSD textual definitions of decomposition relationships are illustrated in the following Figures.

```
<xsd:element name="DecompositionRelation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="MandatoryRelation" minOccurs="0"/>
      <xsd:element ref="OptionalRelation" minOccurs="0"/>
      <xsd:element ref="AlternativeRelation" minOccurs="0"/>
      <xsd:element ref="OrRelation" minOccurs="0"/>
      <xsd:element ref="FeatureCardinality" minOccurs="0" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.6 XSD representing Decomposition relationship

The DecompositionRelation may include different types which are defined with the complex type including <sequence> indicator. Each of the decomposition relation types are referred to their own definitions illustrated as follows.


```

<xsd:element name="MandatoryRelation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element name="MandatoryRelationKeyword" type="xsd:string"
        fixed="Mandatory"/>
      <xsd:element name="Child" type="Feature" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.7 XSD expressing Mandatory relation

```

<xsd:element name="OptionalRelation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element name="OptionalRelationKeyword" type="xsd:string"
        fixed="Optional"/>
      <xsd:element name="Child" type="Feature" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.8 XSD expressing Optional relation

```

<xsd:element name="AlternativeRelation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element name="AlternativeRelationKeyword" type="xsd:string"
        fixed="Alternative"/>
      <xsd:element ref="ChildSet" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.9 XSD expressing Alternative relation

```

<xsd:element name="OrRelation">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element name="OrRelationKeyword" type="xsd:string" fixed="Or"/>
      <xsd:element ref="ChildSet" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.10 XSD expressing Or relation

The Mandatory, Optional, Alternative and Or relationships are defined as a `complexType` including `<sequence>` indicator in which the Parent element should be defined at first, continued with the related keyword and Child/ChildSet definition at the end. The XSD definition for `FeatureCardinality` is illustrated as the following. It consists of two different types, `SolitaryCardinality` and `GroupedCardinality`. They are defined as a complex type including a `<sequence>` indicator which means that the set of definitions should be applied in a specific order. They are demonstrated in Figure 4.12 and 4.13.

```
<xsd:element name="FeatureCardinality">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="SolitaryCardinality" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="GroupedCardinality" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.11 XSD expressing Feature Cardinality

```

<xsd:element name="SolitaryCardinality">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element ref="Cardinality" />
      <xsd:element name="Child" type="Feature"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.12 XSD expressing Solitary Cardinality

```

<xsd:element name="GroupedCardinality">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Parent" type="Feature" />
      <xsd:element ref="Cardinality" />
      <xsd:element ref="ChildSet" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.13 XSD expressing Group Cardinality

```

<xsd:element name="Cardinality">
    <xsd:complexType>
        <xsd:attribute name="Min" type="xsd:integer" use="required"/>
        <xsd:attribute name="Max" type="xsd:integer" use="required"/>
    </xsd:complexType>
</xsd:element>

```

Figure 4.14 XSD expressing Cardinality

Figure 4.14, illustrates the definition of the Cardinality which includes two attributes. "Min" illustrates the minimum number of occurrences of the cardinality and "Max" is utilized in order to represent the maximum number of cardinality. The use of the attributes are expressed as "required" emphasizing that the inclusion of the attributes is obligatory.

The definition of the ChildSet is illustrated in the following Figure. The minimum number of children is two and it can be more than two.

```

<xsd:element name="ChildSet">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="Child" type="Feature"
                minOccurs="2" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>

```

Figure 4.15 XSD expressing ChildSet

4.2.3 XSD for Expressing Complex Cross-Tree Relationships

This section will cover the XSD definitions expressing the basic and complex cross-tree relationships including feature-feature, feature-attribute, and attribute-attribute relationships. The definitions and CFG rules of the relationships are given in section 3.2.3. In Figure 4.16 the different types of the complex cross-tree relationship is illustrated.

```
<xsd:element name="CrossTreeRelation">
<xsd:complexType>
<xsd:sequence>
  <xsd:element ref="Excludes" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="Requires" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="ComplexConstraint" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="GuardedConstraint" minOccurs="0" maxOccurs="unbounded"/>
  <xsd:element ref="GuardedConstraintCombination"
    minOccurs="0" maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Figure 4.16 XSD expressing cross-tree relationships

4.2.3.1 XSD for Condition Relationship

The condition may be in the form of $Expression_1 \text{ relop } Expression_2$, or $Feature.attribute = truth-value$. The XSD is defined as an element including a `<choice>` indicator in order to be chosen which is illustrated in Figure 4.17.

```

<xsd:element name="Condition">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="ExpressionRelOpExpression"/>
      <xsd:element ref="BoolAttributeDesequaltoTruthValue"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

Figure 4.17 XSD expressing Condition relationship

The structure of two alternatives expressing condition expression is illustrated in Figure 4.18 and 4.20. The first element is "ExpressionRelOpExpression" defined as a complex type including a `<sequence>` indicator in order to illustrate a set of expressions in a specific order. The sequence includes the left hand side Expression followed by RelOp and right hand side Expression. The RelOp is defined as a simple type in order to define the set of acceptable values ($relop \in \{=, \neq, <, \leq, >, \geq\}$). The structure of the Expression is expressed in Figure 4.19 as well. The Expression may be expressed as an integer constant, attribute designator, or any well-formed formula by combining integer constants and/or attributes' values with the integer operators. IntegerOperator is defined as a simple type with restrictions on a set of acceptable values.

```

<xsd:element name="ExpressionRelOpExpression">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="ExpressionStructure"/>
      <xsd:element name="RelOp">
        <xsd:simpleType>
          <xsd:restriction base="xsd:string">
            <xsd:enumeration value="="/>
            <xsd:enumeration value="#"/>
            <xsd:enumeration value="&lt;"/>
            <xsd:enumeration value="&lt;="/>
            <xsd:enumeration value="&gt;"/>
            <xsd:enumeration value="&gt;="/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:element>
      <xsd:element ref="ExpressionStructure"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.18 XSD expressing ExpressionRelOpExpression


```

<xsd:element name="ExpressionStructure">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="IntegerConstant" type="xsd:integer"/>
      <xsd:element ref="NumericAttributeDes"/>
      <xsd:sequence>
        <xsd:element ref="ExpressionStructure" />
        <xsd:element name="IntegerOperator">
          <xsd:simpleType>
            <xsd:restriction base="xsd:string">
              <xsd:enumeration value="+"/>
              <xsd:enumeration value="-"/>
              <xsd:enumeration value="*"/>
              <xsd:enumeration value="div"/>
              <xsd:enumeration value="mod"/>
            </xsd:restriction>
          </xsd:simpleType>
        </xsd:element>
        <xsd:element ref="ExpressionStructure" />
      </xsd:sequence>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

Figure 4.19 XSD expressing the structure of Expression

The condition expression may also be in the form of "BoolAttributeDesequaltoTruthValue" illustrated in Figure 4.20. It is defined as a

complex type including <sequence> indicator, as the order of expressions is important. The "DotKeyword" and "EqualKeyword" are defined as a fixed value and will be always illustrated by their defined values. "TruthValue" is expressed with a Boolean type. The structure of the "NumericAttributeDes" is given in Figure 4.21.

```
<xsd:element name="BoolAttributeDesequaltoTruthValue">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="FeatureName" type="Feature"/>
      <xsd:element name="DotKeyword" type="xsd:string" fixed="."/>
      <xsd:element name="AttributeName" type="xsd:string"/>
      <xsd:element name="EqualKeyword" type="xsd:string" fixed="="/>
      <xsd:element name="TruthValue" type="xsd:boolean"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.20 XSD expressing BoolAttributeDesequaltoTruthValue

"NumericAttributeDes" is expressed with a sequence starting with a feature name continuing with a fixed value of "." keyword and attribute name at the end of sequence. There are two kinds of Attribute Designators; Numeric and Boolean. The value of the NumericAttributeDes will always numeric whereas the Boolean one can be illustrated with two values true or false.

```

<xsd:element name=" NumericAttributeDes ">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="FeatureName" type="Feature"/>
      <xsd:element name="DotKeyword" type="xsd:string" fixed="."/>
      <xsd:element name="AttributeName" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.21 XSD expressing NumericAttributeDes

4.2.3.2 XSD for Expressing Excludes Relationship

The Excludes relationship is expressed in XSD as illustrated in Figure 4.22. It is defined in a sequence which may be occurred zero or more times. The left hand side and right hand side of the "excludes" keyword are features following the structure defined for a basic feature in Figure 4.3.

```

<xsd:element name="Excludes">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="LeftFeatureOfExclude" type="Feature"/>
      <xsd:element name="ExcludeKeyword" type="xsd:string" fixed="excludes"/>
      <xsd:element name="RightFeatureOfExclude" type="Feature"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.22 XSD expressing Excludes relationship

4.2.3.3 XSD for Expressing Requires Relationship

The XSD definition for "Requires" relationship structure is expressed in Figure 4.23. The "Requires" relation is defined as a complex type in which the <sequence> indicator is used as the order of the expressions is important. The relationship may or may not occur. The left hand side structure of the "requires" keyword is a feature all the time following the properties defined in the XSD for features in Figure 4.3. The right hand side is identified as a "BooleanFormula". The structure of the "BooleanFormula" is illustrated in Figure 4.24.

```

<xsd:element name="Requires">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="ReqFeature" type="Feature"/>
      <xsd:element name="RequireKeyword" type="xsd:string" fixed="requires"/>
      <xsd:element ref="BooleanFormula"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.23 XSD expressing Requires relationship

```

<xsd:element name="BooleanFormula">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element name="FormulaFeature" type="Feature"/>
      <xsd:element ref="Condition"/>
      <xsd:element ref="NegationBooleanFormula"/>
      <xsd:element ref="BooleanFormulaConBooleanFormula"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

Figure 4.24 XSD expressing the structure of Boolean Formula

A "BooleanFormula" can be expressed with a "Feature", "Condition" relationship, "NegationBooleanFormula" or a combination of features and/or conditions with

propositional logic connectives expressed in Figure 4.26. The "NegationBooleanFormula" is defined in order to express the negation structure illustrated with "not" keyword.

```
<xsd:element name="BooleanFormulaConBooleanFormula">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="BooleanFormula"/>
      <xsd:element ref="BinConnective"/>
      <xsd:element ref="BooleanFormula"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.25 XSD expressing BooleanFormulaConBooleanFormula

```
<xsd:element name="BinConnective">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="and"/>
      <xsd:enumeration value="or"/>
      <xsd:enumeration value="conditional"/>
      <xsd:enumeration value="biconditional"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Figure 4.26 XSD expressing Propositional Logic Connectives

4.2.3.4 XSD for Expressing Complex Constraint Relationship

The XSD definition for expressing the Complex Constraint relationship is illustrated in Figure 4.27.

```
<xsd:element name="ComplexConstraint">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="Requires" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Excludes" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="NegationCP" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="CPConCP" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Figure 4.27 XSD expressing ComplexConstraint relationship

The "ComplexConstraint" is defined as a complex type including <choice> indicator as it can be expressed as a "Requires" relationship, "Excludes" relationships, or Complex Part (CP) which may be illustrated as a negation CP, with "not" keyword, or combination of CPs with propositional logic connectives. The structure of CP is illustrated below in Figure 4.28.

```

<xsd:element name="CP">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="Requires" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="Excludes" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="NegationCP" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="CPConCP" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="CPFeature" type="Feature"
        minOccurs="0" maxOccurs="unbounded"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

Figure 4.28 XSD expressing Complex Part (CP)

The CP definition restricts the occurrence of a feature solely. It can be expressed only as a combination with another feature using propositional logic connectives. "NegationCP" illustrated with a fixed "not" value followed by the structure of CP.

4.2.3.5 XSD for Expressing Guarded Constraint Relationship

The XSD definition for expressing the Guarded Constraint relationships is illustrated in Figure 4.29. It is defined as a complex type applying the <sequence> indicator as the sequence of inclusion is important in the structure. Firstly, the "IfKeyword" is defined which has a fixed value, "If" followed by "Guard". The XSD expression of "Guard" is illustrated in Figure 4.30. The definition continues with "ThenKeyword" with a fixed value of "Then". The termination definition expressing is the "ComplexConstraint" which is defined in section 4.2.3.4.


```

<xsd:element name="GuardedConstraint">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="IfKeyword" type="xsd:string" fixed="If"/>
      <xsd:element ref="Guard"/>
      <xsd:element name="ThenKeyword" type="xsd:string" fixed="Then"/>
      <xsd:element ref="ComplexConstraint"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

```

Figure 4.29 XSD expressing GuardedConstraint

```

<xsd:element name="Guard">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="GuardConGuard"/>
      <xsd:element ref="Condition"/>
      <xsd:element ref="NegationGuard"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>

```

Figure 4.30 XSD expressing the structure of Guard

As it is illustrated in Figure 4.30, the "Guard" is any Boolean combination of conditions which may be expressed as "Condition" or any well-formed formula by combining conditions with propositional logic connectives.

4.2.3.5.1 XSD for Expressing Guarded Constraint Combination Relationship

Guarded Constraints can be combined with the propositional logic connectives in order to build more complex constraints. The definition is expressed in Figure 4.31.

```
<xsd:element name="GuardedConstraintCombination">
  <xsd:complexType>
    <xsd:choice>
      <xsd:element ref="GuardedConstraint"/>
      <xsd:element ref="NegationGuardedConstraint"/>
      <xsd:element ref="GuardedConstraintConGuardedConstraint"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

Figure 4.31 XSD expressing GuardedConstraintCombination

"GuardedConstraintCombination" is defined as a complex type expressing with a <choice> indicator. It can be represented as a "GuardedConstraint", a "NegationGuardedConstraint" expressed with a "not" keyword expressed in Figure 4.32. It may be also represented as "GuardedConstraintConGuardedConstraint" which is a combination of guarded constraints with propositional logic connectives.

```
<xsd:element name="NegationGuardedConstraint">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="NegationKeyword" type="xsd:string" fixed="not"/>
      <xsd:element ref="GuardedConstraint"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Figure 4.32 XSD expressing NegationGuardedConstraint

CHAPTER 5

EXTENSIBLE MARKUP LANGUAGE INSTANCES EXPRESSING EXTENDED FEATURE MODELS

In this section a complete sample of a feature model will be illustrated. According to the feature model and XSD definitions as a meta-model, the XML instances will be proposed. The syntax of the language we proposed is implemented as XML schema and as a result each feature model is an XML document conforming to the XSD. The feature model includes four different kinds of decomposition relationships (Mandatory, optional, Alternative and Or). In addition to the decomposition relationships, cross-tree relationships will be illustrated as well. Furthermore, complex cross-tree relationships will be utilized for the sample feature model which may include complex feature-feature, feature-attribute and attribute-attribute relationships as defined in [37]. The CFG rules and XSD definitions of the abstract syntax is covered in previous sections. Features may include attributes in order to express more information about the features.

5.1 eXtensible Markup Language (XML)

XML is a World Wide Web Consortium (W3C) recommendation since February 10, 1998 and is designed to carry and save data [6]. XML data stores in a plain text format causing software/hardware independent way for data storing. It is a complement to Hyper Text Markup Language (HTML) for web pages [40] which is used in order to display the data.

5.1.1 XML Elements

XML has a simple and comprehensive syntax rules expressed with tags. Tag is a markup construct which starts with "<" and ends with ">". Tags are expressed in three flavors: start tags <start>, end tags </end> and empty element tags such as <line-break> [6]. The tags are case sensitive, opening and closing tags should be written with the same cases. Everything includes among the start and end tag is referred to an element in XML. An element may consist of another element, text, attributes or it may be a mix of all these types.

Some characters are not allowed to be written in their original syntax in XML. For instance if you want to insert a character "<" inside an XML element demonstrating "less than", it will cause an error as the parser interprets it as the start of a new element [6]. In order to avoid these kinds of errors, "entity references" are illustrated in table 5.1 from [6].

Table 5.1 Entity references [6]

Symbol	Description	Entity reference
<	Less than	<
>	Greater than	>
&	Ampersand	&
'	Apostrophe	'
"	Quotation mark	"

5.1.2 XML Attributes

Attributes are expressed in order to provide additional information about the elements. Attribute values should be always quoted. It may be a single or double quote. The both alternatives are illustrated in Figure 5.1.

```
<person sex="female">  
<person sex='female'>
```

Figure 5.1 XML instance expressing Attribute value

XML attributes cannot be expressed in a tree structure. In addition multiple values are not allowed to be used for attributes. On the other hand, utilizing a tree structure and multiple values is convenient for the elements of the XML document.

5.1.3 XML Tree

XML documents establish a tree structure that starts with "root" and branches to the lowest levels with "leaves" hierarchically. As an example, consider the Figure 5.2 accessed from [6].

```
<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>
```

Figure 5.2 XML tree structure [6]

5.2 XML Instances for Extended Feature Models

This section will cover the XML instances according to the feature model illustrated in Figure 5.3. The feature model covers all types of the decomposition relationships,

basic and complex cross-tree relationships as well. Some of the features may include attributes identified with their name, domain and value. The relationship among the features may include feature and group cardinality which was covered in previous sections.

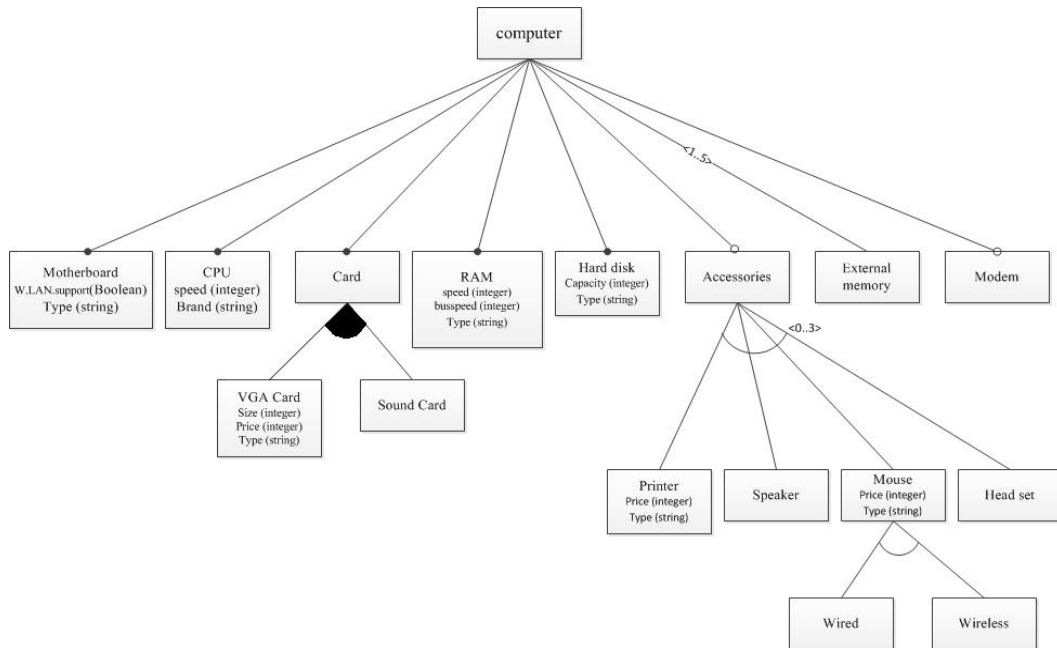


Figure 5.3 A sample feature model for a Computer

The XML instances represented in this section are written based on the XSD definitions expressed in previous chapter. All of the XML documents and the XSD definition are checked in order to control that they are valid or not. This process is done by the tool that will be discussed in next chapter.

5.2.1 XML Instances for Expressing Features

The feature diagram we selected representing in this section consists all the basic decomposition and complex cross-tree relationships defined in [37]. The decomposition relationship is consisting of Mandatory, Optional, Alternative and Or relationship. Besides theses, it may include cardinality-based relationships [15]. The feature cardinality is the relationship among the parent feature and child feature. In Figure 5.3 the feature cardinality is illustrated among the computer feature and

external memory feature with [1..5] cardinality. It means that the inclusion of computer feature implies the inclusion of external memory at least 1 and at most 5 numbers of times. The group cardinality is a relationship among the parent feature and list of child features in which the inclusion of the parent feature implies the inclusion of set of children according to the minimum and maximum cardinality. For instance, in Figure 5.3 the group cardinality is among the Accessories as a parent feature and a set of child features. The inclusion of accessories implies the inclusion of at least zero and at most three number of set of children. In addition complex cross-tree relationships including feature-feature, feature-attribute and attribute-attribute relationships are covered which will be discussed in consequent sections.

```

<?xml version="1.0"?>
<AllFeatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AllFeatures.xsd">
<Features Name="Computer" />
<Features Name="Motherboard" >
<FeatureAttribute Name="WirelessLANSupport" Domain="Boolean"
Value="true,false"/>
<FeatureAttribute Name="type" Domain="string" Value="ASUS,INTEL"/>
</Features>
<Features Name="CPU" >
<FeatureAttribute Name="speed" Domain="integer" />
<FeatureAttribute Name="brand" Domain="string" Value="AMD,INTEL"/>
</Features>
<Features Name="HardDisk" >
<FeatureAttribute Name="capacity" Domain="integer" />
<FeatureAttribute Name="type" Domain="string" Value="Maxtor,Hitachi"/>
</Features>
<Features Name="RAM" >

```

Figure 5.4 XML instance expressing the Feature Model (continued)


```

<FeatureAttribute Name="speed" Domain="integer" />
<FeatureAttribute Name="busspeed" Domain="integer" />
<FeatureAttribute Name="type" Domain="string" Value="SDRAM,RDRAM"/>
</Features>
<Features Name="Card" />
<Features Name="VGAcad" >
<FeatureAttribute Name="size" Domain="integer" />
<FeatureAttribute Name="price" Domain="integer" />
<FeatureAttribute Name="type" Domain="string" Value="ASUS,CLUB,MSI"/>
</Features>
<Features Name="SoundCard" />
<Features Name="Accessories" />
<Features Name="Printer" >
<FeatureAttribute Name="price" Domain="integer" />
<FeatureAttribute Name="type" Domain="string" Value="HP,CANON"/>
</Features>
<Features Name="Speaker" />
<Features Name="Mouse" >
<FeatureAttribute Name="price" Domain="integer" />
<FeatureAttribute Name="type" Domain="string" Value="wired,wireless"/>
</Features>
<Features Name="Wired" />
<Features Name="Wireless" />
<Features Name="HeadSet" />
<Features Name="ExternalMemory" />
<Features Name="Modem" />
</AllFeatures>

```

Figure 5.4 XML instance expressing the Feature Model

Figure 5.4 represents all the features existing in Figure 5.3. Some of the features include attributes consisting name, domain and values, in some cases. The relationship among the features will be discussed in consequent sections.

5.2.2 XML Instances for Expressing Decomposition Relationships

This section will cover the XML instances in order to demonstrate the structure of decomposition relationships of the Figure 5.3. When inclusion of a feature implies the inclusion of its child feature, the relationship among them will be Mandatory. For instance, the parent feature Computer implies the inclusion of MotherBoard, CPU, HardDisk, RAM that are expressed as its children. The relationship among them is a mandatory relationship which is expressed in Figure 5.5.

```
<MandatoryRelation>
  <Parent Name="Computer" ParentName=""/>
  <MandatoryRelationKeyword>Mandatory</MandatoryRelationKeyword>
  <Child Name="Motherboard" />
  <Child Name="CPU" />
  <Child Name="HardDisk" />
  <Child Name="RAM" />
  <Child Name="Card" />
</MandatoryRelation>
```

Figure 5.5 XML instance for expressing Mandatory relation

The relationship among the computer feature as a parent feature and card and accessories as child features is optional as the inclusion of the child features is optional means they may or may not be included. The relationship among them is expressed in Figure 5.6.

```

<OptionalRelation>
  <Parent Name="Computer" ParentName=""/>
  <OptionalRelationKeyword>Optional</OptionalRelationKeyword>
  <Child Name="Accessories" />
  <Child Name="Modem" />
</OptionalRelation>

```

Figure 5.6 XML instance for expressing Optional relation

Another decomposition relation instance is the Alternative relation which is occurred among the parent feature and set of child features. In Figure 5.7, this kind of relationship is illustrated among the speaker feature as a parent feature. The child set features are wired and wireless in which if the speaker feature is included, exactly one of the child features must be included.

```

<AlternativeRelation>
<Parent Name="Mouse" ParentName="Accessories"/>
<AlternativeRelationKeyword>Alternative</AlternativeRelationKeyword>
<ChildSet>
  <Child Name="Wired" />
  <Child Name="Wireless" />
</ChildSet>
</AlternativeRelation>

```

Figure 5.7 XML instance for expressing Alternative relation

The Or relationship is occurred among the parent feature and set of child features in which the inclusion of the parent feature implies the inclusion of nonempty set of child features. According to Figure 5.3, the Or relation in this example is among the card feature as a parent feature and VGA Card and Sound Card as a set of child features.

```

<OrRelation>
  <Parent Name="Card" ParentName="Computer"/>
  <OrRelationKeyword>Or</OrRelationKeyword>
  <ChildSet>
    <Child Name="VGACard" />
    <Child Name="SoundCard" />
  </ChildSet>
</OrRelation>

```

Figure 5.8 XML instance for expressing Or relation

The feature cardinality is expressed in Figure 5.9 and 5.10 including both solitary cardinality and grouped cardinality. The solitary cardinality is occurred between the parent feature and child feature. As it is illustrated in the sample Figure, the solitary cardinality relationship is among the computer feature and external memory feature with the cardinality <1..5> means that the child feature may be included at least one and at last five number of times. The XML instance is illustrated as below in Figure 5.9.

```

<FeatureCardinality>
  <SolitaryCardinality>
    <Parent Name="Computer" ParentName=""/>
    <Cardinality Min="1" Max="5" />
    <Child Name="ExternalMemory" />
  </SolitaryCardinality>
</ FeatureCardinality>

```

Figure 5.9 XML instance for expressing SolitaryCardinality

An XML instance demonstrating grouped cardinality is illustrated in Figure 5.10. The parent feature is Accessories and the child set to be chosen is printer, speaker, mouse, and headset in order. The minimum and maximum cardinality is defined as <0..3> means that at least zero and at most three of the children of Accessories may be included while including Accessories.

```

<FeatureCardinality>
  <GroupedCardinality>
    <Parent Name="Accessories" ParentName="Computer"/>
    <Cardinality Min="0" Max="3" />
    <ChildSet>
      <Child Name="Printer" />
      <Child Name="Speaker" />
      <Child Name="Mouse" />
      <Child Name="HeadSet" />
    </ChildSet>
  </GroupedCardinality>
</FeatureCardinality>

```

Figure 5.10 XML instance for expressing GroupedCardinality

5.2.3 XML Instances for Expressing Complex Cross-Tree Relationships

This section will cover the XML instances representing the cross-tree relationships of the feature model which is illustrated in Figure 5.3. The relationships may construct feature-feature, feature-attribute, and attribute-attribute relationships. The XML instances are expressed according to the XSD definitions which are derived from the definitions of the relationships covered in [37].

5.2.3.1 XML Instances for Expressing Condition Relationships

As it is illustrated in section 4.2.3.1, the condition relationship may be expressed in two different forms $Expression_1 \text{ relop } Expression_2$ and $BoolAttributeDesequaltoTruthValue$. Consider the following constraints for instance:

C1: The price of the *mouse* feature will always be more than \$40. The XML instance of C1 will be as follows:

```

<?xml version="1.0"?>
<Condition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Condition.xsd">
  <ExpressionRelOpExpression>
    <ExpressionStructure>
      <NumericAttributeDes>
        <FeatureName Name="Mouse" />
        <DotKeyword>.</DotKeyword>
        <AttributeName>price</AttributeName>
      </NumericAttributeDes>
    </ExpressionStructure>
    <RelOp>&gt;</RelOp>
    <ExpressionStructure>
      <IntegerConstant>40</IntegerConstant>
    </ExpressionStructure>
  </ExpressionRelOpExpression>
</Condition>

```

Figure 5.11 XML instance expressing Condition relationship including ExpressionRelOpExpression

C2: *Motherboard* supports wireless LAN. Figure 5.12, illustrates the condition which is in the form of *BoolAttributeDesequaltoTruthValue*.

```

<?xml version="1.0"?>
<Condition xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Condition.xsd">
  <BoolAttributeDesequaltoTruthValue>
    <FeatureName Name="Motherboard" />
    <DotKeyword>.</DotKeyword>
    <AttributeName>WirelessLANSupport</AttributeName>
    <EqualKeyword>=</EqualKeyword>
    <TruthValue>>true</TruthValue>
  </BoolAttributeDesequaltoTruthValue>
</Condition>

```

Figure 5.12 XML instance expressing Condition relationship including BoolAttributeDesequaltoTruthValue

5.2.3.2 XML Instances for Expressing Excludes Relationship

This section will cover an XML instance utilizing the Excludes relationship. Consider the following constraint as a basic one demonstrated in C3.

C3: *External Memory* excludes *Accessories*. The XML instance is written in Figure 5.13.

```
<?xml version="1.0"?>
<Excludes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Excludes.xsd">
    <LeftFeatureOfExclude Name="ExternalMemory" />
    <ExcludeKeyword>excludes</ExcludeKeyword>
    <RightFeatureOfExclude Name="Accessories" />
</Excludes>
```

Figure 5.13 XML instance expressing Excludes relationship

5.2.3.3 XML Instances for Expressing Requires Relationship

The Requires relationship may be illustrated in more than one way which is defined in section 4.2.3.3. This section will cover all the defined situations of the relationship. As some simple instances, consider the following constraints.

C4: speaker requires *motherboard* and *CPU*. This constraint will be illustrated as the following as an XML instance.


```

<?xml version="1.0"?>
<Requires xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Requires.xsd">
  <ReqFeature Name="Speaker" />
  <RequireKeyword>requires</RequireKeyword>
  <BooleanFormula>
    <BooleanFormulaConBooleanFormula>
      <BooleanFormula>
        <FormulaFeature Name="Motherboard" />
      </BooleanFormula>
      <BinConnective>and</BinConnective>
      <BooleanFormula>
        <FormulaFeature Name="CPU" />
      </BooleanFormula>
    </BooleanFormulaConBooleanFormula>
  </BooleanFormula>
</Requires>

```

Figure 5.14 XML instance expressing Requires relationship

C5: *external memory* requires 3 GHz CPU speed and *hard disk*. (*external memory* requires CPU.speed=3 GHz and *hard disk*.) In Figure 5.15 the related XML instance is illustrated.

```

<?xml version="1.0"?>
<Requires xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="Requires.xsd">
  <ReqFeature Name="ExternalMemory" />
  <RequireKeyword>requires</RequireKeyword>
  <BooleanFormula>
    <BooleanFormulaConBooleanFormula>
      <BooleanFormula>
        <Condition>
          <ExpressionRelOpExpression>
            <ExpressionStructure>
              <NumericAttributeDes>
                <FeatureName Name="CPU" />
                <DotKeyword>.</DotKeyword>
                <AttributeName>speed</AttributeName>
              </NumericAttributeDes>
            </ExpressionStructure>
          <RelOp>=</RelOp>
          <ExpressionStructure>
            <IntegerConstant>3</IntegerConstant>
          </ExpressionStructure>
        </ExpressionRelOpExpression>
      </Condition>
    </BooleanFormulaConBooleanFormula>
  </BooleanFormula>
</Requires>

```

Figure 5.15 XML instance expressing Requires relationship (continued)

```

        </BooleanFormula>
        <BinConnective>and</BinConnective>
        <BooleanFormula>
        <FormulaFeature Name="HardDisk" />
        </BooleanFormula>
        </BooleanFormulaConBooleanFormula>
    </BooleanFormula>
</Requires>

```

Figure 5.15 XML instance expressing Requires relationship

5.2.3.4 XML Instances for Expressing Complex Constraint Relationship

The Complex Constraint relationship can be a Requires/Excludes relationship which was discussed in section 5.2.3.1 and 5.2.3.3. In addition it can be expressed as any well-formed formula constructed by combining features and/or Requires/Excludes relationships with the propositional logic connectives. This section will cover the XML instances of Complex Constraint relationship according to the sample feature model illustrated in Figure 5.3. As an instance consider the following relationships and their explanations.

C6: speaker and headset implies soundcard.

Figure 5.16 illustrates the XML instance expressing the relationship.

```

<?xml version="1.0"?>
<ComplexConstraint
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="ComplexConstraint.xsd">
  <CPConCP>
    <CP>
      <CPConCP>
        <CP>
          <CPFeature Name="Speaker" />
        </CP>
      <BinConnective>and</BinConnective>
    </CPConCP>
    <CP>
      <CPFeature Name="HeadSet" />
    </CP>
  </CPConCP>
</CP>
<BinConnective>conditional</BinConnective>
<CP>
  <CPFeature Name="SoundCard" />
</CP>
</CPConCP>
</ComplexConstraint>

```

Figure 5.16 XML instance expressing Complex Constraint relationship

C7: (*CPU* requires ((*RAM.speed*>266 MHz and *motherboard*)) or (*RAM.speed*>466 MHz)) and (*CPU* excludes *speaker*). The XML document of this relationship is written as follows in Figure 5.17.

```

<?xml version="1.0"?>
<ComplexConstraint xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="ComplexConstraint.xsd">
  <CPConCP>
    <CP>
      <Requires>
        <ReqFeature Name="CPU" />
        <RequireKeyword>requires</RequireKeyword>
        <BooleanFormula>
          <BooleanFormulaConBooleanFormula>
            <BooleanFormula>
              <BooleanFormulaConBooleanFormula>
                <BooleanFormula>
                  <Condition>
                    <ExpressionRelOpExpression>
                      <ExpressionStructure>
                        <NumericAttributeDes>
                          <FeatureName Name="RAM" />
                          <DotKeyword>.</DotKeyword>
                          <AttributeName>speed</AttributeName>
                        </NumericAttributeDes>
                      </ExpressionStructure>

```

Figure 5.17 XML instance expressing Complex Constraint relationship (continued)


```

<ExpressionStructure>
  <IntegerConstant>466</IntegerConstant>
</ExpressionStructure>
</ExpressionRelOpExpression>
</Condition>
</BooleanFormula>
</BooleanFormulaConBooleanFormula>
</BooleanFormula>
</Requires>
</CP>
<BinConnective>and</BinConnective>
<CP>
  <Excludes>
    <LeftFeatureOfExclude Name="CPU" />
    <ExcludeKeyword>excludes</ExcludeKeyword>
    <RightFeatureOfExclude Name="Speaker" />
  </Excludes>
</CP>
</CPConCP>
</ComplexConstraint>

```

Figure 5.17 XML instance expressing Complex Constraint relationship

5.2.3.5 XML Instances for Expressing Guarded Constraint Relationship

This section will cover some XML instances utilized on the feature model which is demonstrated in Figure 5.3. The Guarded Constraint is a relationship in the form:

If *Guard* then *Complex Constraint*. The XML instances of this relationship are shown in the following examples.

C8: If *VGAcad.size* < 1 GB or *RAM.speed* ≤ 100 MHz then *RAM* excludes *Accessories*. The XML instance illustrating the constraint is shown in Figure 5.18.

```
<?xml version="1.0"?>
<GuardedConstraint xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GuardedConstraint.xsd">
  <IfKeyword>If</IfKeyword>
  <Guard>
    <GuardConGuard>
      <Guard>
        <Condition>
          <ExpressionRelOpExpression>
            <ExpressionStructure>
              <NumericAttributeDes>
                <FeatureName Name="VGAcad" />
                <DotKeyword>.</DotKeyword>
                <AttributeName>size</AttributeName>
              </NumericAttributeDes>
            </ExpressionStructure>
            <RelOp>&lt;</RelOp>
            <ExpressionStructure>
              <IntegerConstant>1</IntegerConstant>
            </ExpressionStructure>
          </ExpressionRelOpExpression>
        </Condition>
      </Guard>
    </GuardConGuard>
  </Guard>
</GuardedConstraint>
```

Figure 5.18 XML instance expressing Guarded Constraint relationship (continued)


```

</Guard>
<BinConnective>or</BinConnective>
<Guard>
<Condition>
<ExpressionRelOpExpression>
<ExpressionStructure>
<NumericAttributeDes>
  <FeatureName Name="RAM" />
<DotKeyword>.</DotKeyword>
<AttributeName>speed</AttributeName>
</NumericAttributeDes>
</ExpressionStructure>
<RelOp>&lt;=</RelOp>
<ExpressionStructure>
<IntegerConstant>100</IntegerConstant>
</ExpressionStructure>
</ExpressionRelOpExpression>
</Condition>
</Guard>
</GuardConGuard>
</Guard>
<ThenKeyword>Then</ThenKeyword>
<ComplexConstraint>

```

Figure 5.18 XML instance expressing Guarded Constraint relationship (continued)

```

</GuardConGuard>
</Guard>
<ThenKeyword>Then</ThenKeyword>
<ComplexConstraint>
<Excludes>
    <LeftFeatureOfExclude Name="RAM" />
    <ExcludeKeyword>excludes</ExcludeKeyword>
    <RightFeatureOfExclude Name="Accessories" />
</Excludes>
</ComplexConstraint>
</GuardedConstraint>

```

Figure 5.18 XML instance expressing Guarded Constraint relationship

C9: if the *motherboard* supports wireless LAN then CPU requires *harddisk.type* = Hitachi. The relationship is illustrated in Figure 5.19.

```

<?xml version="1.0"?>
<GuardedConstraint xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GuardedConstraint.xsd">
    <IfKeyword>If</IfKeyword>
    <Guard>
        <Condition>
            <BoolAttributeDesequaltoTruthValue>
                <FeatureName Name="Motherboard" />
                <DotKeyword>.</DotKeyword>
                <AttributeName>WirelessLANSupport</AttributeName>
                <EqualKeyword>=</EqualKeyword>
                <TruthValue>>true</TruthValue>
            </BoolAttributeDesequaltoTruthValue>

```

Figure 5.19 XML instance expressing Guarded Constraint relationship (continued)

```

</Condition>
</Guard>
<ThenKeyword>Then</ThenKeyword>
<ComplexConstraint>
  <Requires>
    <ReqFeature Name="CPU" />
    <RequireKeyword>requires</RequireKeyword>
  <BooleanFormula>
    <Condition>
      <ExpressionRelOpExpression>
        <ExpressionStructure>
          <NumericAttributeDes>
            <FeatureName Name="HardDisk" />
            <DotKeyword>.</DotKeyword>
            <AttributeName>type</AttributeName>
          </NumericAttributeDes>
        </ExpressionStructure>
        <RelOp>=</RelOp>
        <ExpressionStructure>
          <IntegerConstant>2</IntegerConstant>
        </ExpressionStructure>
      </ExpressionRelOpExpression>
    </Condition>
  </BooleanFormula>
</Requires>
</ComplexConstraint>
</GuardedConstraint>

```

Figure 5.19 XML instance expressing Guarded Constraint relationship

Note that `HardDisk` has an attribute and its name is `type`, with the domain `{Maxtor,Hitachi}`, which does not consist of integer values. As only integer values in operands of the `relop` can be included, the following conversion is utilized in which `{Maxtor → 1, Hitachi → 2}`. As a result the new domain of the attribute becomes `{1, 2}`.

5.2.3.5.1 XML Instances for Expressing Guarded Constraint Combination

Guarded Constraint Combination structure is as same as the Guarded Constraint in which Guarded Constraints can be combined using Propositional Logic Connectives. This section will cover an XML instance of Guarded Constraint combination illustrated in the sample feature diagram. The relationship is defined as follows:

C10: (If *Mouse*.price > 30\$ Then *RAM* excludes *Mouse*) and (If *Mouse*.price < 15\$ Then *RAM* requires *Mouse*). The related XML instance is demonstrated in Figure 5.20.

```

<?xml version="1.0"?>
<GuardedConstraintCombination xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="GuardedConstraintCombination.xsd">
  <GuardedConstraintConGuardedConstraint>
    <GuardedConstraint>
      <IfKeyword>If</IfKeyword>
      <Guard>
        <Condition>
          <ExpressionRelOpExpression>
            <ExpressionStructure>
              <NumericAttributeDes>
                <FeatureName Name="Mouse" />

```

Figure 5.20 XML instance expressing Guarded Constraint Combination relationship (continued)

```

        <DotKeyword>.</DotKeyword>
        <AttributeName>price</AttributeName>
    </NumericAttributeDes>
</ExpressionStructure>
    <AttributeName>price</AttributeName>
</NumericAttributeDes>
</ExpressionStructure>
    <RelOp>></RelOp>
    <ExpressionStructure>
        <IntegerConstant>30</IntegerConstant>
    </ExpressionStructure>
</ExpressionRelOpExpression>
</Condition>
</Guard>
<ThenKeyword>Then</ThenKeyword>
<ComplexConstraint>
    <Excludes>
        <LeftFeatureOfExclude Name="RAM" />
        <ExcludeKeyword>excludes</ExcludeKeyword>
        <RightFeatureOfExclude Name="Mouse" />
    </Excludes>
</ComplexConstraint>
</GuardedConstraint>
    <BinConnective>and</BinConnective>
    <GuardedConstraint>
        <IfKeyword>If</IfKeyword>
    </GuardedConstraint>
</Guard>

```

Figure 5.20 XML instance expressing Guarded Constraint Combination relationship
(continued)

```

<Condition>
  <ExpressionRelOpExpression>
    <ExpressionStructure>
      <NumericAttributeDes>
        <FeatureName Name="Mouse" />
        <DotKeyword>.</DotKeyword>
        <AttributeName>price</AttributeName>
      </NumericAttributeDes>
    </ExpressionStructure>
    <RelOp>&lt;</RelOp>
    <ExpressionStructure>
      <IntegerConstant>30</IntegerConstant>
    </ExpressionStructure>
  </ExpressionRelOpExpression>
</Condition>
</Guard>
<ThenKeyword>Then</ThenKeyword>
<ComplexConstraint>
  <Requires>
    <ReqFeature Name="RAM" />
    <RequireKeyword>requires</RequireKeyword>
    <BooleanFormula>
      <FormulaFeature Name="Mouse" />
    </BooleanFormula>
  </Requires>
</ComplexConstraint>
</GuardedConstraint>

```

Figure 5.20 XML instance expressing Guarded Constraint Combination relationship (continued)

```
</GuardedConstraintConGuardedConstraint>  
</GuardedConstraintCombination>
```

Figure 5.20 XML instance expressing Guarded Constraint Combination relationship

CHAPTER 6

FEATURE MODEL MARKUP LANGUAGE VALIDATION

This chapter provides validating process in order to validate the XML notations defined in chapter four according to the XSD expressions. The XSD expressions are written based on the abstract syntax mentioned in [37] and related CFGs presented in chapter three. The XML instances provided in previous section are provided according to the sample feature model illustrated in Figure 5.3 consisting different kinds of relationships among the features which includes basic decomposition and complex feature-feature, feature-attribute, and attribute-attribute relationships. The XML instances are provided according to the schema definitions.

In this chapter the validation of FMML will be represented which is implemented with java eclipse. The validation process is done in two steps. The first part validates the XML instances according to the schema definitions. The second part is implemented in order to control the XML instances according to the design of the feature models. For validation purposes, the user can facilitates the functionalities offered by the validate submenu illustrated in Figure 6.1 including XSD validate and XML validate.

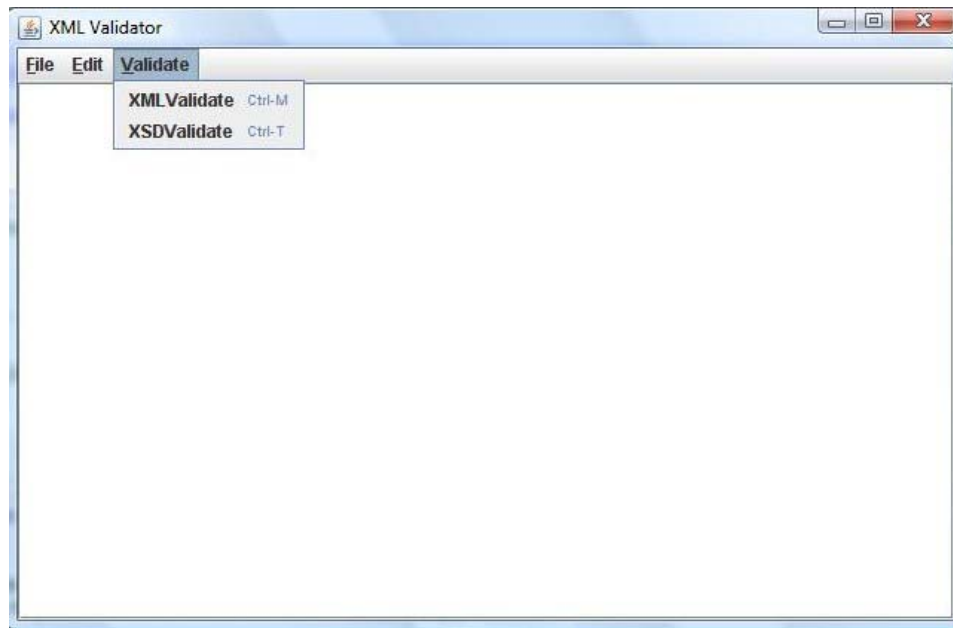


Figure 6.1 Validating processes

6.1 XSD Validation

This section provides the validation process of the XML instances according to the schema definition. The XSD expressions are represented in chapter four which are defined based on the CFG rules expressed in chapter three.

While expressing the schema definition, specific rules and techniques are utilized in order to organize the definitions. For instance, `<sequence>` indicator is utilized to define consecutive symbols. As another example in the case of an event that rewrite rule provides at least two alternative symbols, the `<choice>` indicator is utilized. In addition, occurrence constraints can be used defining repetitive symbols. They are defined as `minOccurs` and `maxOccurs` keywords and can be utilized constraining cardinality. `optional` and `required` indicators are also can be utilized defining the usage of the attributes that may be obligatory or an optional one.

In order to implement the validity control of the XML instances according to the XSD expressions, some of existing Java libraries are utilized illustrated in table 6.1.

Table 6.1 Java libraries applied for validating XML instances according to XSD

Java library	Explanation
<code>javax.xml.transform.stream.StreamSource;</code>	Acts as a holder for a transformation source in a form of a stream of XML markup
<code>javax.xml.validation.Schema</code>	Represents a set of constraints that can be checked against an XML document
<code>javax.xml.validation.SchemaFactory</code>	Reads representations of schema and prepare it for validation
<code>javax.xml.validation.Validator</code>	Checks XML document against Schema

6.1.1 Valid and Invalid Instances

This section provides acceptable and unacceptable XML instances according to the XSD definitions. Consider the following XML instance illustrated in Figure 6.3 illustrating a Mandatory relation according to the Figure 6.2. The XSD expression of Mandatory relation is also defined in Figure 4.7.

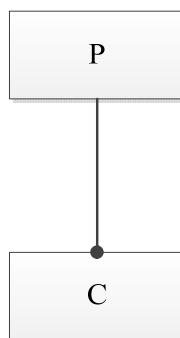


Figure 6.2 Mandatory relation

```
<?xml version="1.0"?>
<MandatoryRelation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MandatoryRelation.xsd">
  <Parent Name="P" />
  <MandatoryRelationKeyword>Mandatory</MandatoryRelationKeyword>
  <Child Name="C" />
</MandatoryRelation>
```

Figure 6.3 XML instance expressing Mandatory relation according to Figure 6.2 and XSD illustrated in Figure 4.7

The XML instance is validated and the output is illustrated in Figure 6.4 which demonstrates the instance is valid.

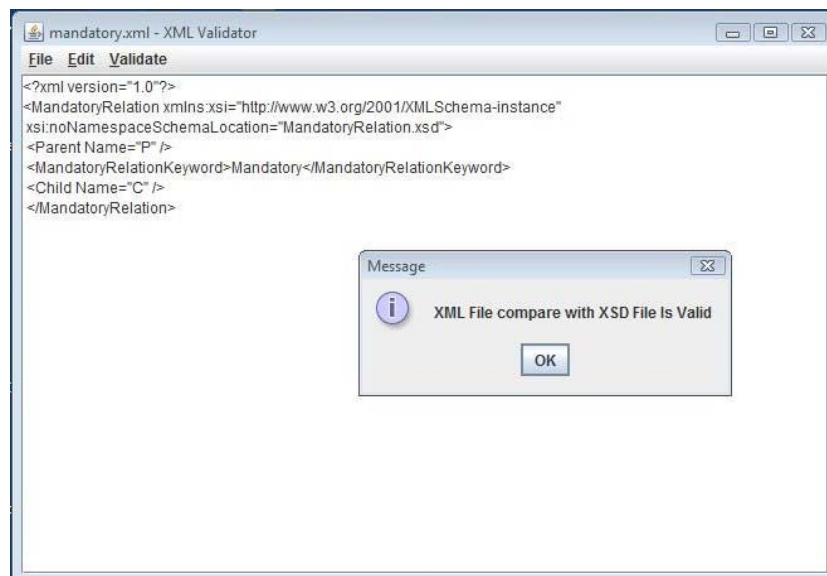


Figure 6.4 Valid XML instance expressing Mandatory relation

As an unacceptable example, consider the following XML instance for defining the Mandatory relation of the sample feature model given in Figure 6.2.

```

<?xml version="1.0"?>
<MandatoryRelation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="MandatoryRelation.xsd">
  <Child Name="C" />
  <MandatoryRelationKeyword>MandatoryRelation</MandatoryRelationKeyword>
  <Parent Name="P" />
</MandatoryRelation>

```

Figure 6.5 XML instance expressing Mandatory relation according to Figure 6.2 and XSD illustrated in Figure 4.7

The result of validating the XML instance according to XSD is unacceptable which is illustrated in Figure 6.6. The Mandatory relation is defined as a `complexType` including a `<sequence>` indicator in which the order of the expressions is important. In order to illustrate the Mandatory relation Parent Name, MandatoryRelationKeyword, and Child Name must be expressed consecutively.

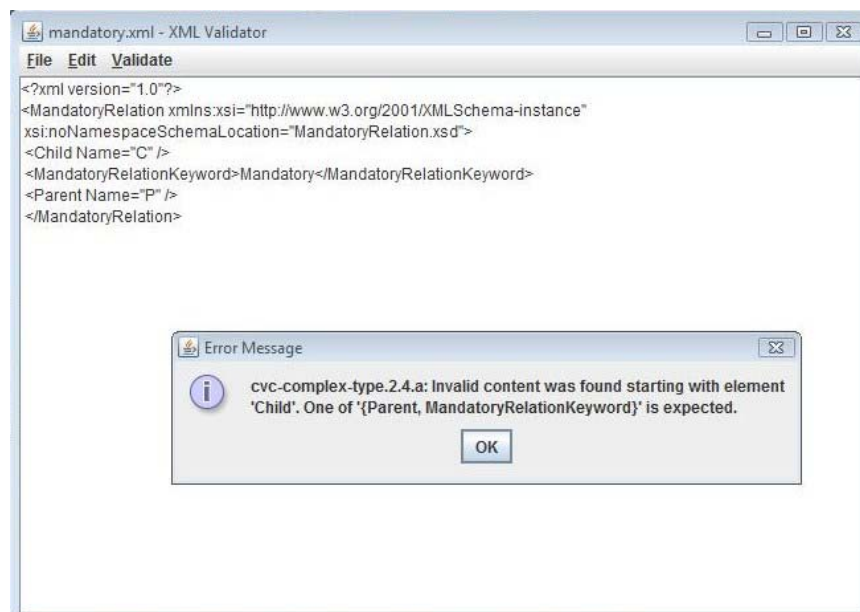


Figure 6.6 Unacceptable XML instance expressing Mandatory relation

As another invalid example, consider a feature including an attribute illustrated in Figure 6.7. The attribute consists of name (id), domain (integer) and value ([1..3500]).

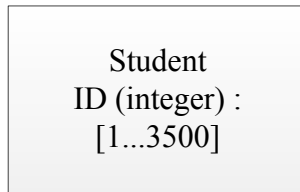


Figure 6.7 A sample feature including Attribute

Figure 6.8 demonstrates an XML instance according to the schema definition expressed in Figure 4.4, and the output is illustrated in Figure 6.9.

```
<?xml version="1.0"?>
<AllFeatures xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="AllFeatures.xsd">
  <Features Name="Student" >
    <FeatureAttribute Name="ID" Value="[1800..3500]"/>
  </Features>
</AllFeatures>
```

Figure 6.8 XML instance expressing feature including Attribute according to Figure 6.7 and XSD illustrated in Figure 4.4

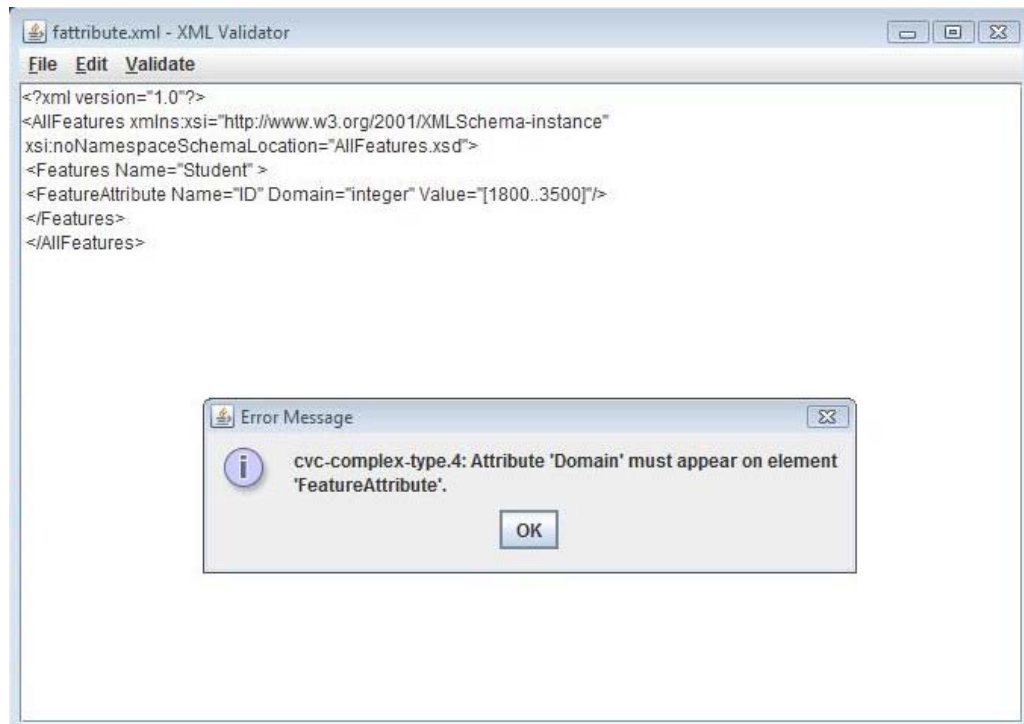


Figure 6.9 Unacceptable XML instance expressing feature including attribute

One of the attributes defined in `FeatureAttribute` element is the `Domain` of an attribute of a feature. The usage of the `Domain` is "required" according to the schema definition.

6.2 XML Well-Formedness Validation

There are other rules a feature model must adhere which are not covered in XSD schema definition. For instance, the schema definition allows declaring a feature which is the parent of another feature to be the child of that feature. As an example, consider the following Figure where the `X` is the parent of the `Y` feature and the decomposition relation among them is optional. According to the XSD expressions the `Y` feature can be defined as the parent of the `X` feature which is unacceptable case illustrating a feature model.

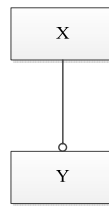


Figure 6.10 Feature model sample

In addition, decomposition cardinalities $\langle \text{min}, \text{max} \rangle$ have to be so that $\text{min} \leq \text{max}$ in both cases of **FeatureCardinality** (**SolitaryCardinality** and **GroupedCardinality**). In the situation of applying **GroupedCardinality**, the value of **max** should not exceed the number of child features.

In order to solve the problems, an XML well-formedness implementation is added to the validity control of the XML instances according to XSD definitions.

In order to implement the required validation processes an existing XML parser for Java is utilized and the validation processes are inserted to the existing code. Figure 6.11 shows a Data Flow Diagram of the XML well-formedness validating process.

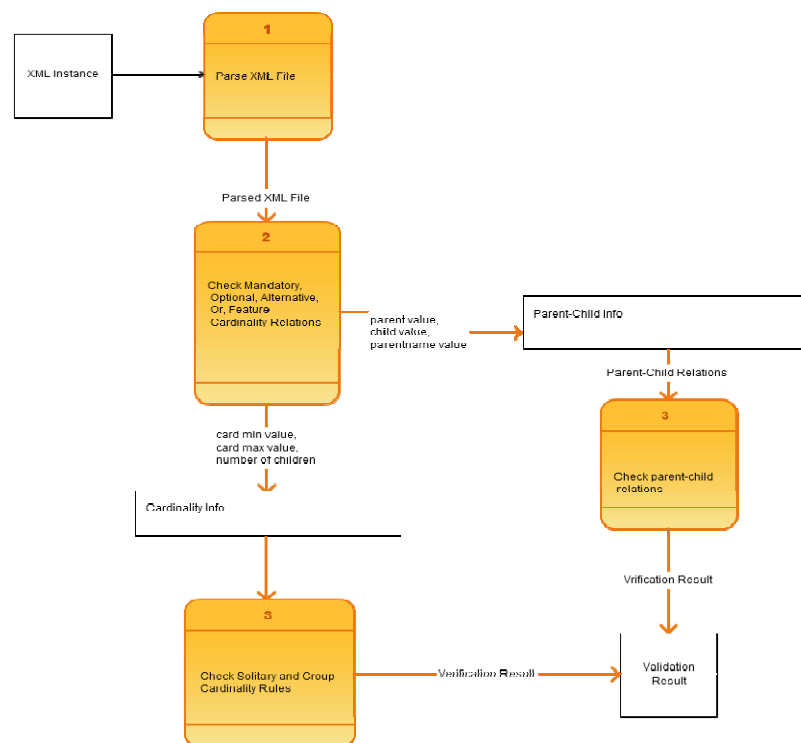


Figure 6.11 XML validating process

For validating process, while parsing the XML document, all the decomposition relationship types are checked one by one in order to control the parent-child relationship. For instance, if a feature (X) is defined as parent feature of other feature(s) (Y) in a relationship, then the child feature (Y) cannot be defined as a parent feature for (X) in another decomposition relationship.

In addition, the cardinality values in cardinality-based relationships are checked as well in validating process. Both the group cardinality and solitary cardinality are validated. In solitary cardinality, the minimum cardinality value should be less or equal to the value of maximum cardinality. In group cardinality, the minimum cardinality value should be less or equal to the value of maximum cardinality. Besides that, the value of maximum cardinality should not exceed the number of child/children. Table 6.2 illustrates the Java libraries imported in order to validate the XML document solely.

Table 6.2 Java libraries applied for validating XML instances

Java library	Explanation
<code>org.xml.sax.SAXException</code>	Encapsulates a basic error or warning information from the XML parser
<code>org.xml.sax.SAXParseException</code>	Encapsulates an XML parse error or warning
<code>org.w3c.dom.*</code>	Constructs a Document Object Model according to w3c standards
<code>javax.xml.parsers.DocumentBuilderFactory</code>	Enables applications to obtain a parser that produces DOM object trees from XML documents
<code>javax.xml.parsers.DocumentBuilder</code>	Defines the API to obtain DOM document instances from an XML document

6.2.1 Valid and Invalid Instances

This section provides acceptable and unacceptable samples in order to illustrate how the XML document is validated according to the constraints expressed in the previous section.

As an instance, consider the sample feature model illustrated in Figure 5.3.

The relationship among the "Computer" and "Motherboard" is Mandatory in which the parent feature is "Computer" and the child feature is "Motherboard". The XML instance illustrating the relationship is shown in Figure 6.12.

```
<?xml version="1.0"?>
<DecompositionRelation xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecompositionRelation.xsd">
  <MandatoryRelation>
    <Parent Name="Computer" ParentName=""/>
    <MandatoryRelationKeyword>Mandatory</MandatoryRelationKeyword>
    <Child Name="Motherboard" />
  </MandatoryRelation>
</DecompositionRelation>
```

Figure 6.12 XML instance illustrating Mandatory relationship according to Figure 5.3 and XSD illustrated in Figure 4.7

The XML instance is validated and the output is as follows shown in Figure 6.13.

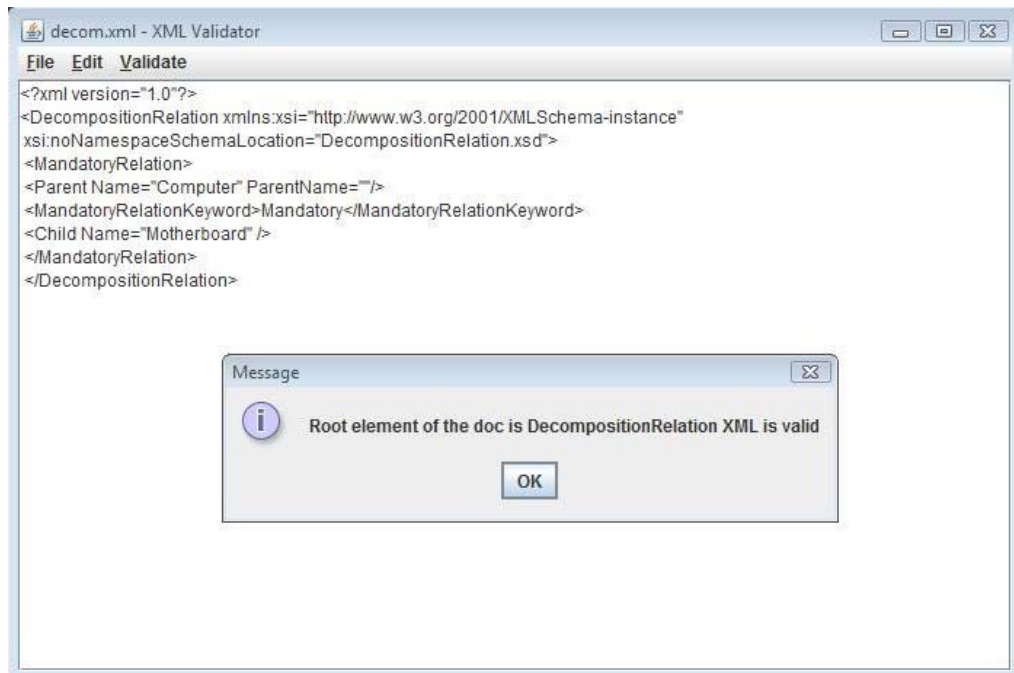


Figure 6.13 Valid XML instance expressing relationship among features

Now, consider the following XML instance, Figure 6.14, in which an optional relationship is expressed as well. The parent feature is "Motherboard" and the child features are "Accessories" and "Modem". In order to illustrate how the XML instance is validated we also add "Computer" feature as a child feature of "Motherboard".

```
<?xml version="1.0"?>
<DecompositionRelation
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecompositionRelation.xsd">
  <MandatoryRelation>
    <Parent Name="Computer" ParentName=""/>
    <MandatoryRelationKeyword>Mandatory</MandatoryRelationKeyword>
    <Child Name="Motherboard" />
  </MandatoryRelation>
```

Figure 6.14 XML instance illustrating Mandatory and Optional relationships according to Figure 5.3 and XSD illustrated in Figure 4.7 and 4.8 (continued)

```

<OptionalRelation>
  <Parent Name="Motherboard" ParentName="Computer"/>
  <OptionalRelationKeyword>Optional</OptionalRelationKeyword>
  <Child Name="Accessories" />
  <Child Name="Modem" />
  <Child Name="Computer" />
</OptionalRelation>
</DecompositionRelation>

```

Figure 6.14 XML instance illustrating Mandatory and Optional relationships according to Figure 5.3 and XSD illustrated in Figure 4.7 and 4.8

The output of validating the XML instance is illustrated as follows in Figure 6.15 which is shown as an unacceptable one.

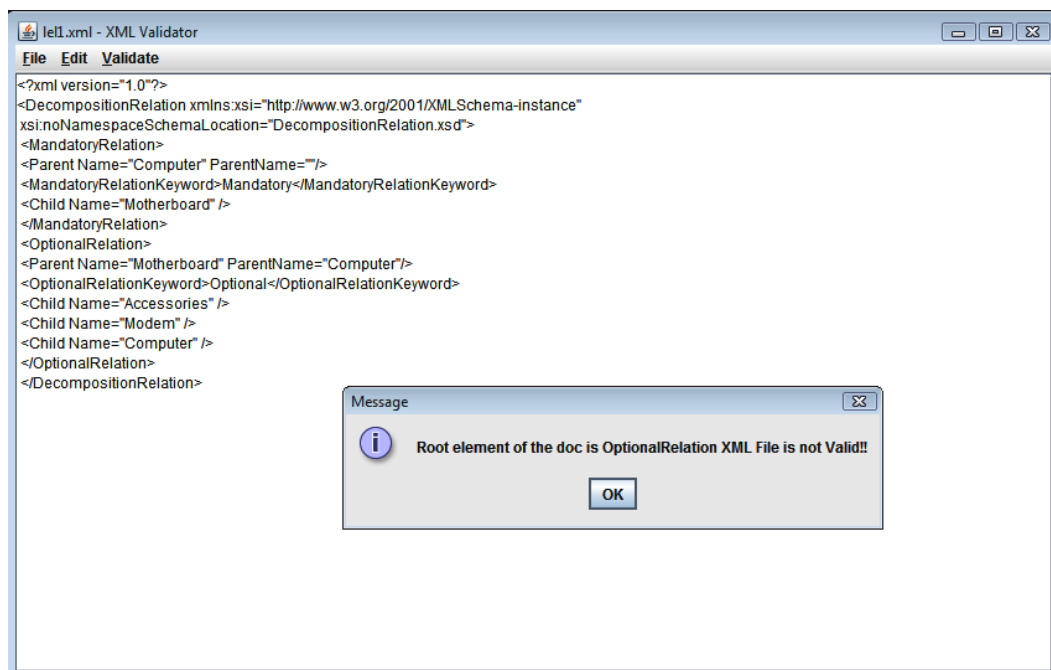


Figure 6.15 Unacceptable XML instance expressing the relationships among features
 As another valid example expressing minimum and maximum cardinality in an acceptable way, consider the following XML instance according to the sample feature model illustrated in Figure 5.3.

```
<?xml version="1.0"?>
<DecompositionRelation
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecompositionRelation.xsd">
<FeatureCardinality>
<GroupedCardinality>
<Parent Name="Accessories" ParentName="Computer"/>
<Cardinality Min="0" Max="3" />
<ChildSet>
  <Child Name="Printer" />
  <Child Name="Speaker" />
  <Child Name="Mouse" />
  <Child Name="HeadSet" />
</ChildSet>
</GroupedCardinality>
</FeatureCardinality>
</DecompositionRelation>
```

Figure 6.16 XML instance expressing Grouped Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.13

The result of validating the XML instance is illustrated in Figure 6.17. It shows that the XML instance is valid according to the rules defined in section 5.2.

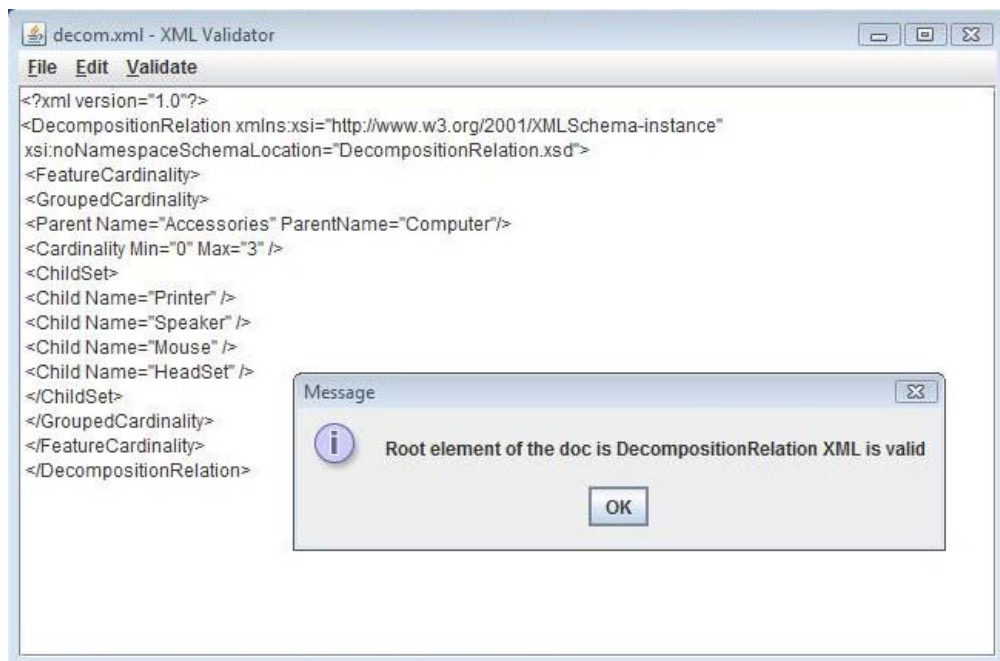


Figure 6.17 Acceptable XML instance expressing the Feature Cardinality

Now consider the following examples illustrated in Figure 6.18 in which the minimum cardinality and maximum cardinality are defined in an invalid way.

```

<?xml version="1.0"?>

<DecompositionRelation
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="DecompositionRelation.xsd">

<FeatureCardinality>

<GroupedCardinality>

<Parent Name="Accessories" ParentName="Computer"/>

<Cardinality Min="3" Max="1" />

```

Figure 6.18 XML instance expressing Feature Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.11 (continued)

```

<ChildSet>
  <Child Name="Printer" />
  <Child Name="Speaker" />
  <Child Name="Mouse" />
  <Child Name="HeadSet" />
</ChildSet>
</GroupedCardinality>
</FeatureCardinality>
</DecompositionRelation>

```

Figure 6.18 XML instance expressing Feature Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.11

The result validating the XML instance expressed in Figure 6.19 is as follows illustrating the instance is not valid. In this example the value of minimum cardinality is three where as the value of maximum cardinality is one. The result is shown as unacceptable XML instance as the min>max.

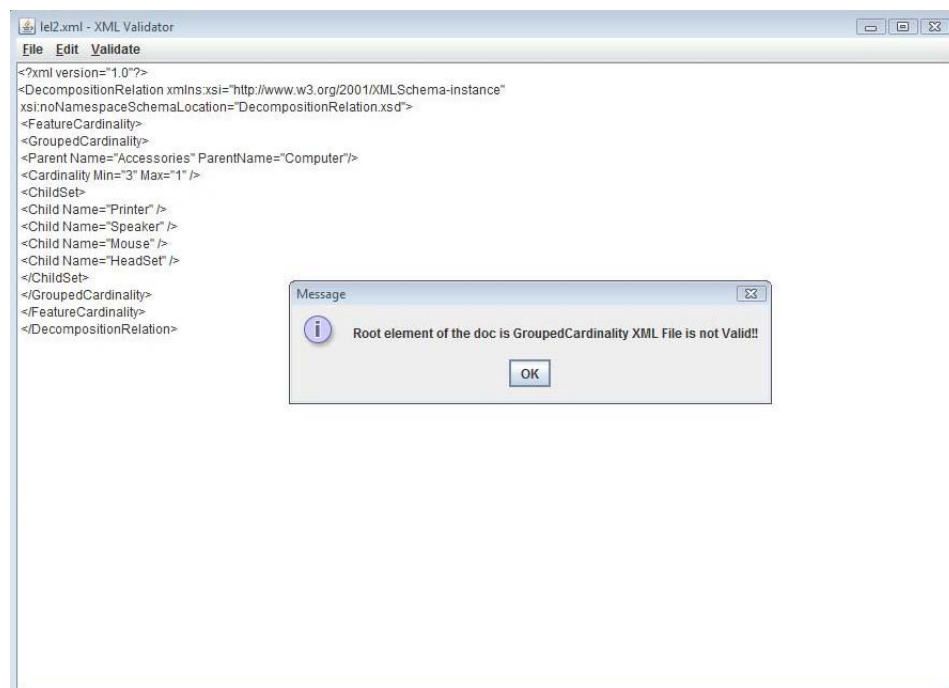


Figure 6.19 Unacceptable XML instance expressing Feature Cardinality

```

<?xml version="1.0"?>
<DecompositionRelation
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="DecompositionRelation.xsd">
  <FeatureCardinality>
    <GroupedCardinality>
      <Parent Name="Accessories" ParentName="Computer"/>
      <Cardinality Min="0" Max="10" />
      <ChildSet>
        <Child Name="Printer" />
        <Child Name="Speaker" />
        <Child Name="Mouse" />
        <Child Name="HeadSet" />
      </ChildSet>
    </GroupedCardinality>
  </FeatureCardinality>
</DecompositionRelation>

```

Figure 6.20 XML instance expressing Feature Cardinality according to Figure 5.3 and XSD illustrated in Figure 4.11

In this example, however, the $\text{min} \leq \text{max}$ the value of maximum cardinality exceeds the number of child set of the parent feature. The output is illustrated in Figure 6.21.

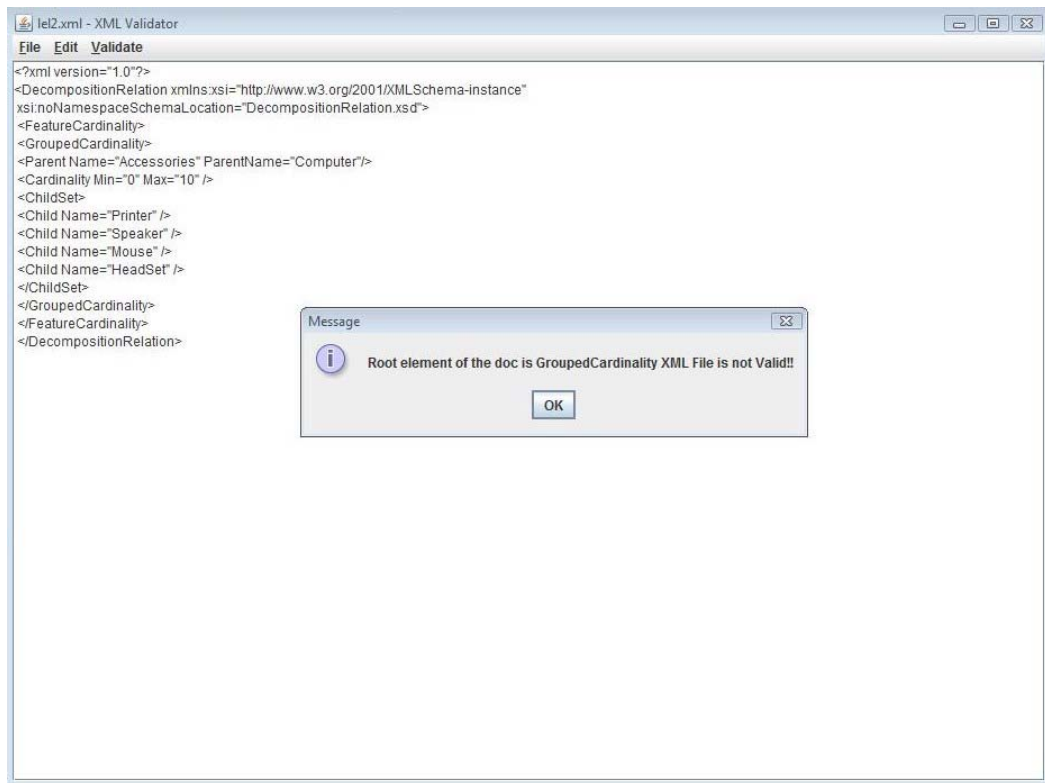


Figure 6.21 Unacceptable XML instance expressing Feature Cardinality

Although all of the samples illustrated in this section are valid according to the XSD definitions, they are not allowed to be expressed according to the well-formedness rules.

CHAPTER 7

CONCLUSION

In this thesis study, an XML-based feature modeling language in order to represent commonality and variability in Software Product Line engineering is provided. Beside the capability of expressing the structure of basic feature models, constructing complex cross-tree constraints including feature-feature, feature-attribute, attribute-attribute, and cardinality-based decomposition is feasible utilizing the proposed language. The approach suggested in this thesis provides CFG rules according to the abstract definitions expressed previously. Consequently, XML schema definitions are provided according to the CFG rules. Next chapter provides XML instances according to a sample feature model including basic and complex relationships in order to illustrate the decomposition and cross-tree relationships. XML instances are validated according to the XSD expressions defined for language. In addition, the XML instances are validated solely in order to examine if they are well-formed according to the defined rules for expressing feature models.

Although graphical notations seem to be more convenient to non-technical stakeholders working with large feature diagrams can be difficult on a two-dimensional surface. In addition, graphical notations do not support expressing constructs like attributes and constraints which are essential factors for feature modeling. Compared with graphical notations, text-based notations can be more convenient to express feature models and automated processing.

The main contribution of this work is the XML representation for the feature modeling language with confidentiality extensions. This style of representation offers advantages that are hard to achieve with other text-based feature modeling languages. Benefiting from XML-based feature modeling language enables the use

of a rich selection of off-the-shelf XML parsers which offers advantages that are not readily available with traditional plain-text encoding. Further, the availability of a wide range of free and commercial tools facilitates the processing of XML-based representations.

The main drawback of this approach is its dependency on the user's experience and knowledge of XML-based representations, which is difficult to interpret mainly due to the overhead caused by XML tags and technical information. However, this problem will be going to be addressed by the envisioned tool utilizing a human readable and user friendly language for the users [8].

For future, this work can be enhanced by expressing the global constraints [41]. It can also be checked by different XML parsers and tools in order to realize whether the features in XSD expressions are tool dependent or not. In addition, a tool can be implemented extending this work in order to implement a graphical environment representing extended feature models including complex cross-tree and cardinality-based relationships and attributes.

REFERENCES

- [1] Magnus Eriksson Alvis Häggblunds, AN INTRODUCTION TO SOFTWARE PRODUCT LINE DEVELOPMENTSE-891 82 Örnsköldsvik, Sweden
- [2] Ezran, M., Morisio, M., Tully, C. (2002). Practical Software Reuse, Springer
- [3] K. Pohl, G. Bockle, and F. J. van der Linden, Software Product Line Engineering: Foundations, Principles and Techniques. Springer, 2005.
- [4] A. VAN DEURSEN, P. KLINT, AND J. VISSER, Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notices*,35 6_:26–36, June 2000
- [5] D. Benavides, P. Trinidad, and A. Ruiz-Cortes. “Automated Reasoning on Feature Models”, Conference on Advanced Information Systems Engineering (CAISE), July 2005.
- [6] XML Tutorial, <http://www.w3schools.com/xml/default.asp>, (*last accessed on 25/03/2011*)
- [7] Nabdell, L., Karataş, A.S., Oğuztüzün, H., Doğru, A., "FMML: A Feature Model Markup Language", In: International Conference on Numerical Analysis and Applied Mathematics (ICNAAM 2011), AIP Conf. Proc. / Volume 1389 / Issue 1, pp. 841-844 (September 2011)
- [8] A. S. Karataş, H. Oğuztüzün, and A. Doğru. "From extended feature models to constraint logic programming", (Submitted for publication)
- [9] Chomsky, Noam (Sept. 1956). "Three models for the description of language". *Information Theory, IEEE Transactions* 2 (3)
- [10] XML Schema Tutorial, <http://www.w3schools.com/schema/default.asp>, (*last accessed on 20/03/2011*)
- [11] P. Schobbens, J.C. Trigaux P. Heymans, and Y. Bontemps. “Generic semantics of feature diagrams”, *Computer Networks*, 51(2):456–479, Feb 2007.
- [12] Danilo Beuche, Holger Papajewski, Wolfgang Schröder-Preikschat "Variability management with feature models *Science of Computer Programming*", Volume 53, Issue 3, December 2004, Pages 333-352

- [13] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. "Feature-Oriented Domain Analyses (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, 1990..
- [14] M. Griss, J. Favaro, and M. d'Alessandro. Integrating Feature Modeling with the RSEB. In Proceedings of the Fifth International Conference on Software Reuse, pages 76-85, Vancouver, BC, Canada, June 1998
- [15] M. Riebisch, K. Bollert, D. Streitferdt, and I. Philippow. "Extending Feature Diagrams With UML Multiplicities", 6th Conference on Integrated Design & Process Technology (IDPT 2002), Pasadena, California, USA, 2002.
- [16] K. Czarnecki, S. Helsen, and U. Eisenecker. "Staged Configurations Using Feature Models", Software Product Lines: Third International Conference, SPLC 2004, Proceedings, Vol. 3154 of Lecture Notes in Computer Science, Springer-Verlag, Heidelberg, Germany, pp. 266–283, Boston, MA, USA, August 30-September 2, 2004.
- [17] K. Czarnecki and C.H.P. Kim. "Cardinality-based feature modeling and constraints: a progress report", International Workshop on Software Factories, San Diego, California, Oct2005.
- [18] K. Kang, S. Kim, J. Lee, and K. Kim. "FORM: A feature-oriented reuse method with domain-specific reference architectures", Annals of Software Engineering, volume 5, pp 143-168, 1998.
- [19] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker. "Generative programming for embedded software: An industrial experience report", Proceedings of the ACM SIGPLAN/ SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'02), Pittsburgh, LNCS 2487, Springer-Verlag (2002) 156–172, October 6–8, 2002.
- [20] Arie van Deursen, Paul Klint: "Domain-Specific Language Design Requires Feature Descriptions", Journal of Computing and Information Technology -CIT 10, 2002, 1, 1–17
- [21] Boucher, Q., Classen, A., Faber, P., Heymans, P.: Introducing TVL, A text-based approach to feature modelling: Syntax and semantics of TVL Science of Computer Programming, Volume 76, Issue 12, 1 December 2011, Pages 1130-1143
- [22] V. Cechticky, A. Pasetti, O. Rohlik, W. Schaufelberger: Xml-based feature modelling presented at the ICSR conference in 2004 and published in the book Software Reuse: Methods, Techniques, and Tools edited by J. Bosch and C. Krueger and published by Springer-Verlag.

- [23] G. Ge, "Rhizome: A Feature Modeling and Generation Platform for Software Product Lines," Department of Computer Science, Ph.D. Thesis, Santa Cruz: University of California, Santa Cruz, 2008.
- [24] M. Antkiewicz, K. Czarnecki, Featureplugin: Feature modeling plug-in for Eclipse, in: Proceedings of the OOPSLA'04 ETX Workshop.
- [25] XPath Tutorial, <http://www.w3schools.com/xpath/>, (last accessed on 10/03/2011)
- [26] D. Benavides, S. Segura, P. Trinidad, A. R. Cortés, FAMA: Tooling a framework for the automated analysis of feature models, in: Proceedings of VaMoS'07, pp. 129–134.
- [27] Edward Tsang. Foundations of Constraint Satisfaction. Academic Press, 1995.
- [28] S. Cook. The complexity of theorem-proving procedures. In Conference Record of Third Annual ACM Symposium on Theory of Computing, pages 151–158, 1971.
- [29] R. Bryant. Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers, 35(8):677–691, 1986.
- [30] Christian Kastner, Thomas Thum, Gunter Saake, Janet Feigenspan, Thomas Leich, Fabian Wielgorz, Sven Apel. FeatureIDE: A Tool Framework for Feature-Oriented Software Development, (ICSE) 2009.
- [31] D. Benavides, S. Segura, A. Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review", Information Systems, Volume 35, Issue 6, pages 615-636, 2010.
- [32] M. Mannion. Using First-Order Logic for Product Line Model Validation. In Proceedings of the Second Software Product Line Conference (SPLC2), LNCS 2379, pages 176–187, San Diego, CA, 2002. Springer.
- [33] W. Zhang, H. Zhao, and H. Mei. A propositional logic-based method for verification of feature models. In J. Davies, editor, ICFEM 2004, volume 3308, pages 115–130. Springer-Verlag, 2004.
- [34] SMV, <http://www.cs.cmu.edu/~modelcheck/smv.html>, (last accessed on 28/03/2011)
- [35] D. Batory. Feature models, grammars, and propositional formulas. In Software Product Lines Conference, LNCS 3714, pages 7–20, 2005.
- [36] P. Klint. A meta-environment for generating programming environments. ACM Trans. Softw. Eng. Methodol., 2(2):176–201, April 1993.

[37] Karataş, A.S., Oğuztüzün, H., Doğru, A., "Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains", In: Proceedings of the 14th International Software Product Line Conference (SPLC 2010), LNCS Volume 6287/2010, pp. 286-299 (2010)

[38] Backus-Naur Form, http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form,
(last accessed on 28/03/2011)

[39] R. S. Scowen, Extended BNF – A Generic Base Standard (EBNF), ISO-14997

[40] HTML Tutorial, <http://www.w3schools.com/html/default.asp>, (last accessed on 11/04/2011)

[41] Karataş, A.S., Oğuztüzün, H., Doğru, A., "Global Constraints on Feature Models", In: Proceedings of the 16th International Conference on Principles and Practices of Constraint Programming (CP 2010), LNCS Volume 6308/2010, pp. 537-551 (2010)

APPENDIX A

GRAPHICAL XML SCHEMA STRUCTURE FOR THE EXPRESSIONS DEFINED IN CHAPTER 4

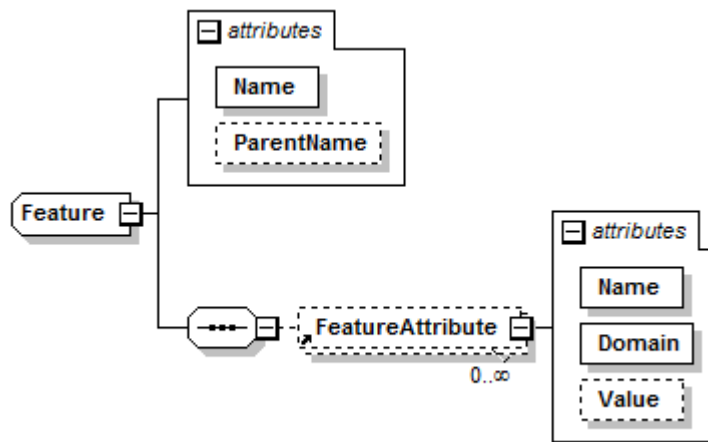


Figure A.1 Graphical XSD expressing feature

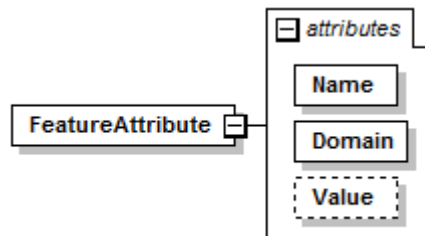


Figure A.2 Graphical XSD expressing feature attribute

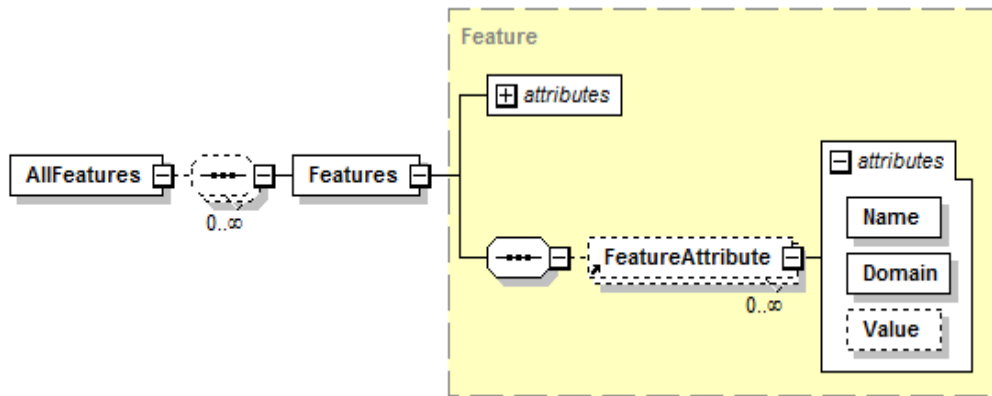


Figure A.3 Graphical XSD expressing All Features

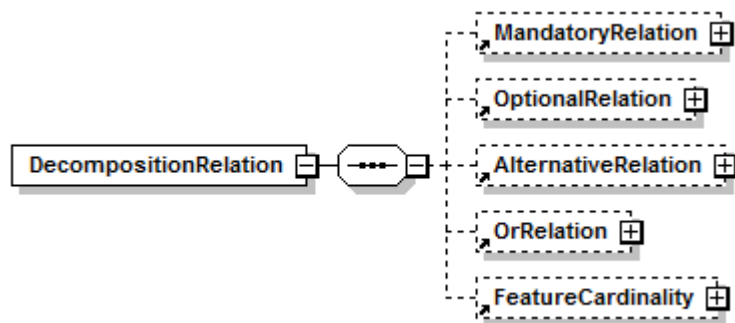


Figure A.4 Graphical XSD expressing Decomposition relation

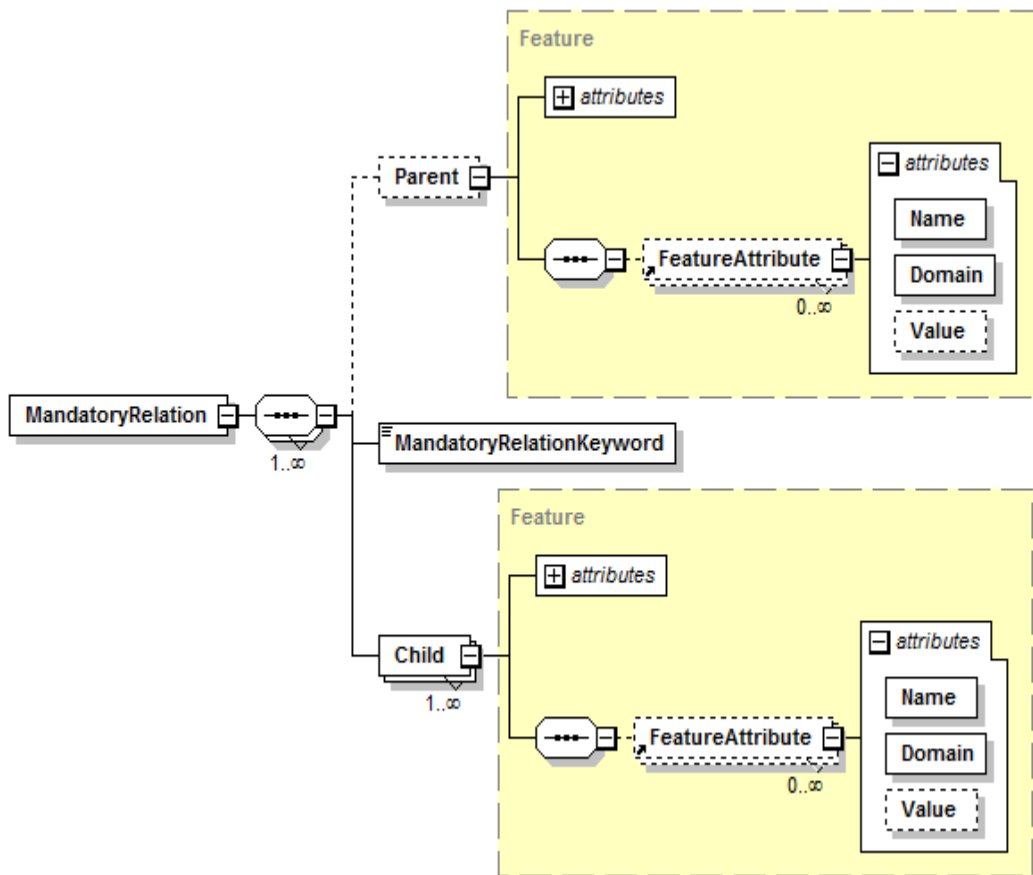


Figure A.5 Graphical XSD expressing Mandatory relation

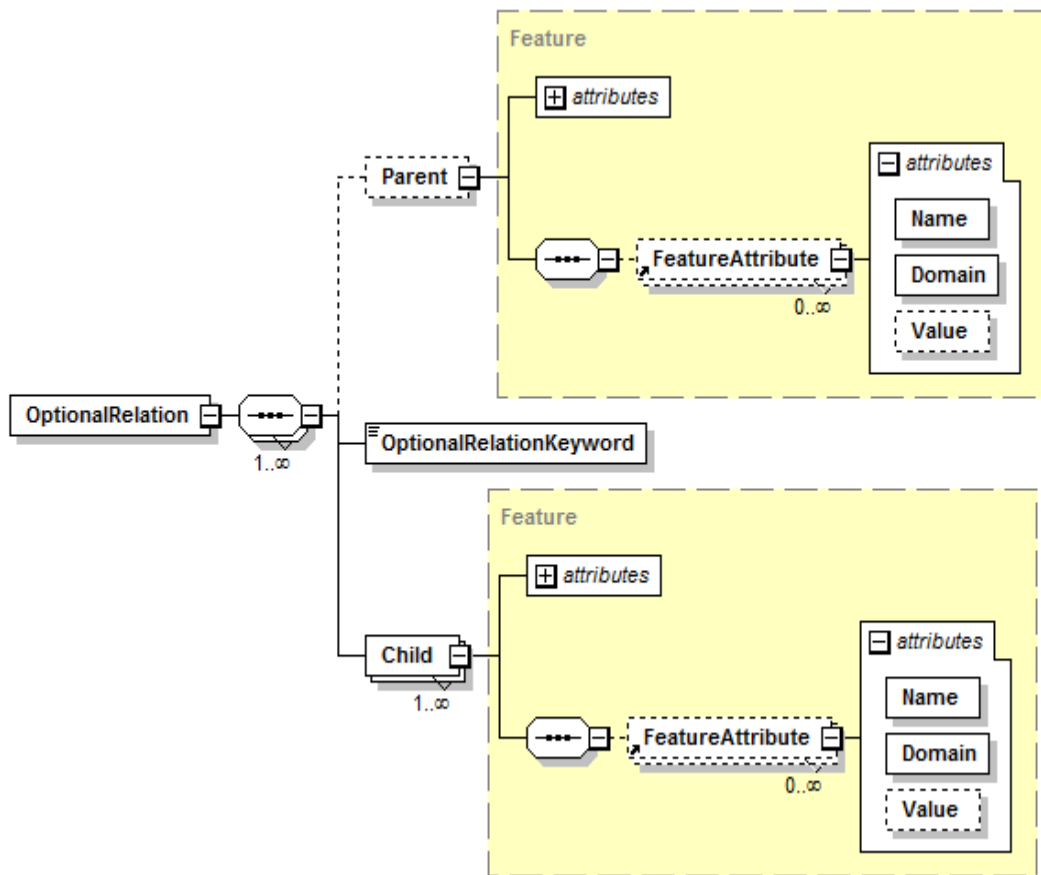


Figure A.6 Graphical XSD expressing Optional relation

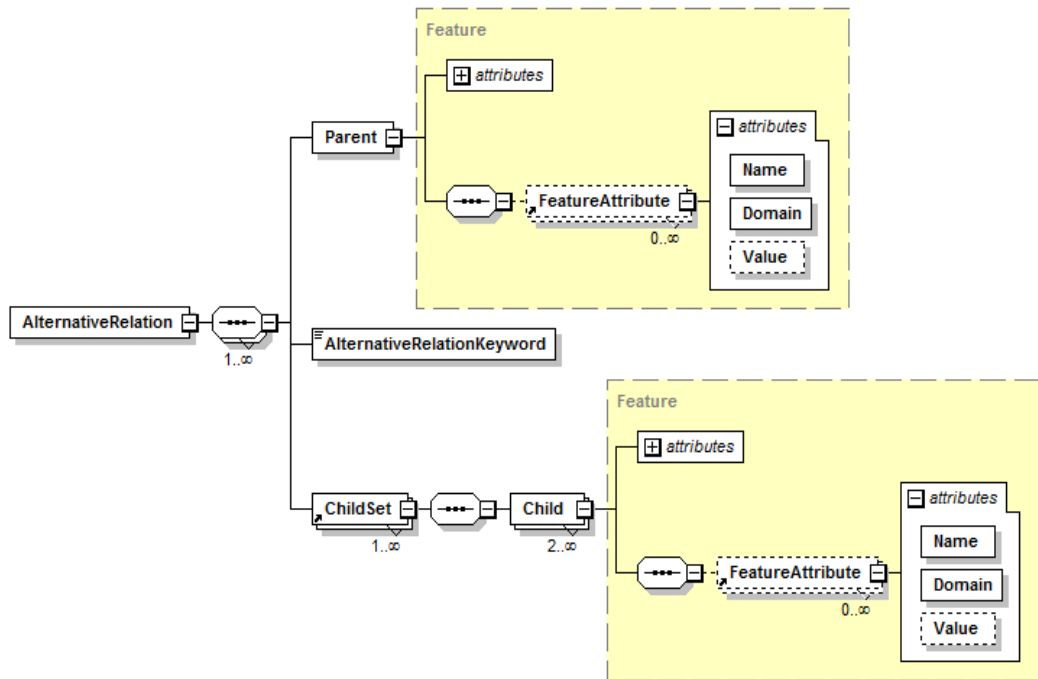


Figure A.7 Graphical XSD expressing Alternative relation

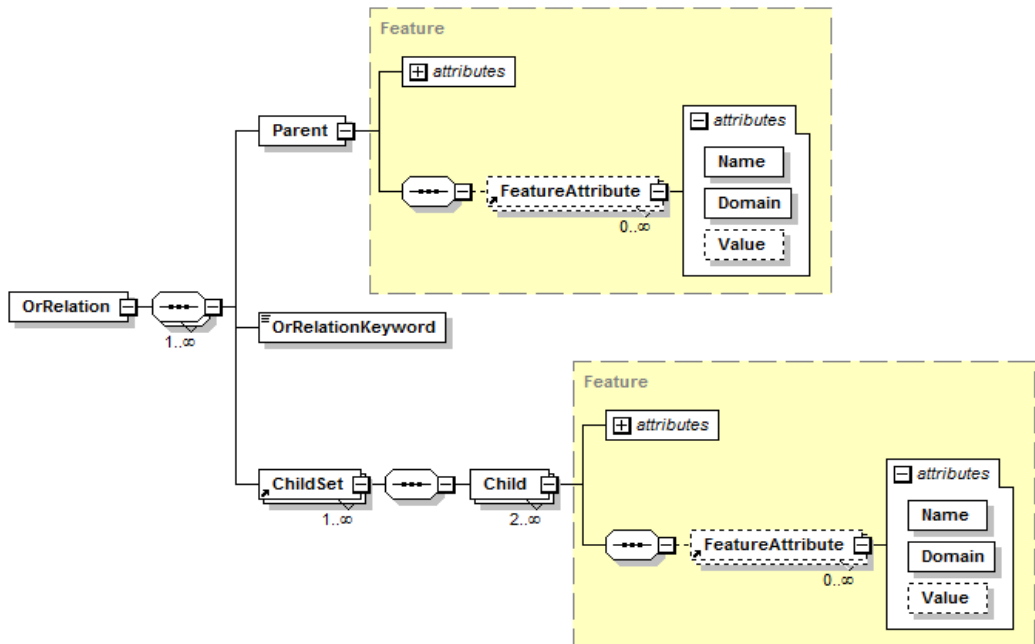


Figure A.8 Graphical XSD expressing Or relation

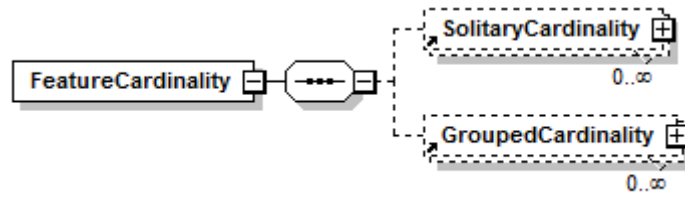


Figure A.9 Graphical XSD expressing Feature Cardinality

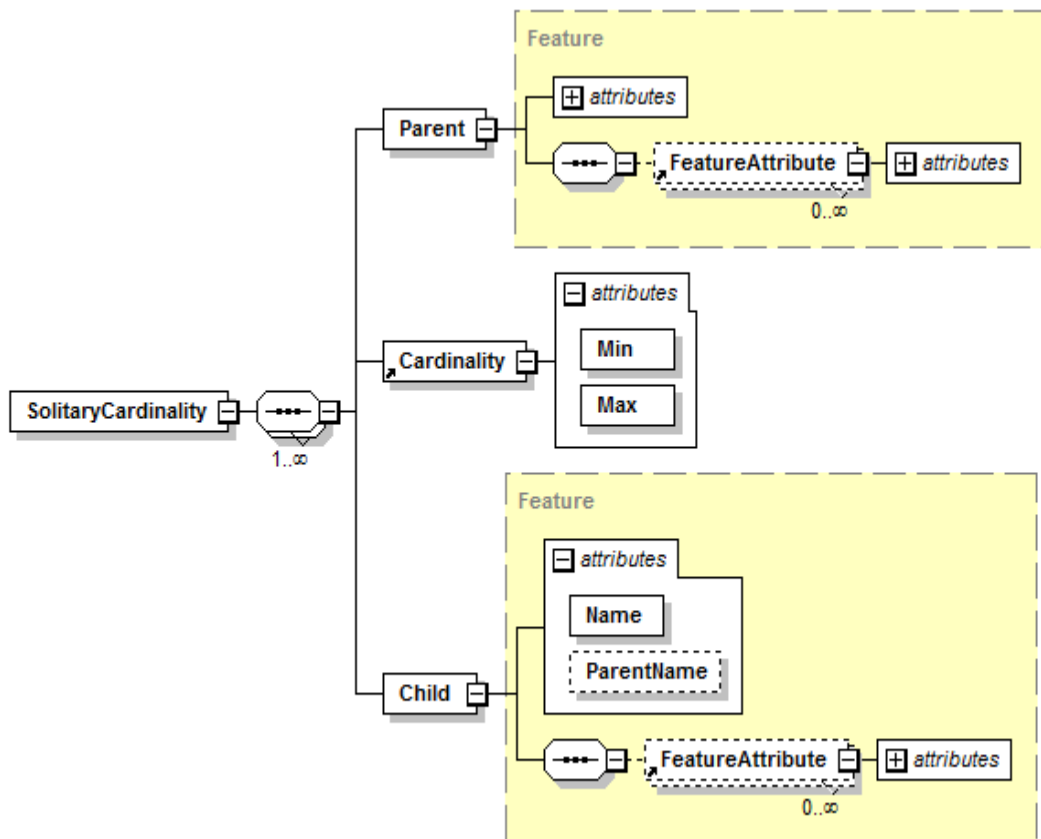


Figure A.10 Graphical XSD expressing Solitary Cardinality

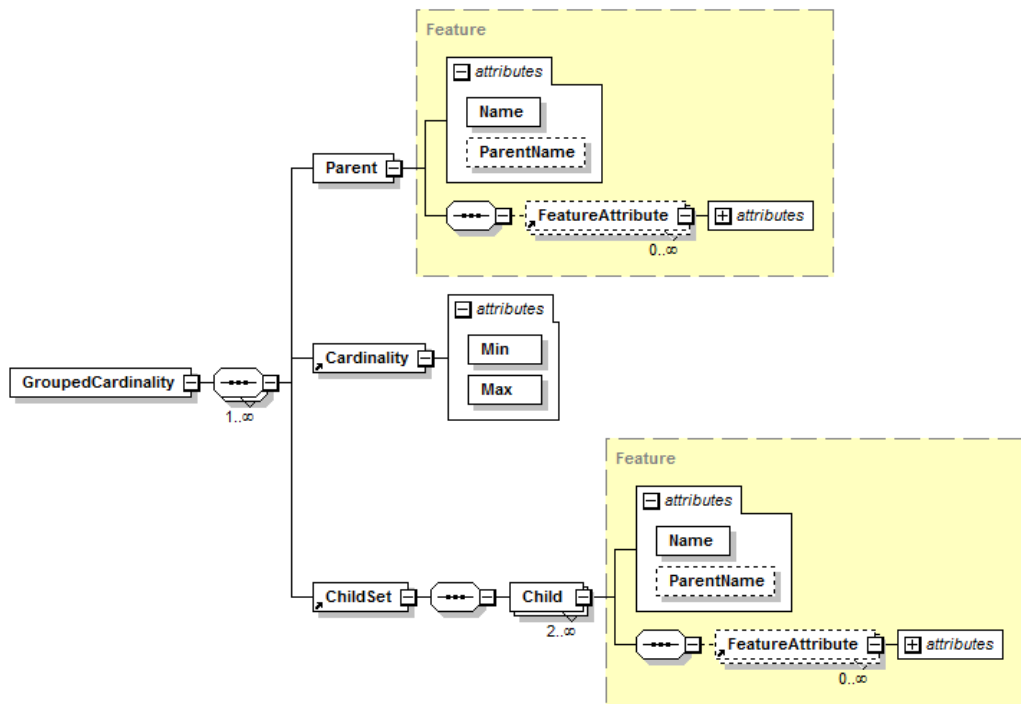


Figure A.11 Graphical XSD expressing Group Cardinality

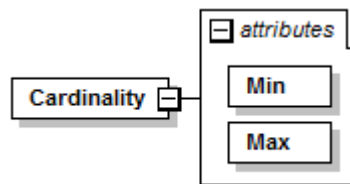


Figure A.12 Graphical XSD expressing Cardinality

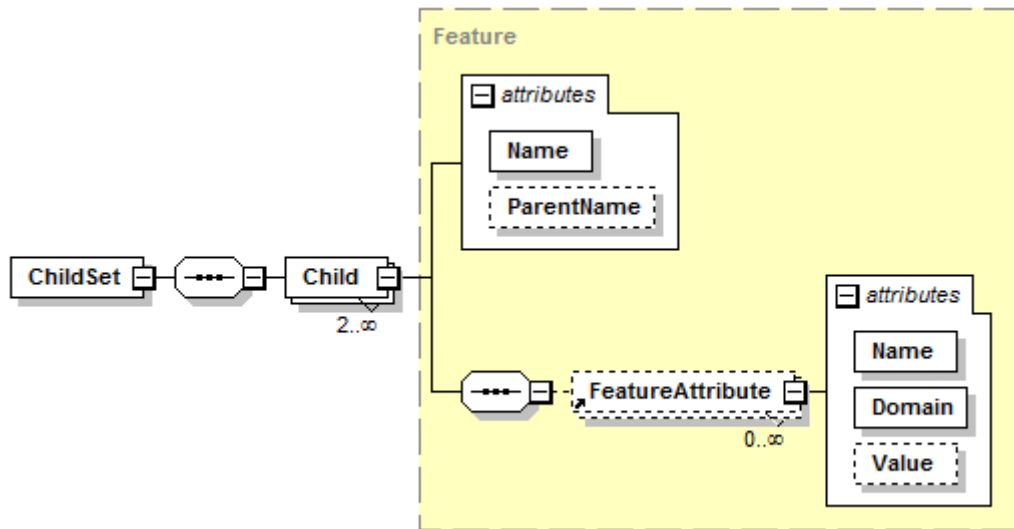


Figure A.13 Graphical XSD expressing Child Set

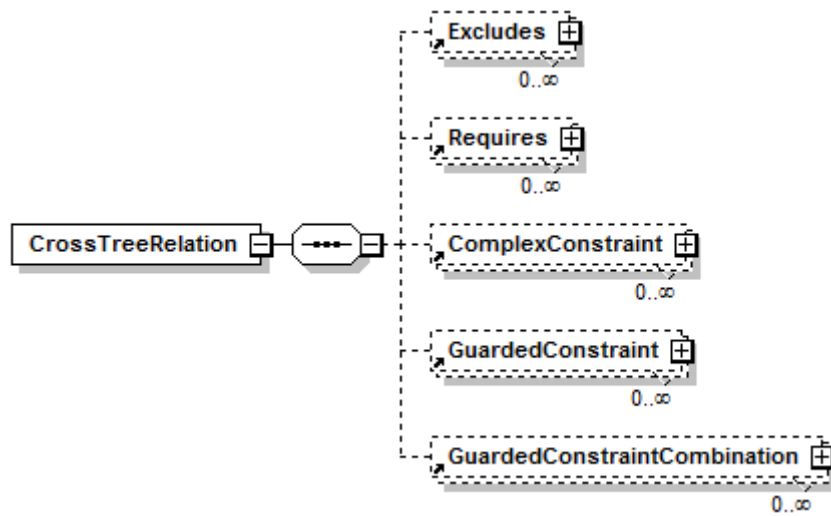


Figure A.14 Graphical XSD expressing Cross Tree relation

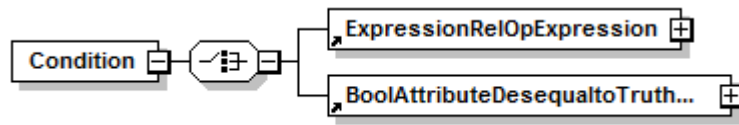


Figure A.15 Graphical XSD expressing Condition

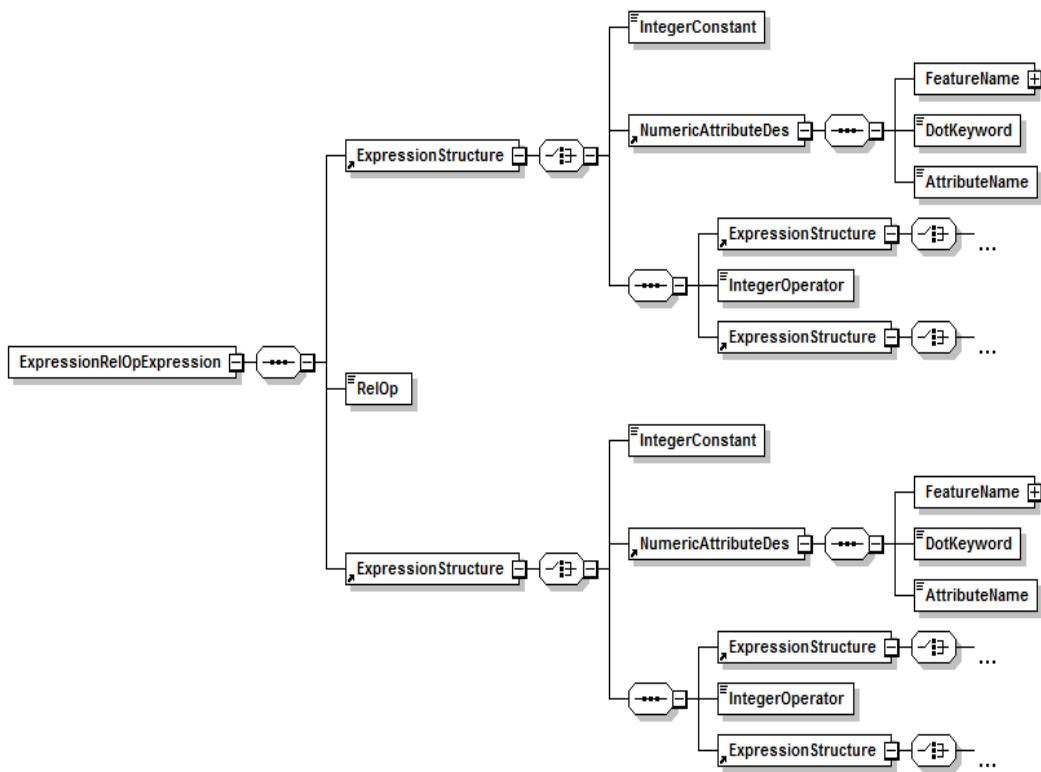


Figure A.16 Graphical XSD expressing Expression RelOp Expression

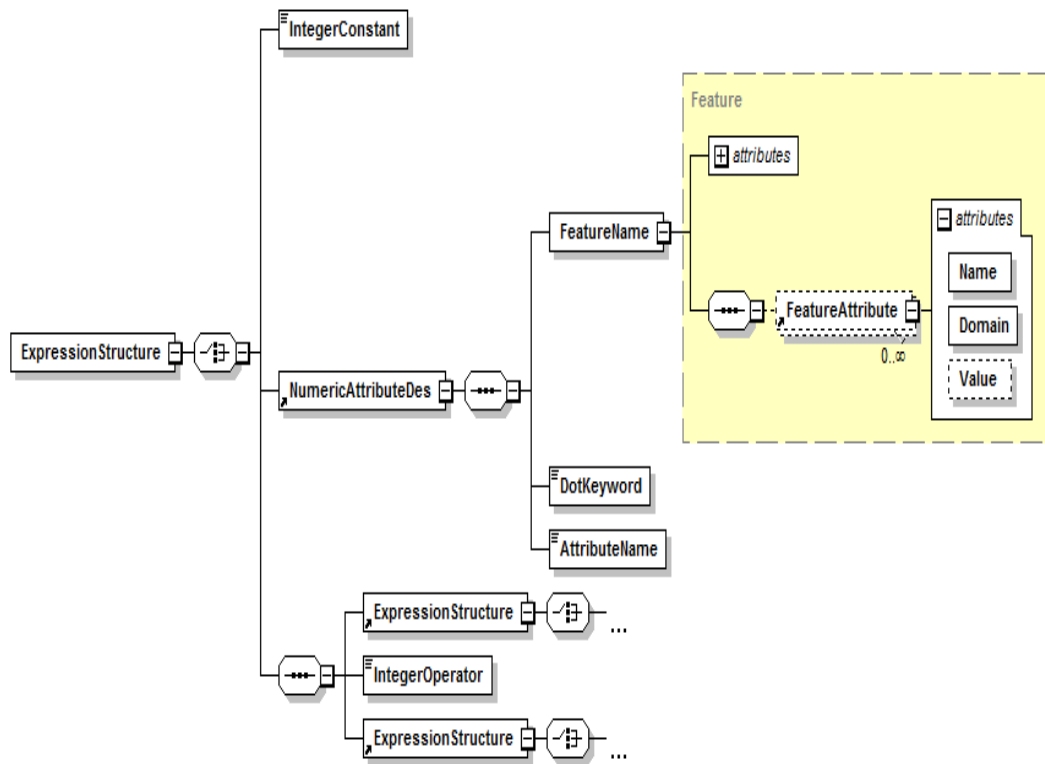


Figure A.17 Graphical XSD expressing Expression Structure

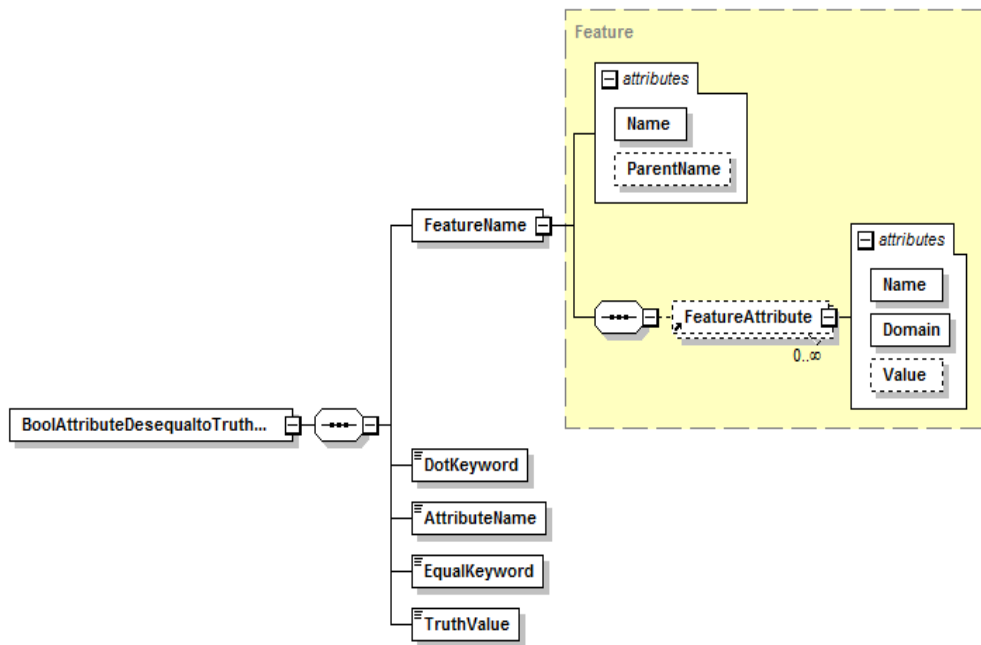


Figure A.18 Graphical XSD expressing BoolAttributeDes=TruthValue

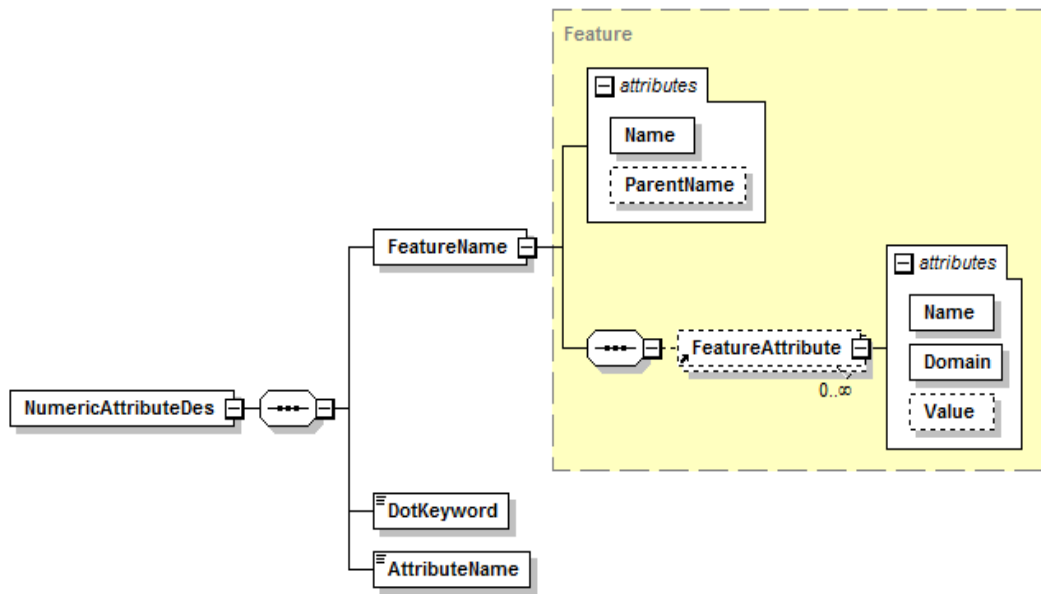


Figure A.19 Graphical XSD expressing Numeric Attribute Designator

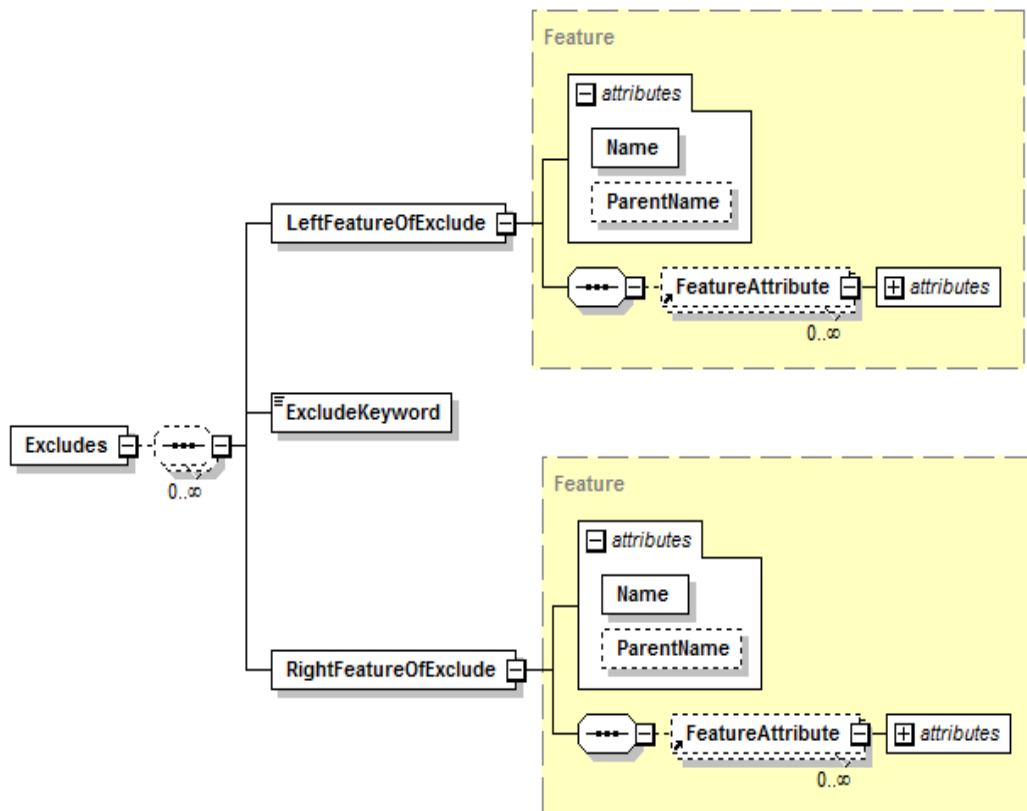


Figure A.20 Graphical XSD expressing Excludes relationship

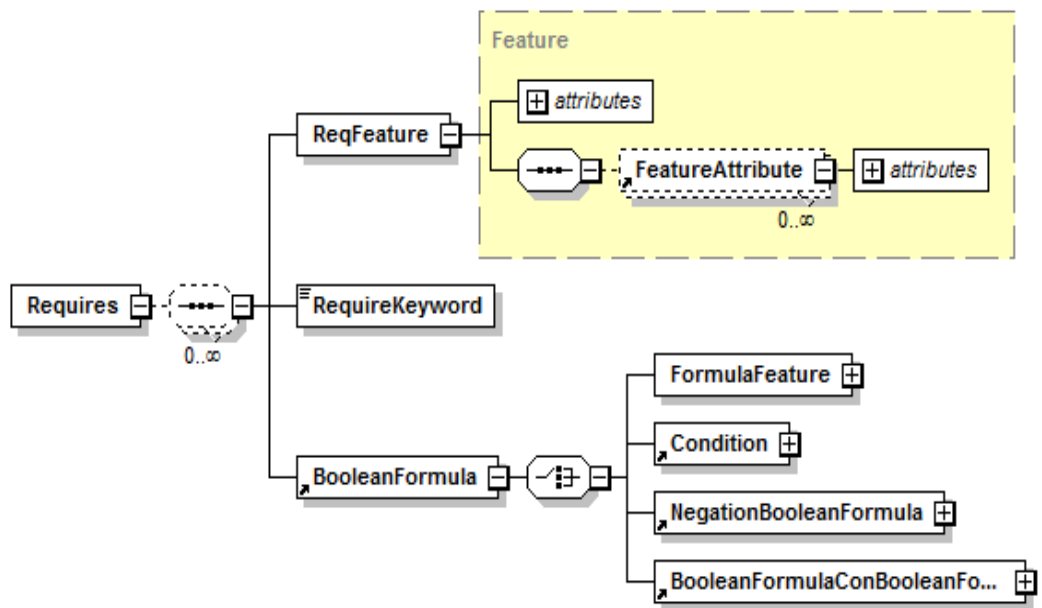


Figure A.21 Graphical XSD expressing Requires relationship

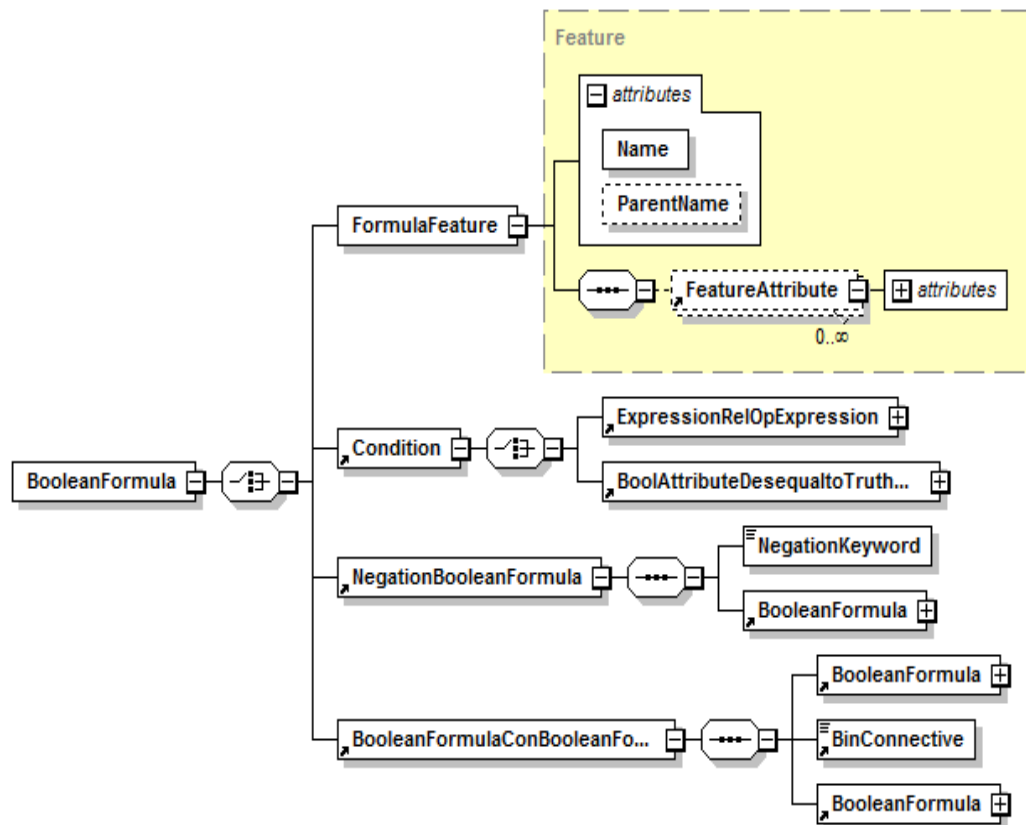


Figure A.22 Graphical XSD expressing Boolean Formula

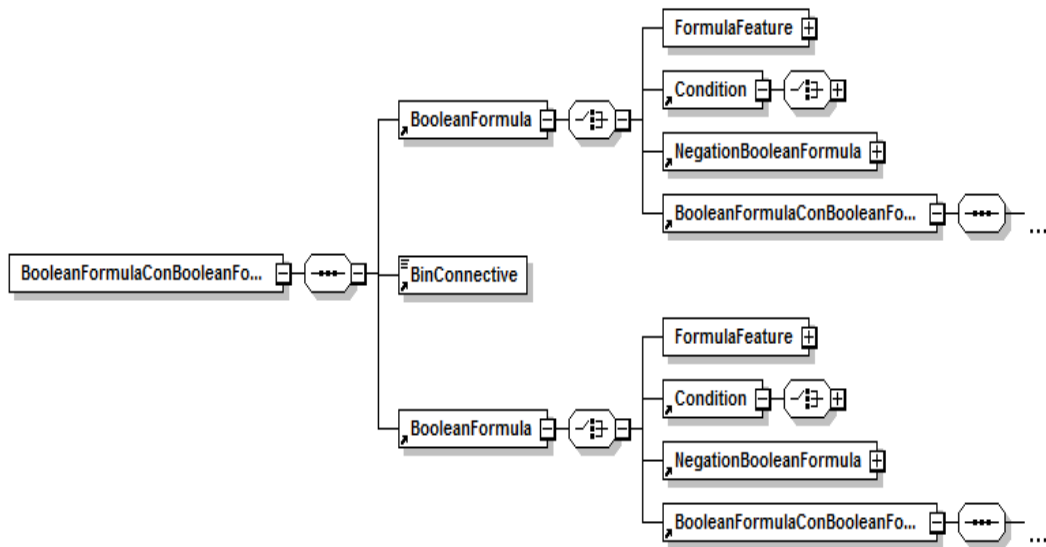


Figure A.23 Graphical XSD expressing BooleanFormulaConBooleanFormula

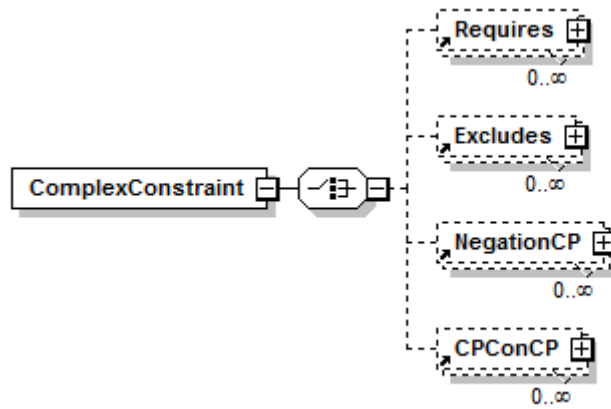


Figure A.24 Graphical XSD expressing Complex Constraint

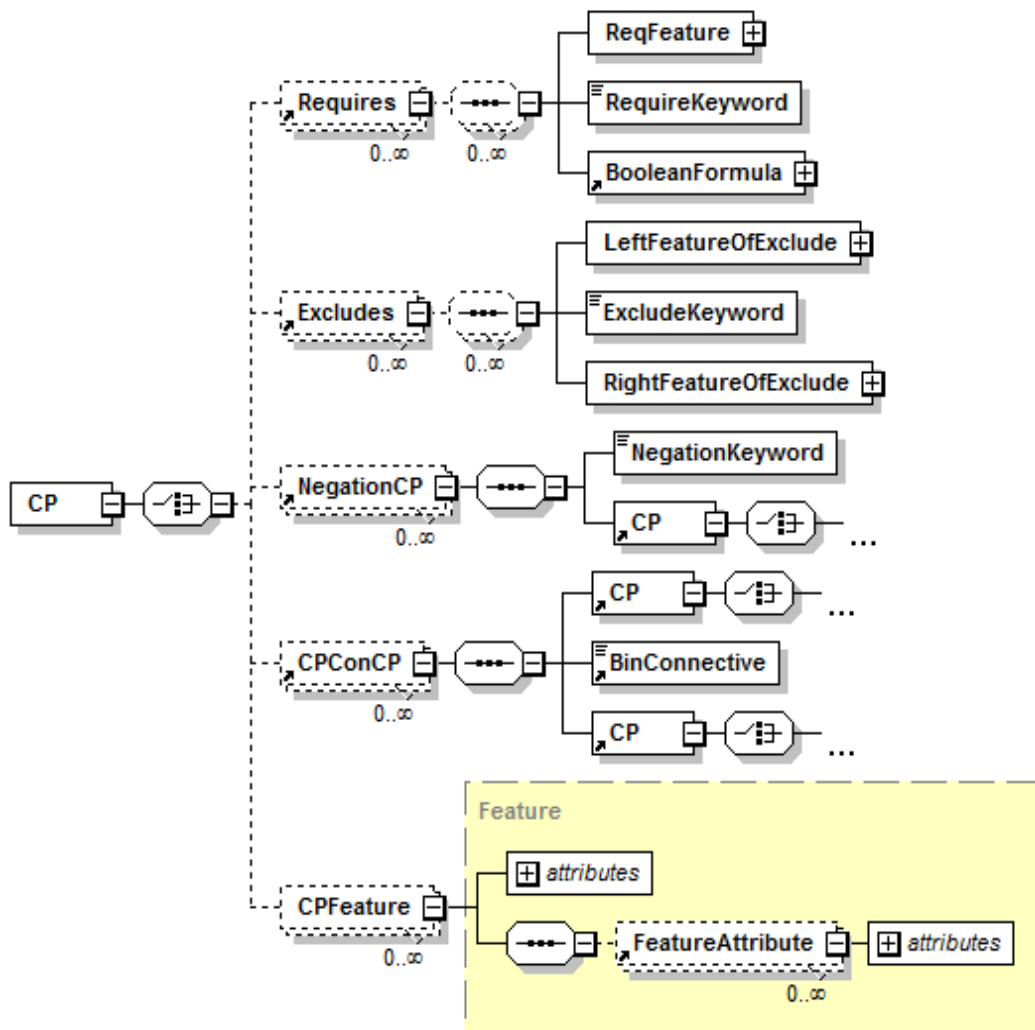


Figure A.25 Graphical XSD expressing CP

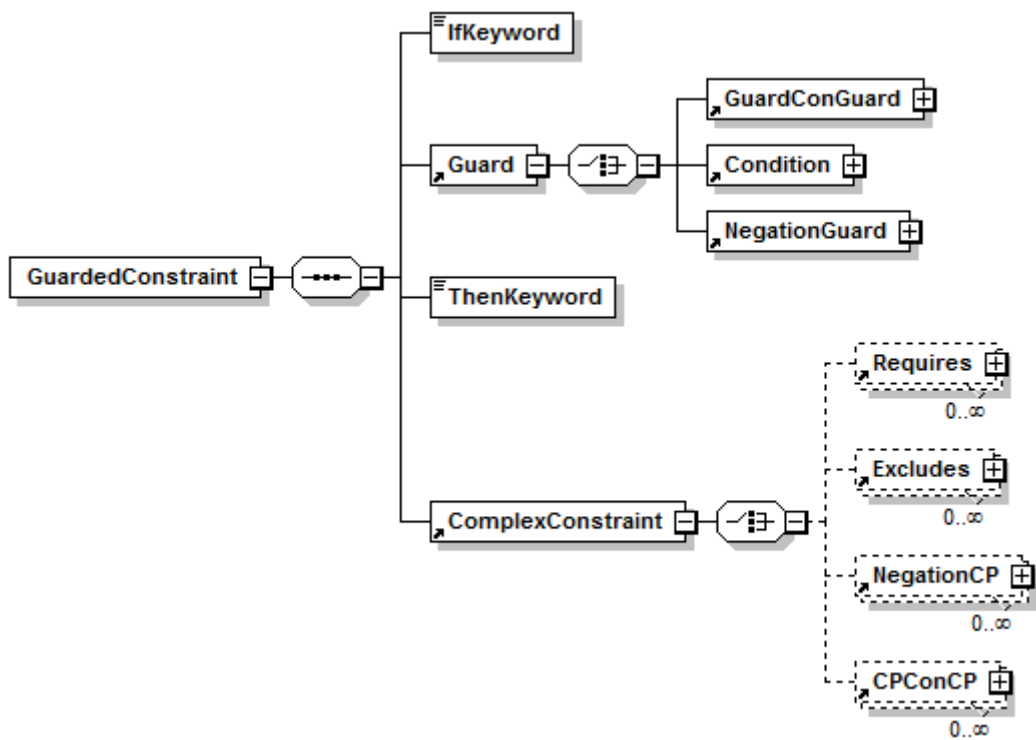


Figure A.26 Graphical XSD expressing Guarded Constraint

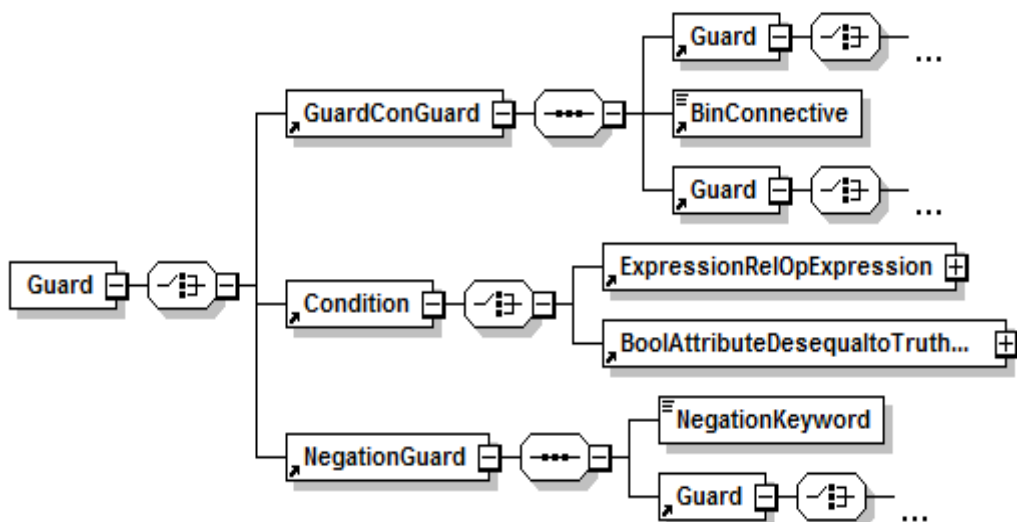


Figure A.27 Graphical XSD expressing Guard



Figure A.28 Graphical XSD expressing Guarded Constraint Combination

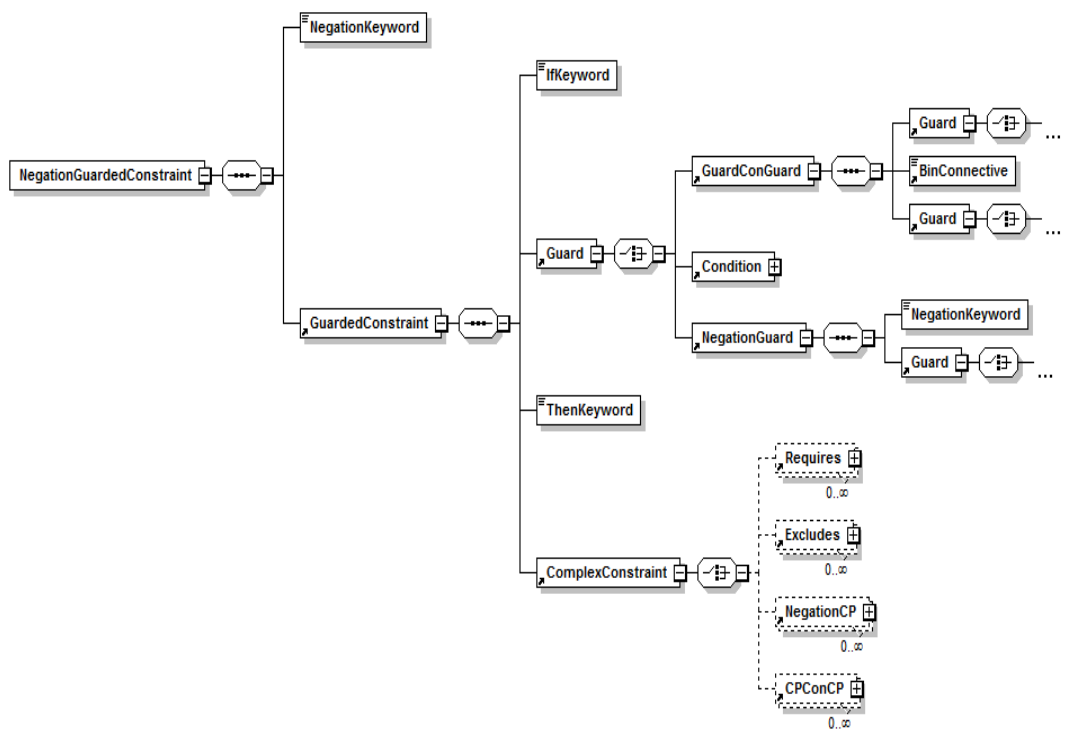


Figure A.29 Graphical XSD expressing Negation Guarded Constraint