# TOWARDS AUTOMATED FEATURE MODEL CONFIGURATION WITH OPTIMIZING THE NON-FUNCTIONAL REQUIREMENTS

by

Samaneh Soltani

B.Sc., Azad University of Mashhad, 2005

A Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science

in the
School of Interactive Arts and Technology
Faculty of Communication, Art and Technology


SIMON FRASER UNIVERSITY
Summer 2012

## APPROVAL

**Name:** Samaneh Soltani

**Degree:** Master of Science

**Title of Thesis:** Towards Automated Feature Model Configuration with Optimizing the Non-Functional Requirements

**Examining Committee:**

**Steve DiPaola**
Associate Professor
Chair

---

**Dr. Marek Hatala**
Senior Supervisor
Associate Professor

---

**Dr. Dragan Gasevic**
Supervisor
Associate Professor, School of Computing and Information Systems, Athabasca University

---

**Dr. Uwe Glasser**
External Examiner
Professor, School of Computing Science

**Date Approved:** _____

## Partial Copyright Licence

# Abstract

A Software Product Line is a family of software systems in a domain, which share some common features but also have significant variabilities. A feature model is a variability modeling artifact, which represents differences among software products with respect to the variability relationships among their features. Having a feature model along with a reference model developed in the domain engineering lifecycle, a concrete product of the family is derived by binding the variation points in the feature model (called configuration process) and by instantiating the reference model. However, feature model configuration is a cumbersome task because of: 1) the large number of features in industrial feature models, which increases the complexity of the configuration process; 2) the positive or negative impact of the features on non-functional properties; and 3) the stakeholders' preferences with respect to the desirable non-functional properties of the final product. Several configuration techniques have already been proposed to facilitate automated product derivation. However, most of the current proposals are not designed to consider stakeholders' preferences and constraints especially with regard to non-functional properties. In this work we address the feature model configuration problem and propose a framework, which employs an artificial intelligence planning technique to automatically select suitable features that satisfy both the functional and non-functional preferences and constraints of stakeholders. We also provide tooling support to facilitate the use of our framework. Our experiments show that despite the complexity involved in the simultaneous consideration of both functional and non-functional properties, our configuration technique is scalable.

*Keywords:* Software product line, feature model, configuration process, planning techniques.

*To my parents and husband for their continual encouragement and support.*

# Acknowledgments

First and foremost I would like to thank my advisers, Dr. Marek Hatala and Dr. Dragan Gasevic for their enthusiastic guidance and support throughout this research and writing of this thesis. Their constant motivation and creative ideas have highly inspired and encouraged me in conducting this research.

I would also like to extend my appreciation to my examining committee member, Dr. Uwe Glasser for his time and constructive and invaluable comments. I am grateful to Dr. Ebrahim Bagheri for his invaluable feedback and comments on this research.

I would also like to thank students of the Ontological Research Lab for the support and ideas they provided, especially I would like to mention Bardia Mohabbati and Melody Siadaty.

My most deepest gratitude must go to my husband, Mohsen Asadi for all his patience, love, support, and encouragement he gave me over the years. Finally, I wish to give my special thanks to my parents for their endless support and encouragement.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

This chapter introduces the research context and research problem. We discuss gaps in the existing research on the feature model configuration and the motivation behind our work. We also explain how we address those gaps and provide an overview of the proposed approach. We conclude this chapter by introducing the structure of the thesis.

## 1.1 Overview of the Research Context

Software Product Lines Engineering (SPLE) aims at developing a set of software systems that share common features and satisfy the requirements of a specific domain [26]. SPLE decreases development costs and time to market, and improves software quality through strategic reuse of assets within a domain of interest. A technique adopted in SPLE for managing reusability is commonality and variability modeling through which common assets and their variabilities are formalized.

A software product line lifecycle encompasses a *domain engineering* process and an *application engineering* process. In the *domain engineering* process, a comprehensive formal representation of the products of the domain is developed. This includes a variability model and the core assets of the product family. *Feature models* are among the prevalent variability modeling techniques in SPLE and represent variability in terms of the differences between the features of the products that belong to a software family. A feature is a logical unit of behavior specified by a set of functional and non-functional requirements [9].

On the other hand, the *application engineering* process is responsible for capturing the requirements of the target application, deriving a concrete product from the variability

model through a configuration process, and deploying the product into users' environment [30]. Using feature models as variability modeling tools, the configuration process selects a suitable set of features to satisfy the stakeholders' requirements.

## 1.2 Motivation and Research Problem

The focus of existing research in software product line has been mostly on modeling and managing product lines, while there are only a few works which in particular concentrate on effectively utilizing product lines. Although there are a number of algorithms for product line configuration [16][8][61], there is very little coverage of non-functional requirements in them. This lack of available approaches and tools for optimizing feature model configuration with respect to non-functional requirements builds the motivation behind our research.

Selecting a proper set of features among a feature model is a complex and time consuming task because:

- there are several types of relations and integrity constraints between the features.
- the number of possible configurations has exponential growth. Even in small feature models the number of possible configurations can be very high. Industrial feature models may consist of hundreds of features which increases the complexity of feature model configuration.
- features may have either positive or negative impact on the different business concerns of a product, and hence expose different quality attributes. We refer to business concerns of a product (e.g., *security* and *customer satisfaction*) and quality attributes of a product (e.g., *performance* and *cost*) as non-functional properties (NFP). For example, a feature may have a negative impact on *security*, but a positive impact on *customer satisfaction* or it could have *high performance* but *low reliability*.
- in addition to functional requirements, stakeholders may have several constraints and preferences over non-functional properties in the product derivation. For example, one stakeholder may ask for a product with *high security*, *high customer satisfaction*, and specific amount of *cost*; and can mention that the *customer satisfaction* is more important than *security*; which would make the configuration process more difficult and complex.

The aforementioned difficulties in the configuration process strongly motivated us to address the following research question:

*How can a product be automatically derived from a feature model in such a way that it satisfies stakeholders' requested functionality and at the same time optimizes their preferences and requirements over the non-functional properties?*

## 1.3 Overview of Proposed Approach

To overcome the complexity of feature model configuration we developed an automatic method to select the proper set of features fulfilling stakeholders' functional and non functional requirements and preferences. To this end, we looked into preference-based planning techniques. Various preference-based planning techniques exist that produce a plan by optimizing a set of given preferences [51]. Hierarchical Task Network (HTN) planning is a popular planning technique, which is suited for domains with hierarchical task decomposition [34]. The HTN Planning technique generates plans from a developed hierarchical network of domain tasks and actions [51]. That is, having modeled tasks, actions, and their constraints in the HTN formalism, HTN planners produce a sequence of actions that perform some given tasks.

By way of analogy between the feature model configuration process and the HTN planning problem, we were motivated to investigate the applicability of HTN planning for the product line configuration problem. Hence, we hypothesized that HTN planning can form the basis for a configuration technique that can answer our research question. Consequently, we proposed and developed a framework, similar to [50], which extends feature models with annotations reflecting different non-functional properties. We then used HTN planning [35] to select a set of features which: 1) satisfy the stakeholders' functional requirements; and 2) optimize the non-functional requirements and preferences of the stakeholders.

The general overview of the proposed approach is illustrated in Figure 1.1. As shown in the figure, our approach captures functional requirements and non-functional requirements of stakeholders for final application. Non-functional requirements are captured in terms of relative importance of non-functional properties along with constraints over the non-functional properties (Chapter 2.1.4). Then, we employed an extension of Analytical Hierarchy Process (AHP)[45] algorithm, called Stratified-Analytical Hierarchy Process (S-AHP) to calculate the ranks of the features based on the relative importance of non-functional

properties assigned to the features (Chapter 3.2.1). Next, we generated the HTN planning domain and problem from the feature model and stakeholders' requirements (Chapter 3.3).

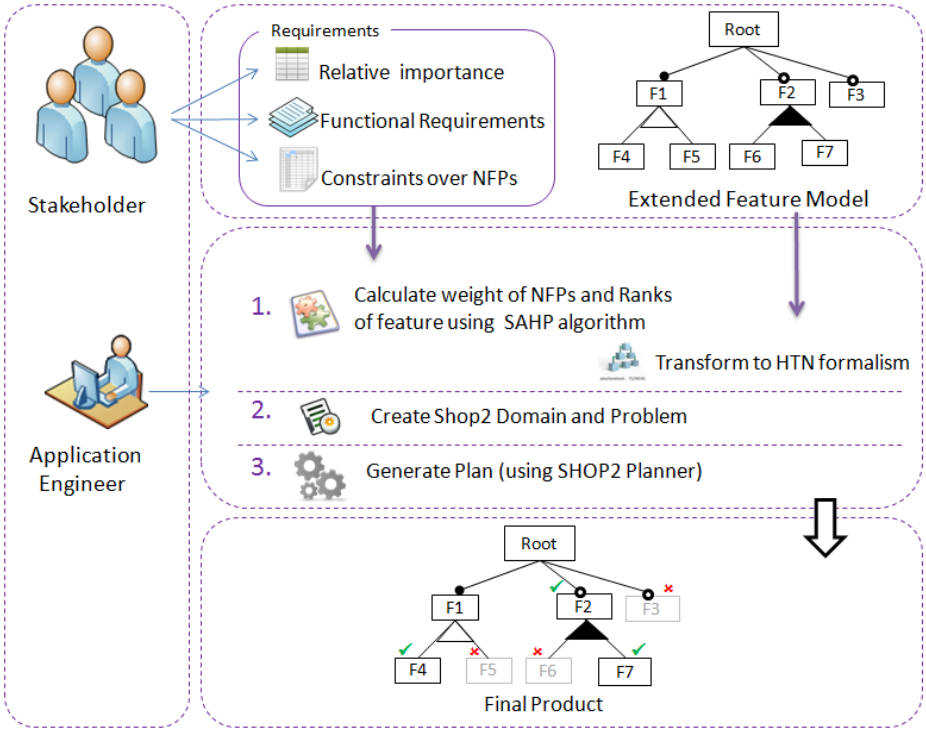



Figure 1.1: Automatic feature model configuration process in application engineering phase

We apply the SHOP2 planner [35], an HTN-based planning system widely used for planning problems, to identify an optimal plan (i.e., a plan with the best overall cost) To produce the final configuration, the features chosen by the SHOP2 planner are selected and represented in the visual view.

In comparison to existing configuration approaches, our approach has the benefit of not only satisfying the structural and syntactic constraints of feature models during the configuration process, but also taking both qualitative and quantitative NFPs as well as the relative importance of NFPs into account.

In general, the main contributions of this thesis are as follows:

- Modeling non-functional properties in the context of software product lines; and extending feature models to cover NFPs as well as functional properties.

- Introducing an easy-to-understand formalism for capturing the stakeholders' preferences over non-functional properties represented in terms of relative importance; and utilizing Stratified Analytic Hierarchy Process (SAHP) to calculate the weight of NFPs.
- Producing an optimal feature model configuration, by transforming a feature model and stakeholders' preferences and constraints into a planning domain and problem by considering both functional and non-functional requirements.
- Developing a tool to support the proposed approach for the feature model configuration.
- Conducting several experiments to evaluate the performance and usefulness of the proposed approach and the developed tool.

## 1.4 Dissertation Organization

1. **Introduction**. This chapter introduces this thesis and presents an overview of the research context, research problem and the motivations of our research as well as a brief introduction to our proposed approach.
2. **Literature Review**. This chapter introduces the fundamentals and basic domain concepts including software product line, feature models, non-functional properties, feature model configuration process, and hierarchical task network. It also presents a detailed overview of existing related research in this domain.
3. **Automated Feature Model Configuration**. The details of the proposed approach along with underlying concepts are explained in this chapter.
4. **Tooling Support**. This chapter introduces the developed tool and explains its characteristics.
5. **Result and Evaluation**. This chapter evaluates the proposed approach and developed tool in terms of scalability and effectiveness. It also compares our approach with other existing approaches.
6. **Conclusions and future work**. This chapter concludes the thesis and mentions the open issues and future works.

# Chapter 2

# Literature Review

In this chapter, we describe the fundamentals and basic domain concepts which are used in the thesis. We first briefly introduce software product lines and feature models as well as the feature model configuration process. Next, we explain the hierarchical task network planning and its related concepts. Finally, we terminate the chapter by providing an overview of the existing researches and works relevant to the feature model configuration.

## 2.1 Background

This section introduces the research context including: software product lines, feature models, non-functional requirements, feature model configuration process, and hierarchical task network planning.

### 2.1.1 Software Product Line

Mass customization is defined as "a large-scale production of products tailored to individual customers' needs" [30]. The main precondition for the success in mass-customization is flexibility - adapt products to fit into different requirements [30]. Moreover, large-scale reuse is another success key in mass-customization. Software product line approaches facilitate the mass-customization through providing variability mechanisms for flexibility and core assets for reusability.

Software product line can be defined as "a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market

segment or mission and that are developed from a common set of core assets in a prescribed way" [14]. The fundamental difference between the software product line and traditional software development is shift from individual software and system to product line [39]. In software product line we have a view of domain of interest as a family of products; each product has some feasible features and properties of a domain and fulfills a specific purpose in that domain. Software product line by aid of reuse assets and components accelerates the development of software systems. Therefore, software product line can be easily changed in order to cope with growing technologies and new needs. For example, CelsiusTech applied the software product line engineering paradigm to develop naval control systems named ShipSystem 2000. CelsiusTech could quickly develop the new avionic systems because it reused 40% of its code directly from Ship System 2000 [39].

Product line notion can be traced back to Parnas' work [37] (called product families) in 1970 which aimed at managing non-functional variability in software systems. Kang et al. [26] fully introduced the product line notion in 1990 by developing Feature Oriented Domain Analysis method (FODA). Afterwards, many industries and academic researchers in both USA and Europe have contributed in the maturity of software product line engineering. Offering improvement in both process oriented aspects (such as cost and time-to-market) and product quality leads companies to adapt software product line for developing software [39]. For an example, by using software product line techniques, Nokia overcame its main challenge (i.e., being the high pace of market demand and customer taste change) and increased its production capacity for new cellular phone models from 5-10 to around 30 models per year [14]. Reducing development cost and time to market are achieved by large-scale reuse during the development of software in the software product line.

There are basically two lifecycles in the software product line engineering, namely *Domain engineering* and *Application engineering* [30]. *Domain engineering* (development for reuse) is concerned with the process of understanding the target domain and developing a relevant, suitable, and comprehensive formal representation of the concepts in that domain. Domain engineering process contains five key sub-processes including [39]:

- **Product management:** This phase determines the scopes of product lines and identifies benefits and costs of applying software product line. At the end, a product line roadmap which contains variable and common features of the product line is generated.
- **Domain requirements engineering:** This phase analyzes, captures, models, and

verifies the domain requirements. Additionally, commonality and variability between products are identified and modeled using variability modeling languages.

- **Domain design:** According to the requirements of a domain, a reference architecture of the product line along with reusable assets are developed during this phase. The variability model is refined and enriched with design variability options.
- **Domain realization:** This phase implements the reusable assets in a domain and enriches the variability model with the implementation variability.
- **Domain testing:** This phase, taking into account the domain requirements and the reusable assets, validates and verifies the reusable assets.

On the other hand, *application engineering* receives the reusable artifacts developed in the domain engineering process and creates an appropriate application instance for given requirements. The application is produced by carefully choosing and instantiating the right elements of the formal representation of the domain model [4]. The application engineering process consists of four sub-processes [39]:

- **Application requirements engineering:** This phase encompasses actvities for capturing, modeling, and validating the requirements of a specific application. To develop the application requirements, the domain requirements model is utilized.
- **Application design:** This phase resolves the variability of the reference architecture based on the specific application requirements and instantiates an application architecture.
- **Application Realization:** This phase creates the final application through instantiating and adapting reusable assets to fit them to application requirements.
- **Application testing:** This phase validates and verifies the final application against its requirements.

### 2.1.2 Feature Model

In Software product line Engineering (SPLE), a feature model is used mainly for representing commonality and variability between products. A feature model is created during the domain requirements engineering process by modeling common and variant features among

products in a specific domain. Later on in application engineering process, the feature model is configured according to the requirements of a target application. The notion of feature model was introduced by Kang et al. in the Feature Oriented Domain Analysis (FODA) report back in 1990 [26]. In this work, products in the product line are differentiated by their features. The features are defined as important distinguishing aspects, qualities, or characteristics of a family of systems [28]. Other researchers proposed different definitions for features including:

- Kang et al. [27]: "a distinctively identifiable functional abstraction that must be implemented, tested, delivered, and maintained";
- Czarnecki et al. [17]: "a distinguishable characteristic of a concept (e.g., system, component, and so on) that is relevant to some stakeholder of the concept";
- Bosch et al. [9]: "a logical unit of behavior specified by a set of functional and non-functional requirements";
- Batory et al. [6]: "a product characteristic that is used in distinguishing programs within a family of related programs";
- Zave et al. [65]: "an optional or incremental unit of functionality".

Among aforementioned definitions, Bosch's definition covers both functional and non-functional aspects. Hence, we consider this definition as we intend to configure the feature model based on both functional and non-functional requirements.

A feature model provides a formal and graphical representation of features as well as the variability relations, constraints, and dependencies defined over the product lines' features. It has a tree-like structure [17] in which features are typically classified as:

- **Mandatory feature:** if a parent feature is selected, its mandatory child feature must be also selected in the configuration process (Figure 2.1(a)).
- **Optional feature:** if a parent feature is selected, its optional child feature may or may not be selected in the configuration process (Figure 2.1(b)).
- **OR feature group:** one or more features in the OR feature group must be selected in the final configuration of the feature model(Figure 2.1(c)).

- **XOR feature group:** one and only one of the features in the XOR feature group must be selected in the final configuration of the feature model(Figure 2.1(d)).

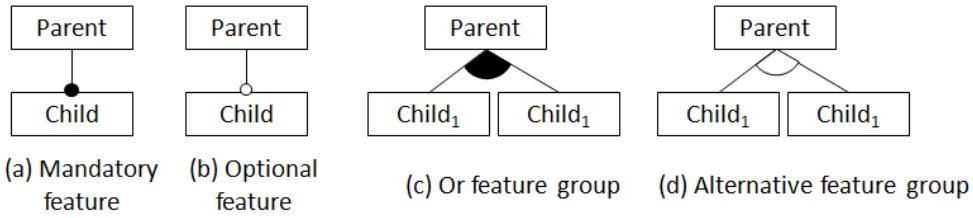



Figure 2.1: Different types of features in feature model

In addition to the relations between a parent feature and its child features, a number of relations are defined to represent mutual inter-dependencies (also referred to as *integrity constraints*) between features. The two most widely used integrity constraints are [17]: *requires*-the presence of a given (set of) feature(s) requires the inclusion of another (set of) feature(s); and *excludes*-the presence of a given (set of) feature(s) requires the exclusion of another (set of) feature(s). For example, if "feature A requires feature B" and feature A is selected then feature B has to be selected as well. Opposite to this, if "feature A excludes feature B" and feature A is selected then feature B has to be unselected. In feature model tree, features without any children called *atomic* (or leaf) features and features which are decomposed into sub-feature(s) called *non-atomic* (or intermediate) features.

Feature models address *commonality* through core features - mandatory features whose parents are all mandatory as well - and *variability* through variability relations described above. It models all possible products of a software system in a specific domain. In contrast with traditional information models which represent a single product in a model , feature models represent a family of products in the same model. A feature model can be formally represented as follows:

**Definition 2.1.1** ***(Feature model).*** *A feature model is a sextuplet* $FM = (F, F_O, F_M, F_{IOR}, F_{XOR}, F_{ic})$ *where 1)* $F$ *is a set of features; 2)* $F_O \subseteq F \times F$ *is a set of parent and optional child feature pairs; 3)* $F_M \subseteq F \times F$ *is a set of parent and mandatory child feature pairs; 4)* $F_{IOR} \subseteq F \times P(F)$ *and* $F_{XOR} \subseteq F \times P(F)$ *are sets of pairs of child features and their common parent feature* $P(F)$ *grouping the child features into or and alternative groups, respectively; 5)* $F_{ic} \subseteq F \times F$ *is a set of integrity constraints (i.e., requires or excludes).*

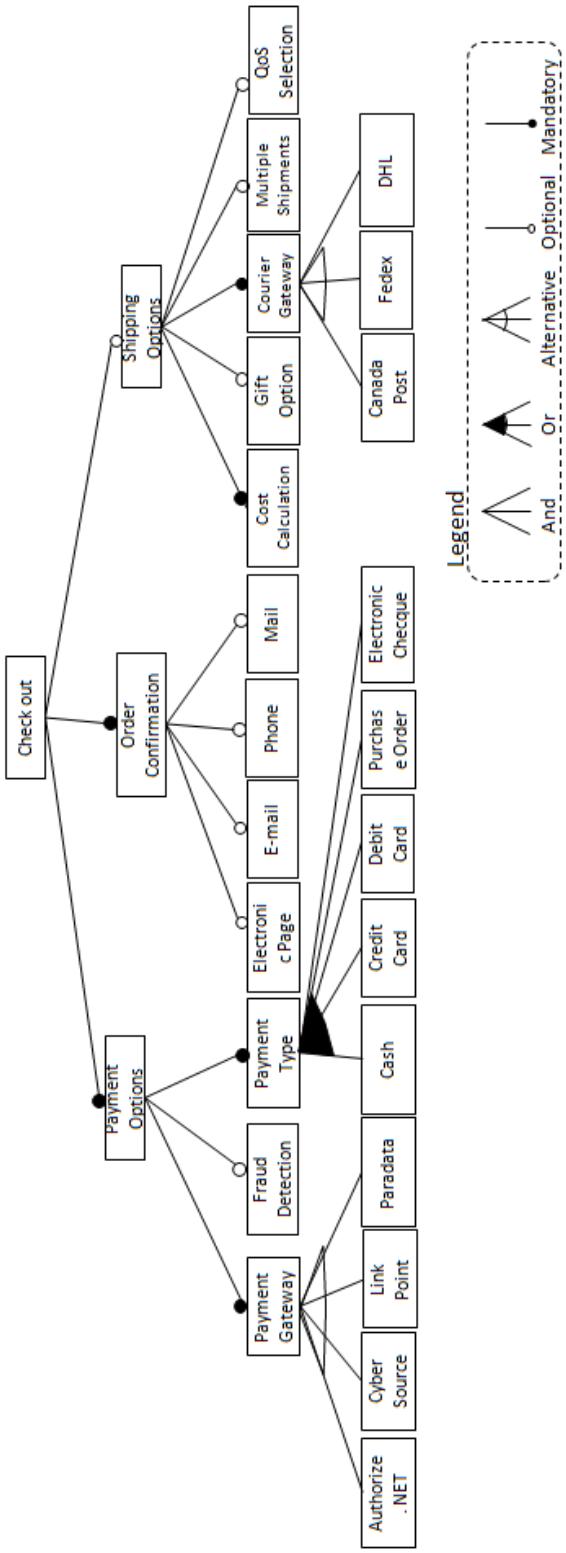



Figure 2.2: A subset of on-line shopping systems feature model

Figure 2.2 depicts a Check out feature model - a part of on-line shopping feature model. This example is taken from SPLOT feature model repository[1] which is used extensively for research works. It consists of three main features: Payment Options; Order Confirmation; and Shipping. Features Payment Option and Order Confirmation are *mandatory* features. The Shipping feature is *optional* and may or may not be selected for the products containing the Checkout feature. That is, all on-line shopping systems containing check out feature must have payment and order confirmation features and may or may not have shipping feature. The Payment Type feature is divided into five *Or-features*, namely Credit Card, Debit Card, Cash, Purchase Order, and Electronic Cheque. Hence, for products having payment feature, one or more payment type(s) (including credit card, debit card, cash, purchase order, and electronic cheque) should be selected. Courier Gateway is divided into three *alternative features* - Canada Post, UPS, and Fedex. That is, for products having shipping feature, one and only one courier gateway (including Canada Post, UPS, and Fedex) must be selected.

Payment Options, Order Confirmation, Payment Gateway, and Payment Type are the *core* features in the feature model. In other words, all on-line shopping systems containing check out feature contain Payment options, Order confirmation, Payment gateway, and Payment type features. Features Electronic Page, Email, Phone, and Mail are the examples of atomic features; and features Payment Type, Payment Gateway, and Shipping Options are the examples of non-atomic(or intermediate) features

### 2.1.3 Non-functional Properties

Handling non-functional requirements (NFRs) has been a major challenge in both requirements engineering and software engineering. Requirements engineering is concerned with capturing and modeling non-functional requirements which refer to constraints over the non-functional properties of a target system. NFRs are essential and critical for the success of a project [22] and neglecting NFRs may lead to serious project failures [31].

Unlike existing consensus on the definition of functional requirements, several different definitions have been given in the literature for non-functional requirements. Glinz [22] analyzed these definitions and found out not only terminological, but also major conceptual discrepancies. In [31], Mairiza and Zowghi performed a systematic analysis of the notion of

[1]SPLOT - Software Product Line On-line Tools, http://www.splot-research.org

non-functional requirements and came up with two main classes of definitions: 1) requirements that describe the properties, characteristics or constraints that a software product must exhibit; and 2) requirements which describe the quality attributes of the software product. According to the first group, non-functional requirements describe any requirements which are not functionality of the system. This includes several aspects such as business rules, development constraints, external interfaces, and quality attributes [31]. On the other hand, the second perspective covers a narrow range of non-functional requirements, (i.e., quality attributes) as a subset of the first group.

From the software engineering point of view, during design and implementation phases, we are concerned with either adding functionalities to realize non-functional requirements or defining constraints over the non-functional properties of a software based on the non-functional requirements. For example, in order to realize high security non-functional requirement, we may need to add a number of functionalities such as login management system or verification components into design and implementation models. Additionally, we can define non-functional properties (also known as quality attributes) for different elements of design and implementation and limit their possible values based on requirements. For example, the response time property may be defined for different functional elements of the system and the property values are limited to a certain interval based on the expected requirements.

### 2.1.4 Stakeholders' Preferences and Constraints

Various stakeholders may have different priorities over non-functional properties. For some stakeholders, a subset of non-functional properties may be more important and relevant than the other non-functional properties. Moreover, in some situations, a conflict may arise between requirements defined over non-functional properties that have an opposite behavior, in case of which the stakeholder needs to choose between the competing options. One approach for specifying the priority between non-functional properties is through formalizing their relative importance relations. Relative importance is defined as follow [36].

**Definition 2.1.2** ***(Relative Importance)***. *Relative importance between non-functional properties $a$ and $b$ is: $a \succ^{\alpha} b$ if non-functional property $a$ is more important than non-functional property $b$ with coefficient $\alpha$.*

Usually, the degree of importance of options (non-functional properties) is represented using values 1, 3, 5, 7, and 9 corresponding to equality, slight value, strong value, very strong and extreme value, respectively [44]. For example, relation *Security* $\succ^3$ *Performance* represents that *security* is slightly more important than *performance*. We refer to the relative importance between non-functional properties as stakeholders' preferences. Prioritizing the stakeholders' preferences, formalized in terms of relative importance of non-functional properties, can have a dramatic impact on the design of the system.

In addition to preferences, stakeholders may define constraints over non-functional properties of a system. For example, a stakeholder may define a constraint that an on-line shopping system has to be at least medium secure and cost less than $1000. Thus, the final system should not have any functionality (i.e., feature) that provides positive impact on security less than the medium level. We assume that a product has some level of qualitative non-functional property (e.g., medium security), if all of its features have at least that level of non-functionality (i.e., all the features must have at least medium security). Formally, we define stakeholders' preferences and constraints as:

**Definition 2.1.3** ***(Stakeholders' Preference and Constraint Model)***. *A stakeholder preference and constraint model is a quadruple* $SPCM = (NFP, V, RI, CO)$ *where 1)* $NFP$ *is a set of non-functional properties; 2)* $V$ *is a set of values of NFPs, which can be either numeric values or qualifier tags based on the type of property 3)* $RI$ *is a set of relative importance relations between non-functional properties (Definition 2.1.2); and 4)* $CO \subseteq NFP \times V$ *is a set of constraints over the values of non-functional properties.*

### 2.1.5 Feature Model Configuration Process

After developing a feature model and implementing features in the domain engineering phase, software products are derived in the application engineering phase by configuring the feature model and instantiating reusable assets according to the requirements of a target application. A feature model represents the configuration space of a system family [16]. An application engineer may create a final software system by selecting a proper set of features based on the stakeholders' needs, and deselecting irrelevant features. In other words, application engineers need to go over the entire feature model and make a decision at each variation point to select the best variant(s).

In selecting a set of features for a product not only the functional requirements should

be satisfied, but non-functional requirements should also be satisfied. Figure 2.3 illustrates the feature model configuration process in the application engineering life-cycle. As shown in the figure, stakeholders specify requirements over functional and non-functional aspects. Application engineers formulate requirements in terms of constraints and preferences and make configuration decisions to select proper features. This process is conducted in an iterative manner including several specialization steps [16]. The configuration process is a vital

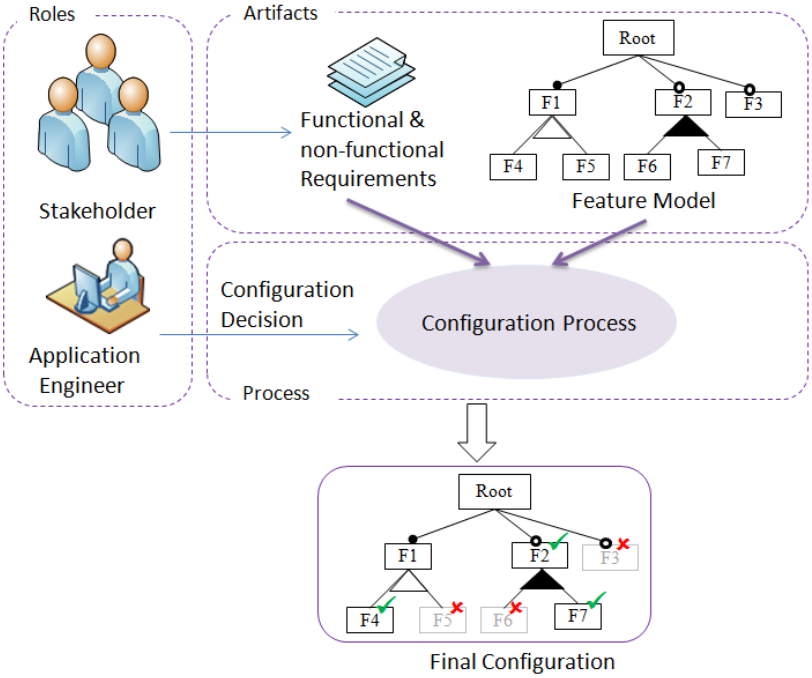



Figure 2.3: Manual feature model configuration process in application engineering phase

task in SPLE, because selecting the right and suitable features for a software product is important for the success of the project at the hand. Moreover, the configuration is a complex process because: 1) industrial feature models are large and may contain hundreds of features; 2) selecting a proper set of features requires close consideration of many factors such as technical limitations, implementation costs, or stakeholders' requests and expectations[2]; 3) features may have partial or full (positive or negative) impacts on non-functional properties; 4) Non-functional requirements may conflict with each other and trade-off decision are required based on the preferences of stakeholders.

### 2.1.6 Hierarchical Task Network (HTN) Planning

Given an initial world description, goal conditions, and actions, planning problem is to find a plan leading from start to the goal. Hierarichal Task Network(i.e., HTN) Planning is among several planning techniques proposed in artificial intelligence, which fits well with domains consisting of low level actions and high level tasks. High level tasks are hierarchically refined into lower level tasks and finally into actions. HTN planning consists of a planning domain, planning problem, and an output plan [52]. Figure 2.4 shows a simple example of HTN domain [34] and initial states; in this example the goal is traveling between home and park. As shown in the figure, the initial task (i.e., travel(me, home, park)) is decomposed into two sub-tasks (i.e., travel-by-foot, and travel-by-taxi) and finally each sub-task is refined into some actions. Definition 2.1.4 formalizes an HTN domain [52].

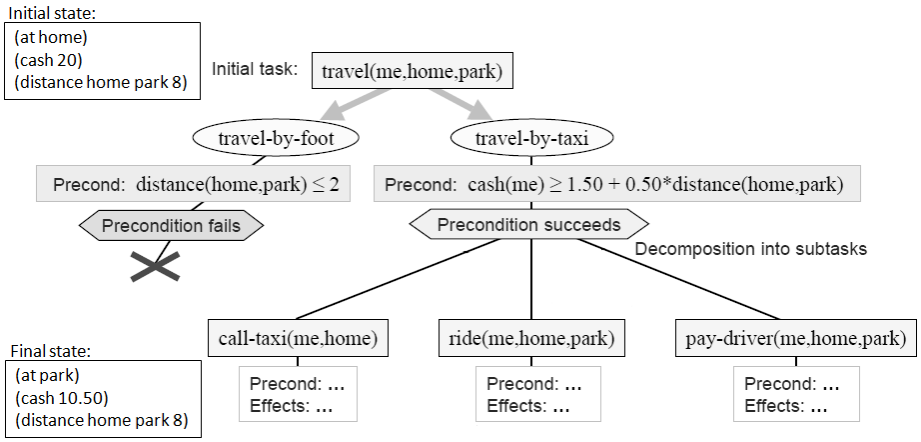



Figure 2.4: An example of HTN domain [34]

**Definition 2.1.4** ***(HTN Planning Domain)***. *An HTN planning domain is a quadruple* $D = (O, T, M, V)$ *where 1)* $O$ *is a set of operators; 2)* $T$ *is a set of tasks; 3)* $M$ *is a set of methods; and 4)* $V$ *is a set of domain predicates.*

An *operator* (denoted as $o$) represents a low level action (e.g., ride-taxi(me, home) in the Figure 2.4), which can be executed in the domain and is formally defined as a quintuple $o = (name(o), pre(o), eff(o), del(o), value(o))$. $pre(o)$ defines a pre-condition (e.g., $cash(a) > 1.50 + 0.50 \times distance(x, y)$ as a precondition for operator pay-taxi) for every operator, which represents required conditions for performing the operator. The effect of performing the operator (e.g., $cash(a) \leftarrow cash(a) - 1.50 + 0.50 \times distance(x, y)$ as an effect of performing

operator pay-taxi) can also be represented by using a post condition $eff(o)$. $del(o)$ or operator's delete list shows what becomes false after performing the operator. For every operator, an optional value $value(o)$ can be defined, which shows a required cost for the execution of that operator. The total value of an output plan is the sum of the values of the operators in the plan.

The *task* construct (denoted as $t$) represents higher level activities in HTN and can recursively be decomposed into the lower level tasks, and finally operators. In HTN, only operators can be executed and tasks can only be reduced into sub-tasks and operators [34]. Refinement of a task into sub-tasks is done using one or more methods (denoted as $m_t$) corresponding to the task. So, every method defines how a task is decomposed into lower level tasks or operators. A method is a quadruple $M = (name(m), task(m), pre(m), dec(m))$ where $task(m)$ is a parent task, $pre(m)$ is a pre-condition, and $dec(m)$ is a list of sub tasks into which the parent is decomposed. A precondition $pre(m)$ defines a required condition for decomposing the parent task. A method is applied only when its precondition is satisfied [52]. For example, method travel-by-taxi can be defined as following:

- *task*: travel(a, x, y)
- *precond*: $cash(a) > 1.50 + 0.50 \times distance(x, y)$
- *subtasks*: call-taxi(a, x) $\rightarrow$ ride(a, x, y)$\rightarrow$ pay-driver(a, x, y)

The planning problem describes characteristics of a required plan - the objective, initial state, and constraints. The HTN planning problem is formally defined as follows [52] :

**Definition 2.1.5** ***(HTN planning problem)****. An HTN planning problem is a triple $PP = (S, T, D)$ where 1) $S$ is a set of logical atoms (initial state); 2) $T$ is a set of initial tasks; and 3) $D$ is a planning domain description defined in Definition 2.1.4.*

As a result of applying a planning technique, a plan containing a sequence of actions that satisfies the objective and the constraints defined in the planning problem is produced. The HTN planning formulates the plan by recursively decomposing the tasks into sub tasks until it reaches the primitive tasks, which can be performed [34]. Similar to [52], we define an HTN plan as follows:

**Definition 2.1.6** ***(HTN Plan)****. Let $PP = (S, T, D)$ be a planning problem defined according to 2.1.5. A plan for planning problem $PP$ is a double $\pi = (O', c)$ where 1) $O' \subseteq O$*

*is a set of operators that will achieve tasks T from initial state S in domain D; and 2) c is the total value of the plan.*

## 2.2 Related Work on Feature Model Configuration

The configuration process has been a center of interest in software product line research community for a long time. Many research papers have been proposed to guide application engineers for configuring feature models as well as automate parts of the configuration process and optimize non-functional properties in the product derivation.

The first attempt, important for our research, has been done by Czarnecki et al. [16] who introduced a stage configuration process. In that work, a number of specialization steps are introduced to remove variability from feature models. In each specialization step, some parts of the feature model variability are resolved. These steps include: refining feature cardinality, refining group cardinality, removing a sub-feature from a feature group, selecting a feature from a feature group, assigning an attribute value, and cloning a solitary sub-feature. In order to help application engineers to derive a valid configuration using this approach, tooling support and validation approaches are provided by Czanrecki et al. [16]

Second group of related works focuses on automating configuration process and considers functional and non-functional requirements of the target application when configuring the feature model. Benavides et al. [7] developed an automated reasoning technique over extended feature models (i.e., feature models with extra-functional features). Using their extension, they were able to assign extra-functionalities such as price range or time range to features. The purpose of their technique is to find a product of a model based on given constraints. Their technique is based on mapping feature models to Constraint Satisfaction Problems (CSP)s and use of CSP solvers [7][8].

Batory et al. [5] integrated feature models with grammars and propositional formulas. Moreover, they used Logic Truth Maintenance Systems (LTMS) and satisfiability solver (SAT), in feature models. The SAT techniques are used for checking the violation of the feature model constraints during the configuration process. Therefore, the SAT techniques can be used as peripheral means during configuration process.

Siegmund et al. [50] have developed a technique called SPL conqueror which extends feature models with non-functional properties and applies CSP to find an optimal configuration based on user defined objective functions. In their technique, a number of preprocessing

steps are taken to reduce the search space for optimal configuration. Their approach differs from Benavidas' approach on types of non-functional properties they manage. Siegmund et al. considered both qualitative and quantitative non-functional properties while Benavidas et al. concentrate on quantitative non-functional properties.

White et al. [61] used a Filtered Cartesian Flattening (FCF) method to select optimal feature sets according to the resource constraints. In their method, they map the feature selection problem to a multi-dimensional multi-choice knapsack problem (MMKP) [61]. By applying the existing MMKP approximation algorithms, they provide partially optimal feature configurations in polynomial time. Filtered Cartesian Flattening consists of different steps including: 1) producing a number of independent MMKP sets (i.e. cutting the FM diagram); 2) converting feature model parts to XOR relations because MMKP just supports XOR relation; 3) flattening with Filtered Cartesian Products; 4) handling Cross-tree Constraints; 5) MMKP Approximation. Although their approach is successful for large feature models, they have a limitation on the number of resources and the amount of resources consumed by features. Moreover, for performing their approach, multi-core processors and parallel computing are required.

White et al. [62] also formalize the stage configuration and introduce a Multi-step Software Configuration probLEm solver (MUSCLE) in which they provide a formal model for multi-step configuration and map it to CSPs. Hence, CSP solvers were used to determine the path from the start of the configuration to the desired final configuration. They consider non-functional properties such as cost constraints between two configurations and formalize them as CSP constraints. Their approach is only applicable for multi-stage configuration and focuses on creating new configurations from an already derived product configuration.

Mendonca et al. [32] proposed a translation of basic feature models into propositional logics and used Binary Decision Diagrams (BDD) as the reasoning system. Their approach concentrates on validating feature models and does not offer a facility for automated configuration. It can be used in a multi-stage configuration process to validate the results of every specialization of a feature model (called interactive configuration). An interactive configuration only checks the structural constraints of feature models and does not consider preferences and non-functional requirements of stakeholders. A tool is implemented to support software developers in validation.

Guo et al. also addressed the challenge of optimizing feature model configuration in their work [23]. They proposed an approach in which Genetic Algorithms are employed to

optimize Feature Selection (GAFES). In the GAFES algorithm the first population is created by randomly generating a string with $n$ binary digits (which represents a chromosome so that a value of 1 and 0 for a digit indicates the selected and deselected feature, respectively). Next, the arbitrary feature selection is transformed into a valid feature selection by applying FesTransform (Feature selection Transform) algorithm which is a key component of their approach. The results of their empirical study show that their proposed algorithm can achieve an average of $86 - 90\%$ optimality for feature selection.

# Chapter 3

# Automated Feature Model Configuration

In this chapter we introduce our configuration approach and explain its details. Our basic idea is to employ planning techniques for solving the configuration problem. To this end, we transform feature models into hierarchical task network formalisms and use a HTN planner to automatically select proper features based on stakeholders' requirements. We also describe how non-functional properties can be optimized during the configuration process, according to the needs of stakeholders .

## 3.1 Integrating Feature Models with Non-Functional Properties

Due to the fact that a product line is developed for an entire domain, different products satisfy very diverse functional and non-functional requirements. So far, the main focus of software product line development approaches is on functional variability of products. Diversity of non-functional properties and their values in different products have been neglected in many existing software product line approaches. However, in order to deliver a product which fits to both functional and non-functional requirements of stakeholders, it is crucial to model non-functional properties along with functional properties in the product line development. This section introduces the proposed solution for incorporating non-functional

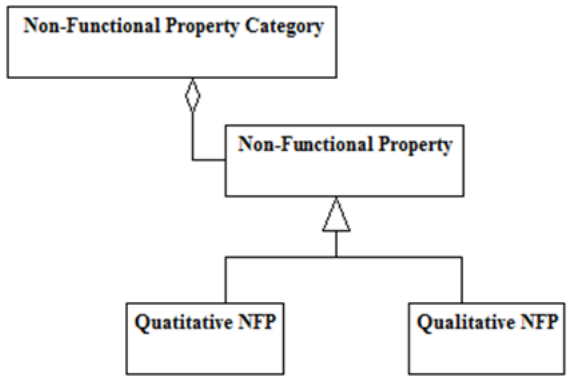



Figure 3.1: Non Functional Property Model

properties into software product line models. The solution consists of a non-functional property model (as a separate model) and extended feature models (feature models with features containing non-functional properties).

### 3.1.1 Modeling Non-Functional Properties

Based on the existing research on non-functional properties in software engineering and software product line, we have proposed a non-functional property model (see Figure 3.1). In the proposed model, based on the nature of non-functional properties, we categorized them into two main categories: *Quantitative* and *Qualitative*. Most non-functional properties are quantitative and there may be some qualitative properties of software systems [19]. In order to make the model more flexible, we let users define several categories to classify the non-functional properties based on their interests and domain.

A *quantitative* property is countable, measurable, or comparable and can be shown as a numeric value. Cost, Performance, and Memory usage are examples of quantitative properties. This category of properties itself has some types like ratio or interval. Metric based values are defined for quantitative non-functional properties and can be measured for a product. For example, performance can be measured for and assigned to the specific feature.

After measuring a non-functional property value for features, the NFP value of a product is computed by aggregating the NFP value of features involved in the product. Based on the nature of non-functional properties, different aggregation functions can be applied [43][63][33]. An aggregation function is a statistical function such as: mean, max, min,

summation, multiplication and so on. For example, to compute the non-functional values of a product, for some non-functional properties such as cost and response time, the values are summed; while for others like availability and reliability values are multiplied. Table 3.1 adapted from [43] shows a set of quantitative non-functional properties and their corresponding aggregation functions.

Table 3.1: Quantitative properties and corresponding aggregation functions

| NFP | Unit | Aggregation function |
|---|---|---|
| Response Time ($q_{rt}$) | msec | $\sum_{i=1}^{n} q_{rt}(f_i)$ |
| Cost ($q_c$) | $ | $\sum_{i=1}^{n} q_c(f_i)$ |
| Availability ($q_{av}$) | percent | $\prod_{i=1}^{n} q_{av}(f_i)$ |
| Accuracy ($q_{ac}$) | percent | $\prod_{i=1}^{n} q_{ac}(f_i)$ |
| Throughput ($q_{tp}$) | invocation/sec | $min\{f_1, \ldots, f_n\}$ |

*Qualitative* properties are the second category of non-functional properties which cannot be exactly measured. Qualitative properties are represented by a label which is called qualifier tag [3] (for example High, Medium, and Low are the values of security property). A qualifier tag represents a possible impact of functionality on a qualitative non-functional property. For example, the Credit card feature (i.e., functionality) has *high* impact on *international sale*; hence it can be annotated with the *High* qualifier tag for *international sale*. Security, Reliability, and Usability are examples of qualitative properties.

### 3.1.2 Extended Feature Model

In feature oriented software product line, feature models are mainly used for representing functional variability between different products. In order to enable representing non-functional properties and their different values in the feature model, we need to extend the standard feature model. Some researchers have extended the feature model notation with non-functional properties to represent the non-functionality aspect [7][50]. Similarly, we extended the feature model with the notion of non-functional properties which can be either qualitative or quantitative. In this model each feature can be annotated with several non-functional properties and corresponding values of those properties. Figure 3.2 shows the extended feature model representing non-functional properties as well as functional properties (i.e. features) and their relationships.

As described in the previous section, the qualitative non-functional properties such as

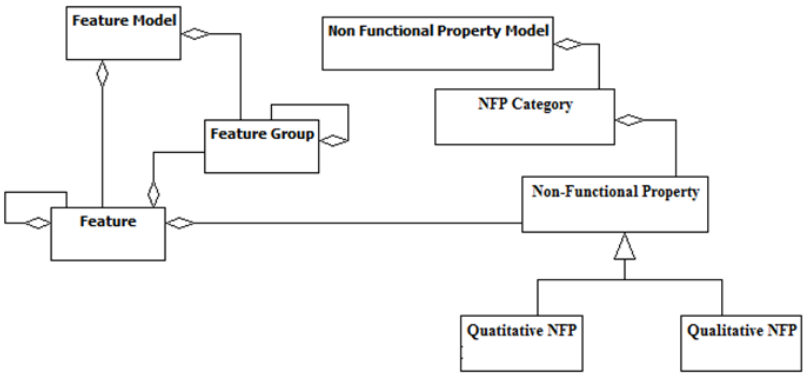
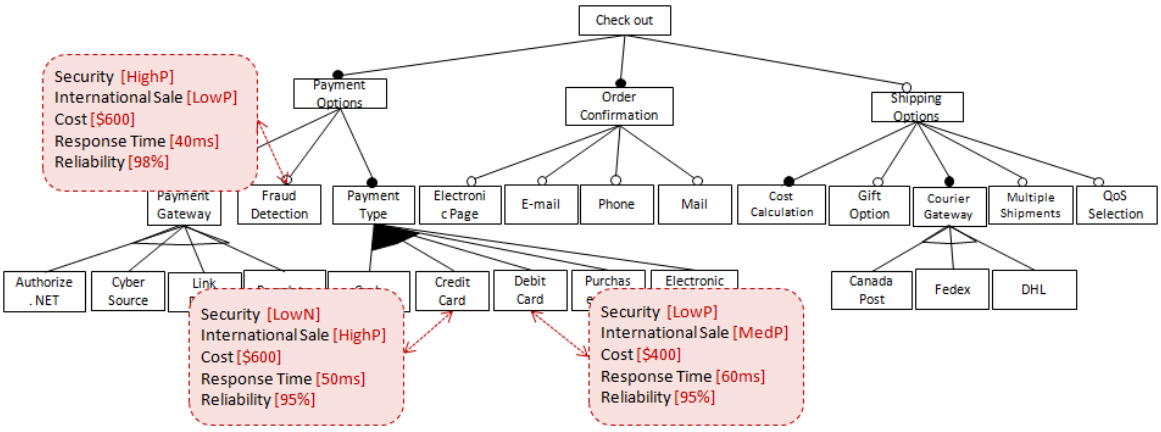



Figure 3.2: Integration of Feature Model with Non-Functional Property Model

*customer satisfaction* or *user friendliness* can be described using an ordinal scale consisting of a set of predefined qualitative values which we refer to as qualifier tags. For example, *High negative*, *Medium negative*, *Low negative*, *Low positive*, *Medium positive*, and *High positive* form the qualifier tags defined for *customer satisfaction*. A qualifier tag represents a possible impact of functionality on a qualitative non-functional property. On the other hand, metric based values are defined for the quantitative non-functional properties and can be measured for a product.

Figure 3.3b shows non-functional properties employed in the on-line shopping product line. *Response time*, *cost*, *availability*, and *reliability* are quantitative non-functional properties and c*ustomer satisfaction*, *international sale*, and *security* are the qualitative non-functional properties. In our approach, we assume that atomic features (i.e., leaf features) in a feature model have concrete implementations. Non-atomic (i.e., non-leaf) features are used for variability and composition relationships of the atomic features. Hence, non-functional properties are defined for the leaf features. If an intermediate feature contains implementations and non-functional properties, we create a mandatory child feature for the intermediate feature and assign the non-functional properties to the child feature. After identifying domain features, developing a feature model and implementing its atomic features, we can then analyze the impact of features on non-functional properties.

In the proposed extension (see Figure 3.2), for each atomic feature a user can define several non-functional properties and specify the value for each one. For qualitative non-functional properties, based on existing domain knowledge, the impact of each feature on non-functional properties can be identified and proper qualifier tags can be assigned to each




| Features \ NFP | Security | International Sale | Customer Satisfaction | Response Time | Cost | Availability | Reliability |
|---|---|---|---|---|---|---|---|
| Credit Card | LowN | HigP | MedP | 50ms | $600 | 90% | 95% |
| Debit Card | LowP | MedP | MedP | 60ms | $450 | 90% | 95% |
| Fraud Detection | HighP | LowP | HighP | 40ms | $600 | 80% | 98% |
| ⋮ | | | | | | | |

Figure 3.3: Integrated Feature Model with Non-Functional Properties

feature's non-functional properties. On the other hand, quantitative non-functional properties for the features can be measured using a suitable metric and assigned to the features. We assume that some techniques, like those proposed in [49], can be employed to measure non-functional properties for each feature. For example, as shown in Figure 3.3, feature Credit Card is annotated with *low negative security*, *high positive international sale*, and *medium positive customer satisfaction* and its estimated *cost*, *response time*, and *availability* are \$600, 50ms, and 90%, respectively.

Extended feature models are formally defined as follows:

**Definition 3.1.1** ***(Extended Feature Model)***. *An extended feature model is a decuple* $EFM = (F, F_O, F_M, F_{IOR}, F_{XOR}, F_{ic}, F_A, NFP, V, AQ)$ *where 1)* $F$, $F_O$, $F_M$, $F_{IOR}$, $F_{XOR}$, $F_{ic}$ *stand for a feature model (FM) according to Definition 2.1.1 ; 2)* $F_A \subseteq F$ *is a set of atomic features; 3)* $NFP$ *is a set of non-functional properties; 4)* $V$ *is a set of values of NFPs, which can be either numeric values or qualifier tags based on the type of non-functional property ; 5)* $AQ \subseteq F_A \times (NFP \times V)$ *is a set of pairs of atomic features and their annotation pairs of non-functional properties and their value.*

## 3.2 Optimizing Non-Functional Properties Based on Stakeholders' Preferences

In the application engineering process, a concrete product is generated by configuring a feature model based on the target application requirements and by instantiating the reference architecture based on the configured feature model. Stakeholders of each application may have different constraints and preferences over non-functional properties that must be considered in the configuration process. Configuring a feature model based on the stakeholders' requirements and preferences usually means selecting features such that a feature model configuration satisfies the stakeholders' functional requirements and constraints and optimizes their preferences. To optimize the configuration with respect to the preferences, feature ranks must be computed based on their impact on the non-functional properties which may be of different importance for the target stakeholders. Additionally, both qualitative and quantitative non-functional properties must be considered in the computation of feature ranks. Following sections describe the way that non-functional properties are optimized based on the preferences and constraints of stakeholders.

### 3.2.1 Stratified Analytic Hierarchy Process

To calculate the ranks of features based on relative importance of non-functional properties assigned to the features, we applied the Stratified Analytical Hierarchy Process (S-AHP) algorithm proposed in [3]. S-AHP is based on the Analytic Hierarchy Process (AHP) [44], which is a well-known pairwise comparison method used to calculate the relative ranking of different options based on stakeholders' judgments. There are several multi-criteria decision making algorithms including: 1) the weighted sum model (WSM) which is the earliest method; 2) the weighted product model (WPM) which is a modification of the WSM; 3) and analytic hierarchy process (AHP) which is a later development and wildly used in several applications [59]. We used S-AHP because it enables ranking non-functional properties based on the defined relative importance between them; according to the study in [3], S-AHP is easy to use and does not need too much effort from stakeholders; and finally, it significantly reduces the number of needed pairwise comparisons. In the following, the process of calculating the weight of non-functional properties based on stakeholders' preferences using AHP algorithm is explained.

As mentioned in Section 2.1.4, the stakeholders' preferences are in the form of relative

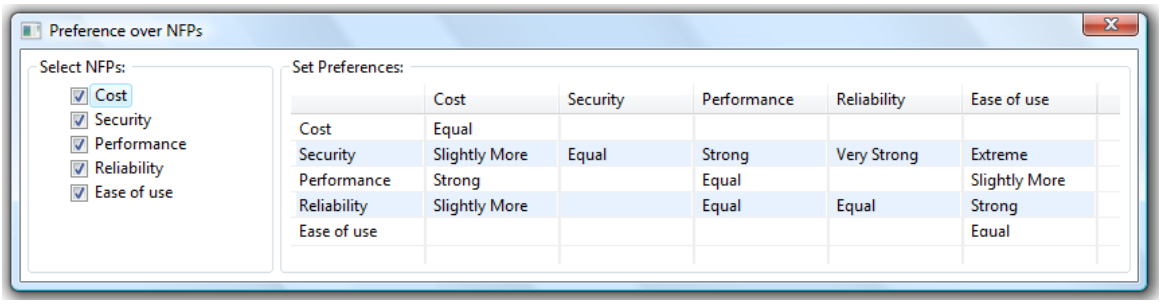



Figure 3.4: Non-functional properties prioritization

importance over non-functional properties. After the stakeholders specify their preferences, a square matrix $RI_{|N|\times|N|}= \{RI[i,j] = \alpha \mid 1 \leq i,j \leq |N|$, $\alpha$ is relative importance of NFP i to NFP j, and $|N|$ shows the number of NFPs on which the stakeholder defines the relative importance $\}$ is created. The value of $\alpha$ (i.e., degree of importance) in each cell of above matrix may be 1, 3, 5, 7, or 9 corresponding to equality, slight value, strong value, very strong or extreme value, respectively [44]. Figure 3.4 shows the non-functional properties prioritization defined by a stakeholder.

According to the AHP algorithm, the elements of matrix RI are normalized by dividing the elements of each column by the sum of the elements of that column; next, to estimate the principal eigenvector, the averages of normalized column are calculated. The principal eigenvector for each matrix, when normalized, becomes the vector of priorities for that matrix [45]. Finally, the weight of non-functional properties is calculated based on eigenvalues. As an example, suppose that a stakeholder defines the relative importance as shown in Figure 3.4; then, RI matrix and RI matrix with normalized values are created (see Table 3.2 and Table 3.3).

Table 3.2: Relative importance of non-functional properties.

| | Cost | Security | Performance | Reliability | Ease of use |
|---|---|---|---|---|---|
| Cost | 1.00 | 0.33 | 0.20 | 0.33 | 3.00 |
| Security | 3.00 | 1.00 | 5.00 | 7.00 | 9.00 |
| Performance | 5.00 | 0.20 | 1.00 | 1.00 | 3.00 |
| Reliability | 3.00 | 0.14 | 1.00 | 1.00 | 5.00 |
| Ease of use | 0.33 | 0.11 | 0.33 | 0.20 | 1.00 |

The final weights of NFPs are as following:

Table 3.3: Normalized values of relative importance of non-functional properties.

| | Cost | Security | Performance | Reliability | Ease of use | Sum |
|---|---|---|---|---|---|---|
| Cost | 0.08 | 0.19 | 0.03 | 0.03 | 0.14 | 0.47 |
| Security | 0.24 | 0.56 | 0.66 | 0.73 | 0.43 | 2.63 |
| Performance | 0.41 | 0.11 | 0.11 | 0.10 | 0.14 | 0.90 |
| Reliability | 0.24 | 0.08 | 0.08 | 0.10 | 0.24 | 0.80 |
| Ease of use | 0.03 | 0.06 | 0.06 | 0.02 | 0.05 | 0.20 |

$$\left\{\begin{array}{cc} Security & 0.66 \\ Performance & 0.22 \\ Reliability & 0.20 \\ Cost & 0.12 \\ Ease\ of\ use & 0.05 \end{array}\right\}$$

### 3.2.2 Utility Function for Calculating Features' Rank

To calculate the rank of features, both qualitative and quantitative properties should be taken into account. To consider qualitative non-functional properties in feature ranks, they should be mapped to the real values. The first way is that, the stakeholders or application engineers provide a mapping function from qualifier tags onto real values. For example, for *customer satisfaction*, one can define the following mapping function.

$$M_{customerSat.}(QT) = \left\{\begin{array}{cc} -1 & High\ negative \\ -0.5 & Medium\ negative \\ -0.25 & Low\ negative \\ 0.25 & Low\ positive \\ 0.50 & Medium\ positive \\ 1 & High\ positive \end{array}\right\}$$

The other way for calculating the corresponding real-numbers for the qualifier tags of a qualitative non-functional property is to use the AHP algorithm. In this way, stakeholders specify the relative importance between the qualifier tags of each non-functional property and AHP calculates the rank of each qualifier tag. The detail of calculating weight of options using AHP is explained in the Section 3.2.1. For example, the stakeholder specifies that for the *international sale* property, *High positive* $\succ^3$ *Medium positive*, which means a feature with high positive impact on *international sale* is slightly more important than the feature

with medium positive on international sale. In both the methods (i.e., both the mapping function and AHP), we assume the values are normalized into the $[-1, +1]$ range.

After defining the ranks of non-functional properties and the mapping function for the qualitative non-functional properties, we describe the following utility function to calculate the ranks of each feature based on an extension to [64]. The proposed utility function calculates the rank of features based on their impact on non-functional properties by considering the preferences of stakeholders formulated in terms of the weight of non-functional properties.

**Definition 3.2.1** ***(Utility function)***. *Let us assume there are $\alpha$ quantitative NFPs to be maximized, $\beta$ quantitative non-functional properties to be minimized, and $\theta$ qualitative non-functional properties whose impact needs to be maximized. The utility function for feature $f$ is defined as:*

$$R(f) = \sum_{i=1}^{\alpha} w_i \times \frac{q_i(f)-\mu_i}{\sigma_i} + \sum_{j=1}^{\beta} w_j \times (1 - \frac{q_j(f)-\mu_j}{\sigma_j}) + \sum_{k=1}^{\theta} w_k \times M_k(QT(f))$$

*where $w$ is the weight of each non-functional property calculated by S-AHP such that $0 \leq w_i, w_j, w_k \leq 1$ and $\sum_{i=1}^{\alpha} w_i + \sum_{j=1}^{\beta} w_j + \sum_{k=1}^{\theta} w_k = 1$ and $M_k(QT(f))$ returns the real number corresponding to quality tags of $k$-th non-functional property. $\mu$ and $\sigma$ are the average values and the standard deviation of the quantitative non-functional properties for all atomic features in the feature model.*

The overall rank of a product is calculated by aggregating the ranks of selected features for the product. The aggregation function used for calculating the product rank depends on the aggregation functions which exist over non-functional properties of a feature. As discussed in Section 3.1.1, some quantitative non-functional properties such as *response time* are additive and the quality of a composition of features is computed by adding up the quality of features [43][63][33]. The second type of aggregation functions defined on quantitative non-functional properties is multiplication when the quality of composition is calculated by multiplying the quality of features involved in the composition. The multiplication type can be converted into an additive type by computing the logarithm values of non-functional values. For qualitative non-functional properties, a qualifier tag assigned to a feature represents a qualitative impact of the feature on the non-functional property. Considering the mapping function that maps the qualifier tags into real numbers, we can

calculate the overall impact of a composition of features by adding the impacts of the features involved in the composition. Hence, the aggregation function over the utility functions of features is an additive function and the overall rank of a product is computed as $\mathbb{R} = \sum_{f_i \in P} R(f_i)$. In order to derive an optimal configuration (product) $P$, we need to select the features, which maximize $\mathbb{R}(P)$.

## 3.3 Automated Feature Model Configuration Based on HTN planning

Having annotated a feature model with NFPs, the process for deriving a new product starts by selecting and deselecting features based on the stakeholders' requirements reflected through desired features and preferences expressed in terms of relative importance between NFPs. The configuration problem is concerned with selecting features that satisfy the functional requirements (i.e., the requested functionality) and constraints and optimize the preferences. Formally, the configuration problem and the configuration are defined as follows.

**Definition 3.3.1** ***(Configuration Problem)***. *Let* $SPCM = (NFP, V, RI, CO)$ *and* $EFM$ $= (F, F_O, F_M, F_{IOR}, F_{XOR}, F_{ic}, F_A, NFP, V, AQ)$ *be a: i) stakeholder's preference and constraint model; and ii) extended feature model as per Definitions 2.1.3 and 3.1.1. A configuration problem is a triple* $CP = (EFM, SPCM, F'_A)$*, where* $EFM$ *is an extended feature model,* $SPCM$ *is the set of preferences and constraints over NFPs defined by the stakeholders, and* $F'_A \subseteq F_A$ *is a set of required atomic features.*

**Definition 3.3.2** ***(Configuration)***. *Let* $CP = (EFM, SPCM, F'_A)$ *and* $EFM = (F, F_O, F_M, F_{IOR}, F_{XOR}, F_{ic}, F_A, NFP, V, AQ)$ *be a configuration problem and extended feature model, respectively. A configuration is a double* $COF = (F", \mathbb{R}(F"))$ *where 1)* $F' \subset F" \subset F$ *is a set of selected features; and 2)* $\mathbb{R}(F")$ *is the total rank of the configuration.*

Therefore, in the configuration process, an application engineer deals with the selection of a set of features containing the stakeholders' required functionality. The total rank of the selected features is the maximum rank based on the stakeholders' preferences.

To automate the configuration process, we define transformation rules to convert a configuration problem to a planning problem. To do so, we develop transformation rules to

represent extended feature models in the HTN formalism. The transformation is done in two steps: 1) generating an HTN domain model from a feature model, and 2) generating a planning problem from a configuration problem.

When transforming a feature model and the corresponding configuration problem into the HTN domain and HTN problem, we need to convert the maximization problem into a minimization problem, because the SHOP2 planner, used in our implementation, works on minimization only (Negating the ranks of features can do this).

### 3.3.1 Transforming Feature Model into HTN

Considering the analogy between tasks in HTN and features in a feature model, there is a need to define task decompositions to reflect feature relations. The HTN *method* element is used to define decomposition of tasks into sub-tasks or operators. As mentioned in Definition 2.1.4, a method contains the parent task, a list of children, and a precondition. Here, we have two options for decomposing a task into sub-tasks:

1. we can define different methods with a common parent task. In this case, an HTN planner (i.e., SHOP2) for decomposing a task into sub-tasks selects just one method for each plan. Hence, this option is suitable for mapping the XOR relation to HTN;

2. we can also define one method for a decomposition in which all children can be considered a list of sub-tasks in the method. In this case, the HTN planner performs the method if and only if the preconditions of all sub-tasks or operators are satisfied. We can then use this option for defining AND-decomposition with mandatory child features.

According to abovementioned, in HTN, a *method* does not support OR-decomposition (i.e., selecting one or more tasks from sub-task list). Moreover, in HTN the concept of *optional* tasks or operators is not defined; that is, in the sub-task list of a method, we can have just mandatory features. In other words, OR group and optional features cannot directly be transformed into HTN formalism. Hence, before transforming a feature model into an HTN domain, a preprocessing step should be done to replace optional and OR features. To this end, similar to [29] for optional goals, every optional feature $f_o$ is replaced with a new feature $f_p$ which is decomposed into two alternative features $f_o$ and $f^d$ where $f^d$ stands for a "dummy" feature (i.e., an operator without any effect) (Figure 3.5a). Regarding OR-decomposition, every OR group in the feature model is converted into a set of optional

features and a mandatory feature ($f_{or}$) with AND relations between them (Figure 3.5b). Feature $f_{or}$ is added to ensure that at least one of the features in the OR group is selected (more details are explained in the "Generating Tasks and Methods" subsection). Consequently, an OR group can be easily transformed into HTN using a method with sub-tasks containing "dummy" features.

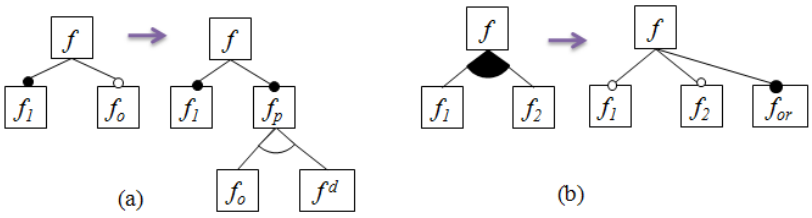



Figure 3.5: The preprocessing step for transforming feature models into HTN domains: (a) optional features (b) OR feature groups

After performing the above changes in the feature model, we transform the feature model into an HTN domain using the following three types of transformation rules:

- domain predicate transformation;
- operator transformation;
- method and task transformation.

**Generating Domain Predicates.** In the HTN planning domain, methods and tasks involve *logical expressions* which are the combination of *logical atoms* and *logical connectives*. *Logical atoms* involve a *predicate symbol* plus a list of *terms*. These logical expressions can be used as either precondition formulae or effect formulae. Based on possible preconditions for selecting and decomposing features, we can identify possible domain predicates.

In the feature model, we have two groups of preconditions including: 1) precondition for checking integrity constraints (i.e., requires and excludes); and 2) preconditions for checking the stakeholder's constraints with respect to non-functional properties. To generate these preconditions, we need domain predicates for each quantitative NFPs, qualifier tags of the qualitative NFPs, and atomic features in the feature model.

For example, the $v_{secnh}$ and $v_{secph}$ predicates are created for *high negative* and *high positive* qualifier tags of the *security* non-functional property, respectively; and $v_{cost}$ predicates is created for *cost* NFP. These predicates can be involved in the logical expressions

to generate preconditions of methods and operator; using these preconditions we can check whether a feature covers a required NFPs or not. Moreover, for features Credit Card and Debit Card, we generate domain predicates $v_{cc}$ and $v_{dc}$, respectively. For each non-atomic feature, a propositional formula (called *attainment formula* [29]) is created according to the features that exist in its sub-tree (Figure 3.6). For instance, in the on-line shopping feature model, which is shown in Figure 2.2, the attainment formula for feature Payment Gateway is $\varphi_{PG} = v_{AN} \oplus v_{CS} \oplus v_{LP} \oplus v_P$, which shows alternative relation between Authorize. NET, Cyber Source, Link Point, and Paradata Features. These formulae are later used as preconditions of the other features in the feature model in order to investigate the integrity constraints (requires and excludes relations among features).

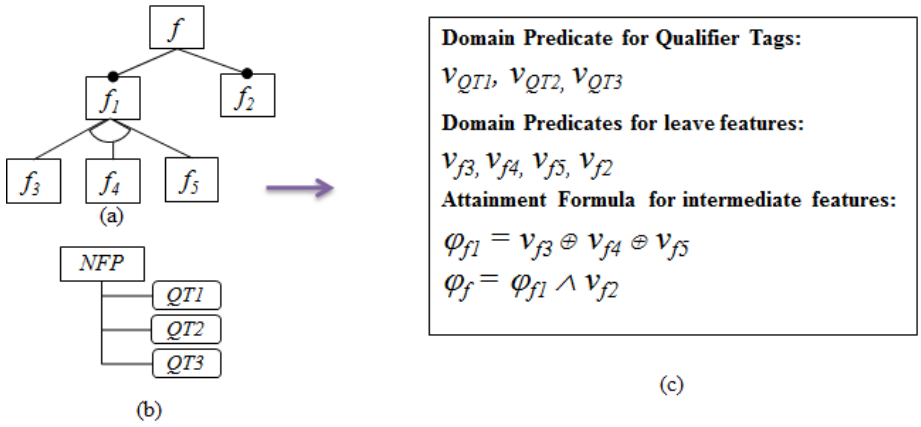



Figure 3.6: Generating Domain Predicates.(a) feature model, (b) a qualitative NFP and its qualifier tags, (c) Domain Predicates in HTN

**Generating Operators.** HTN operators represent the lowest level activities in the domain. Hence, in process of feature model transformation, HTN operators can be generated from atomic (i.e., leaf) features in the feature model tree. Each atomic feature $f$ is transformed into an operator $o_f$ and the rank of the feature calculated by the utility function is transformed into $value(o_f)$. A precondition is defined for each operator based on the non-functional properties and integrity constraints defined over its corresponding feature. The preconditions are defined as logical AND expressions of:

1. domain predicates corresponding to qualifier tags of qualitative non-functional properties with which the feature is annotated;

2. an evaluation expression to check whether the feature is allowable to be selected or

not based on quantitative non-functional properties constraints;

3. attainment formula of features having *requires* and *excludes* relations with the feature (see Figure 3.7b).

Next, the domain predicate, which corresponds to feature $f$ (i.e., $v_f$) is added as an effect of the operator $o_f$ (i.e., $eff(o_f) = v_f$). Whenever operator $o_f$ is performed, its corresponding domain predicate becomes true (i.e., $v_f = true$).

For handling quantitative non-functional property constraints, a logical expression is created showing the maximum available value of each corresponding non-functional property during the planning process. At the first, this value is set by the requested value of stakeholders and added to the initial state of the planning problem. For example, if a stakeholder has a limitation on *cost* (e.g. maximum \$1000), we add a logical expression "($MaxCost$ 1000)" to the initial state of the planning problem. Then, in the effect of each operator, this value is updated based on the assigned non-functional property value to the feature. For example, in Figure 3.7 feature f2 has *cost* NFP with value \$98; hence, after selecting this feature the maximum available *cost* should be reduced by 98. Moreover, before selecting feature f2 we should check whether we have enough available *cost* or not. To this end, we add logical expression like "($MaxCost - 98 > 0$)" to the precondition formula of feature f2. The value of the maximum available *cost* is updated by adding "($MaxCost = MaxCost - 98$)" to the effect formula of feature f2.

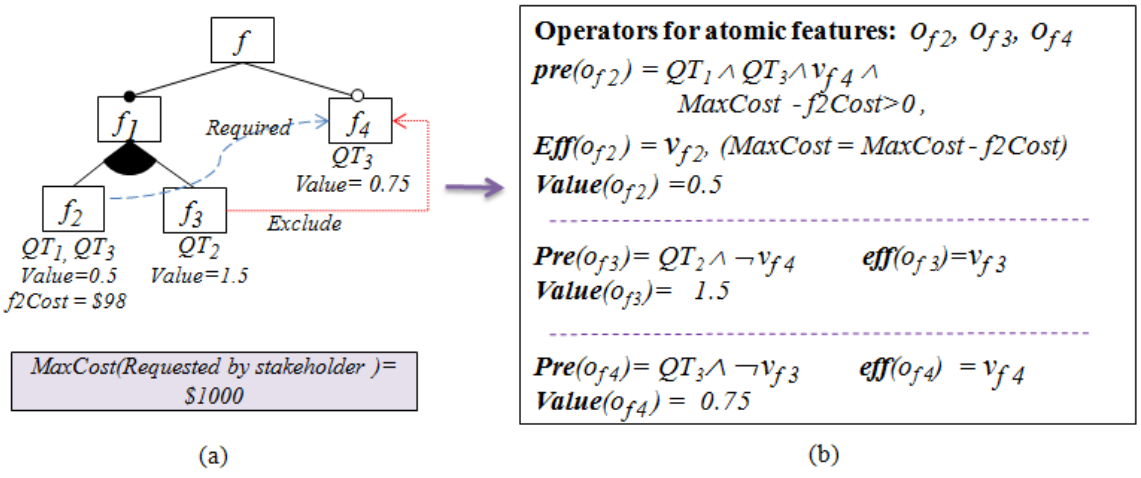



Figure 3.7: Translating Atomic features into Operators. (a)feature Model, annotated with NFPs and feature ranks, (b) Operators generated for atomic features.

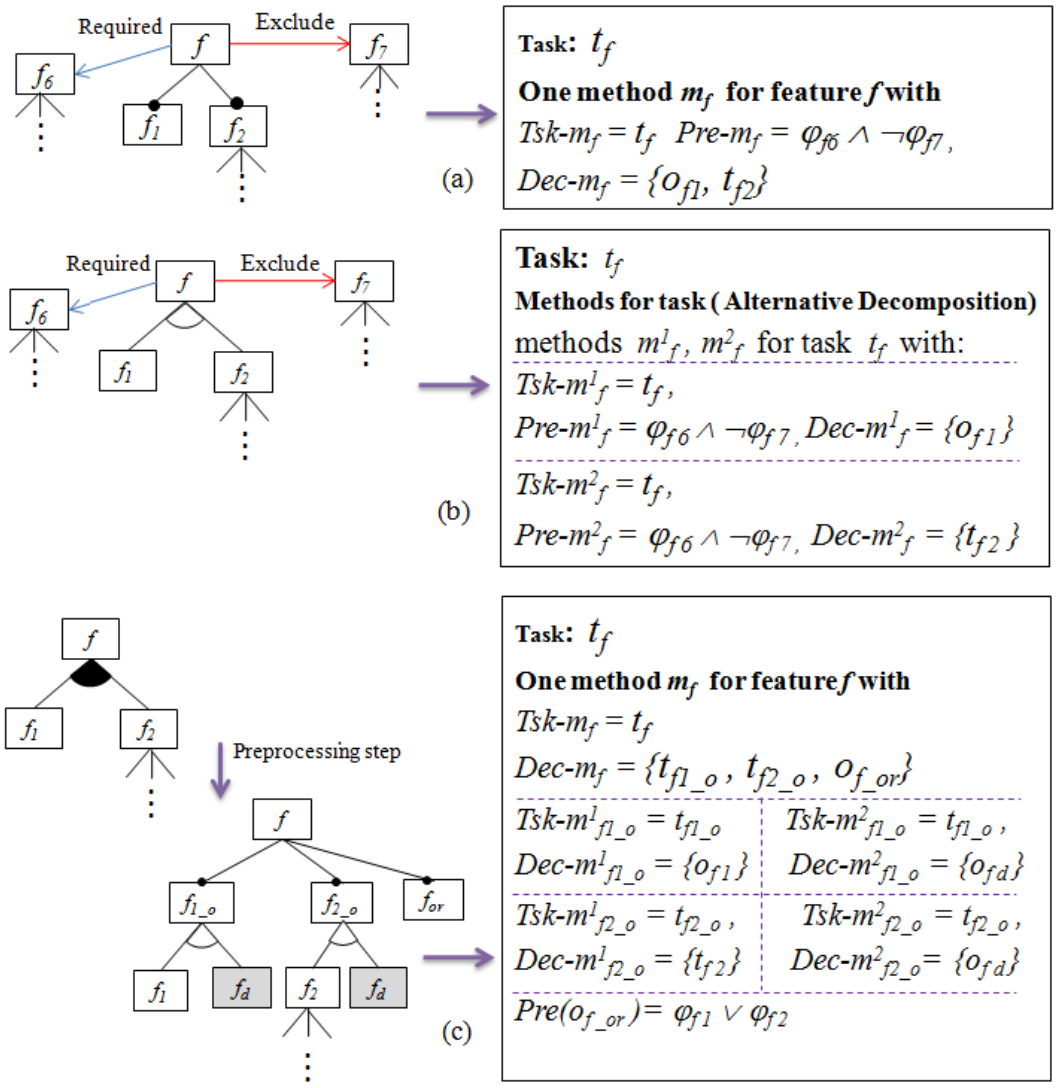



Figure 3.8: Generating tasks and methods from non-atomic features. (a) And Decomposition and corresponding task and method, (b) Alternative Decomposition and corresponding tasks and methods, (c) Or Decomposition and corresponding tasks and methods

**Generating Tasks and Methods.** *Tasks* in the HTN planning domain represent high level activities which need to be decomposed into smaller tasks using a *method*. A feature model also has a hierarchy structure in which features are decomposed into sub-features. Therefore, every non-atomic feature $f$ can be mapped into a task $t_f$; and method(s) ($m_{tf}$) can be defined for different types of decomposition in feature model (i.e., AND, OR, and XOR).

To create tasks corresponding to features, we use the pre-processed feature model from which OR feature groups and optional features have been removed (c.f. Figure 3.5). Based on the type of a feature group(i.e., XOR group or AND-decomposition), we may define one or more methods.

If a feature $f$ is AND-decomposed into features $f_1$ and $f_2$, we define one method $m_f$ which connects corresponding task $t_f$ to tasks or operators corresponding to $f_1$ and $f_2$, (Figure 3.8a). As shown in Figure 3.8a, $Dec - mf$ denotes the task list of method $m_f$ which contains operator $O_{f_1}$ and task $T_{f_2}$ which are corresponding HTN representations for features $f_1$ and $f_2$, respectively. Moreover, the precondition of method $m_f$ contains the attainment formulae of $f_6$ and $f_7$ to check corresponding integrity constraints (i.e., feature $f$ required feature $f_6$ and exclude feature $f_7$).

For alternative feature groups with $n$ sub-features (i.e., $f = XOR(f_1, f_2, .., f_n)$), $n$ methods are defined with a common parent task $t_f$ corresponding to feature $f$. Each method connects task $t_f$ to one operator $o_{f_i}$ or task $t_{f_i}$ corresponding to the sub-feature $f_i$ of the parent feature $f$ (see Figure 3.8b). The precondition for every method is specified based on the attainment formulae of features required by that feature and features excluded by that feature.

Regarding OR feature groups, in the feature model configuration at least one of features in the group must be selected if a parent feature is selected. For example, in Figure 3.8c if feature $f$ is selected, at least one of its sub-features (i.e., $f_1$ or $f_2$ or both) must be selected. To this end, we add a precondition for operator ($o_{f_{or}}$) which is a logical OR expression of attainment formula of the features in the OR feature group (see Figure 3.8c).

Table 3.4 gives the overview of the mapping rules between extended feature models (after eliminating OR and optional features) and the HTN domain.

Table 3.4: Mapping between constructs in extended feature models and HTN Domain models – OR and Optional feature Groups are replaced in the preprocessing step.

| *Extended Feature Model* | | *HTN Domain* | |
|---|---|---|---|
| ***Formal Rep*** | ***Semantics*** | ***Formal Rep*** | ***Semantics*** |
| $(NFP_i, qt_j) \in \text{NFP} \times QT$ | *Qualitative NFP and Qualifier tags* | $v_{NFP_i qt_j} \in \mathcal{V}$ | *Domain predicates* |
| $f_i \in \mathcal{F}_A$ | *Atomic feature* | $o_{fi} \in \mathcal{O}$ | *Operator* |
| $R(f_i)$ | *Rank of atomic feature* | $value(o_{fi})$ | *Property value of operator* |
| $(f_i, (NFP_j, qt_k)) \in AQ$ | *Annotated feature with qualitative NFP and qualifier tag* | $pre(o_{fi}) \leftarrow v_{NFP_j qt_k}$ | *Domain predicate as precondition of operato1r* |
| $(f_i, (NFP_j, v)) \in AQ$ | *Annotated feature with quantitative NFP and its value* | $pre(o_{fi}) \leftarrow Maxv - v > 0$ $eff(o_{fi}) \leftarrow (Maxv = Maxv - v)$ | *Evaluation as precondition and effect of operator* |
| $(f_i excludes f_j) \in \mathcal{F}_{ic}$ | *Integrity constraints** | $pre(o_{fj}) \leftarrow \neg v_{ofi}$ $pre(o_{fi}) \leftarrow \neg v_{ofj}$ | *Pre-condition for operator* |
| $(f_i requiers\ f_j) \in \mathcal{F}_{ic}$ | *Integrity constraints** | $pre(o_{fi}) \leftarrow v_{ofj}$ | *Pre-condition for operator* |
| $f = Xor_n^{i=1} f_i$ | *Alternative feature group f* | $t_f, \cup_n^{i=1} m_f^i$ | *One task and n method one for each alternative child* |
| $f = AND_n^{i=1} f_i$ | *A feature f with set of mandatory child* | $t_f, m_f,$ $Dec\text{-}m_f = \cup_n^{i=1}\ t_{fi}$ | *One task, and one method with a set of sub-tasks* |

**For simplicity in the representation, we considered the includes and excludes relations with atomic features, but features can also be non-atomic*

### 3.3.2 Planning Process

After defining the planning domain, the feature model configuration problem is transformed into the HTN planning problem (Table 3.5 summarizes the mappings). The transformation is done by considering the constraints over the NFPs ($CO$) and setting their corresponding domain predicates as true to form initial state ($S$). For example, if a stakeholder asks for at least *medium security*, and a specific *cost* (e.g., \$1000), the domain predicates $v_{secph} = true$ and $v_{secpm} = true$ and the rest of qualifier tags are set to false; the logical atom "($cost$1000)" is also added as an initial state. Next, the set of required atomic features ($F_A^{'}$) and the root of the feature model are translated into initial tasks of the planning problem ($T$).

The SHOP2 planner starts with the first task in the initial tasks, if the task is a *non-primitive* task, the planner chooses an appropriate method and instantiates it to decompose

the task into sub-tasks; if the task is a *primitive* task, the planner chooses an applicable operator and instantiates it to perform an action. A solution plan is found if all constraints are satisfied, otherwise the planner backtracks and chooses the other methods. Consequently, the planner tries all possible paths in the feature model tree and returns the best features (defined as operators in the planning domain) in the feature model. Then, based on the selected atomic features in the produced plan, intermediate features are selected using a propagation algorithm in which all intermediate features are selected if they have a selected sub-feature(s).

Table 3.5: Mappings between the configuration problem and the HTN planning problem.

| ***Configuration Problem*** | | ***HTN Planning problem*** | |
|---|---|---|---|
| ***Formal Rep.*** | ***Semantics*** | ***Formal Rep.*** | ***Semantics*** |
| $EFM$ | *Extended feature model* | $\mathcal{D}$ | *Domain description* |
| $(NFP_i, qt_j) \in CO'$ | *Selected NFPs and qualifier tags* | $(v_{NFP_i qt_j} = \text{true}) \in \mathcal{S}$ | *Initial state* |
| $(NFP_i, v) \in CO'$ | *Constraint over quantitative NFPs* | $(NFP_i = v) \in \mathcal{S}$ | *Initial state* |
| $f_i \in \mathcal{F}'_{\mathcal{A}}$ | *Requested Feature* | $t_{fi} \in \mathcal{T}$ | *Intial tasks* |
| $f_{root}$ | *Root of extended feature model* | $t_{root} \in \mathcal{T}$ | *Initial tasks* |

# Chapter 4

# Tooling Support

Tools play an important role in both domain and application engineering phases. Having an effective tooling support facilitates software developer tasks in those phases. Although, researchers have already contributed in developing tooling support for defining, managing, and configuring feature models [1][15], there is still the need for product configuration tooling support [41]. Moreover, one common capability lacking in each of these tools is the use of interaction and visualization techniques with respect to the feature model configuration based on the non-functional properties. Hence, in our work we aimed at developing a tool which:

1. provides a facility to integrate the feature model and non-functional properties;
2. facilitates and enhances the feature model configuration process by :
   (a) automating the feature model configuration process based on non-functional requirements;
   (b) employing visualization and interaction techniques, which provides software developers with clear and understandable view of the generated configuration.

For providing tooling support, we extended a well-known feature model plug-in (fmp) [15]. The fmp tool is an eclipse plug-in which supports editing and configuring feature models. It provides very basic visualization and (i.e. it used indented list to show feature model) interaction techniques, however, it provides a good facilities to define a feature model and manage the variabilities. Moreover, the tool is accessible for academic researchers. Hence,

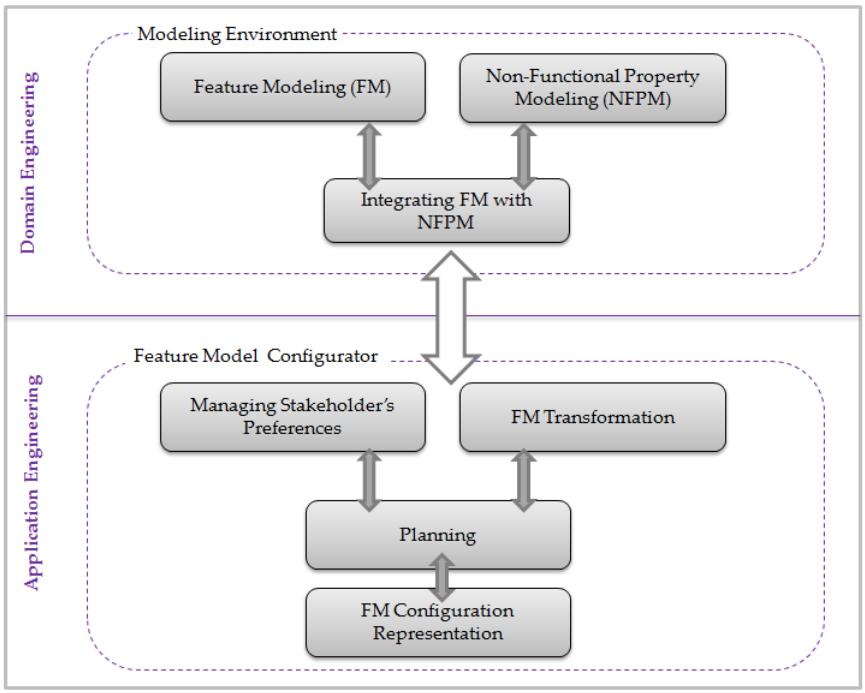



Figure 4.1: High level architecture of the *Vis-fmp*

we consider fmp as a basis and extend it to support more functionalities such as integrating feature model with non-functional properties and automating the configuration process.

Figure 4.1 illustrates the architecture of the tool and its components. In the following sections we discuss different components of the tool in more detail including: 1) integrating the feature model with non-functional properties, 2) managing stakeholders' preferences and constraints, 3) configuration generator (i.e., planning), and 4) visual and interactive view for representing the generated configuration.

## 4.1 Integrating Feature Model with Non-Functional Properties

In order to generate the feature model configuration based on non-functional requirements and provide information about the non-functional properties of the final product, the effect of each feature on the NFPs should be specified. Accordingly, the total value of NFPs for the final product can be calculated based on the effect of selected features on the NFPs. For example, if the cost of each feature in a feature model is specified, the total cost of the product

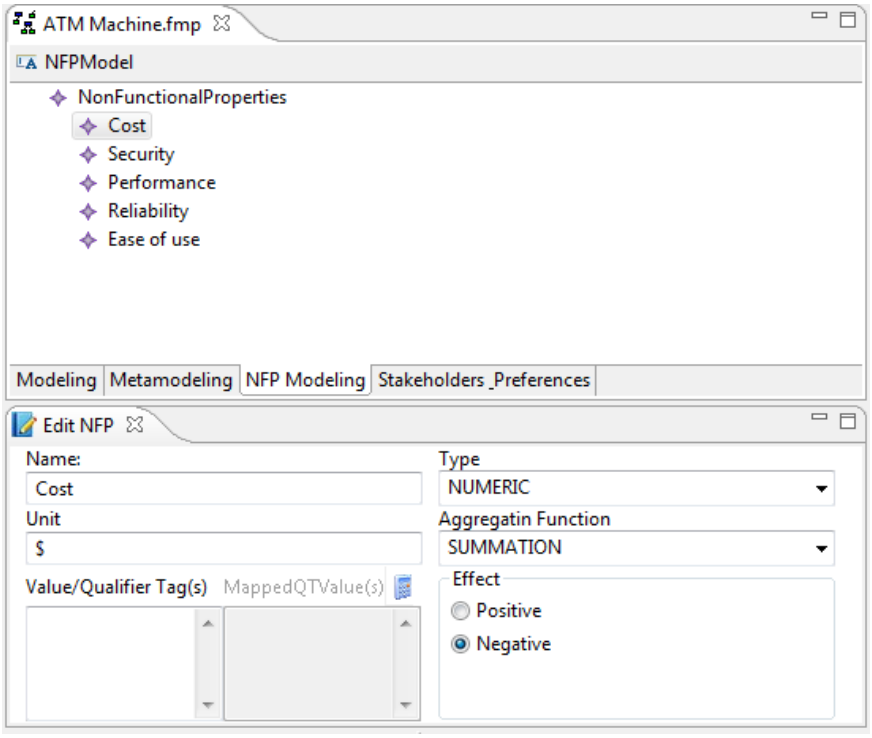



Figure 4.2: NFP model definition view

is calculated by summation of the selected features' cost. To this end, we extend the feature model in such a way that software developers can define non-functional properties as well as functionalities in the feature model. In the developed tool, as shown in Figure 4.2 a user can define the non-functional property model(i.e., all possible non-functional properties in the domain and their characteristics such as type, unit, aggregation function, default values, and so on). After defining the NFP model, features can be annotated with corresponding NFPs and the value of each properties is specified based on the features' implementation.

## 4.2 Managing Stakeholders' Preferences

In the configuration process, the application engineer captures the stakeholders' functional requirements including the required feature set ($F'_A$), the preferences (i.e., the relative importance of non-functional properties $RI$), and the constraints over NFPs ($CO$). By applying S-AHP and the utility function (3.2.1), which are implemented in the developed tool, the ranks of NFPs ($w_i$) and features $R(f_j)$ are computed, respectively. The developed tool has

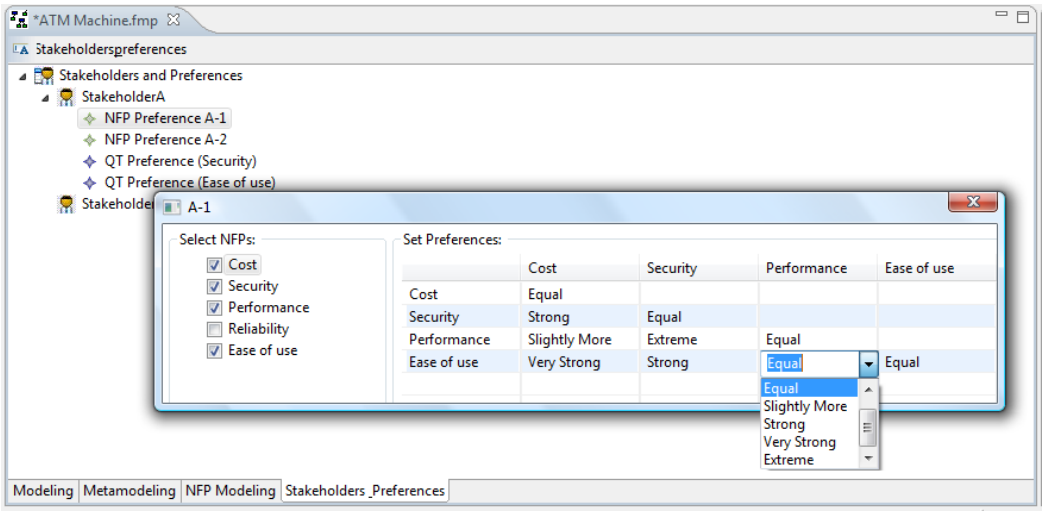



Figure 4.3: A snapshot of the view for managing stakeholders and their preferences

facilities to define stakeholders and managing their preferences (See Figure 4.3).

## 4.3 Configuration Generator

To generate an optimal configuration we employ an efficient planner (i.e. SHOP2), which uses a search-control strategy called ordered task decomposition to perform reasoning on the HTN planning domain [34].

The SHOP2 planner – a domain independent HTN planner – is automatically executed with its required inputs (i.e., HTN planning domain and HTN planning problem). The output of the planner is a set of plans (i.e., sequences of operators for the given tasks), which satisfy the initial conditions and have optimal value. The optimal configuration is achieved by selecting the features corresponding to the operators in the optimal plan.

## 4.4 Visualizing the Result of the Planner

After generating the configurations using the SHOP2 planner, in addition to having an ability to export the results, the results are shown to the stakeholders in a visual view that highlights the selected features and shows the corresponding NFPs along with features. Using the visual view, the stakeholders can navigate the configuration and perform changes in the configuration. Figure 4.4 shows a snapshot of a generated configuration in the visual

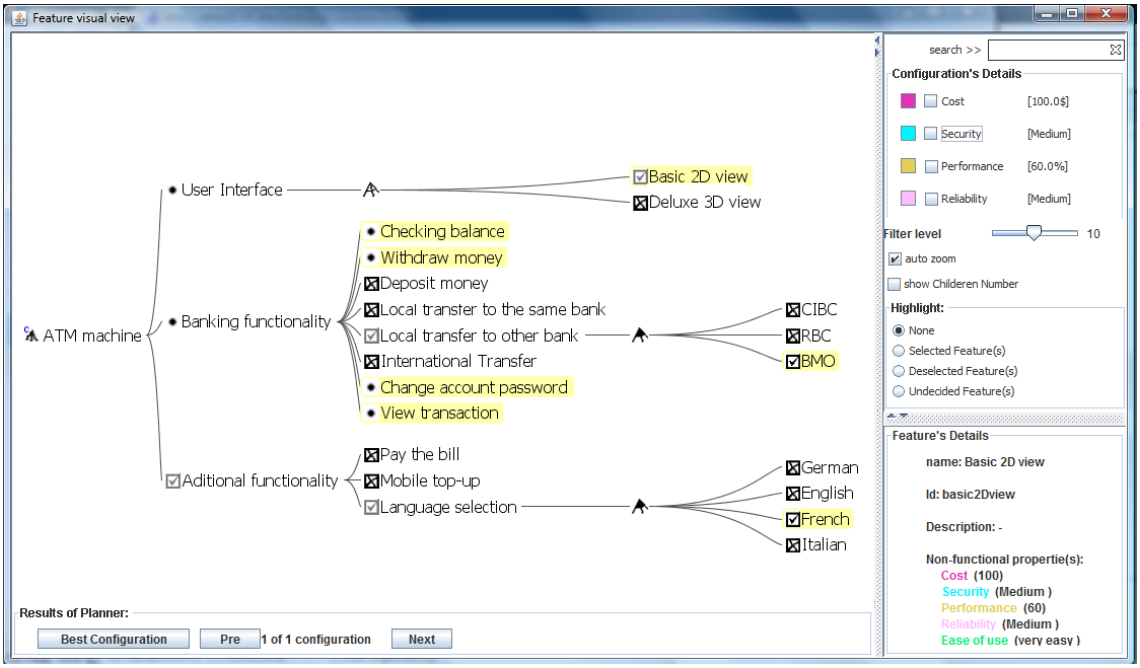



Figure 4.4: The result of planner in the visual view

view. The view not only assists software developers to comprehend the generated configurations better, but it lets users to manipulate the configuration and create necessary changes. In the remainder of this section we discuss more about the visualization and interaction techniques used in the tool.

With respect to the feature model representation, we can either use explicit representation or implicit representation for showing the feature model. In the former, hierarchy relations between nodes are illustrated by explicit links between nodes (i.e., node-link representation). In the latter, the hierarchy relations are shown by special arrangement of nodes [10] (i.e., space-filling representation). Examples of implicit representation are Tree-maps [25] and SunBurst [55]. The cone-trees [42] and space trees [38] are examples of explicit representation.

According to the size of the feature model and weaknesses of implicit drawing for showing large trees, we decided to employ explicit representation. In explicit representation, we have an option of using either 2D space or 3D space. Although 3D visualization, due to the use of three dimensions is more appropriate for showing large scale hierarchies, we preferred using 2D visualization. The reasons for this choice are: 1) the purpose of visualization

was not only showing the feature model, but also we aimed at showing non-functional properties on top of the feature model as well as providing mechanism to configure feature models visually; 2) some researchers addressed that it is not desirable to consider a third dimension when 2D is enough for representing information (i.e., feature model, NFPs, and configuration) [40]; 3) 3D visualization has problems including intensive computation, more complex implementation, hard user adaptation, and orientation [57]; 4) 3D visualization increases difficulty of performing actions and complexity of interactions [60][11]; and 5) finally, occlusion as a common problem in 3D representation may distort software developers during the configuration [13].

To implement the tool, we used prefuse [24], a Java graphic library with different tree and graph views. One of the 2D tree-views in prefuse fulfills all of our requirements discussed above. We used this tree as an initial representation and extended it for our purposes (i.e., visualizing feature model models, NFPs, and configurations). In the used tree, as a layout of this visualization, nodes represent the features of the feature model and links show the composition and variability relations between features.

In information visualization, color is used to group the items and show more information. The literature reviews showed that proper use of color can enhance and clarify a presentation [12][56]. We used color to show the NFPs related to each feature along with the feature. To this end, we assigned predefined colors to each NFPs. However, software developers can interact with the tool and determine desired colors for the NFPs. Having indicated colors corresponding to the NFPs (by the tool or software developers), we apply the coloring on the features of the feature model for showing the NFPs with which they were annotated. Moreover, when software developers select a NFP, Vis-fmp unfolds the features and expands the tree to show all features having the selected NFP. The visual view makes it easy for software developers to see which NFPs are covered with the specific features; which features contain specific NFPs; or how the coverage of NFPs is in the feature model.

In addition to the representation of feature models and their associated NFPs, other visualization and interaction tasks are provided in the visual view including:

1. *Overview*: This technique includes zoomed out views of each data type to see the entire collection along with an attached detail view [48]. In our case, user can see the zoomed out view of a configured feature model with the detail of that configuration. The detail view contains the total value of NFPs which are calculated based on the

selected features in a configured product.

2. *Zooming*: User can zoom in or out on the feature model in order to focus on the specific part of it or view the entire feature model. Moreover, auto-zooming is used to fit the tree to the screen after expanding.

3. *Filtering*: Industrial feature models may contain many features organized in different levels of granularity. Unfolding the whole feature model may cause a visual clutter. Moreover, software developers may not need to view the entire feature model. Therefore, Vis-fmp initially shows high-level features (top three levels). Next, we provide an incremental browsing by two options for the users: 1) Filter-level – software developers can set the level of the feature model tree in which they are interested; then, the tool unfolds all the features between the tree root and the indicated level; and 2) Fisheye Tree Filter – using this option, software developers can explore the feature model by unfolding sub-features of the features they are interested in. When software developers select a feature for further exploration, Vis-fmp closes other nodes in the same level in order to create more space to show the opened nodes. To assist software developers for identifying features require further unfolding, the tool shows the number of children of each feature along with their name.

4. *Highlight*: This technique is employed to highlight the features based on their states (i.e., selected, deselected, or undecided features).

5. *Detail on demand*: In addition to configuration detail , detail of each feature is shown including the general information about the feature and annotated NFPs and their values.

6. *Interactive Configuration*: Software developer can interact with this view by clicking on the features and changing their states (selected, deselected, or undecided). By each change the system updates the detail view of a configuration.

# Chapter 5

# Results and Evaluation

In this chapter we evaluate the proposed approach in terms of scalability and effectiveness. Afterwards, the benefits of the proposed method are discussed via a criteria-based comparison with other existing approaches.

## 5.1 Performance Evaluation

To assess our technique and the corresponding tool *Vis-fmp*, we formulated the following research questions:

- **RQ1 (Scalability)**: Can the approach configure feature models, in a reasonable time, based on functional and non-functional requirements and preferences?
- **RQ2 (Effectiveness)**: How effective is the approach in producing a feature model configuration? **RQ2-1**: Does the approach generate reliable results for application engineers? **RQ2-2**: What is the automation level of the approach?

### 5.1.1 RQ1 (Scalability)

The purpose of this research question is to evaluate whether a feature model configuration can be performed in a reasonable amount of time. Hence, we conducted several experiments to investigate this research question.

### Objects of Study

To evaluate the configuration approach, we adapted Betty FM Generator[1] which enables the random generation of highly-customized feature models [47], to generate feature models with different characteristics (e.g., number of features, probability of mandatory and optional features, probability of OR and XOR groups, and percentage of integrity constraints). We set the characteristics of generated feature models as: 50%, 25%, and 25% for the probability of being features in AND, OR, and XOR groups, respectively. Moreover, 50% of features in AND groups are optional features. The branching factor is also set to 10. These characteristics are backed up by most of surveyed feature models [23][58] to reflect the characteristics of real feature models.

Generating optimal configurations based on stakeholders' preferences and constraints is NP-hard. Hence, HTN planners similar to CSP solvers have problems in finding an optimal solution for large-scale problems. Although the SHOP2 planner applies some heuristics for improving the search time for finding an optimal plan (See ref. [35]), due to explicit representations of states in the memory, SHOP2 runs into memory problems for large domains. However, our experiments showed that for feature models with less than 200 features, SHOP2 returns an optimal plan in a feasible time. According to the results of an investigation on non-functional properties done by Mairiza et al. [31], the number of relevant non-functional properties for different application domains is at most 11. Moreover, Sommerville and Sawyer [54] highlighted that the effective number of non-functional properties is around six. Considering these two studies, we defined 10 non-functional properties; six quantitative; and four qualitative non-functional properties. Five qualifier tags were defined for each non-functional property.

### Experimental Setup

For each feature model, we applied our tool to configure the feature model based on randomly generated preferences and constraints. The evaluation was performed on a computer with an Intel Core DUO 2.2 GHZ CPU, 4GB of RAM, Windows Vista, Java Runtime Environment v5.0, SHOP2 v2.8, and SBCL (Steel Bank Common Lisp) v1.0.55.

To configure the feature models, we considered three independent variables including *number of features*, *number of constraints*, and *integrity constraints*; and *time* as a dependent

[1]Betty Feature Model Generator Version 1.1, http://www.isa.us.es/betty

variable. For each feature model in the study, features were annotated with quantitative non-functional properties and their values were produced by a random function with normal distribution. From a practical point of view, only some of the features may have an impact on a qualitative NFP like security. Hence, to reflect this point, features were randomly annotated with 0 to 4 qualitative non-functional properties with normal distributions, that is, most of the features had one or two non-functional properties. Based on our analysis on SPLOT repository [2], which shows an average of 18% of integrity constraints for real feature models [4], we considered two distributions of integrity constraints: 10% and 20%. Finally, for the constraints over NFPs, four cases were considered: no constraint and constraints over 2, 4, and 6 NFPs to reflect the effect of various constraints over NFPs at run time.

**Experimental Results**

Figure 5.1 illustrates the average time for configuring feature models with different numbers of features and percentages of integrity constraints. Our experiments reveal that, for feature models with less than 200 features, the planner returns results in a feasible time (around 16 second). The number of possible configurations exponentially grows by increasing the number of features. That is, the search space for finding an optimal plan by SHOP2 is boosted by increasing the size of feature models. For example, the total number of expansions in feature models with 150 and 200 features are around 50,000 and 130,000. Therefore, the SHOP2 planner needs more time to return an optimal plan.

We also investigated the effect of constraints over NFPs on the running time of the configuration technique proposed in Section 3. To this end, we run the tool with feature models containing 100 features and 20% integrity constraints. The results are shown in Figure 5.2. In all of these situations, the process of generating optimal configurations was successful.

According to the experiment results, all three independent variables (i.e., *number of features*, *number of constraints*, and *integrity constraints*) have an impact on running time. As shown in Figure 5.1, for the fixed number of integrity constraints, increasing the number of features raises the time for finding an optimal configuration, as increasing the number of features expands the search space for finding an optimal plan. Also, with the same number of features, increasing integrity constraints from 10% to 20% causes an increase in

[2]SPLOT - Software Product Line On-line Tools, http://www.splot-research.org

the time for finding a plan, as the planner needs to check more pair combinations of tasks and operators for optimizing a final plan.

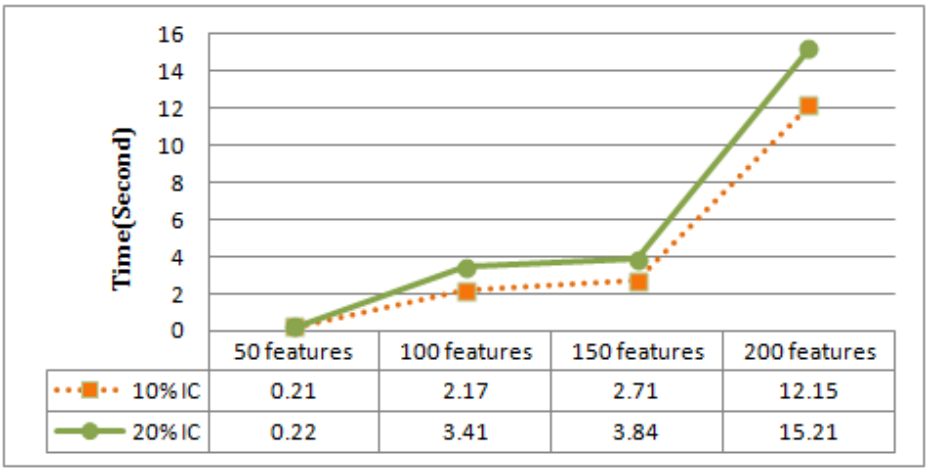



Figure 5.1: Running time of the configuration technique for feature models with different numbers of features and integrity constraints.

Finally, as illustrated in Figure 5.2, the number of constraints over NFPs has an impact on increasing the time for configuration, even if the number of features and integrity constraints is fixed. According to the results of the experiment, the impact of constraints over NFPs is higher than the impact of integrity constraints.

Since generating an optimal feature model configuration based on stakeholders' preferences and constraints is NP-hard, it may need a long time to produce a configuration for large feature models. Moreover, the manual configuration process is very hard and time-consuming task for application engineers. Consequently, the reported time in the Figure 5.1 and 5.2 can be considered as an acceptable time from the perspective of user.

### 5.1.2 RQ2 (Effectiveness)

This research question aims to investigate if application engineers can trust the results returned by the planner and if the approach is beneficial for stakeholders to facilitate their tasks. Regarding the reliability of the results, our approach is based on transforming feature models into the HTN formalisms and applying SHOP2 to find an optimal configuration. According to the framework proposed in [18], we can investigate if this representation can be considered as a good surrogate for representing feature models and preferences. As shown,

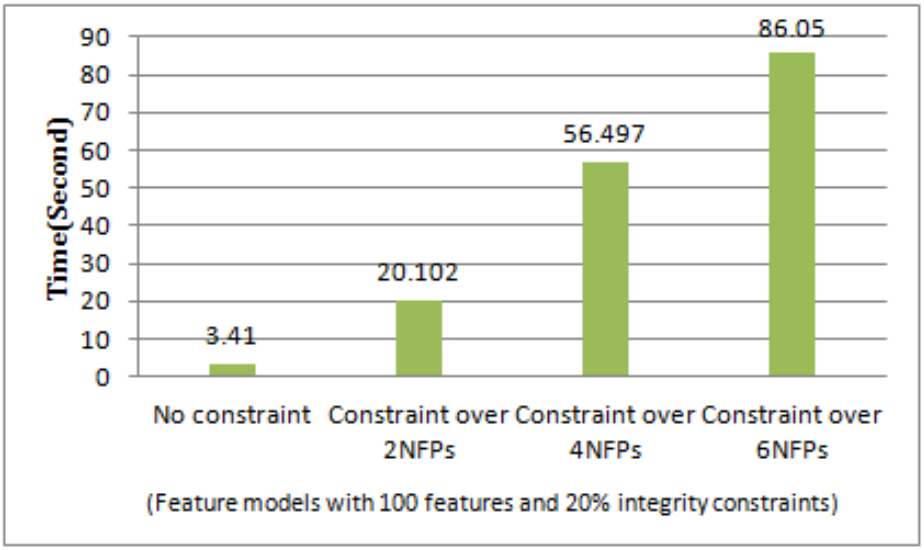



Figure 5.2: Running time of configuration technique based on different number of constraints over NFPs.

we can represent feature model relations and constraints using existing constructs in HTN such as method, tasks, and operator. Additionally, operators provide properties to define a feature rank. The constraints over non-functional properties and integrity constraints can be defined as preconditions of the operators and methods. Hence, we can consider HTN to be a surrogate for feature models. With respect to the ontological commitment, HTN views the world as a set of actions, tasks, constraints between them; this makes it suitable for representing both feature models and constraints. With respect to the planner, SHOP2 has been extensively applied in many projects [34] in government laboratories (e.g., evacuation planning), industry (e.g., evaluating of enemy threats), and academia (e.g., automated composition of Web services), which shows the usefulness of the returned plans and their corresponding configurations.

One way to facilitate the application engineers' tasks during the configuration process is through automating their task. With respect to the automation level, our approach requires only very few manual interventions. The main tasks of the application engineers are: 1) specifying the relative importance of non-functional properties; 2) creating the mapping function for qualitative non-functional properties; and 3) specifying the atomic features that must be included based on the functional requirements. Computing ranks of non-functional properties based on preferences and finding optimal solution are fully automated

in our approach.

In order to investigate the usability of the tool with respect to the employed visualization and interaction techniques, we conducted a controlled experiment. In the experiment we compare the effectiveness of the proposed visualization techniques (implemented in the visual tool) with the basic feature modeling tool (fmp). The results reveal that the employed visualization and interaction techniques significantly improve completion time of comprehension and changing the feature model configuration. More details about the experiment are explained in the technical report [53].

## 5.2 Comparing the Approaches

To systematically compare the proposed approach with other existing approaches, we devise a number of criteria that need to be supported by configuration techniques in order to be effective for application engineering. To define the criteria for systematic comparisons, similar to [46], we applied bottom-up and top-down approaches. Following the bottom-up approach, we identified various important aspects of feature model configuration in the description of existing related works and added them to the criteria set. Following the top-down approach, we used an existing survey on configuration of software product lines. We do not claim that this criteria set is complete, but it provides appropriate aspects to compare our work with related works. These criteria include: 1) Managing functional and non-functional requirements; 2) Modeling stakeholders' preferences; 3) Optimization; 4) Considering stakeholder constraints; 5) Providing tooling support; 6) Automating configuration process; 7) Ensuring the feature model constraints; 8) Effective representation of results to stakeholders; 9) Time efficiency. In the following subsections, we review existing configuration techniques and compare them with respect to the above criteria. Table 5.1 summarizes the results of the comparison of the approaches based on the criteria identified earlier.

**Managing functional and non-functional properties.** Stage configuration [16] and the work in [32] provide no guideline for configuration based on non-functional properties. FCF [61], MUSCLE [62], and the technique from [7] support selection of features only based on quantitative non-functional requirements. Our approach and SPL conqueror [50] guarantee the selection of features based on functional and non-functional properties. Furthermore, only SPL conqueror [50] and our approach consider both qualitative and quantitative non-functional properties.

**Modeling stakeholders' preferences.** CSP based approaches [50][62], FCF [61], and GAFES [23] model stakeholders' preferences in terms of user defined objective functions. Considering the diversity of non-functional properties, it is not easy for stakeholders to define an objective function, which reflects their preferences. However, our approach provides a systematic and easy technique to capture stakeholders' preference in terms of relative importance and defines the objective function using these inputs. To capture stakeholders' preferences we utilize the AHP algorithm which is used in many applications; and it is simple enough, so that stakeholders can use it without any formal training [20]. Stage configuration does not provide any support for stakeholders' preferences.

**Considering stakeholders' constraints.** Stakeholders may define constraints based on the resources that are available to them and level of non-functionality. These constraints need to be considered and only a configuration which satisfies the stakeholders' constraints and optimizes the preferences must be produced. FCF [61], GAFES [23], and CSP based approaches [7][50][62] support constraints on the stakeholders' resources. On the other hand, our approach handles constraints over both qualitative and quantitative non-functional properties.

**Optimization and time efficiency.** Generating optimal configurations based on the stakeholders' preferences and constraints is NP-hard. All CSP approaches [7][50][62] and our approach ensure optimality of the solution, but they require high computation time. To compare the running time of our approach with CSP approaches, we had limitations. The tool developed by Benavides et al [7], called FAMA, does not support optimization which makes it hard to compare with. The SPL conqueror tool is not publicly available and MUSCLE does not have tool support. FCF [61] and GAFES [23] provides partially optimal solutions in a polynomial time. Stage configuration and the work in [32] do not support optimization of the stakeholders' requirements.

**Tooling support and automation.** All the approaches, except FCF, provide tooling support. Stage configuration provides tooling support, but little automation for feature model configuration is provided. With respect to the usability, our approach and SPL conqueror [50] apply visualization techniques to present the configuration results to the stakeholders. Our tool shows the quality level of selected features along with them in the same view (Figure 4.4). Other tools provide basics views for representing configurations to the stakeholders.

**Feature model integrity constraints.** All approaches, except [7] ensure the satisfaction of integrity constraints (i.e., requires and excludes relations) during configuration.

In conclusion, our approach covers all criteria and the only limitation of the approach in comparison with other approaches is the completion time. Due to the NP-hard nature of the feature model configuration problem, among the existing feature model configuration approaches, some have limited scale for generating optimal feature model configurations (e.g., the approach in [7] and our approach) and some require special hardware (e.g., FCF by White et al. [61]) or return partial optimal configurations (e.g., GAFES by Guo et al. [23]). For very large feature models finding an optimal configuration is not feasible because the number of possible configurations has exponential growth. For example, for feature models with $1,000$ number of variation points, $2^{1,000}$ configurations must be evaluated. Although the AI approaches like planning techniques apply heuristics to improve completion time for large problem sizes, we still have restrictions over the size of feature models.

Table 5.1: Comparative analysis of related works ( (+) criterion met, (-) criterion not met, (+/-) criterion partially met).

| Approach \ Criteria | FR/NFR | Preference | Optimization | Constraints | Automation | Integrity constraints | Tooling support | Time efficiency |
|---|---|---|---|---|---|---|---|---|
| Stage Configuration Czarnecki et al. [16] | +/- | - | - | - | - | + | + | - (no result was reported) |
| CSP - Benavides et al. [7][8] | +/- | - | + | + | + | - | + | - (up to 52 features, optimal results) |
| FCF White et al. [61] | +/- | +/- | +/- | + | + | + | - | + (up to 10,000 features, results with 90% optimality) |
| SPL Conqueror Siegmund et al. [50] | + | +/- | + | + | + | + | + | +/- (not exactly mentioned the number of feature) |
| MUSCLE White et al. [62] | +/- | +/- | + | + | + | + | - | +/- (up to 500 features) |
| Mendonca et al. [32] | +/- | - | - | - | + | + | + | + (up to 2,000 features) |
| GAFES Guo et al. [23] | +/- | +/- | +/- | + | + | + | + | + (up to 10,000 features, results with 86-90% optimality) |
| Our approach | + | + | + | + | + | + | + | +/- (up to 200 features, optimal results) |

# Chapter 6

# Conclusion & FutureWork

We target an open research question in software product lines: *how to select a suitable set of features from a feature model based on both the stakeholders' functional and non-functional requirements and preferences?* We investigated the notion of non-functional properties in software product lines and developed a non-functional model in which two main categories of non-functional properties are considered (i.e., quantitative and qualitative NFPs). Feature models were extended with the notion of NFPs. We formalized an extended feature model as an Hierarchical Task Network so that features and relations among features in the feature model were mapped into the components of the HTN planning domain (i.e., operator, task, and method). This transformation covers all relations in the feature model including: *AND*, *OR*, and *XOR* decompositions as well as *require* and *exclude* relations among features. We also formalized the configuration problem as an HTN planning problem and employed an existing HTN planner (SHOP2) to generate an optimal configuration based on the stakeholder preferences and constraints over non-functional properties.

The proposed feature model configuration approach has the following improvements in comparison with other existing approaches:

1) In addition to functional requirements, constraints over both qualitative and quantitative non-functional properties are taken into account; 2) we capture the preferences of stakeholders in terms of relative importance between non-functional properties which is an easy way for stakeholders to specify; 3) we calculate the weight of NFPs by utilizing SAHP as a weighting algorithm, which significantly reduces the number of needed pairwise comparisons and does not need too much effort from stakeholders; 4) we introduce the concept of artificial intelligence planning in the context of feature model configuration and provide

the transformation rules to convert feature models into Hierarchical Task Network; 5) we developed a tool which automates feature model configuration process and provides several visualization and interaction techniques to facilitate the configuration process; 6) the experimental results revealed that our approach can be useful because: *i*) it provides an optimal configuration based on stakeholders preferences and constraints over non-functional properties; and *ii*) it has a good performance on feature models with less than 200 features.

This work can be extended in several directions:

- For larger feature models, the approach is computational demanding, similar to other related approaches in the literature. Another HTN planner (called HTNPLAN-P [51]) applies new heuristics and can support larger planning space problems, so that the practical performance time can be much improved as shown in [51]. Since HTNPLAN-P is not yet publicly available, we could not test our work on larger feature models. In the future work, HTNPLAN-P can be employed for finding optimal plans which will improve scalability of our approach.

- Other planning techniques such as PDDL [21](i.e., Planning Domain Definition Language) based approach can be employed. A wide range of planning engines support PDDL; if the feature model can be successfully transformed into PDDL, we can utilize more powerful planning engines. Moreover, PDDL supports optimization so that the specific metrics (i.e., NFPs in our context) can be minimized or maximized during plan generation.

- According to our initial study, effective visualization techniques improve the configuration process and ease the task for application engineers. Therefore, more interaction and visualization techniques can be applied on the tool and an experimental study can be done in an industrial environment to examine the effects of the applied techniques.

# Bibliography


[1] Variant management with pure:: variants. Technical report, Pure-systems GmbH, 2003.

[2] Mohsen Asadi, Ebrahim Bagheri, Dragan Gašević, Marek Hatala, and Bardia Mohabbati. Goal-driven software product line engineering. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 691–698. ACM, 2011.

[3] Ebrahim Bagheri, Mohsen Asadi, Dragan Gasevic, and Samaneh Soltani. Stratified analytic hierarchy process: Prioritization and selection of software features. In Jan Bosch and Jaejoon Lee, editors, *SPLC*, volume 6287 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 2010.

[4] Ebrahim Bagheri and Dragan Gasevic. Assessing the maintainability of software product line feature models using structural metrics. *Software Quality Journal*, 19(3):579–612, 2011.

[5] Don Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *SPLC*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.

[6] Don S. Batory, Roberto E. Lopez-Herrejon, and Jean-Philippe Martin. Generating product-lines of product-families. In *ASE*, pages 81–92. IEEE Computer Society, 2002.

[7] David Benavides, Pablo Trinidad Martn-Arroyo, and Antonio Ruiz Corts. Automated reasoning on feature models. In Oscar Pastor and Joo Falco e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 491–503. Springer, 2005.

[8] David Benavides, Sergio Segura, Pablo Trinidad Martn-Arroyo, and Antonio Ruiz Corts. Using java csp solvers in the automated analyses of feature models. In Ralf Lmmel, Joo Saraiva, and Joost Visser, editors, *GTTSE*, volume 4143 of *Lecture Notes in Computer Science*, pages 399–408. Springer, 2006.

[9] Jan Bosch. *Design and use of software architectures - adopting and evolving a product-line approach.* Addison-Wesley, 2000.

[10] Goetz Botterweck, Steffen Thiel, Daren Nestor, Saad bin Abid, and Ciarán Cawley. Visual tool support for configuring and understanding software product lines. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC '08, pages 77–86. IEEE Computer Society, 2008.

[11] Doug A. Bowman, Ernst Kruijff, Joseph J. LaViola, and Ivan Poupyrev. An introduction to 3-d user interface design. *Presence: Teleoper. Virtual Environ.*, 10(1):96–108, February 2001.

[12] Brewer C. Color use guidelines for data representation. In *Proceedings of the Section on Statistical Graphics, American Statistical Association*, pages 55–60, 1999.

[13] Mei C. Chuah, Steven F. Roth, Joe Mattis, and John Kolojejchick. Sdm: selective dynamic manipulation of visualizations. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, UIST '95, pages 61–70. ACM, 1995.

[14] Paul C. Clements and Linda Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison-Wesley, August 2001.

[15] K. Czarnecki and P. Kim. Cardinality-Based Feature Modeling and Constraints: A Progress Report. In *Proceedings of the International Workshop on Software Factories at OOPSLA 2005*. ACM, 2005.

[16] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[17] Krzysztof Czarnecki and Eisenecker Ulrich. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[18] H. Schrobe R. Davis and P. Szolovits. What is knowledge representation? *AI Magazine*, 14(1):17–33, 1993.

[19] Huáscar Espinoza, Hubert Dubois, Sébastien Gérard, Julio Medina, Dorina C. Petriu, and Murray Woodside. Annotating uml models with non-functional properties for quantitative analysis. In *Proceedings of the 2005 international conference on Satellite Events at the MoDELS*, MoDELS'05, pages 79–90. Springer-Verlag, 2006.

[20] Ernest H. Forman and Saul I. Gass. The Analytic Hierarchy Process: An Exposition. *Operations Research*, 49(4):469–486, July 2001.

[21] Malik Ghallab, Craig K. Isi, Scott Penberthy, David E. Smith, Ying Sun, and Daniel Weld. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998.

[22] M. Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE '07. 15th IEEE International*, pages 21 –26. IEEE, oct. 2007.

[23] Jianmei Guo, Jules White, Guangxin Wang, Jian Li, and Yinglin Wang. A genetic algorithm for optimized feature selection with resource constraints in software product lines. *Journal of Systems and Software*, 84(12):2208–2221, December 2011.

[24] Jeffrey Heer, Stuart K. Card, and James A. Landay. prefuse: a toolkit for interactive information visualization. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, CHI '05, pages 421–430. ACM, 2005.

[25] Brian Johnson and Ben Shneiderman. Tree-maps: a space-filling approach to the visualization of hierarchical information structures. In *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291. IEEE Computer Society Press, 1991.

[26] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[27] Kyo C. Kang, Sajoong Kim, Jaejoon Lee, Kijoo Kim, Gerard Jounghyun Kim, and Euiseob Shin. Form: A feature-oriented reuse method with domain-specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.

[28] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. Feature-oriented product line engineering. *IEEE Software*, 19:58–65, 2002.

[29] Sotirios Liaskos, Sheila A. McIlraith, Shirin Sohrabi, and John Mylopoulos. Integrating preferences into goal models for requirements engineering. In *Proceedings of the 2010 18th IEEE International Requirements Engineering Conference*, RE '10, pages 135–144. IEEE Computer Society, 2010.

[30] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., 2007.

[31] Dewi Mairiza, Didar Zowghi, and Nurie Nurmuliani. An investigation into the notion of non-functional requirements. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 311–317. ACM, 2010.

[32] Marcilio Mendonca, Andrzej Wasowski, Krzysztof Czarnecki, and Donald Cowan. Efficient compilation techniques for large scale feature models. In *Proceedings of the 7th international conference on Generative programming and component engineering*, GPCE '08, pages 13–22. ACM, 2008.

[33] Bardia Mohabbati, Dragan Gasevic, Marek Hatala, Mohsen Asadi, Ebrahim Bagheri, and Marko Boskovic. A quality aggregation model for service-oriented software product lines based on variability and composition patterns. In *The 9th International Conference on Service Oriented Computing (ICSOC 2011)*, pages 436–451. Springer, 2011.

[34] Dana Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, Hector Munoz-Avila, J. William Murdock, Dan Wu, and Fusun Yaman. Applications of shop and shop2. *IEEE Intelligent Systems*, 20(2):34–41, March 2005.

[35] Dana Nau, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. Shop2: An htn planning system. *Journal of Artificial Intelligence Research*, 20:379–404, 2003.

[36] Ivana Ognjanovic, Dragan Gašević, Ebrahim Bagheri, and Mohsen Asadi. Conditional preferences in software stakeholders' judgments. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, SAC '11, pages 683–690. ACM, 2011.

[37] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15:1053–1058, December 1972.

[38] Catherine Plaisant, Jesse Grosjean, and Benjamin B. Bederson. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In Pak Chung Wong and Keith Andrews, editors, *INFOVIS*, pages 57–64. IEEE Computer Society, 2002.

[39] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques*. Springer, 2005.

[40] Helen C. Purchase. Which aesthetic has the greatest effect on human understanding? In *Proceedings of the 5th International Symposium on Graph Drawing*, GD '97, pages 248–261. Springer-Verlag, 1997.

[41] Rick Rabiser, Paul Grünbacher, and Deepak Dhungana. Requirements for product derivation support: Results from a systematic literature review and an expert survey. *Information and Software Technology*, 52(3):324–346, March 2010.

[42] George G. Robertson, Jock D. Mackinlay, and Stuart K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194. ACM, 1991.

[43] Florian Rosenberg, Predrag Celikovic, Anton Michlmayr, Philipp Leitner, and Schahram Dustdar. An end-to-end approach for qos-aware service composition. In *Proceedings of the 13th IEEE International Conference on Enterprise Distributed Object Computing*, EDOC'09, pages 128–137. IEEE Press, 2009.

[44] Thomas L. Saaty. How to make a decision: The analytic hierarchy process. *European Journal of Operational Research*, 48(1):9–26, September 1990.

[45] T.L. Saaty. *The Analytic Hierarchy Process, Planning, Piority Setting, Resource Allocation*. McGraw-Hill, 1980.

[46] Klaus Schmid, Rick Rabiser, and Paul Grünbacher. A comparison of decision modeling approaches in product lines. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '11, pages 119–126. ACM, 2011.

[47] Sergio Segura, José A. Galindo, David Benavides, José A. Parejo, and Antonio Ruiz-Cortés. Betty: benchmarking and testing on the automated analysis of feature models. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS '12, pages 63–71. ACM, 2012.

[48] Ben Shneiderman. The eyes have it: a task by data type taxonomy for information visualizations. *Proceedings 1996 IEEE Symposium on Visual Languages*, 0(UMCP-CSD CS-TR-3665):336–343, 1996.

[49] Norbert Siegmund, Marko Rosenmuller, Christian Kastner, Paolo G. Giarrusso, Sven Apel, and Sergiy S. Kolesnikov. Scalable prediction of non-functional properties in software product lines. In *Proceedings of the 2011 15th International Software Product Line Conference*, SPLC '11, pages 160–169. IEEE Computer Society, 2011.

[50] Norbert Siegmund, Marko Rosenmüller, Martin Kuhlemann, Christian Kästner, Sven Apel, and Gunter Saake. SPL Conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, pages 1–31, June 2011.

[51] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. Htn planning with preferences. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1790–1797. Morgan Kaufmann Publishers Inc., 2009.

[52] Shirin Sohrabi, Jorge A. Baier, and Sheila A. McIlraith. Htn planning with preferences. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 1790–1797. Morgan Kaufmann Publishers Inc., 2009.

[53] Samanh Soltani, Mohsen Asadi, Dragan Gasevic, and Marek Hatala. The effects of visualization and interaction techniques on feature model configuration. Technical report, School of Intractive Art and Technology, SFU, 2011. `https://files.semtech.athabascau.ca/public/TRs`.

[54] Ian Sommerville, Ian Sommerville, Pete Sawyer, and Pete Sawyer. Viewpoints: principles, problems and a practical approach to requirements engineering. *Annals of Software Engineering*, 3:101–130, 1997.

[55] John Stasko and Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *Proceedings of the IEEE Symposium on Information Vizualization 2000*, INFOVIS '00, pages 57–. IEEE Computer Society, 2000.

[56] Maureen Stone. Choosing colors for data visualization, 2006. `http://www.perceptualedge.com/articles/b-eye/choosing_colors.Pdf`.

[57] Alfredo R. Teyseyre and Marcelo R. Campo. An overview of 3d software visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):87–105, January 2009.

[58] Thomas Thum, Don Batory, and Christian Kastner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE '09, pages 254–264. IEEE Computer Society, 2009.

[59] E. Triantaphyllou, B. Shu, S. Nieto Sanchez, and T. Ray. *Multi-Criteria Decision Making: An Operations Research Approach*, volume 15, pages 175–186. 1999.

[60] Andries van Dam, Kenneth P. Herndon, and Michael Gleicher. The challenges of 3d interaction. In Catherine Plaisant, editor, *CHI Conference Companion*, page 469. ACM, 1994.

[61] Jules White, Brian Dougherty, and Douglas C. Schmidt. Selecting highly optimal architectural feature sets with filtered cartesian flattening. *Journal of Systems and Software*, 82(8):1268–1284, August 2009.

[62] Jules White, Brian Dougherty, Doulas C. Schmidt, and David Benavides. Automated reasoning for multi-step feature model configuration problems. In *Proceedings of the 13th International Software Product Line Conference*, SPLC '09, pages 11–20. IEEE, 2009.

[63] Tao Yu and Kwei-Jay Lin. Service selection algorithms for web services with end-to-end qos constraints. In *Proceedings of the IEEE International Conference on E-Commerce Technology*, CEC '04, pages 129–136. IEEE Computer Society, 2004.

[64] Tao Yu and Kwei-Jay Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *Proceedings of the Third international conference on Service-Oriented Computing*, ICSOC'05, pages 130–143. Springer-Verlag, 2005.

[65] Pamela Zave. *An experiment in feature engineering*, pages 353–377. Springer-Verlag New York, Inc., 2003.