MASTER

Architecture variability and multi-criteria optimization

Zhao, J.

Award date:
2015

Link to publication

Disclaimer
This document contains a student thesis (bachelor’s or master’s), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
Architecture Variability and Multi-criteria Optimization

Author:
Jianfeng Zhao

Supervisor TU/e:
Prof. Johan LUKKIJEN

Supervisor TNO-ESI:
Bart THEELEN

Supervisor Philips:
Martijn OPHEIJ

Supervisor TUD
Nicolas DINTZNER

Department of Mathematics and Computer Science
System Architecture and Networking

Eindhoven University of Technology

August 18, 2015
This work was supported in part by the European Union’s ARTEMIS Joint Undertaking for CRYS\-\-TAL – Critical System Engineering Acceleration – under grant agreement No. 332830.
Abstract

Philips Image Guided Therapy systems are used for diagnostic and interventional treatment of heart and vascular diseases. To meet various medical needs in very different markets, Philips offers a wide range of product variants to their customers. And the system complexity and size are increasing over time. With complex market and system data, manual process is insufficient to evaluate all product variants in the system with huge amount of variabilities. A tool that can automate the evaluation process is needed by Philips. We address the needs of Philips using variability management and multi-criteria optimization techniques.

We formalize the problem in the Philips Image Guided Therapy domain to explore variability in the architecture, and implement a tool to automate the evaluation process to give the optimization result of user concerned criteria. The tool proposed in this thesis allows users to represent the product information and market data in a compact and consistent manner, and it offers the possibility to support automatic visualization and comparison of product variants. Finally we discuss the possible future work which can make the tool more powerful.
Acknowledgements

I would like to express my sincere gratitude to my supervisor at the university, Prof. Johan Lukkien for introducing me to this project, and for his support and guidance throughout the thesis work. Without his insights and support, the work would not be possible.

I want to thank my internship supervisors Bart Theelen and Martijn Opheij for giving me the opportunity to have this meaningful experience and for the valuable comments and technical support in the implementation.

I would like to express my sincere appreciation to my daily supervisor Nicolas Dintzner for his guidance and help in all aspects of the internship work.

Lastly my thanks and appreciations go to my family and friends for all their support.
# Contents

Abstract iii

Acknowledgements iv

List of Figures vii

List of Tables ix

Abbreviations xi

1 Introduction 1
   1.1 Image Guided Therapy Systems ........................................... 1
   1.2 Problem Description ....................................................... 4
       1.2.1 Architecture Sweetspot ............................................. 5
       1.2.2 Focus Area ............................................................ 7
   1.3 Outline ............................................................................. 9

2 Literature Review 10
   2.1 Variability Management ..................................................... 10
   2.2 Multi-criteria Optimization ................................................ 11

3 Formalization 13
   3.1 Domain model .................................................................... 13
   3.2 Instantiation ...................................................................... 15
   3.3 General definitions ........................................................... 16

4 Case Study 19
   4.1 Dataset .............................................................................. 20
   4.2 Data processing .................................................................. 22
       4.2.1 Manual evaluation process ............................................ 23

5 Tool Design and Implementation 25
   5.1 Solution ............................................................................. 25
   5.2 Design decisions ................................................................. 27
       5.2.1 Feature Model ............................................................. 28
       5.2.2 Product Domain Language ........................................... 30
       5.2.3 Grammar of the Product Domain Language ...................... 33
   5.3 Implementation .................................................................... 36
   5.4 Multi-criteria optimization .................................................. 38
       5.4.1 Weighted Sum Algorithm .............................................. 39
       5.4.2 Dominance and Pareto Ranking ..................................... 41
       5.4.3 Discussion ................................................................. 43
# List of Figures

1.1 An example of IGT systems .............................................. 2
1.2 AD7 tiltable patient table of IGT .................................... 3
1.3 AD7 non-tiltable patient table of IGT ................................ 3
1.4 Architecture sweet spot .................................................. 6
1.5 Problem diagram in the IGT domain ................................... 7
1.6 An example of the problem diagram in the IGT domain .......... 8

3.1 Domain model ............................................................ 14
3.2 An instance of the domain model ...................................... 16
3.3 Generalized formalization ................................................. 17

4.1 Components and composition rules .................................... 19
4.2 Configuration Property .................................................. 20

5.1 Development view ........................................................ 28
5.2 An feature model example in FeatureIDE ............................ 30
5.3 Overview of DSL framework ............................................. 32
5.4 Concrete Syntax example ................................................ 32
5.5 Abstract Syntax example ............................................... 33
5.6 Textual Input example ................................................... 33
5.7 Model example ............................................................ 34
5.8 Grammar for expressing component, component properties, and composition rules ........................................... 35
5.9 Grammar for expressing evaluation rules ............................ 35
5.10 Grammar for expressing market information and fulfillment rules .................................................... 36
5.11 The flow chart ............................................................. 37
5.12 The sequence diagram .................................................... 38
5.13 Rules to evaluate a configuration regarding a user need .... 40
5.14 Rules to evaluate a configuration property ....................... 40
5.15 Pareto ................................................................. 42

6.1 Select concerned market segments .................................... 44
6.2 Merged user needs ......................................................... 44
6.3 Output of the tool ........................................................ 45
6.4 Visualization .............................................................. 46
6.5 Comparison of configuration 2 and configuration 3 .......... 47
6.6 An example of complex expressions .................................. 48
6.7 Feature Model of AD7 patient table ................................... 49

8.1 A DSL instance validation example ................................... 52
List of Tables

4.1 Component properties ........................................... 21
4.2 Evaluation rules for configuration non-functional properties .... 21
4.3 Evaluation rules for configuration functional properties .......... 21
4.4 User needs and associated evaluation rules ...................... 22
4.5 Importance of user needs in different market segments .......... 22
4.6 Valid configurations ........................................... 23
4.7 Examples of invalid configurations ............................. 23
4.8 Configurations and related properties ........................... 23
4.9 Fulfillment results ............................................. 24
5.1 Importance values of multiple criteria .......................... 41
5.2 The criterion rank of configuration 1 ........................... 41
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IGT</td>
<td>Image Guided Therapy</td>
</tr>
<tr>
<td>STT</td>
<td>Standard Tabletop</td>
</tr>
<tr>
<td>ETT</td>
<td>Extended Tabletop</td>
</tr>
<tr>
<td>TB</td>
<td>Tablebase</td>
</tr>
<tr>
<td>SDF</td>
<td>Standard Floorplate</td>
</tr>
<tr>
<td>SWF</td>
<td>Swivel Floorplate</td>
</tr>
<tr>
<td>LM</td>
<td>Longitudinal Motorized Option</td>
</tr>
<tr>
<td>E&amp;M</td>
<td>Economizers &amp; Minimizalists</td>
</tr>
<tr>
<td>QO</td>
<td>Quality Optimizers</td>
</tr>
<tr>
<td>PDL</td>
<td>Product Domain Language</td>
</tr>
<tr>
<td>DR</td>
<td>Data Repository</td>
</tr>
</tbody>
</table>
1 | Introduction

As the market leader in systems for Image Guided Therapy (IGT), Philips aims to provide integrated solutions that advance minimally invasive medical procedures by helping doctors to decide, guide and confirm the right therapy for the right patient at the point of care. To meet various medical needs in very different markets, Philips offers a wide range of product variants to their customers. While offering such a wide range of products helps to increase customer satisfaction, each variant of the product must be designed, developed and tested which adds to the overall development costs and time. The number of variants increases when some medical needs change or when a new market is identified. After several iterations, it becomes interesting to assess whether the set of available configurations developed and maintained by Philips is "optimal" with respect to customer satisfaction and maintenance and engineering costs associated with them. A configuration represents a specific product variant. A formal definition of configuration will be introduced in Chapter 3. Maintaining "optimal" configurations can reduce time to markets and total costs.

In this thesis, we explore the means to identify the optimal set of configurations of Philips IGT systems to determine the cost of architecture alternatives with respect to several evaluation criteria which are possibly conflicting with each other. The usage of multi-criteria optimization is to facilitate the identification of the optimal set of configurations as a means to optimize the architecture of the system by reconstructing and optimizing variabilities. A tool is needed to give the architect insights in different configurations and their associated costs and values, and to find the optimal set of configurations which balances different costs given certain criteria.

1.1 Image Guided Therapy Systems

Philips IGT systems are used for diagnostic and interventional treatment of heart and vascular diseases. IGT systems are designed to help doctors perform different medical operations on both adults and children in various countries. An IGT system example
is shown in Figure 1.1. There are 50 basic modules and around 250 sub-modules in an IGT system. Each module has its own variants which result into many possibilities that increase the complexity of the system. Figure 1.2 shows one module of IGT included in Figure 1.1 - the patient table. The patient table is available in several versions such as a tiltable version as shown in Figure 1.2 and a non-tiltable version as shown in Figure 1.3. Both tables belong to the AD7 patient table module of IGT. The AD7 patient table contains 15 possible components, and now has 30 basic configurations that are available on the market. The dataset we will use in Chapter 4 is derived from it.

Currently the IGT product line covers 12 basic product variants. A system product line is a group of related products manufactured by a single company, all of which share some core functionality, yet each of which differs in some specific features (Henard, Papadakis, Harman, and Le Traon, 2015). So available products decide both the capability to fulfill various user needs and market segments, and the total cost.

For a medical product, the design and testing procedure of each variant must follow strict validation procedures imposed by market regulation, e.g., F.D.A. in USA market.
Figure 1.2: AD7 tiltable patient table of IGT

Figure 1.3: AD7 non-tiltable patient table of IGT
Because of limited resources and time constraints, what products to build is usually decided on forehand which is neither adaptive to changes nor optimized for all market segments.

1.2 Problem Description

System complexity and size are increasing over time and in the meantime the innovation speed is also accelerating. Variability management is a driving force for companies to be more competitive in the market. Capilla, Bosch, and Kang (2013) indicate that the balance of power is increasingly shifting to the customer and the ability of the customer to demand products that specifically address customer-specific adaptations to the products. This has lead to a situation where many companies are stuck in a fire-fighting mode where the cost of developing new products increases constantly due to increased system size and complexity while, on the other hand, the number of products and customer-specific adaptations required increases constantly.

The forces above are faced by many companies. To maintain a competitive position, a company has to invest more into R&D. But more investment does not always work out since the market is changing so fast that a long term development is no longer suitable and competitive. In a competitive market, it is essential to reduce time to markets and maintain product quality. Also different user needs from various market segments require different product specifications. Using system product line to manage products is a countermeasure to this problem. The architecture defines how the various products derived from the product line can vary. According to Capilla et al. (2013), a key success factor of system product lines is that it addresses business, architecture, process and organizational aspects of effectively sharing system assets within a portfolio of products.

Variability management is in this case defined in two aspects: product aspect and process aspect. The product aspect focuses on how to manage all variabilities of the product. For instance in the patient table module we described in the previous section, its variability pool contains all components which can be included into a table product. The main concern here is to find the most efficient way of constructing the variability pool. E.g., some variabilities always appear together, so they can be grouped into one variability; or one variability provides different functions in different configurations, so it may be
suggested to be split into multiple ones. And concerns regarding the fabrication and maintenance also affect the construction of the variability pool. In the process aspect, the concern is to find the optimal variability set, i.e., a configuration, to construct the product which can fulfill various requirements. To find the optimal variability set, we use multi-criteria optimization for ranking qualified configurations with regard to user needs of different market segments. Also the results of the process aspect can be fed to the product aspect as feedback to give insights about the variability pool.

For targeted market segments, there are one or more qualified configurations which satisfy multiple criteria with different levels. Multi-criteria optimization aims to find and rank those configurations regarding multiple, possibly conflicting criteria. So the results of multi-criteria optimization are helpful for Philips and its customers making their decisions. An extreme situation can be that each market segment is best suited by a unique configuration. Technically, all variations of the product can be produced, but as we mentioned in Chapter 1, high diversity requires more development time, cost and maintenance efforts of the company. So it is common to release only a limited number of product variants on the market. Each product variant is composed by a set of components, i.e. a configuration. We refer to all configurations that can be mapped to product variants on the market as the configuration set. Given a system with various variabilities, whether a configuration should be included into the configuration set is a decision to balance market needs against development costs, time, and maintenance efforts. And which configuration is the most suitable one for specific market segments is a decision based on the knowledge of market information and user needs. If the set of offered configurations is optimal with respect to cost, benefit, and user needs perspectives, it is referred to as the Architecture Sweetspot by system architects of Philips.

1.2.1 Architecture Sweet spot

To illustrate the idea of the Architecture Sweetspot more explicitly, we use Figure 1.4 from Opheij (2014) as a reference. The X-axis stands for the number of unique configurations, and the Y-axis stands for the value and cost. The blue line in Figure 1.4 stands for the total cost related to development, test & release, manufacturing and sustaining costs. Adding more unique configurations will increase the total cost since all relevant costs are increased. The green line represents the total value related to sales. With more
available configurations, customers who need certain optional functionality appreciate it and are willing to pay extra for it (added value), while customers who do not need it are not served with this option (reducing total material cost) without lowering the perceived value. The sales can be increased because of the customization. Because there exists the market threshold, the total value will tend to be steady after a certain number of configurations. Finally the red line stands for the benefit the company can get.

We summarize the information as follows:

- **Total cost**: is sum of all costs related to a product line (this includes development cost, material cost, manufacturing overhead)
- **Total value**: represents the sales value for the product domain
- **Benefit**:

  \[
  Benefit = Total\ value - Total\ cost
  \]  
  \hspace{1cm} (1.1)

![Figure 1.4: Architecture sweetspot](image)

The equation in 1.1 shows the calculation of the benefit which is the main concern of Philips. The company always wants to get more benefit. So Philips wants to find the Architecture Sweetspot because the Architecture Sweetspot denotes the configuration set at which the benefit in Figure 1.4 is maximal. The configuration set needs to have the optimal number of configurations which lowers the cost, and also needs to include
right configurations which can satisfy multiple user needs of different customers. We also refer to this configuration set as the optimal configuration set.

1.2.2 Focus Area

The optimal configuration set we discussed in the previous section is influenced by many factors. It has to contain configurations that satisfy multiple criteria, such as matching user needs, minimizing product cost and satisfying technical and marketing constraints. Figure 1.5 shows the connection between market information and product selection. The properties of a product is decided by the configuration. In this thesis, a product refers to a certain configuration of components. Market information expresses that each market requires several user needs with corresponding importance values. Each product satisfies different user needs with corresponding scores. Figure 1.6 shows an example where 3 products P1, P2 and P3 serve 2 markets M1 and M2. The 2 markets require 4 user needs UN1, UN2, UN3 and UN4. Market M1 requires user need UN1 with an importance of i1.1. Product P3 satisfies user need UN4 with a score of s3.4.

![Figure 1.5: Problem diagram in the IGT domain](image)
The information in Figure 1.6 is used to evaluate products by their fulfillment regarding user needs of the concerned markets. Each product variant will get a final score with regard to each market. The product with the highest final score for a certain market is the most suitable one. This is the process to evaluate a product for a certain market. For multiple markets, the user needs will be merged into a bigger set. The details of this approach will be explained in Chapter 5.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure1_6.png}
\caption{An example of the problem diagram in the IGT domain}
\end{figure}

The evaluation process discussed above is conducted manually by Philips system architects together with marketeers based on experience from the product predecessors and market data. With more complex market and system data, manual calculation is insufficient to evaluate all configurations in a system with huge amount of variabilities. So system architects of Philips want to have an automated tool to generate valid configurations, rank configurations with respect to known market segments and user needs and also give suggestions on the management of variability. From the output of such a tool, the optimal configuration set which can maximizes the benefit can be inferred. The tool should be generalized to allow them to model different products. Since in industrial
size product lines, thousands of components can be involved and a massive amount of configurations are needed, the results proposed by such a tool should be visualized so that Philips architects can have an intuitive comparison among configurations. So in this thesis we will develop a tool to explore architecture variability in the IGT domain and the output of it can be used to find the Architecture Sweetspot.

1.3 Outline

In Chapter 2 a literature survey is conducted. And in Chapter 3, the problem is formalized based on the data and requirements of Philips system architects and marketeers. Then the formalization is instantiated with a dataset of the patient table which is one module of an IGT system in Chapter 4. In Chapter 5 the implementation of the tool and design decisions are discussed. The tool is tested with the dataset as input and the results are discussed in the following Chapter 6. The conclusions of the project are made in Chapter 7. In the last Chapter 8 possible future work is discussed.
2 | Literature Review

2.1 Variability Management

Variability management is a problem that companies face when reducing development cost and maintenance efforts. It is modeled and investigated by many organizations and research groups. Researchers Hallerbach, Bauer, and Reichert (2007) of Daimler AG design a framework called Process Variants by Options (ProVop) that addresses variability issues in the domains of automotive and healthcare. Researchers Dadam and Reichert (2009) at University of Ulm provide a framework for handling of different kinds of run-time variability in the ADEPT project. In Pesic, Schonenberg, Sidorova, and van der Aalst (2007), another framework called DECLARE is proposed for a constraint-based process modeling language and its implementation by researchers of Eindhoven University of Technology.

According to the work in Aiello, Bulanov, and Groefsema (2010), tools and frameworks for variability management can be categorized by the expressive power for expressing variability. Explicit variability management means the ability to express the possible variations. The expressive power of a tool provides an indication of what must be possible to express the variability management (Aiello et al., 2010). The expressive power contains many requirements, among which the requirement of Constraint Expressions are most relevant to the tool that Philips wants to have. The most relevant constraint expressions are the following ones:

- Mandatory: specify which variability that must be included;
- Prerequisite: specify the dependency between two specific variabilities, such as, if A is included then B must also be included in the process;
- Exclusion: specify the dependency between two specific variabilities, such as, if A is included then B must not be included.
• Substitution: specify the implied substitution, such as, if A is not included then B must be included;

• Corequisite: specify the tight relation between two specific variabilities, such as, A and B must be either included or excluded.

The above 6 constraint expressions are important to express variability in IGT systems. But the frameworks Provop and ADEPT are tools focusing on the expressive power regarding structural changes of a system instead of constraints expressions. DECLARE is more powerful in the constraint expressions aspect but it does not cover all the 6 constraint expressions above.

Philips system architects express the constraints information in the system definition diagram as in the Appendix. The dataset we will use in Chapter 4 shows the necessity of supporting those constraint expressions by the tool.

2.2 Multi-criteria Optimization

The composition of a configuration is well-defined after defining the architecture variability in the Philips IGT systems. With several configurations available, a customer has to choose one which suits his/her needs best. These user needs are the multiple criteria which affect the selection of configurations. In addition, several criteria may be in conflict with each other (e.g., lower cost and more functions).

In 1881, King’s College (London) and later Oxford Economics Professor F.Y. Edgeworth is the first to define an optimum for multi-criteria economic decision making. And the applications of multi-criteria optimization in engineering design grew over the following decades (Alfaris, 2010). Alfaris summarizes the methods of multi-criteria optimization into two fundamental approaches: Scalarization Approaches and Pareto Approaches.

The idea of scalarization is to convert the multi-criteria optimization back to single-criterion optimization. Weighted Sum Approach is a scalarization approach. The equation is shown in Equation 2.1. $J_i$ is the value a solution gets in one optimization criterion, where $1 \leq i \leq z$. $\lambda_i$ is the weight. $sf_i$ is the scale factor. And $S_{MO}$ is the final score of the solution $S$ to those criteria.
\[ S_{MO} = \sum_{i=1}^{z} \frac{\lambda_i}{z} f_i J_i \] (2.1)

In Section 5.4.1, the application of Weighted Sum Approach in the case of Philips IGT domain will be described in detail.

Pareto optimality was first introduced by Fawcett (1888). Finding the Pareto Set is one of the Pareto Approaches. The Pareto Set includes one or more solutions. A solution belongs to the Pareto Set if there is no other solution that can improve at least one of the criterion without degrading any other criteria. This approach can provide all possible optimized solutions of all criteria simultaneously, so system architects can study and compare these solutions in depth. Using the Pareto approach finding the Pareto Set is helpful to filter configurations first if there are many configurations and Philips system architects only want to compare a certain number of configurations. A detailed explanation of using the Pareto approach in the project will be introduced in Section 5.4.2.

A scalarization approach and a Pareto approach are both used in our implementation. The utilization will be described in detail in Chapter 5.
3 | Formalization

Philips IGT is a complex system with many configurations. When designing a system with many possible configurations, domain experts in Philips will evaluate a certain configuration with regard to market segments and user needs in a certain process. To know the process well, we formalize it and develop a tool to automatically process the data. Another motivation of formalization is that there are different names for the same concept, e.g., attributes and properties. To unify the convention on items, we formalize each concept and its name with an example. Unless there is a separate explanation, this thesis consistently uses the terms as defined in this chapter.

3.1 Domain model

Formalization aims not only to capture key concepts of a system, but also describes the behavior of it, and identifies key properties of interest. Figure 3.1 shows the domain model of the system.

The domain model is built based on Philips specification documents and reviewed by domain experts. Each item is followed by an example. The details of the examples will be discussed in Chapter 4.

- **Market**: is a market segment. Market can be clinical domain of Peripheral, hospital domain of Quality Optimizers and geographical region of U.S.A etc.

- **User Need**: a benefit a potential buyer of an IGT system wants to achieve. E.g., capability to treat both Cardio and Peripheral diseases or limited total cost of ownership.

- **Importance**: to what degree (weight) the Market requires a specific user need. The importance range is defined from 0 to 4. 4 means the user need is highly important to the market segment, while 0 or no entry means the user need is not relevant to the segment. For instance, Peripheral requires that Full_body_coverage with an importance of 4 which will be shown in Table 4.5.


• **Component**: is the atom of the system, individually or together with others performs a certain functionality. E.g., Standard Tabletop.

  - *Component property*: can be provided by a component, required by another component or a component property. E.g., cost of the component.

• **Configuration**: is a product variant which can fulfill user needs. A configuration is a set of components to form a product. In this thesis we do not distinguish the concepts of product and configuration. E.g., Configuration A [TableTop, TableBase, FloorPlate, StandardTableTop, StandardFloorPlate]. The example comes from Table 4.6 which will be discussed in Chapter 4.

• **Configuration Property**: is a capability of a configuration. User needs are fulfilled by configuration properties. Configuration properties are divided into functional properties and non-functional properties because of different behaviors and evaluation ways. The evaluation methods will be introduced in the next section.

**Figure 3.1: Domain model**
- **Functional property**: is usually either fulfilled or unfulfilled. E.g., Height_movement is fulfilled by the component Tablebase.

- **Non-functional property**: is usually evaluated to a value. E.g., Cost = 17700 euros.

- *Composition Rule*: expresses how to construct a working product. E.g., Tabletop must be included in any product.

- *Evaluation Rule*: a rule to calculate the property of a configuration. E.g., the Stroke value of a configuration equals the summation of components’ stroke values.

- *Fulfillment Rule*: expresses how well a user need can be fulfilled by configuration properties. E.g., the higher of a configuration Stroke value is, the better the configuration fulfills the user need of Full_body_coverage.

- *Evaluation Function*: a function that scores all configurations with regard to multiple related criteria.

## 3.2 Instantiation

Following the domain model, an example is instantiated in Figure 3.2. All green blocks are instantiated parts of the domain model, while the blue block is not used in the example. The Market Segment USA requires User Need HighPatientWeight with Importance of 4. The evaluation rule means that the PatientWeight property of a configuration (Config.) is the minimum value of PatientWeight property of all components (Comp.) it is composed of. The fulfillment rule in Figure 3.2 is the same as the one in the domain model: the higher of a configuration Stroke value is, the better the configuration fulfills the user need of Full_body_coverage.

All configurations have to comply to the Composition Rules. All configurations are scored by the Evaluation Function which is a Pareto ranking in the example. More details about Evaluation Rule, Composition Rule, Fulfillment Rule and Evaluation Function will be explained in Chapter 5.
3.3 General definitions

In the domain model section we formalized the system problem in the IGT domain. It is specialized based on the configuration evaluation process of Philips which is not applicable to all variability management problems. E.g., user needs of a different product domain may not be expressed as "Market requires a user need with an importance". To generalize architecture variability management, we define the following concepts and notations to express relations and conditions unambiguously. Since in the domain model of Figure 3.1 we use items that are specifically used in the evaluation process of Philips, in this section both the process and items are slightly different from the domain model. The process here does not include market information because it is expressed differently in different domains, and Function is used as the service provided and required by different parts of the system because it can cover more comprehensive concepts than components and properties which play a similar role as Function in the IGT domain model. The
process showing in this section is more generalized, new items which can be mapped to Philips’s concepts will be noted.

A generic formalization model is shown in Figure 3.3.

**Figure 3.3: Generalized formalization**

**Definition 1:** A component \( c \) is the atom element of the system. A component defines its behavior in terms of provided and required functions.

**Definition 2:** A function \( f \) denotes a property. This can range from a unique identification of a component to a complex functional behavior. A set of functions is denoted by \( F \).

A function \( f_1 \) can be realized by another function \( f_2 \) or a set of functions. We denote this by \( f_1 \rightarrow f_2 \) or \( f_1 \rightarrow F \). \( f_1 \) covers \( f_2 \) if \( f_1 \) realizes the same or more than \( f_2 \), or if it refines \( f_2 \). We denote this by \( f_2 \leq f_1 \).

Note: In the domain model, a user need requires one or more properties (i.e. functions), and a property may require one or more other properties, so the role of function is shared by components and properties.
**Definition 3:** A configuration $C$ is a set of components. A valid configuration $C$ contains a valid selection of components that leads to a working product.

A configuration $C$ provides functions. The function set it delivers contains its properties.

**Definition 4:** An evaluation rule $h_e$ expresses a law to evaluate a functional property, or a cost function to evaluate a non-functional property of a configuration. It takes a configuration $C$ as input, and calculates the value with regard to the property as shown in Equation 3.1.

$$Value_{propertyofc} = h_e(C) \tag{3.1}$$

Even the input of the evaluation rule is a configuration, the value $Value_{propertyofc}$ is determined by the rule, which in turn is dependent on the components (i.e. variables in a more general case) of the configuration.

**Definition 5:** A composition rule is a constraint or a relation among components. It is equivalent to the Constraint Expressions we discussed in the Section 2.1. Different product domains use different ways to express the information. A valid configuration has to follow all composition rules.

**Definition 6:** An evaluation function $f_e$ is a mathematical function (different from the function we defined before) to do the multi-criteria optimization. It takes market information, evaluation criteria and rules defined by users as inputs. A $f_e$ in a scalarization approach scores a configuration with respect to certain criteria. The configurations are then ranked based on the scores. While a $f_e$ in a Pareto approach gives a pareto front. The output of an evaluation function is configurations that are optimized ones with respect to user concerned criteria.

As we already introduced in Chapter 1, we will develop a tool to explore architecture variability in the IGT domain. So we will use the specific IGT domain model in the remainder of the thesis. Given user selected Markets, the tool should be able to compute valid configurations and rank them according to the market information. In Section 4.2 we will describe the detailed process.
4 | Case Study

As discussed before, the goal of the project is to provide a tool that can give insights to the variability management of IGT systems and to help to find the optimal configuration set with regard to user needs in different market segments. Since IGT systems are quite complex, we use the patient table module of it for the case study. The dataset for this case contains the necessary information to comply with the domain model we defined in Chapter 3. It is derived from the product of Philips AD7 patient table in the Appendix. The system definition is shown in Figure 4.1.

![Figure 4.1: Components and composition rules](image)

Figure 4.1 shows that a table must have 1 tabletop which could be either Standard Tabletop or Extended Tabletop. Longitudinal Motorized Option is optional. Swivel is optional as well, but as long as it is contained by a table, Swivel Floorplate has to be included into the table configuration as well. The information presented in Figure
4.1 is used to check whether a configuration is valid or not. Detailed analysis of the construction of configurations will be discussed in Section 4.2.

4.1 Dataset

In this system, multiple configurations are possible. As shown in Figure 3.1, each configuration has properties. In this dataset, configuration properties are defined and shown in Figure 4.2.

![Figure 4.2: Configuration Property](image)

Each component has properties which are shown in Table 4.1. A component may not be able to provide all kinds of properties.

As we discussed in Chapter 3, configuration properties are derived from components that it is composed of. The calculation of a configuration property is based on an Evaluation Rule. Evaluation rules for non-functional properties of a configuration are
Table 4.1: Component properties

<table>
<thead>
<tr>
<th></th>
<th>STT</th>
<th>ETT</th>
<th>TB</th>
<th>SW</th>
<th>SDF</th>
<th>SWF</th>
<th>LM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost (€)</td>
<td>2500</td>
<td>3500</td>
<td>15000</td>
<td>10000</td>
<td>200</td>
<td>200</td>
<td>1250</td>
</tr>
<tr>
<td>Patient Weight (kg)</td>
<td>250</td>
<td>200</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
<td>250</td>
</tr>
<tr>
<td>Height (cm)</td>
<td>0</td>
<td>0</td>
<td>74</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Stroke (mm)</td>
<td>1200</td>
<td>1720</td>
<td>0</td>
<td>782</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Length (cm)</td>
<td>319</td>
<td>371</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 4.2: Evaluation rules for configuration non-functional properties

All rules in Table 4.2 are quantitative rules which give values to properties of a configuration. For instance configuration Patient_weight equals to component Patient_weight value of the weakest one which supports the lightest patient weight, while the cost of a configuration equals the summation of costs of its components. Table 4.3 shows that a configuration can perform a functionality if it contains certain combinations of components, otherwise it cannot. So the value of a configuration functional property is either 1 (can perform) or 0 (can not perform).

Table 4.3: Evaluation rules for configuration functional properties

The user needs for a patient table are presented in Table 4.4. A configuration is evaluated by fulfillment rules based on its properties with regard to the user needs. There are two kinds of user needs in Table 4.4: the first 5 user needs are Quantitative User Need which are evaluated quantitatively. E.g., each configuration has a value for the property of Cost, the less the Cost value it has, the better it fulfills the user need UN2; UN6 and
UN7 are Qualitative User Needs which are evaluated qualitatively. For the UN6, only if a configuration has the property of Bolus Chase, it fulfills the user need.

<table>
<thead>
<tr>
<th>User Need</th>
<th>Fulfillment Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN1: Full body coverage</td>
<td>max. Stroke</td>
</tr>
<tr>
<td>UN2: Low working height</td>
<td>min. Working height</td>
</tr>
<tr>
<td>UN3: High patient weight</td>
<td>max. Patient weight</td>
</tr>
<tr>
<td>UN4: Small room size</td>
<td>min. (Stroke + Length)</td>
</tr>
<tr>
<td>UN5: Low system price</td>
<td>min. Cost</td>
</tr>
<tr>
<td>UN6: Entire leg image while contrast flow</td>
<td>Bolus chase</td>
</tr>
<tr>
<td>UN7: Position patient in the iso-center</td>
<td>Lateral movement Longitudinal movement Height movement</td>
</tr>
</tbody>
</table>

**Table 4.4:** User needs and associated evaluation rules

To analyze which configuration suits certain market segments, information of the importance of a user need to a market segment is needed. In **Table 4.5** the importance of each user need to a specific market segment is shown.

<table>
<thead>
<tr>
<th>User Need</th>
<th>Peripheral</th>
<th>U.S.A</th>
<th>Asia</th>
<th>E&amp;I</th>
<th>QO</th>
</tr>
</thead>
<tbody>
<tr>
<td>UN1: Full body coverage</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN2: Low working height</td>
<td></td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN3: High patient weight</td>
<td></td>
<td>4</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN4: Small room size</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>UN5: Low system price</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN6: Entire leg image while contrast flow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UN7: Position patient in the iso-center</td>
<td></td>
<td></td>
<td></td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4.5:** Importance of user needs in different market segments

### 4.2 Data processing

Based on system definition in **Figure 4.1** and components defined in **Table 4.1**, it can be manually verified that there are 8 valid configurations shown in **Table 4.6**.

3 counter examples are shown in **Table 4.7**. According to system definition in **Figure 4.1**, a valid configuration must include 1 and only 1 tabletop, so both configuration A and B are invalid because configuration A does not have a tabletop while configuration B has two tabletops. The dash line in **Figure 4.1** expresses that Swivel implies Swivel Floorplate, so configuration C is invalid.
4.2.1 Manual evaluation process

If a customer wants a product which suits for market Periheral and U.S.A., the selection process goes as follows:

First, Peripheral requires user needs UN1, UN6 and UN7, and U.S.A requires UN2 and UN3. The required user needs and corresponding evaluation rules can be found in Table 4.5. The related configuration properties are Stroke (P2), Working height (P3), Patient weight (P4), Bolus chase (P6), Longitudinal movement (P7), Lateral movement (P8) and Heigh movement (P9).

Second, with component properties in Table 4.1, evaluation rules in Table 4.2 and Table 4.3, valid configurations and concerned configuration properties are listed in Table 4.8.

Third, each configuration is evaluated that how well it fulfills a user need based on fulfillment rules in Table 4.4. For instance, user need Full_body_coverage is fulfilled...
better if a configuration gets a higher value in the property of Stroke than others. So configuration 4 and 8 get the highest value of 2502, both configurations will rank 1st in this user need. While configuration 1 and 5 get the lowest value in Stroke, they are ranked 7th in the user need. The results are shown in Table 4.9

<table>
<thead>
<tr>
<th>Configuration</th>
<th>UN1</th>
<th>UN2</th>
<th>UN3</th>
<th>UN6</th>
<th>UN7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration 1</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 2</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 3</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 4</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 5</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 6</td>
<td>3</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 7</td>
<td>5</td>
<td>1</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Configuration 8</td>
<td>1</td>
<td>5</td>
<td>5</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.9: Fulfillment results

Once we reach this step, we know how well each configuration fulfills related user needs. Since market segments Peripheral and U.S.A. require UN1, UN3, UN6 and UN7 with importance of 4, and require UN2 with importance of 2, if we only consider most important user needs, then configuration 6 is the most suitable one since its overall performance in 4 user needs is the best. But if UN2 is also considered, we have a multi-criteria problem because there are 5 criteria with different importance values now. The process of optimizing a selection differs with different algorithms. Ignorance of unimportant criteria is an algorithm but it is not the only way. In Chapter 5 we will introduce a new algorithm to perform the multi-criteria optimization.
5 | Tool Design and Implementation

In Chapter 4 we described the manual process of evaluating a configuration for certain markets and comparing it with its alternatives. With more variabilities in a system, the workload of the process expands too much to be done manually. In this chapter we will introduce a tool to perform the process automatically.

To automate the evaluation process, the tool has to include the following capabilities:

- Get specifications of the system and market information as input.
- Compute all valid configurations.
- Evaluate a configuration with regard to certain criteria and compare it with its alternatives.
- Deliver the proposed configurations to the user.

In the following sections we will introduce how these capabilities are realized in the tool.

5.1 Solution

Our solution is an Eclipse-based software application. Users of the tool only need to provide the domain information and their concerned market segments, then the tool automatically processes the data and delivers sorted configurations for the market segments. If a given system definition follows the dataset example in Chapter 4, we have the component information and composition rules. If a set \( pC \) contains every possible combination of components \((pc)\), and \( cR \) contains every composition rule \((cr)\), the Algorithm 1 shows the process to get valid configurations. Invalid configurations like shown in Table 4.7 in Chapter 4 are removed from the set \( pC \). In the end all left valid configurations in \( pC \) are stored in the configuration set \( C \).

Now the configuration set \( C \) contains all valid configurations. The user selects concerned market segments. Based on the definition information we can get related user needs
Algorithm 1 Get valid configurations

Input: \( pC, cR \)
Output: \( C \)

1: \textbf{for} each \( pc \) in \( pC \) \textbf{do}
2: \hspace{1em} \textbf{for} each \( cr \) in \( cR \) \textbf{do}
3: \hspace{2em} \textbf{if} \( pc \) does not satisfy \( cr \) \textbf{then}
4: \hspace{3em} \( pC \).remove\((pc)\)
5: \( C \leftarrow pC \)
6: \textbf{return} \( C \)

and configuration properties. With the evaluation rule set \( eR \), Algorithm 2 shows how evaluation rules work. \( Value_{propertyofc} \) contains one property value of a configuration. After being evaluated by all rules in \( eR \), all property values of the configuration are stored in \( Properties_{c} \). In the end all configurations (configs) with their property values are stored in \( Properties_{configs} \).

Algorithm 2 Evaluate configuration properties

Input: \( C, eR \)
Output: \( Properties_{Configs} \)

1: \textbf{for} each \( c \) in \( C \) \textbf{do}
2: \hspace{1em} \textbf{for} each \( h_{e} \) in \( eR \) \textbf{do}
3: \hspace{2em} \( Value_{propertyofc} \leftarrow h_{e}(c) \)
4: \hspace{2em} \( Properties_{c} \).add \( Value_{propertyofc} \)
5: \( Properties_{Configs} .add Properties_{c} \)
6: \textbf{return} \( Properties_{Configs} \)

Now each configuration gets its property values, and is stored together with other configurations. The tool is able to score all configurations with regard to related user needs and then rank them. In Algorithm 3, each configuration first is ranked per user need (un) by the corresponding fulfillment rule \( fr \) in the set of fulfillment rules \( fR \). The result of each configuration with a rank value per user need is stored in \( c_{rankperun} \) which is an item of \( C_{rankperun} \). Then it is scored by the evaluation function \( f_{e} \). All configurations with their final scores are stored into \( Scores_{configs} \). Then the configurations are sorted based on the final score and sorted configurations are stored in \( C_{ranked} \) as the output.

The basic functions of the tool are described by the above 3 algorithms. In the following section we will discuss the design decisions and how the functions of the tool are actually implemented.
Algorithm 3 Score and rank configurations

Input: $C$, $f_R$
Output: $C_{\text{ranked}}$

1: for each $fr$ in $f_R$ do
2:     for each $c$ in $C$ do
3:         $c_{\text{rankperun}} \leftarrow c$ ranked per user need
4:         $C_{\text{rankperun}}$. add ($c_{\text{rankperun}}$)
5:     for each $c_{\text{rankperun}}$ in $C_{\text{rankperun}}$ do
6:         $Score_c \leftarrow f_e (c_{\text{rankperun}})$
7:         $Scores_{\text{configs}}$. add ($Score_c$)
8:     for each $Score_c$ in $Scores_{\text{configs}}$ do
9:         Sort($Scores_{\text{configs}}$) based on $Score_c$
10:    $C_{\text{ranked}} \leftarrow$ configs in Sort($Scores_{\text{configs}}$)
11: return $C_{\text{ranked}}$

5.2 Design decisions

The functions of the 3 algorithms in the previous section are the core parts of the tool. To realize them with a good software functional quality and structural quality, we develop the tool following the development view in Figure 5.1.

- **Problem Instance**: contains all the domain information we defined in Section 4.1.
- **Product Domain Language**: defines the grammar of how to express the dataset. A problem instance has to follow the grammar so that the information can be processed correctly.
- **Feature Model**: is a compact representation of all the configurations of the product line in terms of "features". And in our case, each component can be regarded as a feature of a feature model.
- **Data Repository**: is the intermediate layer of the tool. First it gets all components and composition rules in Figure 4.1 to construct the feature model. Then it gets all configurations from the feature model, which performs the process described in Algorithm 1. It then calculates configuration properties based on component properties, which is the part described in Algorithm 2. In the end it sends these configurations and other necessary informations to the Optimizer.
- **Optimizer**: scores each configuration with regard to the selected market segments that a user wants to optimize, and this is the part described in Algorithm 3.
Figure 5.1 shows the relations among different parts of the tool. All the parts cooperate together to deliver the final output to the users. A flow chart and a sequence diagram with all parts in Figure 5.1 as actors showing how the cooperations work will be explained in detail in Section 5.3.

5.2.1 Feature Model

The Algorithm 1 shows a method which can be implemented in the tool to get valid configurations. It is a simple enumeration method which consider all possible combinations.
It is inefficient to be implemented when dealing with a huge amount of variabilities. Feature Models are useful in modeling of a large number of features and give valid selections of features to compose valid configurations that satisfy constraints.

There are several feature modeling tools available, for example, FeatureIDE, Velvet, Pure:Variants and Formilia. Pure:Variants is a commercial application. Because of the research purpose of the project, it is too early to pay for a commercial application when there are still other free alternatives available. Formilia and Velvet have their own working environment which means that they can not easily be integrated with other parts of the tool. So we choose FeatureIDE. FeatureIDE is an Eclipse-based plugin that supports all phases of feature-oriented software development for the development of System Product Lines: domain analysis, domain implementation, requirements analysis, and software generation.

Besides its easy-to-use editor, FeatureIDE also has the following characteristics (Thüm et al., 2014) that are helpful for our implementation:

- Highlighting of dead and false-optional features and their corresponding constraints based on Sat4j (Le Berre and Parrain, 2010). With this characteristic we can directly check whether the model constructed based on the dataset is correct or not.

- Constraint Editor with content assist, syntax, and semantic checking, e.g., dead feature detection. This is the characteristic that cooperate with the first one. And in our implementation, the constraints are translated and expressed by the Data Repository following the constraint grammar, so the constraint editor helps to identify errors in expressions.

- Configuration Editor to create and edit configurations and with support for deriving valid configuration. This is the core part that is relevant to our implementation. We use FeatureIDE to derive all valid configurations.

- Support for editing on feature models, i.e., categorizing edits into re-factorings, generalizations, specializations or none of these. With this characteristic it is possible to modify the feature model directly on the graph.

- A view displaying statistics about the software product line. The view display helps to check the system model.
FeatureIDE can be used to construct the Feature Model of the product with its graphical and text based Feature Model Editor. An example of feature model in FeatureIDE is shown in Figure 5.2. It is a representation of Figure 4.1 in Chapter 4.

The part that performs the role of Algorithm 1 in FeatureIDE is the integrated SAT solver: Sat4j. Sat4j is a java library for solving boolean satisfaction and optimization problems. Detailed information about Sat4j can be found in the work of Le Berre and Parrain (2010).

![Feature Model Example in FeatureIDE](image)

**Figure 5.2:** An feature model example in FeatureIDE

### 5.2.2 Product Domain Language

For each valid configuration, its components and component properties are needed to compute the properties of the configuration. As in Figure 5.2, a component is a feature in our case, and a component property is called an attribute of a feature in feature modeling. So to express component properties, the feature modeling tool needs to support attributed features. But FeatureIDE does not support attributed features yet. If FeatureIDE is used as the input method of the tool, users have to give properties of components (i.e. feature attributes) in another interface which causes inconsistence and decreases usability. So we propose a Domain-Specific Language (DSL) that is dedicated to the expression of product information as the input format. The implemented DSL in this project is called Product Domain Language as in the Development View in Figure 5.1.

A DSL is specialized to a particular application domain. The advantage of using the DSL in the implementation is that we can define specialized expressions for the target system domain in a compact manner. The concepts and notations can be as close as
possible to what system architects of Philips expect for the solution in the domain. Since the programming language we use is Java, we develop the DSL in Xtext which has an advanced Java integration. Xtext is a framework for development of programming languages and domain specific languages. It covers all aspects of a complete language infrastructure, from parsers, linker, compiler or interpreter to Eclipse IDE integration. It comes with great defaults for all these aspects which at the same time can be easily tailored to individual needs (Friese, Efttinge, and Köhnlein, 2008). Another reason that we choose Xtext is that support is available in the form of, amongst others, user manuals and Internet forums.

A DSL implementation in Xtext is shown in Figure 5.3. The DSL infrastructure is constructed in a meta-level workspace which is the upper part of the figure. A DSL implementation starts by defining the grammar in the Concrete Syntax part in Figure 5.3. Then the Xtext framework is used to generate a parser, an Ecore-based metamodel (Koegel and Helming, 2015) and a textual editor for Eclipse. The Ecore-based metamodel represents the Abstract Syntax in Figure 5.3 and the textual editor is using for Textual Input in Figure 5.3. In the Instance-level instances of the DSL are defined and corresponding XMI models are generated. In the meta-level workspace not only the syntax of the DSL but also the transformations (e.g. to code) are defined (Mooij and Hooman, 2015). And Generated Code of an instance is generated in the instance-level. In our implementation we only use the Concrete Syntax, Abstract Syntax in the meta-level and Textual Input, Model in the instance-level to capture the domain information. More information about the DSL implementation and Xtext utilization can be found in the work by Friese et al. (2008).

Figure 5.4 shows the user need part of the Product Domain Language as an example of the Concrete Syntax in Figure 5.3. As explained in Section 4.1, a user need can be a Quantitative User Need or a Qualitative User Need. The grammar in Figure 5.4 is parsed into the part of Ecore-based model shown in Figure 5.5. The Abstract Syntax example is built up a structured hierarchy. The tree structure is useful for navigation and serialization (Koegel and Helming, 2015).

In the Instance level, the user needs in the IGT domain provided by Philips are shown in Figure 5.6. As an instance of the Product Domain Language, it has to follow the grammar shown in Figure 5.3. The instance of the Ecore-based model is shown in Figure
5.7. The names of user needs and corresponding evaluation rules in the Textual Input part are parsed correctly. The model is an intermediate representation of the information in the input instance. It is generated by Xtext with interfaces and implementations for all of the model’s entities provided by Eclipse Modeling Framework. E.g., an entity’s interface contains getters and setters for the attributes in the model as well as getters for the references. All entities of the generated Ecore-based model are subclasses of EObject (Koegel and Helming, 2015). The model will be used by the Data Repository. So the Data Repository can retrieve all the information, build the feature model and start the optimization process which will be discussed in Section 5.3.

Above we described in detail about how Xtext works to be used for development of the Domain Specific language. In conclusion, we have to define the grammar first, then write an instance following the grammar. After that Xtext will transfer the instance with grammar information to an intermediate model. The Data Repository in the tool will navigate through this model to get instance information.
In the next section we will introduce the grammar we defined for the Philips IGT domain.

### 5.2.3 Grammar of the Product Domain Language

The grammar of the Product Domain Language is defined to express the input information of the system domain. For the Philips IGT domain, the information includes components and component properties, rules, user needs and market information which is shown in the Model part of Figure 5.8. In this part it specifies that a system model can have multiple components and multiple component properties. From the Quantity part to the Component part the grammar is used to express components and component
properties. And in the *Component* part the grammar allows that a component can be expressed as abstract, and the user has to give a name, and possible relations with other components. For example, in the *ComponentRule* part, it shows that a component require another component or several components which is expressed by the *expression* in the grammar. Then the grammar to express composition rules and evaluation rules are defined from the *Variability* part to the *Configuration Expression* part.

Continued in Figure 5.9, the grammar defines how to express the evaluation rules from *Evaluation Rule* part to the *Funcode* part together with the grammar in Figure 5.8. This allows user to express 4 kinds of quantitative evaluation rules which are defined in the *FormulaEvaluationExpression* part. And the qualitative evaluation rule can be expressed with different combinations of components which is shown form *QualitativeEvaluationRule* part to the *Funcode* part.

Figure 5.10 shows the grammar to express the user needs, market importance and fulfillment rules. In the *UserNeed* part, 2 kinds of user need are defined in the *UserNeed* part. 4 kinds of fulfillment rules can be expressed following the grammar defined from the *FormulaNeedExpression* to the *LessNeedExpression* to check the fulfillment of quantitative user needs. And the *BooleanNeedExpression* shows the grammar to express the fulfillment rules for the qualitative user needs. The *Market* part and the
Model:
'Component\' "Properties\'\' (quantities == Quantity)+ '
'Components\'\' ( (components == Component)+ )'
'Rules\'\' (\'for\' names ID ( | \'variables\' == Variability )\')
'Use\'\' \'needs\'\' ( (needs == Needs)+ )
'Markets\'\' ( (markets == Market)+ )

Quantity:
  name == ID \'\' type == QuantityType\'\'

QuantityValue:
  quantity == [Quantity[ID] \'\' value == INT \'\'

Component:
  (abstract => 'abstract') \'\' component\'\' name == ID \'\'\' \'' (values == QuantityValue)+ \'\'

Variability:
  \'\'composition\'\' \'rules\'\' ( (configurations == ConfigurationRule)+ )
  \'\'evaluation\'\' \'rules\'\' ( (evaluations == EvaluationRule)+ )

ConfigurationRule:
  AnyRule | ComponentRule

AnyRule:
  \'\'any\'\' \'requires\'\' expression == ConfigurationExpression \'\'

ComponentRule:
  (option => 'option') \'\' component == [Component[ID] \'\' \'\' \'' (requires\'\' expression == ConfigurationExpression) \'\' \'

ConfigurationExpression:
  components == [Component[ID] \'\' \'\' and\'\' components == [Component[ID]]

---

**Figure 5.8:** Grammar for expressing component, component properties, and composition rules

EvaluationRule:
  QuantitativeEvaluationRule | QualitativeEvaluationRule

QuantitativeEvaluationRule:
  name == ID \'\' expression == FormulaEvaluationExpression \'\'

FormulaEvaluationExpression:
  MinEvaluationExpression | MaxEvaluationExpression | SumEvaluationExpression | AdditionEvaluationExpression

MinEvaluationExpression:
  \'\'\' ( (quantity == [Quantity[ID] )\'

MaxEvaluationExpression:
  \'\'\' ( (quantity == [Quantity[ID] )\'

SumEvaluationExpression:
  \'\'\' ( (quantity == [Quantity[ID] )\'

AdditionEvaluationExpression:
  left == [Quantity[ID] \'\' \' right == [Quantity[ID]

QualitativeEvaluationRule:
  name == ID \'\' requires\'\' expression == BooleanEvaluationExpression \'\'

BooleanEvaluationExpression:
  components == [Component[ID] \'\'\' \'

Funode:
  andfunc = \'\' \' and\'\' | orfunc = \'\' or\'\'

---

**Figure 5.9:** Grammar for expressing evaluation rules

*Importance* part are the grammar to express the importance of a user need to a market.

In this section we showed all the grammar we defined to express the system information. The brief explanation is to map the grammar to the dataset in Chapter 4.
Figure 5.10: Grammar for expressing market information and fulfillment rules

5.3 Implementation

The tool is developed in Eclipse which is a Java Integrated Development Environment with FeatureIDE and Xtext plugins. From the input of the problem instance to the output of proposed configurations, the data is processed following the flow chart in Figure 5.11.

The DSL instance as an input of the tool contains all architecture variability information. It maps to the Problem Instance and Product Domain Language parts in the Development View of Figure 5.1. The input is processed as in the sequence diagram of Figure 5.12. All actors in the sequence diagram can be mapped to parts in Figure 5.1.

The information of the instance is expressed following the grammar defined in the Product Domain Language (PDL) part, and can be divided into three portions: components and composition rules (Figure 4.1); markets related information (Table 4.5) and fulfillment rules (Table 4.4); component properties (Table 4.1) and evaluation rules (Table 4.2 and Table 4.3). When the user starts the optimization, the Data Repository (DR) will
store all the instance information. Components and composition rules are translated into the inputs for FeatureIDE by the Data Repository. Data Repository translates the components and relations among them to features and constraints of the Feature Model.

To get the optimization criteria, another input of the tool – User Inquiry is needed which contains user concerned market segments. The Feature Model then generates all valid configurations to be evaluated based on user inquiry. Then the component properties and evaluation rules stored in the Data Repository are used to calculate configuration properties. Since the Data Repository knows the market segments which are of interest to the user, it will use the Market information (including market importance and user needs relations) and fulfillment rules to grade configurations per user need in selected market segments.

Now all configurations and their scores per user need are clear. The Optimizer in Figure 5.1 will get the configurations and scores per user needs. Every configuration is evaluated by the evaluation function in Optimizer and gets the final score for the market segments.
Now we get the output of the Optimizer – valid, scored, ranked configurations. The visualization part will take these configurations as the input and visualize them.

In the next section, we will discuss the details of multi-criteria optimization performed by the Optimizer in Figure 5.1.

5.4 Multi-criteria optimization

To evaluate an individual configuration, there are one or more market segments to be considered. Among all valid configurations, only based on the comparison of all user concerned criteria or objectives a decision can be made as to the superiority of one configuration over another. We discussed possible multi-criteria optimization approaches
in Section 2.2. The main approach we use in the tool is the Weighted Sum Approach. This is because the optimization criteria for the IGT domain in Philips have different importance values. The method of getting the Pareto Set by a Pareto approach does not handle the importance values well. So we will use a Pareto approach as a supplementary method to filter configurations when dealing with a large dataset. For the dataset we are using now, we can directly apply Weighted Sum Approach since there are only 8 valid configurations. Details about using a Pareto approach will be discussed in Section 5.4.2. For the dataset in Chapter 4, the Weighted Sum Approach is enough. In the following section we will describe how the algorithm in the Weighted Sum Approach works.

5.4.1 Weighted Sum Algorithm

The criteria for optimization in the IGT domain are user needs. Sometimes multiple market segments require the same user need with different importance values. In such cases, the importance values of the user need will be merged to only 1 value which is the highest one of all. For each user need, a fulfillment rule expresses how a configuration fulfills it. An example of such rules is shown in Figure 5.13. In the figure user need of HighPatientWeight is optimized by max(ConfigPatientWeight) which means the tool will get the ConfigPatientWeight property values of all configurations and compare them. The operator max indicates that the higher a configuration scores in ConfigPatientWeight, the higher it will be ranked in the list for that user need. While the LowWorkingHeight property of a configuration is optimized by min(ConfigHeight) means that the lower a configuration scores in ConfigHeight property, the higher it will be ranked for the user need. There is another kind of fulfillment rule shown in Figure 5.13 which are boolean rules to evaluate Qualitative User Needs. E.g., "EntireLegImage requires BolusChase" means the user need of EntireLegImage is only fulfilled if the configuration has the property of BolusChase. In this case a configuration can only score 0 or 1 for the user need. The normalization process is done by ranking configurations based on their scores per user need.

Figure 5.14 shows an example of evaluation rules used to evaluate a configuration property based on its components and component properties. As shown in the figure ConfigHeight property of a configuration is equal to the summation of the Height values of
all components in that configuration. The ConfigPatientWeight property of a configuration is equal to the minimum PatientWeight value of its components. There are also boolean rules to evaluate functional properties. As already explained in Chapter 4, they are either fulfilled if the configuration contains certain combinations of components or not fulfilled at all. The property value is either 1 or 0.

```
FullBodyCoverage is optimized by max(ConfigStroke);
LowWorkingHeight is optimized by min(ConfigHeight);
HighPatientWeight is optimized by max(ConfigPatientWeight);
SmallRoomSize is optimized by min(ConfigStrokePlusLength);
LowSystemPrice is optimized by min(ConfigCost);
EntireLegImage requires BolusChase;
PositionInTecCenter requires LateralMovement and LongitudinalMovement and HeightMovement;
```

**Figure 5.13:** Rules to evaluate a configuration regarding a user need

```
ConfigStroke = sum(Stroke);
ConfigHeight = sum(Height);
ConfigPatientWeight = min(PatientWeight);
ConfigStrokePlusLength = stroke + length;
ConfigCost = sum(Cost);
ConfigLength = max(length);
LongitudinalMovement requires TableTop;
LateralMovement requires TableTop;
HeightMovement requires TableBase;
BolusChase requires (LongitudinalMotor and ( ExtendedTableTop or ( StandardTableTop and SwivelFloorPlate )));
```

**Figure 5.14:** Rules to evaluate a configuration property

If we have 2 user needs (criteria) to consider, then each configuration has criteria scores for every user need in the concerned market segments. Based on the criteria scores, every configuration gets a rank $Rank_i$ in a certain criterion $i$. A rank $Rank_i$ is a serial index and different configurations may have the same value of $Rank_i$ if they fulfill that user need (criterion $i$) with the same level. And each user need (criterion $i$) has an importance value $Importance_i$ considering to the specific market segment. To evaluate how well configurations suit the multiple criteria, we use the Equation 5.1 to score them. The Equation 5.1 is a modified version of Equation 2.1 according to the dataset in the IGT domain. Then configurations are ranked based on the final scores.

The basic idea behind the algorithm is that the higher a configuration ranks, the more it satisfies the user need (criterion). So it will get a higher value of $\frac{Importance_i}{Rank_i}$. In the end the values are summed together so that the higher of the $S_c$ value a configuration gets, the better it suits the market or markets selected.
Now using the same example in Section 4.2, we get the following user needs and importance information for a merged market segment shown in Table 5.1.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>User Need</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UN1: Full body coverage</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>UN2: Low working height</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>UN3: High patient weight</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>UN6: Entire leg image while contrast flow</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>UN7: Position patient in the iso-center</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 5.1: Importance values of multiple criteria

From the Table 4.9, configuration 1 has the following scores for each user need in the merged market segment.

<table>
<thead>
<tr>
<th>Criterion</th>
<th>User Need</th>
<th>Criterion Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UN1: Full body coverage</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>UN2: Low working height</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>UN3: High patient weight</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>UN6: Entire leg image while contrast flow</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>UN7: Position patient in the iso-center</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 5.2: The criterion rank of configuration 1

According to Equation 5.1, the final score of configuration 1 for the segment is calculated as the following equation:

\[ S_c = \sum_{i=1}^{z} \frac{Importance_i}{Rank_i} \]

(5.1)

For each criterion we apply this algorithm to score the configuration and rank them in comparison to each other.

### 5.4.2 Dominance and Pareto Ranking

Pareto ranking is another multi-criteria optimization approach. So apart from the Weighted Sum Algorithm we described in the previous section, Pareto ranking can also be used as a ranking algorithm. We cite the example from Pohlheim (2006) to illustrate the idea of it.
When a company produces products, the ideal case is to keep the production costs low and produce those products quickly. There are various approaches to achieve the ideal case during production planning which may differ in the results. Here the costs $s$ and production time $t$ are criteria which serve as evaluation criteria for each solution.

The superiority of a solution over another can be decided by comparing them. Dominance and Pareto ranking can be illustrated via Figure 5.15. For both $s$ and $t$, smaller values are preferred to larger ones since smaller $s$ means lower cost and smaller $t$ stands for shorter production time.

![Figure 5.15: Pareto](image)

The boxed points in Figure 5.15 represent solutions.

- **Dominance**: a non-dominated solution is one where an improvement in one criteria results in a deterioration in one or more of the other criteria when compared with other solutions. In Figure 5.15, solution C is dominated by solution A because A performs better both on cost and time criteria. While solution A is not dominated
by any other ones since no solutions in the figure can gain improvements in one of the criteria without making the other worse.

- **Pareto ranking**: a solution’s Pareto Rank corresponds to the number of solutions by which it is dominated. E.g. in Figure 5.15 solution A is not dominated by any other solutions, so it’s Pareto Rank value is 1.

- **Pareto set**: is the set of solutions that are not dominated by other solutions on all criteria. In Figure 5.15 all solutions on the red line belong to the Pareto set.

When the dataset of the tool expands a lot, Pareto ranking is extremely helpful to reduce the configuration set. For instance when there are 1000 configurations available, it is not reasonable to show all of them to users of the tool to make their choices. Then a Pareto set is shown to them. All configurations in the Pareto set are not dominated by any others which means they must perform the best in at least one criterion. The tool can use Weighted Sum Approach to evaluate the configurations inside the Pareto set. This will improve efficiency and deliver reasonable amount of proposed configurations to users.

### 5.4.3 Discussion

The algorithm we discussed in Equation 5.2 is not the only scalarization approach to perform the multi-criteria optimization. For example, in the manual process we discussed in Section 4.2, the approach we used is that less important user needs (criteria) are neglected and only the most important criterion is considered. And other approaches are also explored during the implementation process. We chose the algorithm in Equation 5.2 because it can reproduce the result that Philips system architects got manually. In the future more multi-criteria optimization approaches can be explored to see the difference in the output.
6 | Results

6.1 Output

From the description in Chapter 5, the tool takes the user concerned market segments as part of its input. The markets shown in Figure 6.1 are taken from the product instance. In Figure 6.1, there are 5 market segments that can be selected by users. And it shows that the market segments of Peripheral (clinical segment), USA (geographic segment) and QO (i.e. Quality Optimizers, hospital segment) are selected. Users are free to combine any market segments that they want to optimize. Each market has one or more user needs.

```
Please choose from the following markets:
1:Peripheral
2:USA
3:Asia
4:EM
5:QO
1,2,5
```

Figure 6.1: Select concerned market segments

With chosen market segments, the tool will give outputs for each segment and an overall output for the merged markets. As mentioned in Section 5.4.1, the merged segment is combined with all user needs of the selected segments, and if there are duplicate user needs, the value is merged to the higher one as well. With the information in Table 4.5, the related user needs merged by the tool are shown in Figure 6.2 and the output is shown in Figure 6.3.

```
EntireLegImage Merged to 4
HighPatientWeight Merged to 4
SmallRoomSize Merged to 2
LowSystemPrice Merged to 2
LowWorkingHeight Merged to 2
FullBodyCoverage Merged to 4
PositionInIsoCenter Merged to 4
```

Figure 6.2: Merged user needs
In Figure 6.3, the FS value is the final score that the configuration gets for the market segment. E.g., for market Peripheral, configuration 8 gets the highest score (12.00), which means it suits best. The values after the final score are the ranks of that configuration for the user needs. 1 denotes fulfilling the user need best, so it ranks the first, so on. Note that 0 means that the configuration does not fulfill the user need at all. For market Merged, the values after FS are ranks each configuration gets for the user needs following the user needs order in Figure 6.2.
### 6.2 Visualization

The information in Figure 6.3 is not easy to interpret. If we want to know why configuration 2 is better than configuration 3 in the merged segment, it is difficult to distinguish and compare all ranks for the user needs. Visualization techniques solve this problem well. Figure 6.4 is a spider chart representation of the output. The value shown in the spider chart is the $\frac{\text{Importance}}{\text{Rank}_i}$ in the Equation 5.1.

If we choose only configuration 2 and configuration 3 to compare, the spider is as in Figure 6.5. It is straightforwardly visible in the chart that configuration 2 performs much better than configuration 3 in EntireLegimage and HighPatientWeight criteria.

![Figure 6.4: Visualization](image)

### 6.3 Analysis

The output of the tool shows that it can conduct the process automatically. As we discussed in Chapter 1, based on the output the optimal set of configurations can be inferred. For each market segment and combinations of them, we can get the ranked
configurations respectively. Then by examining the frequency of occurrence of a configuration as ranked highest, we can infer how valuable it is to those market segments. Then Philips can decide which configuration should be included in the configuration set. The tool can be expanded to find the optimal configuration set when the decision making process is provided by Philips in the future.

The tool can also be used to compare the system of Philips with its competitor’s product. This is a problem which was proposed by engineers of Philips during the demonstration of the tool. Two datasets can be processed separately by the tool and then the user can compare the scores of configurations. Since we do not have the system information from Philips’s competitors’ products, the comparison is not conducted in this project. But as long as the market related information and the evaluation function are the same, the final scores of configurations are comparable even if they are not from one dataset.
6.4 Validation

The output of the tool is examined by domain experts in Philips. It outputs the same results that they got manually. During the validation phase, we also use a larger dataset to validate our solution. The new dataset is the complete AD7 patient table which is attached in the Appendix. This dataset contains 17 market segments and 14 components. This example gives the feature model shown in Figure 6.7. There are some deficiencies in the Product Definition Language part and in the Data Repository part we found during the validation phase. With some modifications in the original work we successfully conduct the conversion of the new input to the Feature Model in Figure 6.7. For AD7 patient table we do not have the full market and user needs information. So the optimization phase is not entirely conducted. Another problem we found in this phase is that the tool cannot process complex expressions of evaluation rules yet. An example of complex expressions is shown in Figure 6.6. The tool can be expanded to be more powerful in this aspect in the future.

\[
\text{BolusChase requires} \ (\text{LongitudinalMotor} \text{ and} \ (\text{ExtendedTableTop or} \ (\text{StandardTableTop and SwivelFloorPlate}\)))
\]

Figure 6.6: An example of complex expressions
Figure 6.7: Feature Model of AD7 patient table
In this thesis we formalize a variability management problem related to the Philips IGT domain. Variability management is a problem faced by different domains including but not limited to manufacturing companies and software developers. Generalization of the solution is a concern of domain experts of Philips which was proposed at the very beginning of the project. To be able to describe variability management problems in different systems, we defined general concepts in Chapter 3. And the design decisions during the implementation of the tool are made also considering generalization as an important aspect. And as we described in Section 1.2.2, the tool automates the evaluation process.

Now the tool allows users to represent the product information and market data in a compact and consistent manner via the Product Domain Language. And it offers the possibility to support automatic visualization and comparison of configurations. After giving the system information as the input, a user only needs to provide his or her selection of concerned market segments, then human intervention is not needed before getting ranked configurations with regard to market segments.

After the demonstration of the tool in Philips, we got many positive feedback and comments. The following capabilities of the tool are very useful and helpful for system architects and marketeers in Philips.

- It can map uses needs of market segments on different configurations.
- It supports in product definition phase by breaking up variability options.
- It supports to build the business cases to balance between variations and cost.
- Based on the output of the tool, it is possible to optimize the number of configurations.
- It can assist in sensitivity analysis on input data.
Above are the useful capabilities that system architects are interested in. So the goal of the project is achieved. The tool will be delivered to Philips with proper documentations. The documentations will include the explanation of the code, and give suggestions about which part of the code to be modified to have more capabilities. Based on this prototype, system architects and engineers of Philips can expand it to a more powerful and customized tool. There are also parts that can be improved. In the next chapter some improvement possibilities are described.
8 | Future Work

8.1 Validation of DSL instance

The DSL instance has to follow the grammar we defined. If there is an error in the DSL instance, the tool does not provide detailed error message yet. Xtext provides the possibility to validate the instance and to provide very specific and detailed error messages. One simple example is shown in Figure 8.1. In the future if a user uses the tool to express a huge dataset, it is helpful to have the validation part in the DSL implementation.

![Figure 8.1: A DSL instance validation example](image)

8.2 Input format

The input of the tool is currently a textual DSL instance. It is a piece of code which follows the grammar we defined. Successful usage of the tool requires the user to learn how to express required information correctly. Even though the learning effort is small, it is not the format that Philips system architects are familiar with. During the demonstration phase of the tool, it was suggested that a generic and commonly used input format would be more convenient. The required information is currently mostly available in the form of spreadsheet tables (Excel) in Philips. And also the intermediate presentation of the product in the problem instance which is the feature model shown in Figure 5.2 is a more direct way they like to use as the input.
So, they want to have a combination of a graphical, tabular and textual Product Definition Language to give input:

- Graphical part to capture the variability in the diagrams like they originally provide (such as in the appendix).
- Tabular part to capture, for example, the importance of user needs for different markets such as in the Chapter 4 or as in the excel spreadsheets.
- Textual part for expressing the various rules.

### 8.3 Processing complex expressions

Our tool is verified based on a small dataset for the patient table module of IGT systems. Compared to a whole IGT system, the patient table module is only a small part of it. So if we use the tool to express a whole IGT system, it will run into more complex situations which may not be handled well. To make the tool more suitable, there are two improvement points. First, the ability to express complex expressions needs to be improved. Second, the data repository (data processing unit) needs to be expanded as well to process complex expressions in evaluation rules.

### 8.4 Automatic visualization

The visualization of the results shown in Chapter 6 is done manually. An ideal way is to automatically represent the result after the market selection by users. A challenge of the automatic visualization is that users will get no access to full information of results. Visualization techniques are helpful to interpret complex data and it is possible to show the relevant detailed data when a user selects a configuration or a user need in a visualization tool such as TRACE (2015).


Arjan Mooij and Jozef Hooman. Creating a domain specific language (dsl) with xtext. TNO, 2015.


Appendix. AD7 Patient Table Model

The AD7 patient table is a successor of the AD5 patient table for IGT systems. There are many product variants of the AD5 patient table for different user needs and market segments. But none of these versions meet all the needs looking at the clinical trends based on the study of system architects and marketeers in Philips. It is also impossible to evolve the AD5 patient table to an optimal table for all clinical trends, since the AD5 patient table is fully equipped at this moment. Therefore they propose the AD7 patient table. One table that meets all the known user needs and is prepared for all future trends.

The system definition of the AD7 patient table and market segments are shown in the following pages.