A Formal Method to Identify Variation Points in Product Line Assets

Felix Loesch
Robert Bosch GmbH
Corporate Research and Advance Engineering CR/AEY
P.O. Box 30 02 40, D-70442 Stuttgart, Germany
felix.loesch@de.bosch.com

Abstract

Analyzing the provided variability in an evolving product line is necessary to determine if new products can be derived by configuration of existing product line assets. To analyze the provided variability, we need to identify the variation points, the variants, and its constraints. In this paper, we show how formal concept analysis can be used to derive this information.

1 Introduction

Let us assume you are a product line engineer who has been assigned to decide whether the requirements of a new product that should be derived from the product line can be supported by the existing variability in the product line assets. Unfortunately, you are unfamiliar with the existing variation points and variants that can be derived by configuring product line assets. The people that initially developed the product line assets are not available anymore and did not document the variability. Hence, the only reliable source of information are the existing product line assets and the knowledge about variants that is stored in the configuration management system.

Very likely, you are first interested in the *variation points* of specific product line assets, i.e., the locations where a choice can be made and in the *variants* you can choose at these points. In a second step, you want to understand the *constraints* between different variation points that limit the selection of specific variants. The product line consists of hundreds of reusable assets each of which supports a large number of configuration options. Where do you start?

A large number of software configuration management (SCM) systems used in industry support a two-dimensional organization of the version space consisting of *branches* and *revisions* [1]. Typically, *branches* are used to represent variant implementations that exist in parallel, while *revisions* are used to manage incremental changes of specific variants. Each configuration item under version control has an associated version graph consisting of branches and revisions.

One important piece of information for a product line engineer who needs to understand the provided variability is the relationship between *configuration items* and *branches* (variants). For a given set of configuration items e.g. a component defined by the product line architecture, this relationship describes which subsets of configuration items belong to a particular branch or set of branches.

In this paper we show how formal concept analysis applied to this relationship can be used to derive the variation points for a given set of configuration items, to extract the variants at each variation point, and to infer constraints between variation points.

2 Approach

2.1 Overview

Our approach consists of the following steps:

1. The product line engineer selects the configuration items needed for a particular product line asset.
2. For each configuration item f_i, the set of branches B_i associated with f_i is extracted.
3. The data yielded by the extraction of branches is used as input for formal concept analysis.
4. The product line engineer derives the variation points, the variants and constraints by interpreting the concept lattice computed by the concept analysis.

2.2 Concept Analysis of Branches

Formal Concept Analysis (FCA) is a mathematical method to analyze binary relations between objects and attributes. Its foundations were laid by Birkhoff in 1940. Primarily Snelting [3] has recently introduced Formal Concept Analysis to software engineering. For a detailed description of Formal Concept Analysis we refer to [2].

Formal Concept Analysis is based on a relation $I \subseteq O \times A$ between a set of objects O and a set of attributes A. The triple $C = (O, A, I)$ is called a *formal context*. For a set of objects $X \subseteq O$, the set of *common attributes* is defined as $\sigma(X) = \{a \in A \mid \forall o \in X : (o, a) \in I\}$. Similarly, for a set of attributes $Y \subseteq A$, the *common objects* are defined by $\tau(Y) = \{o \in O \mid \forall a \in Y : (o, a) \in I\}$.

In our case, configuration items f are considered objects and branches b are considered attributes. A pair (configuration item f, branch b) is in relation I, if configuration item f has branch b associated to it.

We will use a simple example to demonstrate concept analysis of branches. Suppose we have a software asset A

that consists of six configuration items f_1, f_2, f_3, f_4, f_5, f_6. The following table shows the relationship between configuration items and extracted branches.

	b_1	b_2	b_3	b_4	b_5	b_6	b_7
f_1	×		×				
f_2				×	×	×	×
f_3	×	×	×				
f_4	×		×				
f_5				×		×	
f_6	×	×	×		×		×

A pair $c = (X, Y)$, where $X \subseteq O$ and $Y \subseteq A$ is called a *concept*, if $Y = \sigma(X)$ and $X = \tau(Y)$. Hence, a concept is characterized by a set of objects, called its *extent* denoted by $ext(c)$ and a set of attributes, called its *intent* denoted by $int(c)$ such that all objects share all attributes and all attributes fit to all objects. Informally, a concept is a maximal filled rectangle in the relation table where column and row permutations do not matter.

Formal Concept Analysis applied to our example yields a concept lattice (shown in Figure 1) from which interesting facts can be derived (due to space reasons we only present the most interesting facts):

- Each concept c in the lattice with $ext(c) \wedge int(c) \neq \emptyset$ can be interpreted as variation point for a specific subset of configuration items f. The extent shows the configuration items f that are affected by this variation point. The intent shows the variants for the subset of configuration items that can be chosen at this variation point. For example, the variation point labeled with $\{b_1, b_3\}$, $\{f_1, f_4\}$ affects $\{f_1, f_3, f_4, f_6\}$ and we can choose variant b_1 or b_3.

- A configuration item f exists in only one variant b, if b is the only variant on all paths labeled with f to the top element. This can be interpreted as an *optional variation point* at configuration item level.

- Configuration items that exist in two variants b_1 and b_2, can be identified by the infimum $\mu(b_1) \sqcap \mu(b_2)$. For example, f_6 exists in variant b_2 and b_5.

- If the top element is not empty, the top element describes a variation point that affects all configuration items belonging to the product line asset. In Figure 1 there is no such variation point.

The lattice even reveals important relationships between variation points:

- Variation points (concepts) that appear on the same path from bottom to the top element or have a common infimum are *dependent*, i.e., the selections of variants (branches) at variation points along this path towards the bottom element affect a subset of configuration items already selected at variation points towards the top element because the intersection of affected configuration items of two variation points along a path is always non-empty.

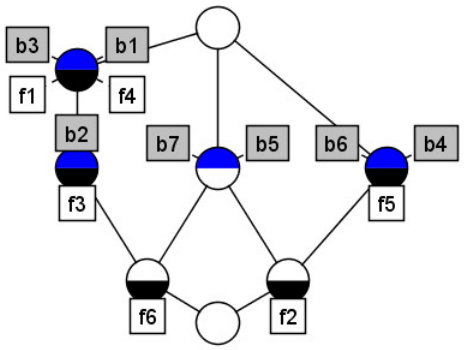

Figure 1: Concept lattice for our example.

- Variation points (concepts) that appear on totally separated paths from the bottom to the top element are *independent*, i.e., the selection of variants at these variation points do not affect each other because the intersection of affected configuration items of two variation points that appear on totally separated paths is always empty.

3 Preliminary Experiences

To evaluate our approach, we selected a reusable product line component from a product line used for electronic control units. The evaluation showed that our method revealed useful information needed for understanding the existing variability in the context of an evolving product line asset.

The information that was derived from the lattice could not only be used to re-document the variation points and constraints but also to restructure and simplify the variability of the evaluated product line asset.

4 Conclusions

The approach presented in this paper allows to derive the variation points, the affected configuration items, and the possible variants that can be chosen at each variation point. Furthermore, our approach supports to derive constraints that exist between variation points.

This information helps the product line engineer to understand the provided variability, to determine if this variability is sufficient to fulfill the requirements of the new product, and to restructure the variability if necessary.

References

[1] R. Conradi and B. Westfechtel. Version models for software configuration management. In *ACM Comput. Surv.*, volume 30, pages 232–282. ACM Press, 1998.

[2] B. Ganter and R. Wille. *Formal Concept Analyis - Mathematical Foundations*. Springer Verlag, 1999.

[3] G. Snelting. Reengineering of configurations based on mathematical concept analysis. *ACM Transactions on Software Engineering and Methodology*, 5(2):146–189, 1996.