Isolated Features Detection in Feature Models [*]

Pablo Trinidad, David Benavides, and Antonio Ruiz-Cortés

Dpto. de Lenguajes y Sistemas Informáticos
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
{trinidad, benavides, aruiz}@tdg.lsi.us.es

Abstract Feature models are commonly used to describe software product lines in terms of features. Features are linked by relations, which may introduce errors in the model. This paper gives a description of isolated features and states the detection of them, as the first step in their treatment. Two implementations are given to automatically support isolated features detection and a third one that uses both and improves the performance.

1 Isolated Features in Feature Models

1.1 Introduction

Software Product Lines (SPL) approach is being used in organizations to shift from developing separated and similar products one by one, to developing products from a set of existing assets. A successful software reuse, higher software quality and lower time-to-market are some of the benefits of applying SPL. A good understanding of the domain where SPL are to be applied is needed. One of the ways of describing the requirements of a SPL is feature modeling.

Feature modeling is a declarative way to represent a domain of applications in terms of visible end-user's features. Features are externally visible characteristics that can differentiate one product from the others. Depending on the context they may refer to requirements, components or source code (feature oriented programming).

When modeling SPL using feature models(FM), it is important to check for the correctness of the model. Errors may be introduced because of badly defined relationships among features. Correctly defining the set of products within the SPL and checking for the absence of features that may not appear in any product are examples of activities to be realized for this debugging. Many times, FM are big enough to avoid a manual handling of them, therefore an automation on debugging FM is needed.

1.2 Preliminaries

There are several notations for representing FM [2,3,4]. Czarnecki's notation defines the domain in terms of features using a hierarchical representation. A feature is drawn

[*] This work was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01 (AGILWEB)

as a box with its representative name. Four different kinds of relations among features may be used: mandatory, optional, alternative and or-relation. Relations are stablished among a parent feature and one (in the case of mandatory or optional) or more (in the case of alternative and or-relation) feature's children. Besides these relations types, two more non-hierarchical relations are defined in order to represent more complex FM: depends and excludes. Figure 1 represents a Home Integration System (HIS) SPL using Czarnecki's notation. When defining one FM some mistakes can be made, usually as a

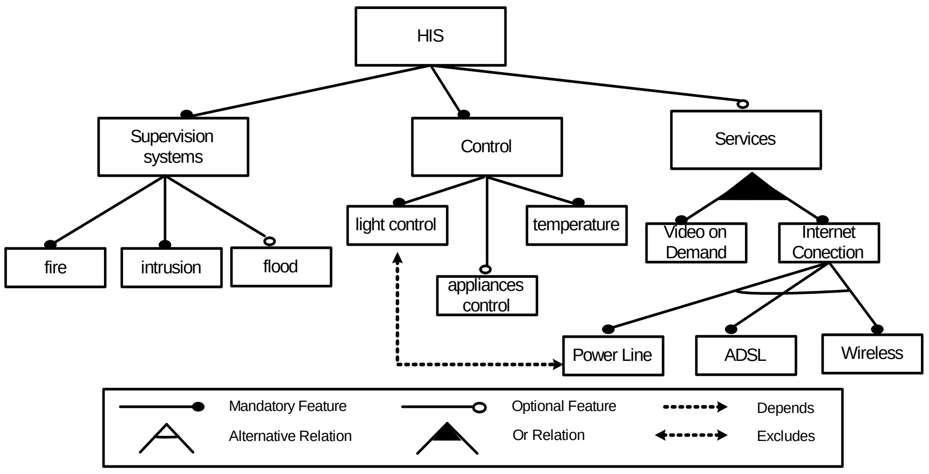

Figure 1. HIS Feature Model. Power Line is an isolated feature

consequence of an unappropiate use of depends or excludes constraints[8]. These constraints can cause some features not to appear in any product in the FM. For example, notice in Figure 1 the mutex between Power Line and Light control features. It means that there could be no product with both features at the same time. Light control is a full mandatory feature, so it appears in every product. Although, Power Line feature is represented as an alternative of Internet Connection, no product can contain it because it is incompatible with Light Control. Therefore, Power Line is an isolated feature.

1.3 Isolated Features Treatment

When defining the relations among features, errors can be introduced in the model. One common error is making a feature not to appear in any product of the product family defined in the FM. It is important to be able to detect such features and being able to give solutions to them. At first, we define the concept of isolated features:

Isolated Feature A feature in a FM which does not appear in any of the products described within the FM.

Detecting isolated features is only the first step in a three-steps treatment process:

1. Detecting isolated features in a FM
2. Detecting the relationship/s that cause isolated features to appear
3. Giving possible solutions to avoid a feature to be isolated.

2 Our proposal

2.1 Isolated Features Detection

In this paper we deal with the automated detection of isolated features in FM. We base our proposal on the definition of feature commonality. In [1], a feature commonality is defined as the ratio of products containing that feature, and is a powerful tool to detect isolated features.

Definition 1 (Feature Commonality). *Number of products where the feature appears respecting to the total number of products of the SPL.*

An isolated feature appears in no product, so it would have a zero commonality. According to this, and based on Definition 1, we define the next concepts :

Definition 2 (Isolated Feature). *Let M be a FM and f the feature which commonality we know. Feature f is isolated iff its commonality is zero.*

$$Isolated(f, M) \iff (Commonality(f, M) = 0)$$

2.2 Implementation

Taking into account the previous definitions, automated detection depends on automated support for calculating commonality for every feature in a FM. We propose three different alternatives for detecting isolated features taking the leverage on CSP and variation degree to calculate commonality:

- **Using Constraint Satisfaction Problems (CSP):** In [1], we used Constraint Satisfaction Problems(CSP) to represent FM, allowing to realize different operations for extracting relevant information and reasoning on them. One of the stated operations is $Comm$, that can be used to calculate commonality. For each feature in the feature model, $Comm$ is used to detect if its commonality is zero.
- **Using Variation Degree:** In [7] variation degree of a feature is defined as the number of possible instantiations below this feature. An algorithm to calculate this variation degree is proposed, but it only determines the exact variation degree with one dependency (depends or excludes relation) or none. If there are multiple dependencies, only an approximation can be given, calculating the low and upper bounds where the variation degree takes values from. The results comming from variation degree algorithm can be taken to calculate commonality[6], but only bounds can also be obtained. If the low bound is zero for a feature, it is potentially isolated, but a precise result cannot be calculated.
- **Using CSP and Variation Degree:** To obtain a better performance in isolated features detection, we propose a detection method in two phases that uses both previous alternatives:
  1. Calculate commonality from variation degree for every feature in the FM considering all the dependencies. If exact values are obtained for some features, it can be used to check if they are isolated or not. In the case of obtaining bounds, if one feature had a zero low bound, it is candidate for being checked if it is isolated or not in next phase.

2. Every potentially-isolated feature detected in the phase one, is checked to be isolated or not using CSPs.

3 Conclusions and Future work

In this paper, we give a definition of isolated feature and the path to follow to treat them. As a first step, Feature commonality calculation is proposed to detect them, and three different implementations are given to automate the detection. There is still a gap to cover in detecting the relationships causing isolated features to appear and giving solutions to them. Our future work will focus on proposing solutions to these problems. We think our proposal can still be improved by finding a better model of the CSP problem and using CSP compilation techniques. Our final objective is the integration of these and other debugging techniques into a tool for feature modeling.

References

1. D. Benavides, Ruiz A. Cortés, and P. Trinidad. Automated reasoning on feature models. *LNCS, Advanced Information Systems Engineering: 17th International Conference, CAiSE 2005*, 3520:491–503, 2005.
2. K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.
3. M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, pages 76–85, Canada, 1998.
4. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
5. J. Henk Obbink and Klaus Pohl, editors. *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*. Springer, 2005.
6. P. Trinidad, D. Benavides, and Ruiz A. Cortés. Automated inconsistencies detection in feature models. *SPLC 2006(submitted)*, 2006.
7. Thomas von der Massen and Horst Lichter. Deficiencies in feature models. In Tomi Mannisto and Jan Bosch, editors, *Workshop on Software Variability Management for Product Derivation - Towards Tool Support*, 2004.
8. Thomas von der Maßen and Horst Lichter. Determining the variation degree of feature models. In Obbink and Pohl [5], pages 82–88.