MAPPING FEATURE DIAGRAMS TO UML MODELS
A Transformational Approach

Miguel A. Laguna, Bruno González-Baixauli and Rubén Fernández
Department of Computer Science, University of Valladolid
Campus M. Delibes, 47011 Valladolid, Spain

Keywords: Feature Models, Software Product Line, Model Transformation.

Abstract: Variability and commonality management is one of the key aspects in the development of software product lines. Feature models embody various different variability facets that must be mapped to UML models to trace the variability from requirements to the architecture (and implementation) of the product line. In this context, this article presents the experiences with pattern identification in feature models and their relation with the corresponding architectural UML counterparts. The work includes the definition and implementation of the meta-model based transformations between these models. Finally, an example of application of the transformations completes the article.

1 INTRODUCTION

The product line (PL) paradigm of software engineering involves many technical and organizational challenges (Bosch, 2000). The development of a product line includes two main categories of software artifacts: the artifacts shared by the members of the product line and the product-specific artifacts. The product line itself is a set of reusable assets, where three abstraction levels can be clearly identified (requirements, design and implementation assets). In the requirement level, one of the key activities is the specification of the variability and communality of the product line. The design of a solution for these requirements constitutes the domain architecture of the product line. Later, in the application engineering process, the application architecture must be derived from the domain architecture. In this process the customer functional and non-functional requirements are used for choosing among alternative features. This activity is essentially a transformation process where a set of decisions at the requirements level generates the initial feature product model and, consequently, via traceability paths, the architecture of the product (Bosch, 2000).

One of the most critical points is the elicitation and analysis of variability in the product line requirements. In addition to the information that expresses the requirements themselves, it is important to know the variability of the requirements, and the dependencies between them. In this context, feature models (Kang et al., 1990) are the basic instrument to analyze and configure the variability and communality of the software family. On the other hand, the PL architecture is documented using UML diagrams that are organized in packages. Apart from the base package, each optional feature must have a counterpart in a package which includes the set of class diagrams, use cases and sequence diagrams that are the solution that achieve this feature. The packages are structured using the UML package merge mechanism. In (Laguna et al., 2007), we explained the application of this technique to the general organization and configuration of the product line architecture.

The experiences so far consist of manually designing and implementing each package of the UML models, adding later the user interface and persistence details. This manual approach has been successfully applied to the development of several product lines in the Web and mobile applications domains. However, the productivity (and the complexity in non trivial product lines) demands to automate the construction and configuration of the diverse product line models, following a Model Driven Engineering (MDE) approach. The transformation from goal to feature model has been treated by Yu et al. in their work on Goal-Oriented

Laguna M., González-Baixauli B. and Fernández R. (2009).
MAPPING FEATURE DIAGRAMS TO UML MODELS - A Transformational Approach.
In *Proceedings of the 4th International Conference on Software and Data Technologies*, pages 295-298

requirement engineering (Yu et al., 2008). Basically they use a catalog of goal patterns and their corresponding feature constructions. This article deals with the analogous transformation of the feature models into architecture models, including the package organization and a first cut of the package contents.

The rest of the paper is as follows: the next Section describes a catalog of feature patterns and their correspondences with architectural UML models and the transformation between these models. Section 3 presents an analysis of the results obtained in a comparative case study. Section 4 introduces related work and Section 5 concludes the paper and proposes additional work.

2 A CATALOG OF FEATURE AND UML PATTERNS

Once the feature diagram of a product line is established, several design level UML models must be developed. Our intention is to build a catalog of commonly used derivations of feature to UML models. Sochos (Sochos et al., 2004) have reviewed recently the approaches apart from proposing a new one. The classical works of Kang (Kang et al., 1998), Czarnecky (Czarnecki & Eisenecker, 2000), Griss (Griss et al., 1998), or Bosh (Bosch, 2000), between others have allowed to identify a set of feature patterns that potentially can populate the catalog. A revision of the literature has revealed that it is naive to pretend a simple and univocal transformation from feature models to UML diagrams. Therefore, we have adopted a pragmatic and multi-view approach: separate the different categories of features in a variability model and treat each of these categories in a different way, with an emphasis in structural features.

In previous work (Laguna & González-Baixauli, 2008) we have described two different approaches of these feature transformations patterns. The first approach, based on the cited literature, directly transforms feature models into classes, relationships, and attributes. The general mapping creates class and attributes from features. Mandatory features imply a 1..1 composition relationship, optional features imply a 0..1 composition relationship and groups of features originate specialization relationships.

The problem is that this approach does not take into account the difference between PL variability and the possible variability of the product, for that reason in we proposed a second approach that combine the existing transformations wiht the package merge mechanism of UML 2. The strategy is based on the three subtypes of *Feature*. The root of every tree in a feature model (*RootFeature*) is transformed into a base package (and an initial class, which will generally be discarded) and a recursive transformation of SolitaryFeatures and FeatureGroups linked to every feature is carried out. The presence of a group implies a class associated to the parent feature that is specialized in several subtypes (one per alternative feature). Previously, a new merging package is created if the feature is optional.

The feature meta-model and the QTV transformations have been presented in (Laguna & González-Baixauli, 2008). The package content must be revised and completed, but the package structure itself can be used afterwards to automatically derive the product model by selecting the desired features. This possibility compensates for the overcharge of complexity that the traceability management and the extensive use of packages in the architectural models entail.

Concerning the implementation details, the C# language and the Microsoft .NET platform have been selected because of the direct support of the package merge mechanism by means of partial classes. For this reason, we have developed, using the Microsoft DSL tools, a specific domain language, functionally equivalent to the *fmp* eclipse plug-in (Antkiewicz & Czarnecky, 2004), integrating all the steps of the process, from feature model definition to UML generation and the package configuration of each final product inside the Microsoft Visual Studio platform. This tool implements internally the transformation of the feature models serializing and manipulating the feature and UML package models as XML (and XMI) files.

3 EMPIRICAL EVALUATION

To validate the proposed catalog and transformations, a case study was selected. In particular, it is a portion of the electronic commerce product developed in (Lau, 2006). This election provides us with an interesting starting point to contrast our techniques, since the manually implemented packages are imposed by an external independent study, avoiding the temptation of proposing a problem once we have a predefined solution. The PL architecture was obtained

therefore in two independent ways: in a fully manual development process (employing the package merge approach) and using the proposed transformations. The first option was carried out by a group of graduate students and the results are available at the GIRO Web. At this moment, the common part of the product line and a dozen packages have been fully developed. Hundreds of e-commerce systems can be generated, from a minimal combination to a typical portal with registered users, shopping cart, credit card secure payment, multiple categories catalogs, search criteria, etc.

Once the manual development was achieved, a derivation of the package an internal structure was carried out, using the mentioned transformations. Comparing the results with the manually obtained, we can have an idea of the utility of the process. Table 1 summarizes the results.

The manual elements column is the reference and, logically, has many more elements than the generated version as this last aims only to be a first cut of the architecture. The present elements column indicates in percentage how many automatically generated elements exist in the manual version.

The coincidence in the basic structure (packages and merge relationships) is almost a 100%, so it looks reasonable to use this automatic transformation to create the framework structure of the PL. The problem comes with classes, attributes and relationships generation. The accuracy between the manual example and the automatic transformation is not so good. Only the 37% of the classes are present in both the manual and generated packages and the percentage of attributes and relationships are even lower.

Table 1: Structural UML elements generated from feature models and percentages of usefulness.

Type of elements	Manual elements	Percentage of present elements	Percentage of useful elements
packages	14	81 %	87 %
merge rel.	12	100 %	100 %
classes	39	37 %	58 %
generalizations	10	30%	100 %
associations	27	16 %	46%
attributes	56	5 %	100 %

However the useful elements (how many of the generated elements would have been used directly to define the architecture) is much better in the class, relationships, and attribute rows. The percentage of useful classes is almost 60%, which is a satisfactory result. The results of attributes are even better, because all attributes generated automatically have been used in the manual development. A similar situation occurs with the generalizations and associations rows (100 and 46% respectively). These observations imply that, in spite of the need of manual completion, many packages can be partially generated. Studying results by packages, typed features and group/grouped combination features are the best candidates to the automatic transformation. On the contrary, the base package is clearly the most difficult to generate or at least the package that will suffer more changes by the manual intervention of the PL developer.

4 RELATED WORK

Schobbens et al. (Schobbens, Heymans, & Trigaux, 2006) have revised the diverse variant of feature diagrams, clarifying the differences and establishing a generic formal semantics. The influence of non functional requirements preferences in variant selection has been faced by several authors. The original FODA proposal uses the feature models for representing all the types of variability, functional and non functional (Kang et al., 1990). Bosch (Bosch, 2000) proposes an assessment method that addresses design decisions and non functional requirements in product-line development. In his approach, different profiles are used in relation to different “ilities” (usage profile for reliability or change profile for maintainability). Finally Yu *et al.* present in (Yu et al., 2008) a model-driven extension to their Early Requirements Engineering tool (OpenOME) that generates an initial feature model from stakeholder goals.

Also the work devoted to relate feature constructions and architectural designs is abundant. Recent proposals express variability with UML models modifying or annotating these models. Structural, functional or dynamical models have been used. Concerning structural models, either the mechanisms of UML are used directly (through the specialization relationship, the association multiplicity, etc.), as in the case of Jacobson (Jacobson et al., 1997) or the models are explicitly annotated using stereotypes. The work of Gomaa is an example of this approach, since it uses the stereotypes <<kernel>>, <<optional>> and <<variant>> (corresponding to obligatory, optional, and variant classes) (Gomaa, 2000). Similarly, Clauß proposes a set of stereotypes to express the

variability in the architecture models: <<optional>>, <<variationPoint>> and <<variant>> stereotypes designate respectively optional, variation points (and its sub-classes), and variant classes (Clauß, 2001).

Finally, we have already mentioned the works of Sochos (Sochos et al., 2004), the FORM method (Kang et al., 1998), the generative approach of Czarnecky (Czarnecki & Eisenecker, 2000), or the RSBE method (Griss et al., 1998) as references in order to populate the feature pattern catalog.

5 CONCLUSIONS

The main contribution of this article is the identification of patterns in the feature models and the mapping of these patterns with the correspondent architectural diagrams. The feature patterns catalog allows the automated creation of traceability links between the product line feature and the architectural models, consequently improving the product derivation process.

Our approach to product lines is based on the MDE paradigm, aiming to automate many of the phases of product line development. In particular and using the proposed catalog, the UML domain models can be obtained from feature models using QVT pattern transformations (though these models must be manually completed). The implementation of the transformations as part of a conventional IDE tool facilitates the work of the product line developers. And, using its configuration capabilities, the tool is also useful for the developer in charge of deriving the final products. A first experience has allowed the contrast with reality and the obtained results are encouraging.

As future work, we consider an alternative configuration process. The design level (instead of fully implemented) packages can be merged into a monolithic model (using existing MDE tools). The resulting (platform independent) model will be used as input to code generators tools. These tools are precisely intended to generate the platform specific models and the final code. We are evaluating some of the best known tools in order to assess the practical possibilities of this product line and MDE alliance.

ACKNOWLEDGEMENTS

This work has been supported by the Junta de Castilla y León (project VA018A07) and Spanish MICIINN (project TIN2008-05675).

REFERENCES

Antkiewicz, M., & Czarnecki, K. (2004). Feature modeling plugin for Eclipse. OOPSLA'04 Eclipse technology exchange workshop.

Bosch, J. (2000). Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach. Addison-Wesley.

Clauß, M. (2001). Generic modeling using Uml extensions for variability. Workshop on Domain Specific Visual Languages at OOPSLA.

Czarnecki, K., & Eisenecker, U. W. (2000). Generative Programming: Methods, Tools, and Applications. Addison-Wesley.

Gomaa, H. (2000). Object Oriented Analysis and Modeling for Families of Systems with UML. IEEE International Conference for Software Reuse (ICSR6), (pp. 89–99).

Griss, M. L., Favaro, J., & d'Alessandro, M. (1998). Integrating feature modeling with the RSEB. Proceedings of the Fifth International Conference on Software Reuse, (pp. 76-85).

Halmans, G., & Pohl, K. (2003). Communicating the Variability of a Software-Product Family to Customers. Journal of Software and Systems Modeling , 15--36.

Jacobson, I., Griss, M., & Jonsson, P. (1997). Software Reuse. Architecture, Process and Organization for Business Success. ACM Press. Addison Wesley Longman.

Kang, K. C., Kim, S., Lee, J., & Kim, K. (1998). FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering , 143-168.

Kang, K., Cohen, S., Hess, J., Nowak, W., & Peterson, S. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21, Software Engineering Institute (Carnegie Mellon), Pittsburgh, PA 15213.

Laguna, M. A., González-Baixauli, B., & Marqués, J. M. (2007). Seamless Development of Software Product Lines: Feature Models to UML Traceability. GPCE 07.

Laguna, M. A. & González-Baixauli, B (2008). Feature Patterns and Product Line Model Transformations. DSDM'08.

Lau, S. (2006). Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates", . MASc Thesis, ECE Department, University of Waterloo, Canada.

Schobbens, P.-Y., Heymans, P., & Trigaux, J.-C. (2006). Feature diagrams: A survey and a formal semantics. RE , 136–145.

Sochos, P., Philippow, I., & Riebish., M. (2004). Feature-oriented development of software product lines: mapping feature models to the architecture. En LNCS 3263 (pp. 138-152).

Yu, Y., Lapouchnian, A., Leite, J., & Mylopoulos, J. (2008). Configuring Features with Stakeholder Goals. ACM SAC RE Track.