INTERNATIONAL CARPATHIAN CONTROL CONFERENCE
ICCC'2010
Eger, Hungary
May 26-29, 2010

FEATURE MODEL-DRIVEN SOFTWARE DEVELOPMENT

Dávid ZAKÁL and László LENGYEL
Budapest University of Technology and Economics, Hungary, {zakal, lengyel}@aut.bme.hu

Abstract: Software product lines enable the development and mass production of a set of similar systems by relying on predictable reuse of components. Instead of general collections of reusable assets the approach applies domain-specific, configurable artifacts for mass customization. Feature modeling provides an implementation-independent, compact representation for entity variants in the modeling domain. Features enable the hierarchical description of commonalities and differences between the entity variants. These characteristics make feature modeling a natural choice for the representation of variabilities in software products. Taking into account the characteristics both of the software product lines and modeling, current paper presents a feature modeling-based approach for software product lines.

Key words: Software Engineering, Software Product Lines, Feature Modeling, Domain Engineering

1 Introduction

As software engineering improves, coping with the challenges of managing complexity and continuous changes, the expectations of stakeholders constantly increase as well. From a technical point of view, this increasing business demand is rooted in the rapid improvement and innovation of platform technologies [1], which offer a lot of novel possibilities. When building a large variety of products for any platform, it is a great challenge to complete them on schedule and within budget, without violating any requirements. The degree of automation and the number of skilled developers is insufficient alone to deal with the enormous demand.

Considering that improving the skills of developers in order to increase productivity does not scale up well [1], the solution lies in systematic reuse of domain-specific assets, supported by automation whenever possible. This also enables a more intelligent distribution of human resources and allocated competence. For example, domain experts do not need to be familiar with the concrete implementation concepts and thus can focus on the product requirements on a high level. Similarly, software implementers do not require thorough knowledge of the product domain.

Instead of specifying and developing each product individually, product families can be created by capturing the common and different features of the products. This domain-specific (and thus properly specified and restricted) variability is implemented and supported by reusable assets. They can be combined to form a concrete product variant of the family. This approach, Software Product Lines, greatly increases productivity, reduces risk and costs, and reserves these advantages to the future variants as well during the evolution of the product line.

Product lines are specified in the modeling space, and realized in the implementation space by relying on reusable assets. This logical separation enables a model-driven approach for the design of product lines.

Feature modeling is a natural choice for the compact and complete high-level specification of product lines. Feature models hierarchically express the common and variable features of the instances of a concept, and represent the interrelations between the variabilities [2]. They ensure that relevant features are always included in the concept, while unnecessary ones are never present. Feature models are visualized by feature diagrams, offering a compact and expressive graphical representation.

In case of Software Product Lines, the concept of a feature model is the product family. The feature hierarchy is determined by the relevant common and variable characteristics of the products in the given family. For example, a mobile phone product family can have the following features: battery, screen, supported radio network, supported communication. They can be further decomposed into subfeatures, e.g. screen to screen size and screen type (normal or touchscreen), or supported communication type to wired and wireless. Wireless communication can be further divided into WiFi, IR (Infrared) and Bluetooth. The refinement is continued as long as it results in relevant features for the product line design. Implementation-specific, low-level details should not be included. The diagram of the presented example can be seen in Fig. 1.

Even though the discussed feature model example (mobile device family) does not originate from software industry, the presented modeling approach and technique can be fully applied to software products as well. Current example was chosen for the sake of clarity, as it is simple and expressive.

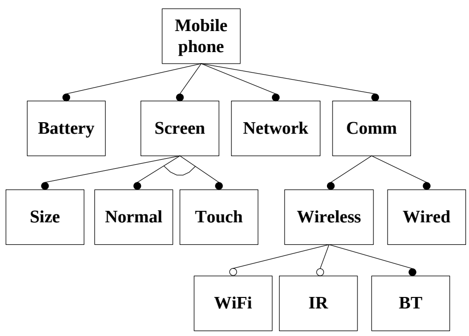

Fig. 1. A feature diagram representing a possible mobile phone product family

This paper discusses a model-driven Software Product Line Engineering approach, placing a great emphasis on the version that relies on feature models. The rest of the work is structured as follows: it presents the general approach in details, introduces the use of feature models through an example, and examines the relevant software engineering aspects. Finally, conclusions and further work are delineated.

2 Contribution

While the separation of model space and implementation space is accomplished along the dimension of *abstraction*, deriving products from the family model changes on the degree of *variability*. Taking into account these two dimensions, we can represent the relationship between products, product lines and their implementations (Fig. 2). Note that the presented model, inspired by [3], is general and does not require or assume the use of feature models.

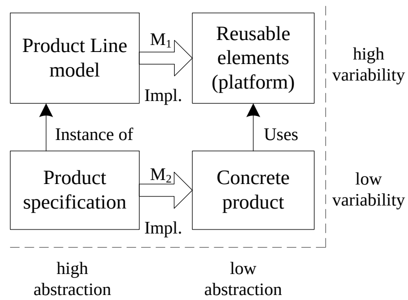

Fig. 2. Model-driven product lines

Product Line models capture the common and variable features of the members of a product family. They serve as the metamodel of product specifications, residing on the same abstraction level as their instantiations. Product Line models can be expressed by feature models or Domain-Specific Languages (DSL).

A product specification defines the actual features of a product variant, conforming to the parent Product Line model. For example, it specifies which optional and alternative features of the family model are present in this variant. Depending on their metamodel, product variant specifications are based on feature models or represented by DSL instance models.

The concrete product variant implements the product specification, relying on the elements and services of a core platform. Note that the current model makes no assumptions about the actual implementation of the platform (e.g. it may be directly integrated into the software package as data and executable code, or may be available as a host runtime or hardware environment). Furthermore, the *platform* in our model is just one source of reusable assets, providing reusable *elements* for product implementations. Other manifestations of reusable production assets (development tools, processes, tests etc.) reside in other parts of the presented model, e.g. in model processing and implementation activities (indicated in Fig. 2 by mapping M_1 and M_2).

2.1 Product from specification

The key point in understanding the model-driven reuse and implementation is the mapping of a product specification to a concrete product (M_2 in Fig 2). This is a complex process that is partially or completely automated (Fig. 3).

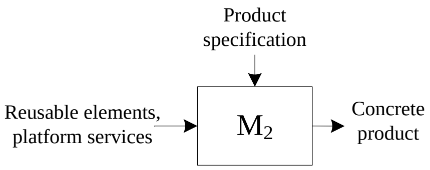

Fig. 3. Implementing a product variant

Once the Product Line platform is complete (mapping M_1 is finished), all the implementation primitives are available as reusable components and platform services. The only remaining task, performed by M_2, is combining the required components and services to build the concrete software product. M_2 can be considered as an intelligent "multiplexer" whose input is a set of components and services, while the selection line (filter condition) is the product variant specification. The output is an assembled product which relies on a specified subset of the components and services that served as the input of M_2.

Due to the model-driven Software Product Lines approach, the discussed implementation process can be highly automated. M_2 is performed and supported by various reusable assets: architectural and mediation patterns, DSL model processors, code generators, etc. These are specific to the Product Line domain, but generic enough to support the production of each product variant, hence highly reusable.

2.2 A feature model-based example

Fig. 4 depicts mobile software development in a feature model-driven Product Lines context. Notice that this is a specialized, domain-specific version of the model presented in Fig. 2.

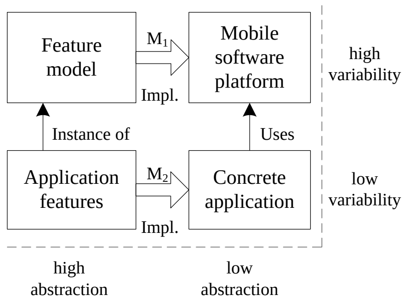

Fig. 4. Feature model-driven Product Lines supporting mobile application development

The feature model serves as the core model for the family of mobile application variants, capturing their high-level, family-specific features and their variabilities. If there are many relevant features and relations to be modeled for the product line, the central model can be composed as a hierarchical set of distinct feature models. This improves the quality of feature diagrams, ensuring that the visual graph representation does not get too complex for the human eye.

Features of a particular application variant are derived from the aforementioned core model by the domain expert of the product family. The software product is customized and localized in this process. For example, features are fixed (or even omitted completely) regarding the technical, cultural and legal environment where the product will be used. As the core feature model provides all information in a known terminology and visual representation, the domain expert does not need any software development skills when specifying the variant.

The implementation of the application includes automated and semi-automated tasks.

Reusable software components are assigned to the modeled features in mapping M_2. This enables building a significant part of the application architecture with automatic code generator tools. It is not worth investing into such a tool for a few uses, but we have many mobile applications variants, especially during the long-term evolution of the Software Product Line. Consequently, code generator software that realizes a part of M_2 is an example of a reusable asset in our mobile Product Line. Architectural patterns shared by all variants are another example.

When specifying M_2, several concepts of feature model theory [2] can directly be applied. Many development steps naturally follow from feature availability, binding sites and binding modes. For instance, our feature model-based variant specification tells whether an element (representing a feature) should be statically linked at build time of the mobile application, or should be dynamically bound at run time.

The mobile software platform in the current model is a composite concept, and covers much more than merely the device platform and operating system that hosts the application. This comprises all reusable elements that the applications use, varying from third party frameworks to internal component libraries and code snippets created specifically for this mobile software Product Line. Since a big codebase is already available, the mapping M_1 does not only create reusable assets (including components) but also makes *references* and scenarios for already existing assets. To summarize, M_1 defines the platform, creating new assets only if necessary.

3 Conclusions and further work

This paper has presented a model-driven approach of Software Product Lines, suggesting the use of feature models as integral parts of product family specifications. It discussed that domain-specific reusability relying on implementation-independent models increases productivity and enables better leveraging of domain and developer competence.

Our further work focuses on automatic Product Line feature implementation by means of domain-specific model processing algorithms and techniques.

Acknowledgement

This paper was supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences.

References

[1.] GREENFIELD, J., SHORT, K. ET AL. 2004. *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*. John Wiley & Sons, 2004,. ISBN 978-0-471-20284-3

[2.] CZARNECKI, K., EISENECKER, U.W. 2000. *Generative Programming: Methods, Tools, and Application.* Addison-Wesley, 2000,. ISBN 978-3-540-43483-2. pp. 86

[3.] BEUCHE, D., DALGARNO, M. 2006. *Software Product Line Engineering with Feature Models*. http://www.methodsandtools.com/archive/archive.php?id=49