Feature Dependencies have to be Managed Throughout the Whole Product Life-cycle

Goetz Botterweck, Kwanwoo Lee

Lero – The Irish Software Engineering Research Centre
University of Limerick
Limerick, Ireland
goetz.botterweck@lero.ie

Hansung University
Seoul, South Korea
kwlee@hansung.ac.kr

Abstract: In this position paper, we discuss feature dependencies as one major challenge in product line engineering. We suggest that (1) feature dependencies should be treated as first class entities and (2) dependencies in various artefacts across the software life cycle should be mapped onto each other.

1 Introduction

Product line engineering [PBL05, ClNo02] is based on the assumption that under certain conditions it is more efficient to *derive* products from a product line than it would be to develop these product from scratch.

However, the make that promise a reality the application engineering processes, which derive the product (and account for the product-specific effort), have to be performed as efficiently as possible. For instance, in the schematic overview shown in Figure 1, we would have to improve the processes *Product Configuration* and *Product Derivation* to make the creation of a *Product* more efficient.

One potential approach would be to group all *interactive* decisions, which require human intelligence and creativity, into the process of *Product Configuration* and try to *automate* the mechanic assembly and generation of the product in *Product Derivation* as much as possible. Then, after we configured a product we could just ``press a button" and the product would be created by automated mechanisms.

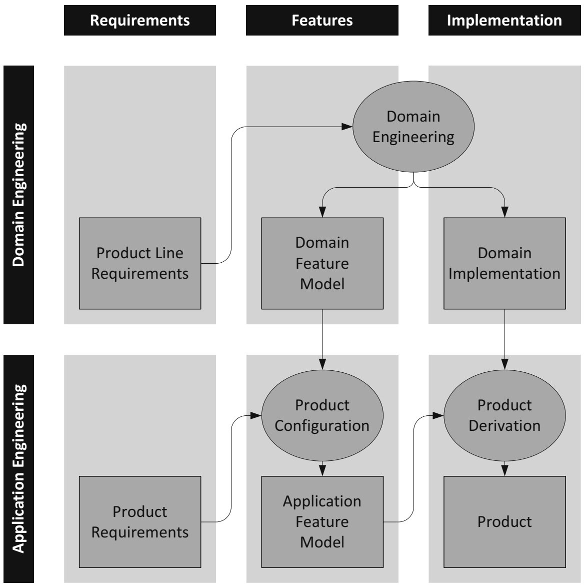

Figure 1: Schematic overview of product line engineering

2 Research problem: Dependencies

So what is stopping us from reaping all the efficiency benefits that product line engineering seems to promise?

In our opinion, one of the main challenges in product derivation lies in dependencies between features. For instance, consider the illustration in Figure 2: In product line engineering it is common engineering practice to describe the product line with various artefacts, such as requirements models, feature models and implementation models. Each of these models contains elements (requirements, features, components) and dependencies between them.

Now, if we want to support the flexible configuration and derivation of products, where we can choose arbitrary configurations (as long as they fulfil the constraints set by the feature model) we have to support all potential combinations and, while doing so, take into account the corresponding dependencies.

If we would follow a naive approach for implementing dependencies, we would create implementation components for all the various cases. For *n* optional features, in the worst case, we might end up with 2^n implementation components. However, we are looking for techniques to turn this into “just *n* implementation components for *n* features". So how do we do that?

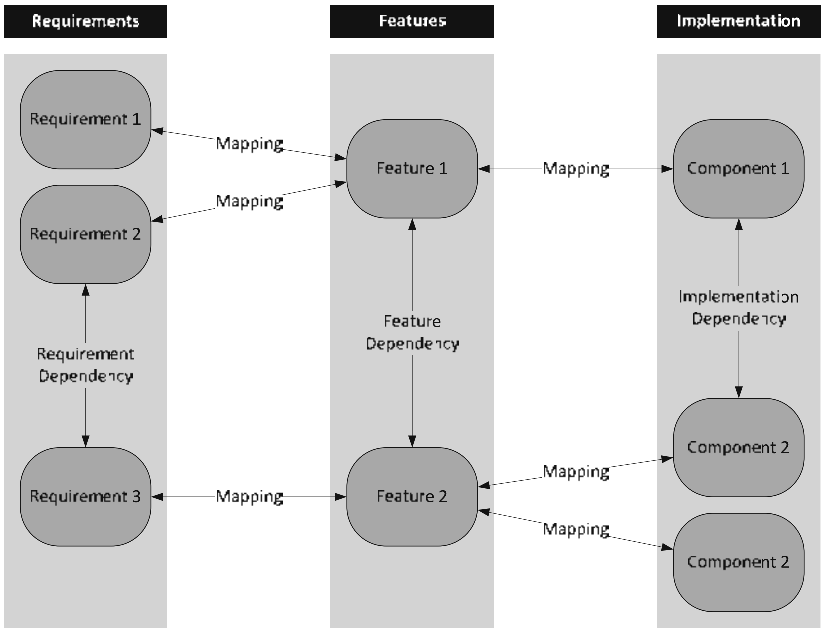

Figure 2: Mappings and dependencies between SPL elements

3 Approach: Managing Dependencies Throughout the Product Life-cycle

One of the things that force us take into account too many combinations are feature dependencies -- because they cause, that the change in the selection of just one feature can influence all kinds of implementation components.

Consequently, in our opinion one potential approach to address the described challenges lies in the improved handling of dependencies. Concretely, we see the following techniques:

- In all involved artefacts throughout the life cycle, dependencies have to be treated as *first class entities*. In other words, *dependencies have to be isolated into their own units of description*. For instance, when coding and structuring the implementation, we should modularize the implementations of features dependencies into separate components. Similar applies to dependencies in feature models and requirements.
- Just like mapping of features to components, we should *map dependencies in various artefacts onto each other*. For instance, each feature dependency should be mapped to a corresponding dependency implementation.
- To allow for flexible combinations of feature implementations and dependency implementations, we have to apply techniques than can generate, assemble or weave implementation units [VoGr07, StVo06]

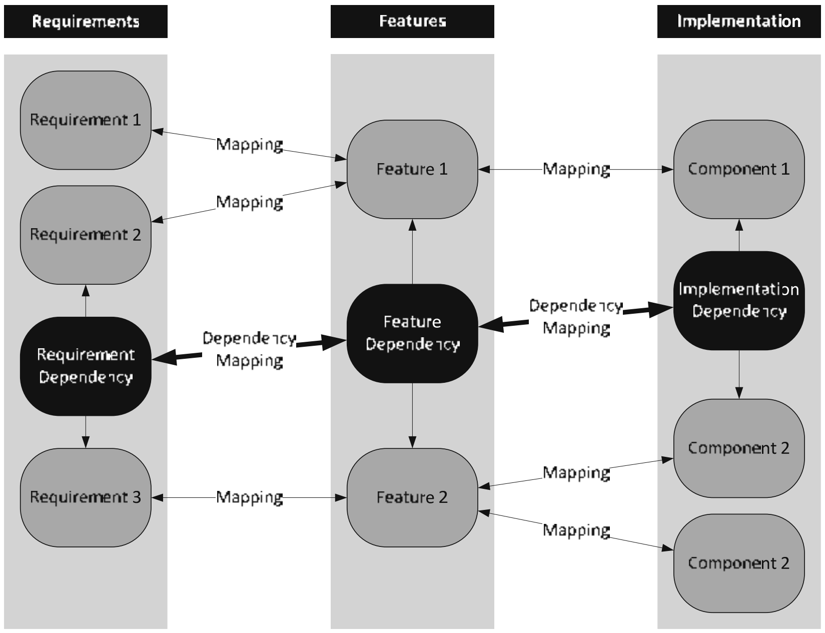

Figure 3: Dependencies as first class elements (with dependency mappings)

- Especially aspect-oriented techniques allow to encapsulate the implementation of a dependency (as a cross-cutting change) into its own unit of description

Starting from the earlier situation (Figure 2) the suggestions described above result in a situation as illustrated in Figure 3, where dependencies are treated as first class entities and dependencies in various artefacts across the life-cycle are mapped onto each other.

4 Sample Case: Calculator Product Line

To illustrate our approach, we want to briefly discuss an example which is taken from earlier research [BLT09], see the diagram in Figure 4.

The upper part of the diagram shows extracts of the feature model of a sample product line of calculators. It contains the optional features `Mode`, `Notation`, `History`, and `NumberSystem` as well as the mandatory feature `Off`. The entities labelled *`d1`* to *`d5`* represent dependencies between features. For instance, *`d1`* describes that the feature `Off` (functionality to switch off the calculator) influences the run-time behaviour of feature `History` (functionality to recall values calculated earlier).

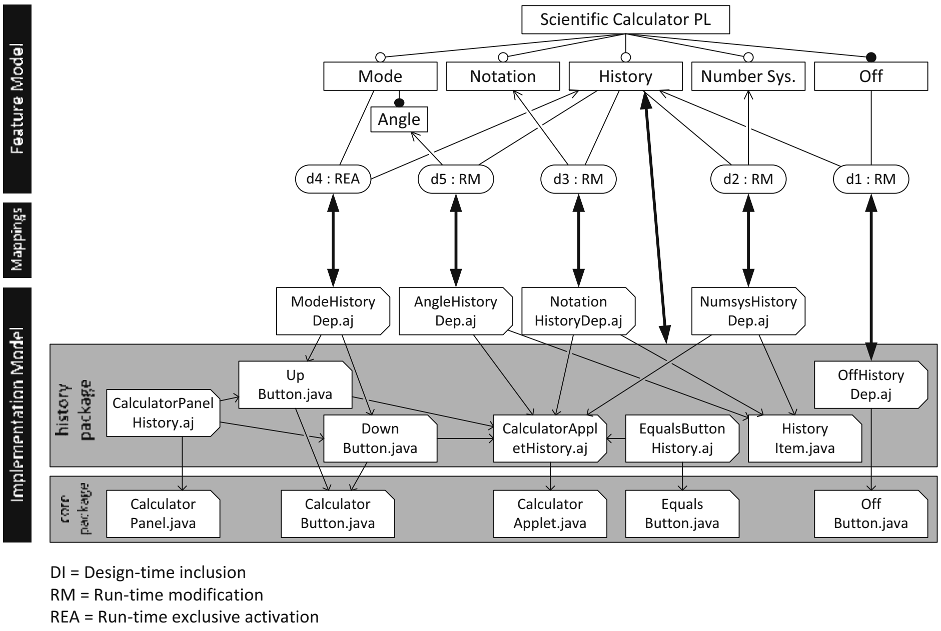

Figure 4: Dependency mappings in a sample product line

The vertical arrows indicate mappings between feature and implementation model. The dependencies *d1* to *d5* are mapped onto corresponding aspectual components. For instance, the aspect `OffHistory.aj` implements the dependency *d1* between the features `Off` and `History`.

In our approach [BLT09], all this information about feature dependencies, their implementation and the corresponding mappings are captured by EMF-based models. Hence, we can apply frameworks such as GMF or openArchitectureWare [oAW] to handle and process these models.

In the long run, such techniques, hopefully, support our goals of making product derivation processes more efficient.

Bibliography

[BLT09] Botterweck, G.; Lee, K.; Thiel, S.: Automating Product Derivation in Software Product Line Engineering, Proceedings of Software Engineering 2009 (SE09), 2-6 March 2009, Kaiserslautern, Germany, 2009.

[VoGr07] Voelter, M.; Groher, I.: Product Line Implementation using Aspect-Oriented and Model-Driven Software Development, 11th International Software Product Line Conference (SPLC 2007), 10-14 September 2007, Kyoto, Japan, 2007.

[StVo06] Stahl, T.; Voelter, M.: Model-driven Software Development : Technology, Engineering, Management, ISBN 978-0470025703, Chichester, England; Hoboken, NJ, USA, John Wiley, 2006.

[PBL05] Pohl, K.; Boeckle, G.; van der Linden, F.: Software Product Line Engineering : Foundations, Principles, and Techniques, ISBN 978-3540243724, New York, NY, Springer, 2005.

[ClNo02] Clements, P.; Northrop, L. M.: Software Product Lines: Practices and Patterns, The SEI series in software engineering, ISBN 978-0201703320, Boston, MA, USA, Addison-Wesley, 2002.

[oAW] openarchitectureware.org - Official Open Architecture Ware Homepage, Web site. http://www.openarchitectureware.org/