

# Towards Engineering Product Lines Using Concerns

Joachim Bayer

Fraunhofer Institute for Experimental Software Engineering (IESE)  
Sauerwiesen 6  
D-67661 Kaiserslautern, Germany  
Joachim.Bayer@iese.fhg.de

**Abstract.** Separation of concerns is accepted as introducing numerous benefits into software development and maintenance. In this position paper, we argue for a method that introduces separation of concerns into product line software engineering. The method covers the complete product line life cycle and integrates the different concerns expressed at the different product line life cycle stages.

## 1 Introduction

Separation of concerns provides the ability to identify, describe, and handle important and critical aspects of a software system separately. Concerns are always related to a goal a stakeholder wants to achieve with a software system or to anticipations or expectations a stakeholder has on a system. A concern can be seen as a perspective that is taken by a stakeholder on a system; only elements that are important for the stakeholder are visible. Some concerns are only relevant in certain development stages, but usually they are relevant during the complete software life cycle.

A clean and explicit separation of concerns reduces the complexity of the description of the individual aspects; thereby it increases the comprehensibility of the complete system. Separation of concerns supports evolution and maintenance, facilitates reuse, it also enables consistency checking and traceability.

The need for separation of concerns has long been recognized in software engineering. The concept was introduced at different software development life cycle stages. In its beginning, separation of concerns was realized at code level and expressed as the need for modularization [12]. Many approaches for separation of concerns have been proposed. Recent examples are Aspect-Oriented Programming [9], Subject-Oriented Programming [6], and Multi-Dimensional Separation of Concerns [17]. Other approaches for separation of concerns include view based requirements engineering (e.g., in [11], [16]), architectural views (the most influencing paper here was [10]), as well as object-oriented analysis and design (Catalysis[5] provides means to compose design models; in OORAM [14] concerns are modeled as roles that can be synthesized).

However, the existing approaches for controlling separation of concerns do not take into account the special situation that the introduction of separation of concerns into a product line engineering method imposes.

In product line development, the need for separation of concerns is stronger than for single systems development. Product line engineering aims at identifying and exploiting commonalities and variabilities within a family of software systems. A system family is a set of software solutions for similar problems in an application area. The identified commonalities and variabilities are captured in the models that are created: the domain model contains the common and variable requirements for the product line, the reference architecture captures commonalities and variabilities of the architectures of the different systems in the product line. An explicit modeling of concerns is also more important, because of the different products that a product line aims for. Variants of models can only be introduced based on the justification that the variant is meaningful and important for a product line member.

Product line engineering increases the number of concerns that have to be taken into account. Each product line member increases the number of customers. The customers of the different product line members are all stakeholders of the product line as a whole. Other sources of concerns arise from the different products that the product line encompasses; it must be possible to build those products from the product line (this must be ensured even after the product line has been changed or evolved). Product line engineering also introduces complexity to separation of concerns. Even though a product line is based on commonalities among its members, the concerns

of the customers and of the product line members can be conflicting. The problem here is that these conflicts can not be resolved without removing products from the product line. Therefore, a method for separation of concerns must be able to handle such conflicts and resulting inconsistencies.

Another important aspect that must be taken into account when introducing separation of concerns to product line development is that product line engineering is a full life cycle approach. It is therefore essential that the concerns that span the complete life cycle can be related to adjacent development steps to be able to reuse information on concerns during development and to trace concerns throughout the life cycle. These goals can be achieved by making concerns explicit and traceable between development stages.

In this paper, an approach is proposed that introduces separation of concerns into product line software engineering. It defines concerns as heterogeneous model sets and provides means to establish relationships within and among these model sets, including relationships between life cycle stages.

## **2 Approach**

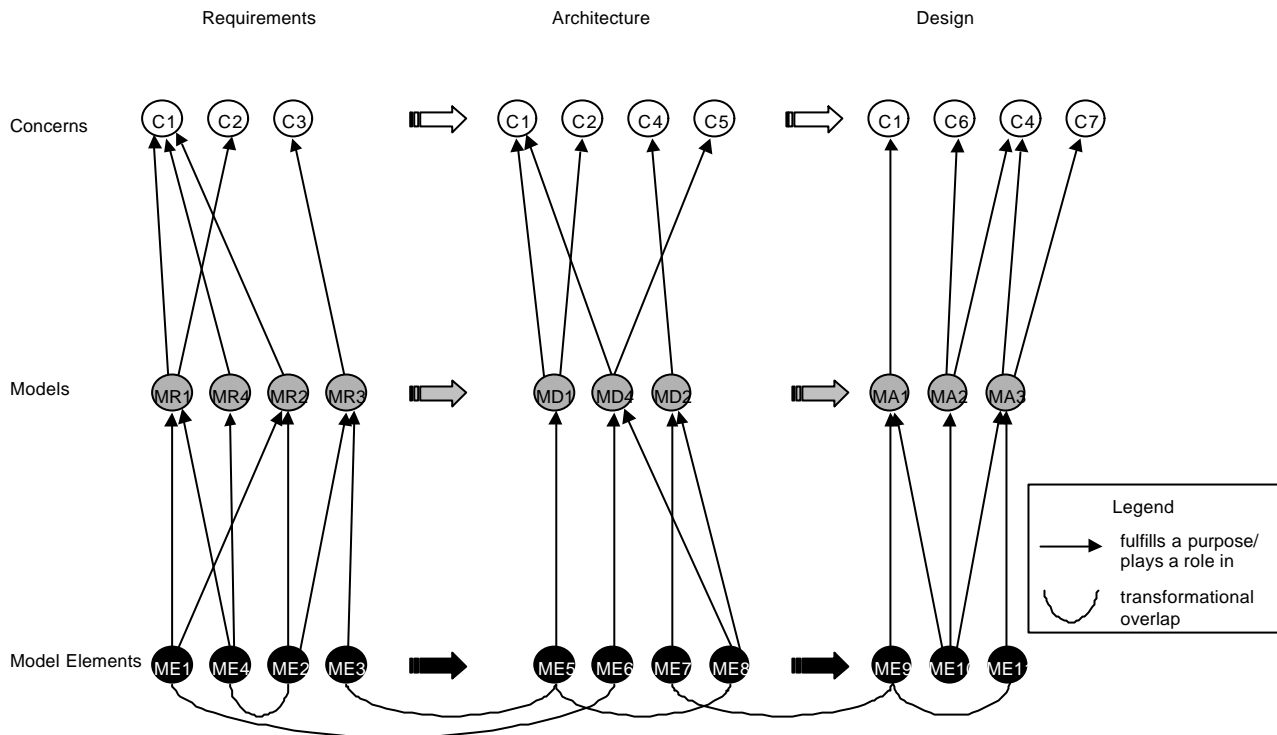
In product line software engineering, a mechanism is needed that allows the explicit definition and description of interrelated and potentially conflicting concerns. This mechanism must be applied to different levels of abstraction in modeling software. Additionally, it must be possible to relate and reuse concern definitions in adjacent development steps. To solve these problems, we develop a method that enables us to introduce the notion of concerns independent of the development stage as an addition to the models already in use at a particular stage. The mechanisms to define and introduce concerns are the same for all stages. This enables the use and evolution of concerns in subsequent phases of product line development.

A concern is a goal, an anticipation, or expectation a stakeholder has on a system. A concern is expressed as a set of requirements associated with the stakeholder. Concerns are persistent over the life cycle of the system, but the requirements they impose in the different life cycle phases are different. A concern is realized in a number of models consisting of a number of model elements that together are able to express how the system fulfills the requirements imposed by the concern.

The principle approach is shown in Figure 1. A heterogeneous model set is a set of models, in which each model describes a software system under consideration from a different perspective. The models in a set together describe the system at a certain life cycle stage completely. The models in different stages can be related to each other; this provides a connection of different life cycle stages. A model's purpose or role in a set is defined by relating stakeholders with their concerns to it. The models together provide views on the system that contain more information than the accumulated information of the individual models. The concerns that are expressed with those models together describe all concerns all stakeholders have on the system. Model sets can be structured hierarchically; the important point is that each model set has a clearly defined purpose in the hierarchy. One consequence of that approach to modeling concerns is that a concern can be cross cutting (i.e., concerns that can not be completely encapsulated) or non cross cutting depending of the model set used to represent it. In other words, the approach is independent of the used paradigm.

Factors that influence the applicability of model sets in a given project include the nature of the application domain, the organizational context, and the structure and aims of the project. Therefore, the individual models and the model set have to be customized to the specific context, in which they will be used. Our approach for separation and composition of concerns is to introduce a method that defines how to encapsulate a concern as a heterogeneous model set. It is devised to support product line engineering. It can be used in all phases of the product line development. Additionally, the concerns at different phases can be related to each other.

To introduce the method to a given project, first the different stages in the software development are investigated for the models that are currently used. The second step is to elicit the relevant concerns. As third step, the models and the concerns are related to each other. A concern is described by a set of models and the purpose and stakeholders related to that concern. A model set itself is defined by defining the relationships of its constituting models based on overlap. Two models are said to overlap, if they designate common aspects. This can be a model element that appears in several models (e.g., methods appear in class diagrams and in collaboration diagrams).



**Figure 1.** Relationships between model elements, models, and concerns

More complex relations among models can be expressed in transformations (e.g., methods are derived from use cases). The description of the overlap is necessary to capture the semantics and overall meaning of the model set and to compose the separated concerns. The model sets depend on the domain and the life cycle phase. Therefore, the definition of overlap must be customized. The overlap definition is also the basis to establish traceability and to define consistency checks. Each goal or concern a stakeholder has on a model set must explicitly be stated. This is broken down to the models that serve a certain purpose within the set. Concerns can be related to each other, too. This is done by hierarchically composing model sets. Using the same mechanism that is used to relate the models and concerns in one life cycle stage, different stages can be related.

### 3 Future work

The method will be applied to two product line development approaches: PuLSE and KobrA.

PuLSE (Product Line Software Engineering) [2] is a customizable product line engineering approach. The life cycle of a product line in PuLSE is split in the phases: initialization, product line infrastructure construction, usage, and evolution. In the initialization phase, PuLSE is customized to the respective context, that is, customized versions of construction, usage, and evolution are created. The customization includes the definition of model sets for the domain analysis [3] and the reference architecture creation [4] in the construction phase. The model sets are used for application engineering in the usage phase and are the basis for the evolution of the product line. PuLSE has been customized for different domains. We are retrospectively eliciting the model sets and the domain-specific concerns. We are applying the method to the domain of financial applications, for which PuLSE is currently customized.

The KobrA method [1] is a component-based product line method. The model set used in KobrA is based on the UML. In addition to the customized UML models, there are a number of models used that stem from the domain of enterprise resource planning or have been added to close gaps in the UML (e.g., operation schemata as used in the Fusion [5] method). The method described in this paper will be applied to KobrA in two case studies.

## REFERENCES

- [1] C. Atkinson, J. Bayer, and D. Muthig. Component-Based Product Line Development: The Kobra Approach. *Proceedings of the First Software Product Line Conference (SPLC-1)*, Denver, Colorado, August 2000.
- [2] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud. PuLSE: A methodology to develop software product lines. In *Proceedings of the Symposium on Software Reuse (SSR'99)*, May 1999.
- [3] J. Bayer, D. Muthig, and T. Widen. Customizable Domain Analysis. In *Proceedings of the International Symposium on Generative and Component-Based Software Engineering*, September 1999.
- [4] J. Bayer, O. Flege, and C. Gacek. Creating Product Line Architectures. To appear in the *Proceedings of the International Workshop on Software Architectures for Product Families. (IW-SAPF-3)*.
- [5] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes, and P. Jeremaes. *Object Oriented Development: The Fusion Method*. Prentice-Hall, 1994.
- [6] D. F. D'Souza and A. C. Wills. *Objects, Components, and Frameworks with UML: The Catalysis Approach*. Addison-Wesley, 1999.
- [7] W. Harrison and H. Tarr. Subject-Oriented Programming (A Critique of Pure Objects). In *Proceedings of the 8th Conference on Object-Oriented Programming: Systems, Languages, and Applications (OOPSLA'93)*, pp. 411-428, Washington, D.C., September 1993.
- [8] V. Issarny, T. Saridakis, and A. Zarras. Multi-View Descriptions of Software Architectures. In *Proceedings of the Third International Software Architecture Workshop*, Orlando, Florida, November 1998.
- [9] G. Kiczales, J. Lamping, A. Mendhekar, C. Medea, C. Lopes, J.-M. Loingtier, and J. Irving. Aspect-oriented Programming. In *Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, pp. 220-242, Jyväskylä, Finland, June 1997.
- [10] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, November 1995, pp. 42-50.
- [11] J.C.S.P. Leite and P.A. Freeman. Requirements Validation through Viewpoint Resolution. *IEEE Transactions on Software Engineering*, 17(12):1253-1269, 1991.
- [12] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053-1058, December, 1972.
- [13] Ran. Architectural Structures and Views. In *Proceedings of the Third International Software Architecture Workshop*, Orlando, Florida, November 1998.
- [14] T. Reenskaug, P. Wold, and O. A. Lehne. *Working with Objects: The OOram Software Engineering Method*. Manning Publications Co., 1995.
- [15] D. Soni, R. L. Nord, and C. Hofmeister. Software Architectures in Industrial Applications. In *Proceedings of the 17th International Conference on Software Engineering*, IEEE Computer Society Press, 1995.
- [16] G. Spanoudakis, A. Finkelstein, and D. Till. Overlaps in Requirements Engineering. *Automated Software Engineering*, 6(2):171-198, April 1999.
- [17] P. Tarr, H. Ossher, W. Harrison, S. Sutton. N Degrees of Separation: Multi-Dimensional Separation of Concerns. In *Proceedings of the 21st International Conference on Software Engineering*, Los Angeles, CA, May 1999.