Towards a More Precise Definition of Feature Models

Matthias Riebisch

Technical University Ilmenau, Germany
matthias.riebisch@tu-ilmenau.de

Abstract. Feature models are a well accepted means for expressing requirements in a domain on an abstract level. They are applied to describe variable and common properties of products in a product line, and to derive and validate configurations of software systems. Their industrial importance is increasing rapidly. However, methodical usage and tool support demands for a more precise definition of features, their properties and their relations within a feature model. This position paper summarizes the state of the discussion and proposes issues for future development. Categories of features and types of their attributes and relations are presented. The represented information is limited to a customer point of view onto the feature models without excluding technically detailed features. Connections of features to other models i.e. design, and to implementation elements are given by traceability links. Approaches for graphical representations and data models for feature models are shown. Proposals of attaching additional information for related tasks like product line evolution, scoping, effort estimation, definition of product configurations and documenting are discussed.

Introduction

The idea of developing software once and using it for a variety of different requirements has been a driving force of software engineering methods for a long time. Software product line methods strive for prefabricating software for multiple applications by providing a reusable platform with variable extensions, i.e. by adaptable parts or by configurable components. However, the future requirements of such applications are unknown, therefore the risk of developing inappropriate software is high.

One way to reduce this risk is provided by Domain Analysis. Concepts, properties and solutions of a domain are analyzed. Based on this information decisions about software development for future applications within such a domain are made. As part of Domain Analysis methods, feature models are used for describing common and different requirements for software systems as instances of a product line.

Feature models describe properties distinguishing between common and variable requirements. They structure requirements by generalizing them by concepts. They provide a very flexible means of description. Meanwhile, they are applied in some industrial projects for describing software for multiple use, like component-based systems, reusable libraries, e.g.

While applying feature models in large industrial projects the need for more methodical and tool support became obvious. There is a deficiency of clear definitions of the feature model elements, their syntax and semantics. As a consequence, ambiguities and inconsistencies occurred. Exploitation of the feature models by tools are impossible unless more formal descriptions are introduced.

In this position paper a more strict definition of feature models is proposed. Its intention is to inspire the discussions at the ECOOP2003 workshop, and to receive feedback for further development. The main goal is to establish a new, more precise but well accepted definition.

Feature Model Basics

As introduced by the FODA method [Kang et al.1990] and by [Czarnecki et al. 2000], a feature model represents a hierarchy of properties of domain concepts. The properties are used to discriminate between concept instances, i.e. systems or applications within that domain. The properties are relevant to end users. At the root of the hierarchy there is a so-called concept feature, representing a whole class of solutions. Below of this concept feature there are hierarchically structured sub-features showing refined properties. Each of the features is common to all instances unless marked as being optional, thus not necessarily being part of all instances. Fig. 1 shows an example for an ATM product line with a feature `ATM` as concept feature. The feature `debit card reader` is a so-called mandatory feature, stating that it is common to all instances of the domain, because every ATM has a reader for debit cards. The feature `receipt printer` is marked as optional by an empty bullet, because there are ATMs in this product line example without a printing device.

Such a feature model represents an abstract view onto properties of all instances of a domain. Every feature covers a set of requirements. By selecting a set of optional features an instance of that domain can be defined. All mandatory features are part of the instance by definition.

Feature models are used for development and application of software product lines, i.e. for defining products and configurations, for describing possibilities of a product line, and for establishing new products and adding new properties to a product line. Several methods for product line development make use of feature models, i.e. FeatuRSEB [Griss et al. 1998], Generative Programming [Czarnecki et al. 2000] or our methodology ALEXANDRIA [Riebisch et al. 2002].

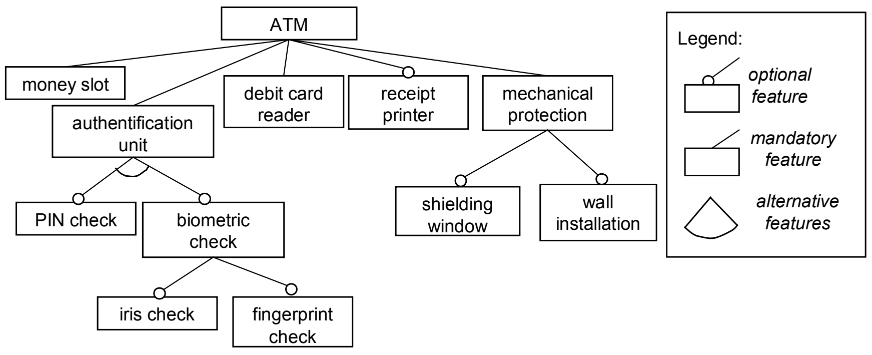

Fig. 1. Feature Model Example

Grouping variable Features

For controlling the selection of optional features FODA introduces alternatives as a relation between two or more features neighboring in the hierarchy. In Fig. 1 the features `PIN check` and `biometric check` are alternative ones; either a `PIN check` or a `biometric check` is provided by a product of this product line.

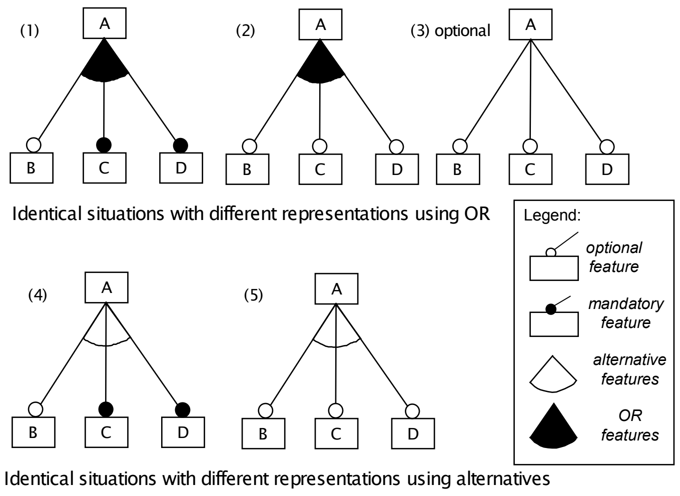
(1)
A
B
C
D
(2)
A
B
C
D
(3) optional
A
B
C
D
Identical situations with different representations using OR
Legend:
optional feature
mandatory feature
alternative features
OR features
(4)
A
B
C
D
(5)
A
B
C
D
Identical situations with different representations using alternatives

Fig. 2. Examples for ambiguities with OR and alternatives

FeatuRSEB extends this by distinguishing between OR and XOR alternatives, where XOR shows mutual exclusion and OR enables more than one feature.

Generative Programming combines OR, XOR and alternatives with designating the member features as mandatory or optional. Fig. 2 shows examples using the graphical representations of [Czarnecki et al. 2000] with filled bullets for mandatory features. Unfortunately, these combinations of mandatory and optional features with alternatives, OR and XOR relations could lead to ambiguities.

To prevent these ambiguities and to enable a more expressive and powerful notation for relations between neighboring features, multiplicities have been introduced by [Riebisch et al. 2002]. These multiplicities are similar to those of Entity Relationship Models ERM and of the Unified Modeling Language UML. In addition to joining some features to a group, all of them are designated as optional to express the possibility of choice.

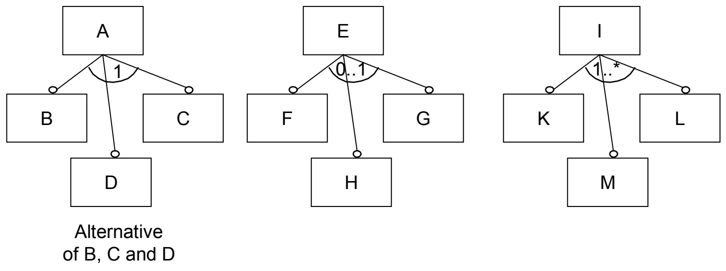

Fig. 3. Grouping neighboring features using multiplicities

Different variants of multiplicities of the groups are possible:

0..1	at most one feature has to be chosen from the set of the sub-features,
1	exactly one feature has to be chosen from the set,
0..*	an arbitrary number of features (or none at all) have to be chosen from the set,
1..*	at least one feature has to be chosen from the set,

Certainly, this list of possible multiplicities in feature diagrams covers the most common cases. In practice, however, we often encounter situations in which a set of features has a multiplicity like “0..3”, “1..3”, or simply “3”. Such multiplicities cannot be expressed using the previous notations.

Feature Categories and Views

When establishing feature models questions about structuring useful characteristics of features arise. A definition of features should answer such questions. According to FODA, a feature is “a prominent or distinctive and user-visible aspect, quality, or characteristic of a software system or systems” [Kang et al.1990]. Features are categorized into *Capabilities* (functional, operational, presentation features), *Operating Environments*, and *Domain Technology Implementation Techniques*.

Czarnecki and Eisenecker extend this to a more general definition. In [Czarnecki et al. 2000] a feature is a "property of a domain concept, which is relevant to some domain stakeholder and is used to discriminate between concept instances". A stakeholder could be any person important for a product line, not only the end-user.

The method FORM [Kang et al. 1998] and its successor FOPLE [Kang et al. 2002] introduce four different views with features classified to the according types:

1. Capability features:
   Service, Operation, Nonfunctional characteristics.
2. Domain technology features:
   Domain method, Standard, Law.
3. Operating environment features:
   Hardware, Software.
4. Implementation technique features:
   Design decision, Communication, ADT.

Unfortunately there is no clear definition for making a distinction between these views. User-visible properties should be covered by the Capabilities view, however some of the properties of this category like graphical user interface components could even be assigned to the Operating Environment view. A service could be designed according to an existing act and therefore assigned to the Domain Technology or to the Capability view. Furthermore, the aims and advantages of these views on FORM and FOPLE are not clear.

Goals and Application

Views and feature categories should correspond to the intended purpose of the feature models. In our opinion, feature models should fill the gap between requirements and the solution. They provide an extra model between requirements specifications (i.e. structured text with glossaries, concept graphs, use case models, decision models etc.) and design models and architectures (UML models, ERM models). In our opinion it is not desirable to extend feature models, with the goal to replace some parts of these well-established descriptions for requirements and design.

According to our experiences from industrial application of feature models, they can successfully support some product line development activities of two groups of stakeholders. The first group, handling software more as a black box, consists of customers, sellers, product managers and company managers. For them, a feature model

- provides an overview over requirements
- distinguishes between common and variable properties
- shows dependencies between features
- enables feature selection for defining new products

- supports decisions about evolution of a product line

The second group of stakeholders is working on the development of a product line, i.e. architects, software developers for reusable components of the product line and developers for single products. They are supported by feature models during

- defining reusable components and separating them according to the Separation of Concerns principle
- assigning reusable components to variable features
- describing dependencies and constraints between components and features
- controlling the configuration of products out of the reusable components

By linking features to elements of design and implementation, additional information about details of the solution domain are provided. By linking features to requirements, detailed information from the problem domain is reachable. These links are built using traceability links [Sametinger Riebisch 2002]. Fig. 4 shows the linkage by an ERM diagram.

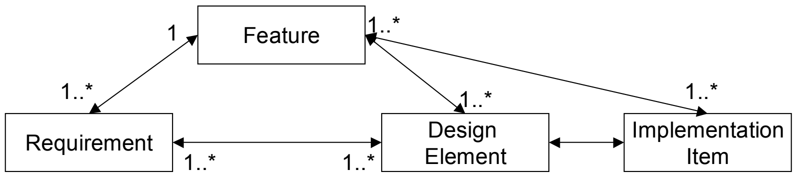

Fig. 4. References between features, requirements, design and implementation

Based on these goals and on the linkage to other models, a customer view is sufficient for the feature model. All other information should be captured in the available descriptions, i.e. requirements specification, design models and implementation documents. A feature model provides an overview over the requirements, and it models the variability of a product line. It is used for the derivation of the costumer's desired product and provides a hierarchical structure of features according to the decisions associated to them. The proposed definition reflects this conclusion.

Proposed Definition - Part 1

A feature represents an aspect valuable to the customer. It is represented by a single term. There are three categories of features:

- *Functional features* express the behavior or the way users may interact with a product.
- *Interface features* express the product's conformance to a standard or a subsystem
- *Parameter features* express enumerable, listable environmental or non-functional properties.

A feature model gives a hierarchical structure to the features. In addition to the mentioned categories, within the hierarchy there could be abstract features to encapsulate *Concept features*. The root of the hierarchy always represents a concept Feature.

According to this definition, not only features describing capabilities of a system are possible. Even very technical concepts - i.e. "common rail fuel injection" for a car engine - can occur as features, if there is the chance that customers will use these concepts for distinguishing between products. In our experience, the decision about including a concept as a feature becomes very clear by asking if a customer is willing to pay for it.

The four categories were defined aiming at a small number of categories and at an unambiguous distinction between them. *Functional features* describe both static and dynamic aspects of functionality. They cover i.e. use cases, scenarios and structure. To give some examples for the automotive domain, features like `Electric seat heating` and `Extra daytrip mileage counter` belong to that category.

Interface features describe connectivity and conformance aspects as well as contained components. From the customers point of view the possibility of connecting a product to other devices and of extending it are valuable categories. Examples for features from this category are `Firewire connection` for an electronic camera and `DDR133 RAM` for memory sockets of a PC. Conformity to standards and certificates are in this category as well, i.e. `USB 2.0 compatible` and `ISO 9000 certified` for a PC. Complete components or subsystems of special quality or by special vendors were added to the same category, because the handling of such features is very similar to interfaces. An example is the feature `Bosch ABS device` for a car, if this is valuable for a customer.

Parameter features cover all features with properties demanding for quantification by values or for assignment to qualities. Examples from the automotive domain are `fuel consumption`, `top acceleration` or `wheel size`.

Concept features represent an additional category for structuring a feature model. Features of this category have no concrete implementation, but their sub-features provide one. The feature `mechanical protection` in fig. 1 represents an example for such a feature.

Relations in a Feature Model

Within a feature model the features are structured by relations. Common to all methods mentioned above are hierarchical relations between a feature and its sub-features. They control the inclusion of features to instances. If an optional feature is selected for an instance, then all mandatory sub-features have to be included as well, and optional sub-features can be included. Additionally, FODA introduces so-called composition rules using "`requires`" and "`mutex-with`". These rules control the selection of variable features in addition to the hierarchical relations. If a feature A is selected for an instance, and there is a relation "`A requires B`" then feature B has to be selected as well. Opposite to this, if a feature A is selected for an instance, and there is a relation "`A mutex-with B`" then feature B has to be unselected. In Generative Programming the latter relation is called "`excludes`" instead of "`mutex-with`".

When applying these relations in practical projects, some deficiencies become visible. When building a hierarchy of features it is not clear how to arrange the features. Frequently it was not obvious whether to express a particular relation between two features by assigning one as a sub-feature of the other or by establishing a "`requires`" relation between them. Semantically, there are only small differences between a feature – sub-feature relation in a hierarchy and a "`requires`" relation. The same is to be observed with alternatives and "`mutex-with`" (see fig. 5). There is only little support for decisions between these possibilities for a relation. The description of more complex dependencies, with more than two participating features or more conditions is impossible.

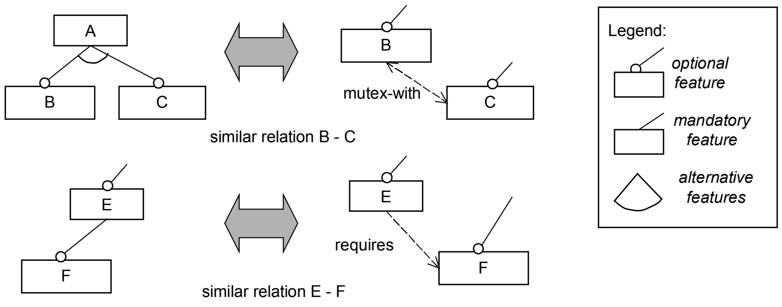

Fig. 5. Similar relations in FODA models

Proposed Definition - Part 2

The features are structured by relations of the following categories:

- *Feature - sub-feature - relations* construct a hierarchy, designed to guide the customer's selection process. The position of a feature within the hierarchy shows its influence on the product line architecture. A hierarchy relation could carry the semantics either of the *requires* relationship or of the *refinement* one. The hierarchy relations distinguish between *mandatory* and *optional* features.
- Constraints between features are expressed either by *multiplicity-grouping relations* for features with the same parent or by *requires* or *excludes* relations for arbitrary features.
- *Refinement relations* lead towards more detailed sub features. They express is-a or part-of semantics.
- Suggestions for additional selections can be expressed by the *hint relation.*

Requires, excludes, refinement and *hint relations* bridge arbitrary features within the hierarchy.

While analyzing feature models of practitioners a triple use of the hierarchical relation was discovered. First, the hierarchical relation is used for refinement, second for decomposition, and third as a "requires" relation as a construction rule.

The hierarchy of features should be established according to the most important usage of the feature model. In our opinion, the selection of features for defining new product line instances represents the purpose of highest importance. Therefore, the hierarchy is built by the composition rules – requires e.g. – at first order. other relations between features like refinement are regarded as less important. In this way the features with more influence on other decisions are arranged nearer to the top of the hierarchy. In the case of a conflict, other relations like refinement have to be expressed as external dependencies and composition rules like requires relations are forming the hierarchy.

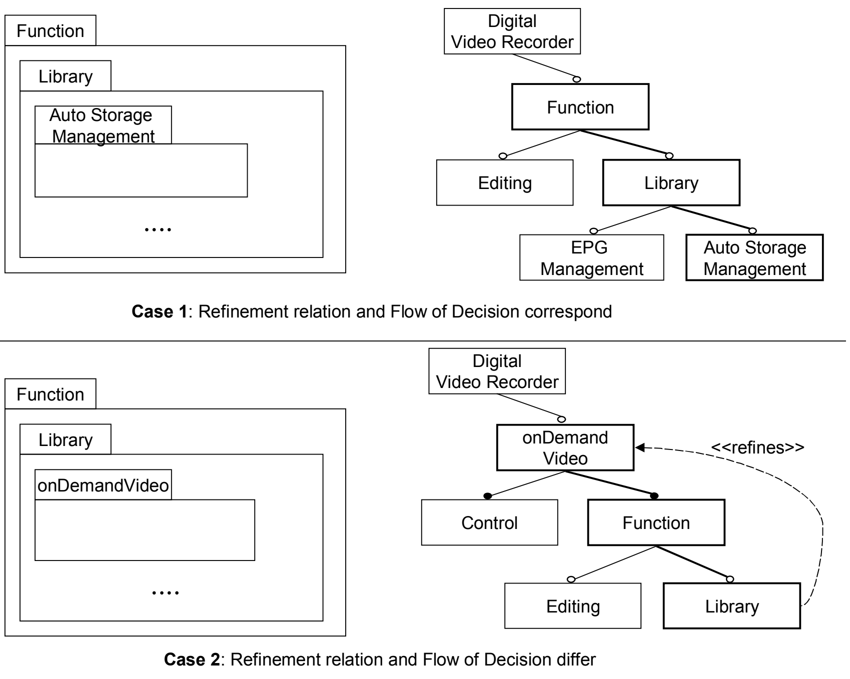

Fig. 6. Corresponding and conflicting relations in a feature model example

Fig. 6 shows two cases in an example with a package structure showing the refinement on the left and the corresponding feature model on the right. In case 1 there is no difference between refinement and requires relation. The latter relation is expressing the sequence of decisions in defining product instances. In this case, the hierarchy relations express both, refinement and sequence of decisions. In case 2 there is a conflict, because the decision about whether a Digital Video System should work as an On Demand video server for clients influences many other decisions, therefore this feature has to be chosen first. However, looking at the refinement structure, the feature `OnDemand video` is a sub-feature of the `Library` feature. There is a conflict between the refinement and the requires relation. In such a case, the hierarchy is arranged according to the requires relation showing the sequence of decisions, and the refinement is expressed by an external relation.

Related Work

As already mentioned, feature models have been introduced as part of the method Feature Oriented Domain Analysis FODA [Kang et al.1990]. It was extended by Generative Programming [Czarnecki et al. 2000]. Slightly different notations are introduced by Bosch as part of his Software Product Line Methodology [Bosch 2000]. The product line method FeatuRSEB [Griss et al. 1998] uses the FODA notation as well. This notation was extended by feature categories and views by the methods FORM [Kang et al. 1998] and FOPLE [Kang et al. 2002]. As discussed above, these extensions lack of clear definitions, thus leading to ambiguous models.

There are some different proposals for graphical representation of feature models [Robak 2003]. Most of them are similar to the FODA notation as shown in Fig. 1. Another approach is using UML class diagram symbols for features and their relations [Clauß 2001]. However, this approach leads to communication problems for software developers using UML in their work. Currently, there is no graphical notation for feature models established as a standard.

Features are mentioned by UML as well. However, the notion is different to the feature models discussed here, because a feature in UML is a property similar to an operation or an attribute, and it is encapsulated, i.e. in an interface or a class [UML 2001].

Frequently, some features interfere with each other leading to a so-called feature interaction [Zave 1999]. In such a case a feature influences the properties or the behavior of another feature in some way, if both features are present in one instance of a product line as a combination. There are different approaches of solving such interactions. These issues were subject of workshops i.e. [FICS 2001]. An overview and a bibliography is provided by [Calder et al. 2003].

Development processes support methods and model notations, and vice versa. For establishing and utilizing feature models there are some descriptions by product line development methods, i.e. by Bosch [Bosch 2000], [Atkinson et al. 2002] and our own methodology ALEXANDRIA [Riebisch Böllert 2003]. Processes for applying feature models in product line development are issues of European ITEA projects ESAPS [ESAPS] and CAFÉ [CAFE], however most of their published process descriptions are fairly general.

Ongoing and Future Work

Tool support in using feature models requires notations based on a formal defined syntax and semantics. Currently, we are developing a language for expressing feature dependencies and constraints [Streitferdt et al. 2003]. For product line configuration purposes, the Object Constraint Language OCL is adapted to express valid configurations of features [Streitferdt 2003].

Most benefits of using feature models for product line development can be obtained by linking features to requirements and solution elements, i.e. architectural and design models and source code. For integrating feature models with CASE tool repositories XML provides a technology. Current works are implementing an XML notation and tool integration for software product lines [Streitferdt 2003].

Feature models can be applied with great success for improving program comprehension. In most reengineering activities source code is the only reliable source of information. Documents of architecture, design and even requirements are mostly outdated or incomplete. In reconstructing this information feature models help to bridge the gap between the very concrete code and the fairly abstract information of the documents mentioned. They are applied in methods for supporting reverse engineering in a hypothesis - verification procedure [Pashov Riebisch 2003].

Conclusion

In this position paper a definition of features and their relations in feature models is proposed. This definition aims at avoiding ambiguities in establishing and exploiting feature models. Categories of features and views in feature models are limited to a customer point of view. Alternatives, OR and XOR groups of features are replaced by multiplicities similar to those of UML and ERM in order to enable more powerful and less ambiguous models. The characteristics of features within a hierarchy and external to it are analyzed. Categories of feature relations are defined. The selection of features for new product line instances was determined as the most important usage of feature models. Therefore, construction rules like requires and excludes relations are favored in the hierarchy in comparison to refinement relations.

The proposed definition is part of the product line development methodology ALEXANDRIA [Alexandria]. It is applied as base of exploiting feature models for evolving product lines, for developing instances of them and for reverse engineering existing components and legacy systems.

Acknowledgements

The definition of features and feature models was developed as part of the product line methodology ALEXANDRIA. The definition was discussed and elaborated as a joint work together with Kai Böllert, Detlef Streitferdt, Ilian Pashov, Perklis Sochos and Michael Hübner. I would like to thank Detlef Streitferdt and Michael Hübner for their hints on an earlier version of this paper.
This research was part of a project funded by the department of Process Informatics chaired by Prof. Ilka Philippow. The definition was approved in co-operation projects for software product lines together with Siemens Dematic AG Constance, and Siemens AG, Corporate Research Munich, Germany.

References

[Alexandria] Software product line development methodology ALEXANDRIA. Project website at http://www.theoinf.tu-ilmenau.de/~pld/

[Atkinson et al. 2002] Atkinson, C., et al.: Component-based product line engineering with UML. Addison Wesley, 2002.

[Bosch 2000] Bosch, J.: Design and use of software architectures – Adopting and evolving a product-line approach. Addison Wesley, 2000.

[CAFE] From Concepts to Application in System-Family Engineering (CAFÉ). Project ITEA 00004 of the Eureka Sigma!2023 Program. Project Homepage http://www.esi.es/cafe/

[Calder et al. 2003] Calder, M.; Kolberg, M.; Magill, M.H.; Reiff-Marganiec, S.: Feature Interaction – A Critical Review and Considered Forecast. Elsevier: Computer Networks, Volume 41/1, 2003. S. 115-141

[Clauß 2001] Clauß, M.: A proposal for uniform abstract modeling of feature interactions in UML. Proceedings FICSworkshop, 15th European Conference on Object-Oriented Programming (ECOOP'01), April 2001. S.. 21-25. Online available at http://www.info.uni-karlsruhe.de/~pulvermu/workshops/ecoop2001/proceedings/FICS2001.pdf

[Czarnecki et al. 2000] Czarnecki, K., Eisenecker, U.W.: Generative Programming. Addison Wesley, 2000.

[ESAPS] Engineering Software Architectures, Processes and Platforms for System Families (ESAPS). ITEA project 99005 of the Eureka Sigma!2023 Program. Project Website http://www.esi.es/esaps/

[FICS 2001] Feature Interaction in Composed Systems. ECOOP 2001 Workshop. Material available online at http://www.info.uni-karlsruhe.de/~pulvermu/workshops/ecoop2001/

[Griss et al. 1998] Griss, M.; Favaro, J.; d'Allesandro, M.: Integrating Feature Modeling with RSEB. Hewlett-Packard Comp., 1998.

[Kang et al.1990] Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A., Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-021, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, 1990.

[Kang et al. 1998] Kang, K., Kim, S., Lee, J., Kim, K., Shin E. and Huh, M.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures, Annals of Software Engineering, 5, 1998, pp. 143-168.

[Kang et al. 2002] Kang, K.C.; Lee, K.; Lee, J.: FOPLE - Feature Oriented Product Line Software Engineering: Principles and Guidelines. Pohang University of Science and Technology, 2002

[Pashov Riebisch 2003] Pashov, I., Riebisch, M.: Using Feature Modeling for Program Comprehension and Software Architecture Recovery. In: Amendment to Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03), Huntsville Alabama, USA, April 7-11, 2003. IEEE Computer Society, 2003.

[Riebisch et al. 2002] Riebisch, M.; Böllert, K.; Streitferdt, D., Philippow, I.: Extending Feature Diagrams with UML Multiplicities. 6th World Conference on Integrated Design & Process Technology (IDPT2002), Pasadena, CA, USA; June 23 - 27, 2002.

[Riebisch Böllert 2003] Riebisch, M.; Böllert, K.: Feature-driven Composition of Software Systems Using UML. Submitted for publication, 2003

[Robak 2003] Robak, S.: Feature Modeling Notations for System Families. ICSE'03 Workshop on Software Variability Management. Portland, Oregon 2003, pp. 58-62

[Sametinger Riebisch 2002] Sametinger, J.; Riebisch, M.: Evolution Support by Homogeneously Documenting Patterns, Aspects and Traces. 6th European Conference on Software Maintenance and Reengineering. Budapest, Hungary, March 11-13, 2002 (CSMR 2002) . Computer Society Press, 2002. S. 134-140.

[Streitferdt et al. 2003] Streitferdt, D., Riebisch, M., Philippow, I.: Formal Details of Relations in Feature Models. In: Proceedings 10th IEEE Symposium and Workshops on Engineering of Computer-Based Systems (ECBS'03), Huntsville Alabama, USA, April 7-11, 2003. IEEE Computer Society Press, 2003. S. 297-304

[Streitferdt 2003] Streitferdt, D.: Family Oriented Requirements Engineering. PhD thesis (to be submitted), TU Ilmenau, Germany, 2003. (in German)

[UML 2001] Object Management Group: Unified Modeling Language Specification*,* Version 1.4. http://www.omg.org, 2001

[Zave 1999] Zave, P.: FAQ Sheet on Feature Interaction. AT&T, 1999. http://www.research.att.com/~pamela/faq.html