# Manifoldness of Variability Modeling — Considering the Potential for Further Integration

Mark-Oliver Reiser[1,2], Ramin Tavakoli Kolagari[2], and Matthias Weber[3]

[1] DaimlerChrysler AG, Research & Technology, GR/ESM,
Alt-Moabit 96a, D-10559 Berlin (moreiser@cs.tu-berlin.de)
[2] Technische Universität Berlin, Fakultät IV, Lehrstuhl Softwaretechnik,
Franklinstraße 28/29, D-10587 Berlin, Germany (tavakoli@cs.tu-berlin.de)
[3] Carmeq GmbH, Carnotstrae 6, D-10587 Berlin, Germany
(matthias.weber@carmeq.com)

**Abstract.** Variability management has become an important concern in software and systems engineering. Especially in industrial settings a rigid management of variability has been identified as an important prerequisite for further optimization of the development process, e.g. for reuse of software sub-systems across vehicle models such as the Mercedes Benz A-Class and C-Class. In response to this growing practical interest, the scientific community has come up with numerous concepts and techniques for modeling variability. However, despite initial attempts to integrate or unify some of these manifold approaches, a clear understanding of how they precisely relate to each other is still not yet achieved.
In the paper, various techniques for variability modeling are elaborated and a basic classification scheme is proposed. From this we derive their common capabilities, which arguably embody the essence of variability modeling in general. On this basis, a discussion is presented that concerns the potential and feasibility of integrating all these diverse techniques into a single, common technique for variability modeling.

Key words: Software product lines, Variability management.

## 1 Introduction

Over the past decade product line engineering became a popular approach to software development both in classical software engineering domains as well as for industrial software-intensive systems. A *software product line* is a set of software products that share a certain degree of commonality while still showing substantial differences and that are developed from a common set of core assets in a prescribed way [1]. In other words, whenever a company has several similar software products on offer it makes sense to consider developing only a single, but variable product instead of developing the products in parallel and independently from one another, thus shifting the focus of development from the individual products to the product line. Key to all product line engineering

PREPRINT OF THE PROCEEDINGS OF CEE-SET 2007, www.cee-set.org, pp. 283 - 295, 2007.

is *variability management*, i.e. the documentation and management of the commonality and variability between the products within the scope of the product line.

According to the paradigm of orthogonal variability modeling [2], the variability between the products in a product line is documented and managed as a separate, orthogonal aspect of development, called *variability dimension*. This dimension is thus clearly set apart from the *artifact dimension*, i.e. the definition of the development artifacts, such as requirements, component diagrams, state charts and test cases.

However, variability is often not only addressed in the variability dimension alone. Instead, it is common to describe the variability's precise impact on the development artifacts within these artifacts themselves, i.e. it is explicitly defined at what location in an artifact certain variability shows up and what alternative forms the artifact can take at that location. The fact that in these cases some aspects of variability are also defined in the artifact dimension need not necessarily be seen as a violation of the orthogonal variability modeling paradigm (even though it is sometimes seen as such), because the definition of variability aspects in the artifact dimension only relates to where and how the artifact is affected by variability. The primary focus of variability management—i.e. the presentation of an overview of the entire product line's variability, definition of dependencies between variations and the global coordination of variability across several artifacts—is still, mainly, the variability dimension.

Over the past decade, a multitude of different techniques have been proposed for both the variability dimension (esp. feature modeling [3–7]; decision tables [8, 9]; decision diagrams/trees [10, 11]) and for defining variability in the artifact dimension (esp. approaches for explicitly defining variation points and their variants in various types of artifacts). Most of these techniques come in a variety of flavors; an attempt to unify some of them has already been undertaken or is currently in progress, e.g. for feature modeling [12, 13]. When considering all these methods, a few basic groups of techniques and thus a few fundamental approaches towards variability modeling can be identified, for example feature modeling and decision tables. Unfortunately, how these main approaches relate to each other is not examined in detail and is not well understood. Are they merely different forms of presenting the same information or are there fundamental differences in how they address variability modeling? Since all these techniques are aimed at variability modeling, this situation is unsatisfactory from a theoretical and conceptual point of view: when proposing different ways to treat variability, it should be clear how they differ and why the distinction is necessary. Moreover, there is also a practical problem with this splitting up of basic approaches: When two or more independent product lines need to be related to each other or integrated into a single higher-level product line, different variability modeling approaches are usually applied in the individual product lines. In this case, it must be clear how these approaches relate to each other. Such product line integration is of particular importance in industrial settings; for ex-

ample, in the automotive industry car manufacturers usually need to integrate the products, i.e. sub-systems, from numerous suppliers' product lines.

In the remainder of this article, we discuss what basic groups of variability modeling techniques can be identified and how they relate to each other. This is done for the variability dimension in Section 2 and for variability in the artifact dimension in Section 3. Then, in Section 4, we discuss the potential and benefit of a further integration of these techniques.

## 2 Variability dimension

Roughly, three forms of variability modeling in the variability dimension can be distinguished: *decision tables*, *decision trees/graphs* and *feature models*. Figure 1 shows an excerpt from a decision table, inspired by an example in [8]. A de-

| Name | Description | Constraints | Resolution | Effect |
|---|---|---|---|---|
| T9 | Does the phone have T9 support ? | | yes<br><br><br>no | step 6 of use case 'send message' is obligatory; extension 6a of use case 'send message' is obligatory<br>step 6 of use case 'send message' is removed |
| Cam | Does the phone have a camera ? | | yes<br>no | …<br>… |
| CamRes | What is the camera's resolution ? | Cam==yes | 1MegPix<br>2MegPix | – |
| | … | | | |

**Fig. 1.** Excerpt from a sample decision table (cf. [8]).

cision table usually refers to one or more variable development artifacts, in the example a use case diagram for the scenario 'send message' (not shown). Each line in a decision table represents a decision to be taken in order to configure the corresponding variable artifact(s) of the table. Each such decision has a name as its unique identifier, a question that formulates the decision to be taken, a list of possible resolutions, i.e. possible answers to the question, and one effect or action per resolution that describes how the corresponding variable artifacts have to be changed in order to configure them in line with the decision taken. Constraints allow defining interdependencies between decisions in order to restrict the available resolutions depending on decisions taken earlier or to hide decisions when they are no longer valid because of some other decision taken earlier. For example, if the decision 'Does the phone have a camera?' was answered with 'no', the decision 'What is the camera's resolution ?' is no longer valid and can be hidden during configuration. The number and precise meaning of each column in a decision table varies from one approach to another, but the example given here illustrates the basic idea of decision tables.

Similarly, decision trees also define decisions to be taken in order to configure one or more variable artifacts. However, the decisions are represented and arranged graphically. Figure 2 shows a small example of such a decision tree. The advantage here is that some selected dependencies between the decisions can easily be defined in this way. For example, the fact that the decision 'What is the camera's resolution?' is invalid if the camera is previously deselected altogether is clearly visible in the tree. Also, the number of possible product configurations is easily ascertainable, because each leaf in the decision tree corresponds to exactly one product configuration. However, this also points at an important problem with decision trees. They tend to become extremely large in complex cases. This can be avoided by using directed acyclic graphs instead of trees. Decision tree approaches (e.g. [14]) differ from one another in many details, but these are not required for the following discussion.

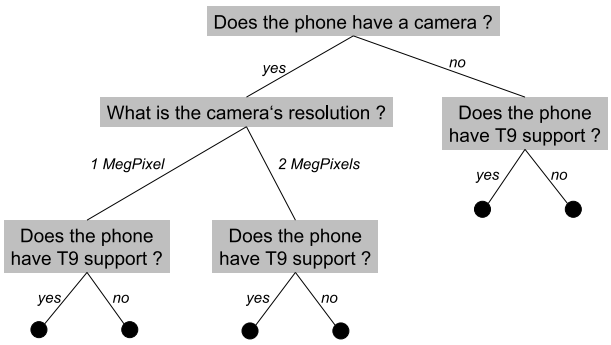


**Fig. 2.** Example of a decision tree.

Feature models are the third form of variability modeling in the variability dimension. A feature is a characteristic or trait that an individual product instance of a product line may or may not have [15]. The purpose of a feature model is to provide an overview of both the common and variable characteristics of the product instances and the dependencies between them. Figure 3 shows an example. Each node in the tree depicts a feature (e.g. `CruiseControl`, `Wiper`). During configuration, features are selected or deselected. Child features may only be selected if their parent is. Each child feature has a cardinality stating whether it is mandatory, i.e. it needs to be selected if the parent is, optional, i.e. it may or may not be selected if the parent is, or if it can be selected more than once (so called cloned features ; e.g. `Wiper`). When a feature is selected more than once, all its descendants can be configured separately each time the feature is selected. For example, if two wipers are selected during configuration of the feature model presented in Figure 3, then the `RainSensor` can be configured independently for each of the two. In addition, several children of a single feature can be grouped to express a certain dependency between them, e.g. the alternativity between `Simple` and `Adaptive` in Figure 3. More general dependencies between features

of different subtrees can be expressed through feature links which usually are depicted as an arrow (e.g. between `RainSensor` and `Radar`). Furthermore, features may be parameterized meaning that if the feature is selected during configuration, a value of a certain type has to be provided, for example when `Radar` is selected, the minimum distance to the next car has to be supplied as an integer value (cf. Figure 3).

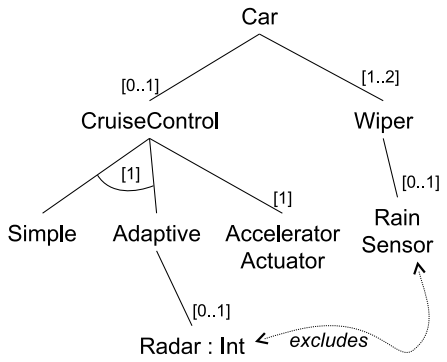


**Fig. 3.** Example of a feature model with advanced concepts.

Again, the details of feature modeling approaches (for an overview refer to [12]) differ greatly; but for the discussion presented here, the basic idea of feature modeling suffices.

Since all three forms of variability dimension modeling basically have the same purpose—presenting an overview of the product line's variability and providing a basis for configuration—it makes sense to ask whether they are basically equivalent and are merely different ways of presentation for the same information. In order to tackle this question more systematically, we examine whether the different forms of variability modeling can be translated into one another without loss of information.

Translating a decision table into a feature model is quite straightforward. For yes/no decisions, a simple feature is created; for value decisions, a parameterized feature is added; and for decisions with a finite set of enumerated resolutions, a parent feature is created together with a child for each of the allowed resolutions. Decisions' constraints are turned into feature links. The problems with this translation are:

(1) The natural-language description expressing the decision to be taken cannot be expressed in the feature model. The features' textual descriptions are not normally formulated in such a way. However, the description of a feature could still be used for this purpose, or an additional attribute could easily be introduced, if desired.

(2) Decision constraints can refer to several other decisions in a complex way. Since feature links are often defined as links from one single source feature to a single destination feature, this technique is less expressive. Again, this is not a fundamental problem for the translation because a more flexible feature linking concept could be provided.

(3) In addition, the feature model created from a decision tree in this way will be very flat. Since we only require that all the information from the decision table can be expressed and is therefore present in the feature model, this is not really an obstacle to such translation. However, it already points to an important problem that we will encounter below when examining translation in the opposite direction.

Despite these limitations, the translation from a decision table to a feature model works relatively well. Unfortunately, this is not true for the opposite direction. Basically, we can create a decision for each feature that is not mandatory as follows: for simple features a yes/no decision is created, and for parameterized features a value decision is provided; alternative features are merged into one decision with one resolution per feature. Parent-child relations are mimicked with decision constraints. While this mapping works well in principle, we identified several critical mismatches and problems during our investigation:

(4) Feature links can easily be formulated as decision constraints. However, in that case the dependency needs to be added to either the source or target decision, while a feature link represents a dedicated entity between the two. Also, one feature link can easily be kept apart from other feature links affecting the same feature and from dependencies that are expressed as parent-child relations, feature groups, etc. In decision constraints, all these dependencies get mingled within a single constraint.

(5) *The hierarchical structuring defined through the parent-child relationships gets lost.* Although the dependency expressed in a parent-child relation (i.e. the child may only be selected if the parent is selected) can be preserved in the corresponding decision constraint, it is not possible to document the fact that this dependency came from a parent-child relation. In other words, when looking at the decision table, it is no longer possible to distinguish between the dependencies that are to be interpreted as parent-child relations or hierarchy and those that are to be interpreted as feature links. This problem could be solved by introducing hierarchy in decision tables. However, it would then no longer be possible to edit them with standard office applications, which is one of the most important advantages of decision tables.

(6) Typed edges, i.e. types of parent-child relations, cannot be expressed in a decision table.

(7) *Cloned features cannot be translated into standard decision tables.* Of course, a similar concept could be incorporated in decision tables—i.e. several lines of the table would be replicated during configuration and then configured separately for each copy—but such a mechanism is not available in any of the common decision table approaches.

(8) *Mandatory features cannot be translated into decision tables.* This results in the most important difference between the forms of variability dimension modeling: in contrast to decision tables and decision trees/graphs, feature modeling does not primarily focus on the decisions to be taken during configuration and the resulting effects on the variable artifacts. Instead, feature modeling focuses directly on the differences and similarities between the product line's individual products. More specifically, feature models list all important characteristics of the individual products and state whether these characteristics are common to all products or vary from one product to another.

To this extent, decision diagrams/trees are very similar to decision tables. There is only one additional difference that arises when comparing them to the other two forms of variability dimension modeling:

(9) Decision diagrams/trees bring all configuration decisions into a certain order. For example, if feature f1 and f2 exclude each other (defined by a feature link), then neither has priority over the other. By contrast, in a decision tree with decisions d1 and d2—corresponding to f1 and f2, respectively—either d1 is asked before d2 (and consequently d2 won't be asked at all if d1 is answered with yes) or d2 before d1 (and d1 is therefore skipped in the case of a positive answer to d2). In the first case, d1 has "priority" and in the second d2. Even though this does not make a difference on a technical level, it is of great importance from a methodical point of view.

In summary, we can say that there are fundamental differences between the three forms of variability dimension modeling.

## 3 Artifact dimension variability

Successful management of variability also includes handling artifact variability, i.e. variability of software development assets on different realization levels. Artifact dimension variability can be thought of as being described in different ways:

- *Internal*: artifact variability is explicitly expressed within the artifact meaning that the possible design decisions and alternatives are explicitly given in the artifact descriptions.
- *External*: artifact variability is described outside the artifact meaning that the possible design decisions are captured elsewhere than within the artifact specification. Here, often the variability specification that forms the variability dimension is used as the location where the artifact variability is specified, i.e. the variability dimension is augmented by information on the variability of the artifact dimension.

Also the configuration of artifacts (process of binding variability) can be managed differently:

- *Generative/constructive artifact configuration*: artifact configuration is realized by generating the final, configured artifact and/or by composing it out of basic elements from an overall pool of (variable) artifact elements.

– *Alterative artifact configuration*: artifact configuration is realized by changing, i.e. enhancing or reducing, a default artifact model.

Bearing these possible differences in mind, one can derive basic artifact dimension variability approaches and essential concepts for the modeling of artifact dimension variability as well as the configuration of variable artifacts. In this section we provide an exemplary overview of artifact dimension variability by introducing a category matrix of artifact dimension variability management approaches, shown in Table 1.

**Table 1.** Category matrix of artifact dimension variability management approaches.

| | Internal | External |
|---|---|---|
| Generative / constructive | — | e.g. decision models, feature models |
| Alterative | e.g. explicit variation points and variants | e.g. aspect-oriented model transformation |

Table 1 gives an overview of basic groups of current approaches for artifact dimension variability definition. The idea is not to have a complete overview of existing approaches but to motivate the differences between these four essential concepts represented as the four fields in the matrix.

### External and generative variability management approaches

The field at the top right of the matrix represents approaches that model variability externally to the artifact and that obtain an artifact configuration by way of generation or through a composition of individual artifact elements or fractions of artifacts (e.g. [16] and [17]). As described above, artifact dimension variability is in this case incorporated into the variability dimension specification, where decision models for a set of assets as gained from domain engineering—as presented in PuLSE-CDA [8]—are an example of such an approach (see also the previous Section 2). Variability and dependency relations are only described in the decision model—not in the artifact elements. The anchor in the artifact elements for the decisions is described as part of the decision rule, e.g. by using a unique identifier for artifact elements. Decisions only refer to the variability or dependency relations at different levels of detail: thus one can either select or deselect a specific element or one may set specific values for parameters (see Figure 1). Invariable elements are thus selected automatically from the element pool, and transitive or technical relations are also applied automatically. An external variability modeling approach for a pool of artifact elements is easily applicable and does not need to be tool-supported in the first place. Complex variability and dependency relations can be modeled and the expressiveness of decision models is high because the complete arrangement of the artifact elements can

be described. Besides decision models as an external variability modeling approach for an elementary artifact pool also feature models can be thought of to be an applicable approach. The problem here is that feature models used for the configuration of artifact elements need explicit links to the artifact elements. Furthermore, it must be described at the artifact elements which constellation of features leads to which configuration of the artifact elements.

**Internal and alterative variability management approaches**

The field at the bottom left of Table 1 contains such approaches that model variability explicitly within the artifact elements of any kind of "default" model, which is changed in the course of the instantiation process, i.e. either the default model is enhancing because new elements are added, or it is reducing because variable elements are dropped from the model. An example of an approach describing variability internally for an enhancing default model is the explicit description of variation points, variants and dependency relationships between them. The default model then comprises all the characteristics (artifact assets) of the whole product family. The asset abundance is constantly reduced through variability binding. Describing variability within a variable default model calls for explicit scoping efforts in an early domain engineering phase because all products with their differences and commonalities are derived from the default model. Furthermore, it is not easy to obtain an overview of the artifact variability from a conceptual point of view because often a single conceptual variability (e.g. the wiper has a rain sensor) affects an artifact at many different locations and this variability's definition is therefore split across many variation points and, similarly, at a single location many different conceptual variabilities can have an impact and are thus mingled into a single variation point. Thus if the differences and commonalities are clearly defined and the artifact variability is complex in the sense that it is local or only technically based, then an artifact-internal variability description is reasonable. In order to obtain an overview of the variability and to facilitate the instantiation process the internal variability description will in this case often be complemented by an external variability modeling approach in the variability dimension, e.g. feature models.

**External and alterative variability management approaches**

The bottom right field in the matrix corresponds to external variability modeling approaches that change a default model. Once again—as described in the previous paragraph—the default model can be reduced or enhanced during the variability instantiation process. An example of such an approach is aspect-oriented model transformation. Differences between products are captured in aspects that are described in the form of a transformation rule. The rule consists of a point-cut and an advice, the point-cut specifying the place in the original model (join-point) that has to be amended with the model fragment described in the advice of the rule. This kind of variability modeling approach can handle complex variability and cope with many shortcomings of the internal approach

as described in the previous paragraph (esp. splitting of a single conceptual variability across many variation points). However, a difficulty with this approach is that things become extremely complex when several transformations affect the same location in an artifact. Basically, external variability modeling changing a default model can be used for all kinds of artifact variability, but its actual power lies in its great expressiveness: in contrast to the internal variability modeling approach, which is bound to the principal design of the default model, an external variability modeling approach can change entire parts of the design and rearrange or exchange completely independent design fragments. Thus especially in cases where variability results in a rearrangement of the design, an external approach would be worth considering.

**Internal and generative variability management approaches**

Remarkably, so far no approaches have been published in the literature describing variability in the artifacts with a generative/constructive artifact configuration (field in the top left of the matrix). This may be due to the fact that a variability description in artifact elements is usually limited to capturing simple variability and dependency relations in order to remain straightforward and manageable. As mentioned above, complex variability and dependency modeling calls for an orthogonal view of the variable artifact elements because coherences with respect to variability and dependency of variable artifact elements cannot merely be described at one element but are rather crosscutting. Here, a possible artifact dimension variability approach would assume a domain with simple and local variabilities and only technical dependencies (e.g. communication relationships). In this case, the variability and dependency relationships can be described locally at the artifact elements, and a configuration can be derived only via the element-based variability and dependency description. Although not an artifact dimension variability management approach, a Java development library can be thought of as the simplest approach to locally describe dependency relations and where the configuration is constructive, though not tool-supported.

**Discussion**

The remainder of this section discusses the results of table 1:

(1) Currently no internal and generative variability management approaches exist. Potential use could be in a case of simple, local occurrence of variability. Such a technique would have to be simple and may influence other variability management approaches to become easier applicable in various practical situations.

(2) External variability management approaches augment the variability dimension with additional, artifact related information and therefore—at least implicitly—establish a link between the artifact and variability dimensions. Use of feature models as well as decision tables proved successful, especially in the case of complex and highly interrelated variability. Use of feature models and an explicit description of the artifact configuration based on a feature selection

ends up in a mixture of internal and external description of artifact variability. Thus, it can be said that there is a continuum rather than a two-valued scale from internal to external variability modeling approaches.

(3) The most flexible technique to manage artifact variability is an internal and alterative approach. A default model containing variation points is changed in the course of the variation point instantiation. This technique is broadly used in programming (e.g. abstract parameters, types) and therefore often the first choice to manage variability of artifacts. In complex cases however, the spreading of variability information throughout the system models my prove infeasible. In this case a mixture between both an internal and external, generative approach would be needed.

(4) A more complex but powerful way to change a default model is the use of model transformation rules. With the help of these rules complete parts of the model can be rearranged and changes in different artifacts at various places can be defined together in a single rule. But because of its complexity model transformation should only be used in cases where this flexibility is needed. In all other cases an internal way to change the default model should be chosen.

It can be observed that most of the approaches applied in practice are mixtures between internal and external, generative and alterative variability management techniques; this is beneficial because the decoupled approaches complement one another.

## 4 Potential for further integration

Based on this survey of fundamental approaches to variability modeling and their interrelations presented in the previous two sections, we can now come back to our initial question whether this diversity of variability modeling techniques is actually required, or whether it would make sense to aim for replacing them with a single, comprehensive variability modeling technique.

First of all, a single technique for variability modeling both in the variability and in the artifact dimension is not realistic. These two cases of variability modeling are of very different nature actually: In the variability dimension, an overview of variations across many artifacts is to be provided on an abstract, conceptual level while in the artifact dimension, the variations of an individual artifact, i.e. the impact of variability, need to be defined precisely. Thus, the technique for the artifact dimension introduces variability in an existing artifact while the technique for the variability dimension constitutes an additional artifact of its own.

Consequently, the highest degree of integration that is conceivable would be to have a single technique for managing variability in the artifact dimension and another for the variability dimension. In the artifact dimension, however, the fundamental approaches towards variability modeling illustrated in Table 1 differ greatly and are aligned with very diverse methodological requirements (as described in Section 3). Therefore an integration would not make sense at this point. The next lower level of integration would be to provide a single,

integrated technique for each of the four cells of Table 1. But also this is not feasible in practice, because of the great diversity of the development artifacts that need to be covered by these techniques. For example a requirements specification may call for very different means to express variability than a test case description; similarly, the concept of aspect-orientation proved feasible for weaving variability into program code but its application to design models is still a challenging research issue. This diversity can be seen as being orthogonal to the two dimensions of Table 1. In order to ensure usability it is necessary to tailor the variability technique to the specific characteristics and needs of the artifact in question. Hence, the artifact dimension does not have great potential for a further integration of techniques.

For the variability dimension, on the other hand, this is much different. The variability dimension represents an artifact of its own and has a global perspective spanning all other development artifacts. It therefore needs to be independent of the artifacts' specificities anyhow. In addition, we identified that the different basic approaches towards variability modeling in the variability dimension all share the same major objective: establishing a global perspective on variations within the product line and defining dependencies between them. Practical considerations also suggest an integration of approaches: While the definition of the precise impact of variability inside an artifact (i.e. the purpose of variability modeling in the artifact dimension) usually is the responsibility of a single team working on the corresponding artifact, the global variability dimension is frequently subject to coordination between teams, departments and companies. Therefore, an integration in this area would be of high practical value.

However, we also identified substantial disparities between the basic approaches for variability dimension modeling, esp. the lack of commonality and hierarchy in decision tables. An integration will therefore be a challenging task and requires significant further research.

## 5 Summary and Conclusion

We surveyed current techniques for managing variability and, by categorizing them, identified several main approaches to variability modeling and examined how these are related. Based on this, we discussed the potential of integrating them into a single, common technique for variability modeling. We argued that such an integration effort should be targeted at the variability dimension only, both for practical and conceptual reasons.

## References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2002)
2. Bachmann, F., Goedicke, M., do Prado Leite, J.C.S., Nord, R.L., Pohl, K., Ramesh, B., Vilbig, A.: A meta-model for representing variability in product family development. In: Proceedings of the 5th International Workshop on Software Product-Family Engineering (PFE 2003). Volume LNCS 3014., Springer (2003) 66–80

3. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (foda) – feasibility study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute (SEI), Carnegie Mellon University (1990)
4. Asikainen, T., Männistö, T., Soininen, T.: A unified conceptual foundadtion for feature modelling. In: 10th International Software Product Line Conference (SPLC 2006). (2006) 31–40
5. Batory, D.: Feature models, grammars, and propositional formulas. Technical Report TR-05-14, University of Texas at Austin (2005)
6. Czarnecki, K., Helsen, S., Eisenecker, U.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practices **10**(1) (2005) 7–29
7. Czarnecki, K., Kim, C.H.P.: Cardinality-based feature modeling and constraints: A progress report. In: Proceedings of the OOPSLA'05 Workshop on Software Factories. (2005)
8. Muthig, D., John, I., Anastasopoulos, M., Forster, T., Dörr, J., Schmid, K.: Gophone – a software product line in the mobile phone domain. IESE-Report 025.04/E, Fraunhofer IESE (2004)
9. Bayer, J., Flege, O., Knauber, P., Laqua, R., Muthig, D., Schmid, K., Widen, T., DeBaud, J.M.: Pulse: a methodology to develop software product lines. In: SSR '99: Proceedings of the 1999 symposium on Software reusability, New York, NY, USA, ACM Press (1999) 122–131
10. TreeAge Software: TreeAge Software Inc. DATA Interactive White Paper. (1999) http://www.treeage.com/DIDocs/start/whitePaper.php3.
11. Apprentice Systems Inc.: Apprentice Decision Modeler. (2005) http://www.apprenticesystems.com.
12. Schobbens, P.Y., Heymans, P., Trigaux, J.C., Bontemps, Y.: Feature diagrams: A survey and a formal semantics. In: Proceedings of the 14th IEEE International Requirements Engineering Conference (RE 2006), IEEE Computer Society (2006) 136–145
13. Reiser, M.O., Tavakoli Kolagari, R., Weber, M.: Unified feature modeling as a basis for managing complex system families. In: Proceedings of the 1st International Workshop on Variability Modeling of Software-Intensive Systems (VAMOS), University of Limerick, Ireland (2007)
14. Tessier, P., Gérard, S., Terrier, F., Geib, J.M.: Using variation propagation for model-driven management of a system family. In: Proceedings of the 9th International Software Product Line Conference (SPLC 2005). Volume LNCS 3714., Springer (2005) 222–233
15. Pohl, K., Böckle, G., van der Linden, F.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Heidelberg (2005)
16. Czarnecki, K.: Overview of generative software development. In Bantre, J.P., ed.: Unconventional Programming Paradigms (UPP). Number 3566 in LNCS. Springer (2004) 313–328
17. Czarnecki, K., Eisenecker, U.: Generative Programming. Addison-Wesley (2000)