

Managing Highly Complex Product Families with Multi-Level Feature Trees

Mark-Oliver Reiser
Technische Universität Berlin
Fakultät IV, Lehrstuhl Softwaretechnik
Franklinstraße 28/29
D-10587 Berlin, Germany
moreiser@cs.tu-berlin.de

Matthias Weber
DaimlerChrysler AG
Research & Technology, REI/SM
Alt-Moabit 96a
D-10559 Berlin, Germany
Matthias.N.Weber@DaimlerChrysler.com

Abstract

Feature trees are a well-established instrument for domain analysis and modeling. But for highly complex product families like a vehicle manufacturer's product range – comprising well-above a thousand technical features – they become very large, and thus cumbersome and inflexible, especially when managing changes to the tree's structure over a long time. Furthermore, a conflict of aims arises when using feature trees in large organizations: while a single global feature tree for the entire company is desirable, local amendments for individual units or projects are indispensable in practice.

In this paper, we present a detailed description of this problem and show its great relevance to the automotive domain. We then provide a detailed definition of multi-level feature trees as a possible solution to the above problem. Finally, we describe scenarios how such multi-level feature trees can be put into practice and introduce a prototypical tool implementation of this concept.

1 Introduction

When applying product family concepts [6, 11, 14, 16] to the automotive domain, some specific characteristics of embedded real-time systems in general and automotive control software in particular have to be taken into account [4, 12, 19]. Among other issues, the great complexity of the organizational environment in which development takes place is of paramount importance here. Many different organizational units of the manufacturer work simultaneously on various vehicle models, and development is also performed by many different suppliers. Especially in such intricate organizational contexts, feature modeling has great potential: features may serve as a link between manage-

ment, marketing and development in a single company and as a link between companies to ease communication; and they can provide a central view of variability in a wide range of development artifacts - such as requirements, design models or test cases - thus becoming the core of all variability and evolution management [2]. But when applied in that manner, feature models tend to grow very large. Therefore, to take full advantage of this potential, it is essential to make complex feature trees more manageable.

A research project at DaimlerChrysler, being conducted in cooperation with the Technical University of Berlin, has examined this problem in several case-studies and elaborated an approach based on a distribution of the feature tree definition over several hierarchical levels. Using this approach, amendments to the feature structure can first be introduced locally, i.e. with respect to a single unit or project only, and later become valid in a broader context, if desired.

After defining several basic terms in the following section, we give – in Section 3 – a detailed description of the difficulties of managing extremely complex product families using the automotive domain as an example. Then, as a possible solution to these challenges, the concept of *multi-level feature trees* is introduced in Section 4. To show their potential in practice, we describe several scenarios of applying them in Section 5, before introducing our prototypical tool implementation of the proposed concepts (Section 6). The final two sections are then devoted to a discussion of related work and concluding remarks.

2 Terminology

Before we go into details, we will define a number of basic terms in this section.

A *product line* is a set of several products which share many common characteristics but also show certain substantial differences. When following a product-line oriented

approach, the individual products are no longer developed independently from one another, but instead only a single, but *variable* product is developed (also called *product line infrastructure*). From this variable product, the actual products (also called *product instances*) can then be derived by configuration. Consequently, development is divided into two separate phases: The development and evolution of the product line infrastructure (called *domain engineering*) and the definition of individual product instances to be derived from that infrastructure (called *product engineering*).

All development artifacts that make up the product line infrastructure (e.g. requirements, design models, test cases) can be defined in a variable form, if their content varies from product to product. This is usually achieved by adding *variation points* to them and defining several *variants* for each such variation point.

The term *product family* is sometimes used synonymously to product line. We prefer to think of a product family as a very complex product line, possibly made-up of several smaller product lines. This will be discussed in more detail in Section 3.

One of the most common concepts of product line methods are *feature models*. They are used to document what characteristics are common to all product instances and which characteristics differ from one product instance to another. Also dependencies between characteristics are defined in feature models. A *feature* in this context is a certain characteristic or trait that each product instance may or may not have. Often, feature models are hierarchically structured as *feature trees*. Apart from documenting the commonalities and differences between product instances, feature models also serve as coarse-grained requirements.

Further details of these basic terms are beyond the scope of this paper but can be found in [6, 8, 9, 16].

3 Automotive Development

To motivate the approach, this section describes common characteristics of the organization of automotive development.

When development of a new vehicle model (e.g. “the new E-Class model to be introduced in 2007”) is initiated, work is based – in nearly all cases – on predecessor models. The only exception to this rule is the introduction of a completely new vehicle line – but even then at least some of the subsystems will be initially taken from existing products. From a strictly technical point of view, development in the automotive domain can thus be seen as a process of amending predecessor vehicle models to optimize them (e.g. to reduce fuel consumption), to meet new requirements or to add innovative functionality. Basically, this process is infinite. It is important to note that what happens here is more than just a versioning of information: the information about the

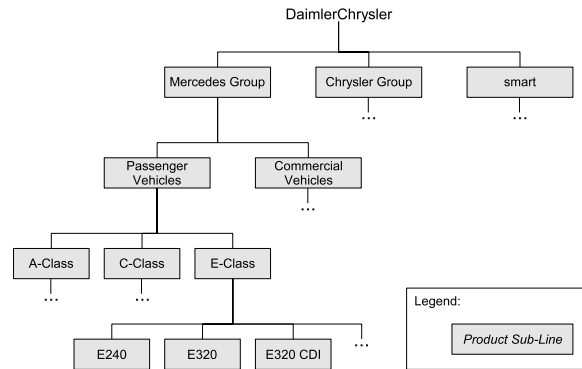


Figure 1. Example product/project structure.

previous models cannot be disposed of because these vehicles will still be in use for many years after production ends. And – even more importantly – the old vehicle model will be developed further, not only for bug fixing but also because some subcomponents may no longer be available as spare parts and therefore have to be replaced by alternative components, which may require changes to the surrounding hardware-/software-system. Thus, the information on the new vehicle model does not replace that on the old one but rather constitutes a new branch in development.

In addition, a car manufacturer’s product range is usually made up of several vehicle lines (e.g. A-Class, C-Class, E-Class), which are themselves often further subdivided in some way (e.g. E240, E320, E320 CDI with diesel engine). In addition, there often is a breakdown into passenger cars, freight vehicles, other commercial vehicles and possibly further categories. Moreover, today’s large corporate groups usually consist of large subgroups or brands (e.g. Mercedes-Benz, Chrysler, smart). This situation is shown in Figure 1. All these models, lines, series etc. are hierarchically related to each other: for example, all models of a particular vehicle line share certain characteristics, just as all vehicle lines of a vehicle series do. Sometimes vehicle models are even further subdivided, leading to additional levels.

In summary, this means there are several, hierarchically structured “contexts” in which development takes place. For these contexts, we use the term *product subline*. Each shaded box in Figure 1 can then be seen as such a product subline. In many cases, they are reflected in a car manufacturer’s organizational structure as divisions, departments or other organizational units or as projects. This gives them a significance that goes beyond methodological considerations.

When applying existing product line concepts – especially feature modeling – to this situation, one would have to choose between two basic approaches:

1. Organizing each *leaf* product subline as a separate

product family/line of its own; these sublines would then be represented by one feature model each.

2. Organizing all product sublines in one global product family; the sublines would then become partial configurations of a single global feature model.

For automotive development, neither possibility is truly satisfactory. In case of the first approach, this is immediately obvious: when treating the product sublines as separate, independent product families/lines, there is no way to document and manage the commonalities and differences between them, which means that the benefits of product-family-oriented software development are not available on the level above the individual (leaf) sublines.

The shortcomings of the second approach are not as obvious. Theoretically, it would be possible to have one global feature model because all the company's brands, lines and models are merely additional forms of variation of an imaginary company-global variable car. But such a monolithic feature tree would be enormously complex. First of all, even for a single car model, well-above a thousand technical features are necessary to discriminate which functionality is mandatory and which is optional in principle – regardless of final marketing decisions. Second, think of the large number of models offered by today's automotive manufacturers such as General Motors, Renault or DaimlerChrysler. And the number of different models is only the tip of the iceberg, because each model is offered in many different markets with differing legislation, in many different styles (e.g. Classic, Elegance, Sport), and so on. Thus, a monolithic feature model would be unmanageable, especially when it comes to adding new product sublines, making temporal changes in the tree's structure or – most importantly – making local amendments, i.e. amendments that are valid for only a single or only a few product sublines.

There are several reasons why such local amendments are so important:

- sometimes a product subline requires amendments that are so far-reaching and so special that they would unnecessarily complicate the feature models of other sublines a lot (e.g. if the structure of the climate control's subfeatures differed fundamentally from A- to S-Class)
- amendments may be needed to solve urgent problems that do not exist in other product sublines or to solve problems that will be treated differently in future models (stop-gap solutions)
- amendments may need to be introduced quickly when the discussion and coordination with the departments responsible for the other product sublines would take too long, is not immediately necessary and can therefore be postponed (local innovations).

One possible solution to this problem is staged configuration [9]. To reduce complexity when working in the context of a particular product subline, a partially configured version of the global feature model could be used. But using this mechanism, only a specialization of the global model can be achieved, e.g. by “narrowing” cardinalities or by masking unnecessary features, whereas other forms of local amendments are often needed, especially structural changes like adding child features or moving features within a tree hierarchy.

Another problem with the global model approach is that it would have to be introduced in a company at once – which is extremely difficult, if not impossible. It would be preferable to be able to harmonize the local feature models over a longer period of time, step by step, and move only those features to a higher-level product subline that are already harmonized and consolidated.

For these reasons, a different procedure for modeling variability in complex organizational contexts is proposed in the next section. It can be seen as a compromise between one global feature model and many independent models.

4 Multi-Level Feature Trees

In this section, the concept of multi-level feature trees is described in detail. Our main goal is to find modeling means to reflect the hierarchical organization of product sublines from Figure 1. This is achieved by introducing the notion of *reference feature models*: traditional feature models are enhanced with the option of having such a reference feature model (described in detail in Section 4.2). The reference model then serves as a template and guideline for the referring model by defining default features together with their default properties and by defining which deviations from these defaults are allowed. Thus, the reference model becomes a means to strategically drive the content of the referring model without inhibiting deviations where this is deemed unobjectionable. It is important to note that reference feature models are no longer used to directly define legal feature selections and thus possible product configurations (the traditional purpose of feature models). Instead, they affect product configuration only indirectly through influencing the contents of the referring model, which is the actual basis for product configuration.

To formulate a feasible framework for allowing or disallowing certain deviations from a reference feature model, we first examine what forms of deviations are possible in principle (Section 4.3). We then decide which of these deviations can be restricted in the reference model and how such restrictions can be defined (Section 4.4).

The fictitious example in Figure 2 is used throughout this section to illustrate these concepts. It shows a CruiseControl (CC) system that comes either in a

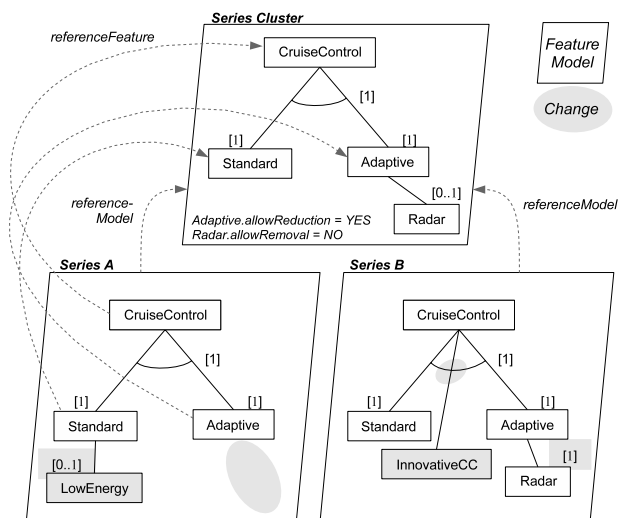


Figure 2. Example of a multi-level tree.

Standard or Adaptive version. In the latter case, vehicle speed is reduced when the distance to the car in advance will fall below a certain threshold. In an advanced version, this is controlled by a Radar. In “Series A” – a low-end model –, the radar is never used and therefore was removed to simplify the local feature model. Moreover, a stopgap solution was needed: for some vehicle configurations, a version of the CC with less energy consumption is needed (LowEnergy). In “Series B” – the high-end model –, adaptive CC is only sold with radar and some innovative third form of CC was introduced, but it is unclear whether or not it will be introduced to “Series A” in the future. While enormously simplified, this example presents typical problems occurring in real-world feature trees.

4.1 Basic Feature Models

The concept of multi-level feature trees can be seen as an add-on to traditional feature models. Therefore, we need to choose a feature modeling technique on which we can base the definition of multi-level feature trees. For the purpose of this paper, we used the technique proposed in [9]: Each feature has a cardinality similar to the cardinalities in UML class diagrams (e.g. $[0..1]$, $[1]$, $[0..*]$, $[0..4, 8..12]$). A cardinality of $[0..1]$ means the feature is optional, $[1]$ means the feature is mandatory and a maximum cardinality above 1 means the feature may appear more than once in a single product instance. Features may have child features. A child feature may only be selected for a product instance if also the parent is selected. Several child features of the same parent may be grouped in a *feature group*. These feature groups also have a cardinality, stating how many features of the corresponding group

must/can be selected for a single product instance. For example, a group cardinality of $[1]$ means that exactly one of the group’s features needs to be selected whenever their parent is selected.

Further details of this modeling technique are beyond the scope of this paper and are not needed in the following; in case you want to read more, please refer to [9]. We chose this technique here because it provides some concepts not found in many other feature modeling techniques (e.g. features with a cardinality above 1). In choosing such a relatively complex technique, we were able to also test our approach with these feature modeling concepts. In any case, the approach can easily be transferred to and used with other feature modeling techniques.

4.2 Reference Feature Models

A *reference feature model* is a normal feature model except that there is some other feature model, called the *referring feature model*, that refers to it as its reference model. In Figure 2 “Series Cluster” is the reference model for the referring models “Series A” and “Series B”. A feature model can only refer to one reference feature model at the most, but several feature models can refer to the same reference model.

The reference feature model can itself refer to a third feature model as its reference model, for example if “Series Cluster” would refer to a company-global reference model called “Company Cluster”. This situation is expressed in the meta-model of Figure 3 by the association named *referenceModel* with a cardinality of $*:0..1$.

Furthermore, a feature in the referring feature model can point to a feature in the reference feature model as its *reference feature*. It is then called a *referring feature*. This is realized by the association *referenceFeature* in Figure 3. In the example, this is only shown for “Series A” and omitted for “Series B” to not clutter the presentation too much. Instead of an explicit association being used, the names of the features could have been employed to find corresponding features in a reference model. But then features with the same name would “automatically” be associated with each others, when an existing feature model is declared the reference model of another already existing feature model. But in such a situation, it is preferable to only assume an association when the designer has actually decided to do so. The disadvantages of explicit associations – mainly the effort needed to edit them and keep them up-to-date – can be eliminated with appropriate tool support. For example, when initially defining these associations, a tool could provide functionality to initialize them depending on a feature name matching; similarly, a tool could hide in the editor all reference feature associations between features with matching names to allow the user to easily grasp

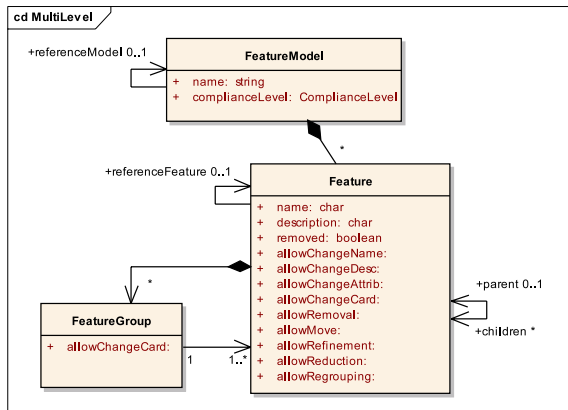


Figure 3. Meta-model for Multi-Level Trees.

all exceptions from that rule.

The concept of a reference model now allows for a hierarchically structured tree of reference feature models that reflect the situation described in Section 3. Each node in the tree shown in Figure 1 would then be represented by a feature model that is both a reference model (for the lower-level models) and a referring model (with respect to the higher levels), except for the root which is only a reference model, and the leafs which are referring models only. Such a tree of reference/referring feature models is called a *multi-level feature tree*.

The only thing left to be done is to provide a means to precisely define in a reference model what deviations from it are allowed or disallowed. To do so, the possible forms of deviation will be examined in the next section.

4.3 Possible Forms of Deviations

Let F and R be two feature models, where R is the reference model for F . Furthermore, let f be a feature in F that refers to feature f_R in R as its reference feature. To make it easier to think about the possible content of F and R , we assume that F was initially created as an exact duplicate of R , where all features in F refer to their corresponding feature in R as their reference feature, and that some changes have subsequently been made to F and R . Now, let us examine in detail how F and R can differ from each other (summarized in Table 1).

First, all differences in F compared to R are called *deviations*. In Figure 2 they are shaded in gray. The simplest form of such a deviation is a difference in a basic attribute of a feature, such as its description. name also belongs to these basic attributes because it is not used as an ID (see above). Similarly, there may be differences with respect to the attribution of a feature, i.e. whether the feature is an attributed feature or not. In particular, the referring feature f can be attributed while the reference feature f_R is not (or

vice versa) or f has a different type than f_R . Then, the cardinalities can differ between the referring feature and its reference feature. We distinguish two cases of differing cardinalities: the cardinality of the referring feature is a proper subset of the reference feature's cardinality, or it is not a subset. Radar in "Series B" is an example of the first case.

Next, there may be a different number of features in the referring model than in the reference model. Precisely speaking, we must distinguish four situations:

1. There is a feature in R for which no *referring* feature is defined in F because the feature in R was *added*.
2. There is a feature in R for which no *referring* feature is defined in F because the corresponding feature in F was *removed*.
3. There is a feature in F for which no *reference* feature is defined in R because the feature in F was *added*.
4. There is a feature in F for which no *reference* feature is defined in R because the corresponding feature in R was *removed*.

Radar in "Series A" is an example for no. 1 & 2 and Low-Energy for no. 3 & 4. Unfortunately situation no. 1 cannot be distinguished from no. 2 and no. 3 not from no. 4 when looking only statically at the two feature models at a certain point in time. But, from a methodological point of view, it makes a great difference whether a newly added feature in R has not yet been considered for inclusion in F – situation no. 1 – or a referring feature has been (intentionally) removed even though the reference feature (still) exists – situation no. 2 above. In the first case, the designer of the referring model still has to make the decision as to whether to include the new feature or not, whereas in the second case this is not true. The same applies to the other two situations: a new feature in F has not yet been considered for representation in R as a reference feature or its reference feature has been intentionally removed from R . To solve these problems, a feature that refers to a reference feature or that is referred to by one or more features as their reference feature is not actually removed from the feature model when deleted, but is only marked as deleted. This is the purpose of the removed attribute in class Feature in Figure 3.

What remains are differences in the structure of the feature trees. These are a bit tricky because all possible forms of structural differences have to be covered without introducing too many types of deviations. First of all, the parent of f can differ from f_R 's parent. Again, we distinguish two situations of differing parents: the parent differs but f_R 's parent is still an ancestor of f (i.e. f has been moved down in the tree) or the parent differs "completely", including the case that f is a root feature (i.e. its parent is set to *null*).

On the other hand, the children of f can differ from those of f_R . Here, we distinguish these cases:

1. f has a child for which there is no corresponding reference feature among the children of f_R :

$$\exists f' \in f.children : f'.referenceFeature \notin f_R.children$$

For example, this is the case for `Standard` in “Series A” and `CruiseControl` in “Series B” but not for `CruiseControl` in “Series A”.

2. f_R has a child for which there is no corresponding referring feature among the children of f :

$$\exists f'_R \in f_R.children : \quad \nexists f' \in f.children : \\ f'.referenceFeature = f'_R$$

In the example, this is the case only for `Adaptive` in “Series A”.

3. Like 2. but there is a corresponding referring feature at least among the descendants of f .

Deviation 1 is called *refinement* of f and means that a child feature was newly created as a child of f or moved from somewhere else within the same feature model to f . We do not have to distinguish between these two cases here because in the second case, and only in the second case, there is an additional deviation with respect to the parent of the moved feature. Deviation 2 is called *reduction* of f and means that a child feature was moved somewhere else or that it was entirely removed from the model while deviation 3 (also called reduction) means that it was moved down in its subtree.

A last form of deviation is related to feature groups and occurs whenever child features are added to or removed from an existing group (including the case that a group is created, deleted or split) or a group’s cardinality is changed. In this context, ungrouped features are treated as being in a single, implicit group of cardinality $[0..*]$. The feature group below `CruiseControl` in “Series B” is an example for this, because `InnovativeCC` has been added.

Using these types of deviation, all differences of a feature model compared to its reference feature model can be described. A few differences are not covered, like the re-ordering of child features in the parent’s list of children. These differences are ignored by the reference model concept presented here, i.e. a feature model differing only with respect to such properties from its reference feature model is deemed identical to it.

Please note that we assumed that F started as a duplicate of R only to simplify the explanation. The definition of the possible deviations would also be applicable without changes if, for example, F was created before R and R initially started as an empty model.

4.4 Restricting Deviations

The concept of multi-level feature trees is all about managing deviations in a referring feature model with respect to its reference feature model. To allow for this, a means has to be provided to allow and disallow certain forms of deviation from within the reference model. This is described in this section.

For each type of deviation, Table 1 lists an attribute that is responsible for specifying whether such a deviation is allowed or not (second column). The possible values of these attributes are listed in column 3. When a certain value is set, a constraint is imposed on the referring model, which is given in column 4. Please note that the attribute’s values are those specified in the reference model, whereas the corresponding constraints must be met by the referring model.

Sometimes there are more choices available than only completely allowing or disallowing a deviation. For the basic feature attributes `name` and `description`, it is possible to specify that only additional text may be appended and the original text may not be removed or altered. Changes in a `Feature`’s or `FeatureGroup`’s cardinality can be restricted to partial configuration of the cardinality as defined in [9], i.e. the cardinality of the feature / feature group in the referring model must be a subset of that of the feature / feature group in the reference model. Finally, moving a feature can be restricted to moving it only in the subtree below it, i.e. moving it down the feature tree hierarchy. This can be specified either in the feature itself – by setting `allowMove` appropriately – or in its parent feature – through setting `allowReduction`. In the latter case, the setting also affects all the feature’s siblings.

With these definitions in the reference feature model, it is possible to decide whether a referring feature model *conforms to its reference model* or not, i.e. whether all deviations from the reference model (if any) are allowed by that reference model or not. Now, three different states can be distinguished, called *conformance states* of the referring model:

1. The referring model is identical to the reference model and therefore conforms to it.
2. The referring model deviates from but still conforms to the reference model.
3. The referring model does not conform to the reference model.

The third case is not technically forbidden, i.e. it is syntactically legal to bring a referring model into that state and this should not be prevented by a modeling tool supporting multi-level feature trees. The modeler should only be urged to make the referring model conformant. But in special cases it can make sense to even tolerate nonconforming

Table 1. Overview of possible forms of deviations and the attributes allowing or disallowing them.

Form of Deviation	Permission Attribute	Values	Constraint
different name	allowChangeName	NO APPEND YES	$f.name = f_R.name$ $\exists s \in \text{String} : f.name = f_R.name + s$ <i>true</i>
difference in other basic attribute, such as a feature's description	allowChange...	<i>as above</i>	...
difference in feature attribution	allowChangeAttrib	NO YES	$class(f.attribute) = class(f_R.attribute)$ <i>true</i>
different cardinality	allowChangeCard (in Feature and FeatureGroup)	NO SUBSET YES	$f.cardinality = f_R.cardinality$ $f.cardinality \subseteq f_R.cardinality$ <i>true</i>
no referring feature for a reference feature (Removal)	allowRemoval	NO YES	$\exists f' \in F : f'.referenceFeature = f_R \wedge f'.removed = false$ <i>true</i>
no reference feature for a referring feature	-	<i>always allowed</i>	<i>true</i>
different parent, including $parent = null$ (Move)	allowMove	NO SUBTREE YES	$f.parent.referenceFeature = f_R.parent$ $f.parent.referenceFeature \in descendantsOf(f_R.parent)$ <i>true</i>
referring feature f has a child without a corresponding reference feature among the children of f_R (Refinement)	allowRefinement	NO YES	$\forall f' \in f.children : f'.referenceFeature \in f_R.children$ <i>true</i>
reference feature f_R has a child without a corresponding referring feature among the children of f (Reduction)	allowReduction	NO SUBTREE YES	$\forall f'_R \in f_R.children : \exists f' \in f.children : f'.referenceFeature = f'_R$ $\forall f'_R \in f_R.children : \exists f' \in f.children : f'.referenceFeature \in descendantsOf(f'_R)$ <i>true</i>
difference in the grouping of the children (Regrouping)	allowRegrouping	NO SUBSET YES	omitted omitted <i>true</i>

referring models. This is a question of the role of the reference model and its precise application in the actual development projects.

In the example “Series A” is not conformant to “Series Cluster”, because Radar was removed which is disallowed (note the permission definition in the lower left corner of “Series Cluster”). If we swapped the values of `Adaptive.allowReduction` and `Radar.allowRemoval` in “Series Cluster”, “Series A” would still be non-conformant. But then all children of “Adaptive” would be affected by the restriction.

The permission attributes introduced here bring about another form of deviation omitted in Table 1: a feature of the referring model may have its permission attributes set to other values than its reference feature ($f.allowXYZ \neq f_R.allowXYZ$). This is only of interest if the referring model is at the same time also used as a reference model, because only in this case do the values of the permission attributes matter. Again we can distinguish two cases of differing permission attributes:

1. The value is different and more restrictive (e.g. APPEND instead of YES or NO instead of APPEND in the case of `allowChangeDesc`).
2. The value is different and less restrictive (e.g. YES instead of APPEND or NO).

Making permission attributes more restrictive is always allowed, making them less restrictive is always disallowed. Thus no additional permission attributes must be introduced for this form of deviation, which is why it was omitted from Table 1.

In summary, the concepts described in this section, namely reference feature models and multi-level feature trees, provide a means to organize a very complex product family as several hierarchically related product sublines, offering a compromise between having one large global product family or having many small, independent product families/lines.

5 Use Scenarios

This section describes some typical use scenarios for multi-level feature trees we encountered during our case studies at DaimlerChrysler. These scenarios are not intended as alternative ways of applying multi-level feature trees but rather give an overview of typical – possibly simultaneously occurring – situations in which they reveal their potential.

The starting point for this is a tree of several product sublines, as shown in Figure 1. Each such product subline

is represented by a feature model, which refers to the parent subline's feature model as its reference model. Together they form a multi-level feature tree as described above.

Now, several use scenarios are conceivable:

(1) A new leaf of the multi-level tree, i.e. a new product or product subline, is set up by creating a copy of the parent or a sibling feature tree, and, for each feature in the copy, deciding whether to reuse it or not. If reused, a link from the feature in the copy to the corresponding feature in the parent tree is created; if not reused the feature is simply removed from the copy.

(2) *Bottom up reuse*: In leaf product sublines, features for innovative functionality can be introduced locally at first. Then, when the new functionality has been consolidated and is to be used in more models, lines, etc., the corresponding features can be propagated up the tree hierarchy step by step to make them visible and valid in a broader context. The reason for introducing new changes locally at first may be that the new feature represents a stopgap solution (e.g. *LowEnergy* in Figure 2) for a problem that only exists locally or will be solved differently in other sublines. Then it makes no sense to clutter the global model with it. Another reason may be that it represents an innovation which may become useful in other contexts (e.g. *InnovativeCC*), but at the moment there is no time or no reason to decide this and discuss it with the personnel responsible for the global model.

(3) *Top down reuse*: Accordingly, features for innovative functionality may also be immediately introduced in the root reference feature model. This often makes sense when the innovation is driven by management decisions and is valid for many different product sublines at once, e.g. "all fellow passenger air-bags in all the company's models must be able to be switched off to avoid accidents with child car seats". When using this mechanism, it is also possible to introduce new functionality in a step-by-step manner by first allowing many deviations from the globally introduced features and then making them more and more restrictive over time.

(4) As a middle course between the previous two scenarios, innovative features may also be introduced to a product subline somewhere below the root and above the leaves, e.g. "fuel cells should be available in C-, E- and S-Class from 2006 onward".

(5) Instead of propagating up the tree hierarchy an innovative feature, that has no corresponding features in sibling product sublines (second scenario above), the concept can also be used to integrate parallel innovations. When two sibling product sublines have developed two versions of a new subsystem in parallel (e.g. a pedestrian safety system), the features related to this subsystem can be integrated by propagating up features that represent a common view of the variability of the new subsystem and defining the local

situation for each subline with local deviations.

In contrast to these scenarios, which are all based on the assumption that a multi-level feature tree is already in use, there is also the interesting scenario of introducing them:

(6) Multi-level feature trees do not have to be introduced all at once. Instead, they can be introduced in small steps. For example, let us assume that two departments are both working on climate control systems, one for the A-Class and smart, the other for the C-, E- and S-Class. These two departments can, as a first step, build their own, local feature trees. In a second step, they can define for them a reference feature model which is empty to begin with. Then, step by step, they see which parts of their local feature models are the same and propagate these features up to the reference model. This can be done not only for features that are completely identical but also for features that are only similar, because each department can still define deviations from the reference model to reflect its legacy situation. Next, they can try to reduce the number of such deviations over time so as to end up with a common feature model. In addition, other departments also working on climate control systems may join in at some time during the process. In simple situations, the local feature trees may eventually even cease to be necessary. And this is not only possible for individual departments but also for vehicle lines, groups, etc. (e.g. C- and E-Class or passenger vehicles and commercial vehicles) or individual subsystems.

6 Tool Implementation

The concepts described in the previous sections were prototypically implemented as a feature-modeling tool. The main purpose of this tool was to allow us to experiment with different common forms of feature modeling in conjunction with multi-level feature trees. We chose to implement a prototype instead of building upon existing tools, because the feasibility of the approach presented above is closely linked to how it is embedded in the overall feature editing facility and we therefore needed to experiment with different editing functionality. The implementation is based on the Eclipse Tools Framework [10].

The functionality of the prototype comprises:

- basic feature modeling (i.e. creating and deleting features, moving features within the feature tree hierarchy, editing properties of features like name, description, cardinality, attribution)
- explorer-style textual views as well as graphical views of the feature models
- saving/loading in XML format
- creating and deleting reference feature models (see Section 4.2)

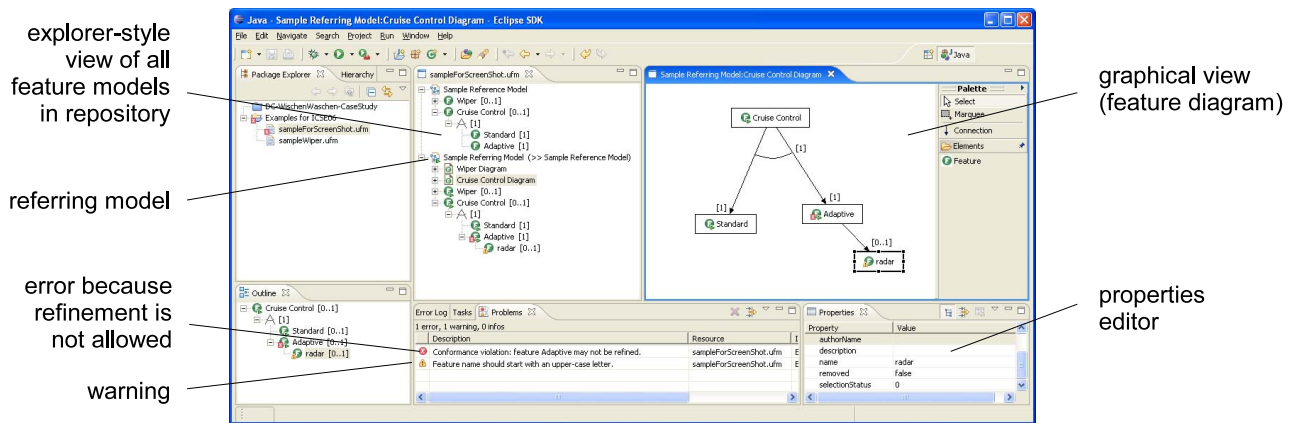


Figure 4. A prototypical tool to manage multi-level feature trees.

- propagating features from a referring model to a reference model and vice versa (see Section 5)
- finding deviations in a referring model with respect to its reference model, either from within the referring model or from within the reference model (Sections 4.3 and 4.4)
- determine the conformance state of a referring model (Section 4.4)

Figure 4 shows a simple feature model in both the explorer-style view and a graphical feature diagram view.

The tool is designed as an Eclipse plug-in. It is based on EMF [3] for implementing the data model and on GEF [10] for the graphical editing functionality. This architecture would enable the tool to evolve into a full-scale feature-modeling environment that can be applied in mission-critical projects. Currently, however, this is only seen as an option and is not one of our primary goals.

7 Related Work

Staged configuration as proposed in [9] is closely related to multi-level feature trees. This fact was already discussed in Sections 3 and 4.

Another interesting approach that also aims to make highly complex product families more manageable is described in [5]. The difference is that this approach tries to achieve its goal through introducing views on variability models, whereas the approach proposed here rather tries to avoid a global, extremely large variability model by interrelating formerly independent models. Also, there is no way to strategically drive the contents of one view from within another and it is not possible to introduce changes locally without affecting the global model. On the other hand, views are a simple, yet powerful concept and probably are in line with many software product family use cases.

In addition, multi-level feature trees are related to hierarchical and n-dimensional product lines as introduced in [17]. However, the approaches differ at a very fundamental point: hierarchical product lines are made-up of sublines in a strict sense, i.e. a valid product of a subline is always a valid product of the higher-level product line, whereas in multi-level feature trees, a valid product of a referring feature model does not necessarily have to be a valid product of the corresponding reference feature model, due to the fact that very far-reaching deviations may be allowed. Also, multi-level feature trees are more aimed at the specific problems that arise from complex organizational structures and on the process of initiating product-line oriented development in these contexts.

Finally, the similarity and difference between multi-level feature trees and version-handling systems like CVS [18] or SVN [7] deserve a few comments. The similarity is that multi-level feature trees also provide support for version handling. The difference is that they also provide the notion of reference feature models. Or to put it differently: in concurrent version systems, reference models and their referring models could be represented by one branch per model. But it would not be possible with these systems to control which deviations are allowed in one branch (representing a referring model) from within another branch (representing the reference model), which is the fundamental idea of multi-level feature trees.

A brief work-in-progress report on multi-level feature trees was given in [1].

8 Conclusion and Further Work

To manage extremely complex product families, we have proposed multi-level feature trees consisting of a tree of feature models in which the parent model serves as a reference feature model for its children. This enables concordance

to be strategically steered from the higher levels down to the leaf feature models and, at the same time, allows local innovations to be introduced locally at first and then made globally visible in a step-by-step process.

Possible future work includes studying feature models that have more than one reference model at the same time and finding means to further support management activities through reference models by assigning relevant meta-information to reference features.

Another very interesting subject for future research beyond the field of feature modeling and product-line engineering would be to investigate whether the notion of reference feature models could be generalized so as to be applicable to all kinds of models. To this end, the reference model definition could be reformulated so that it is based on OMG MOF [15] in much the same way as it was based on feature models in Sections 4.3 and 4.4.

9 Acknowledgments

We would like to thank Stan Bühne, Kim Lauenroth and Klaus Pohl from the University of Duisburg-Essen, Alexander Poth from MB-technology GmbH, Stuttgart and Ali Botorabi, Nikolai Cieslak, Jörg Donandt and Matthias Hoffmann from DaimlerChrysler, Berlin for many fruitful and stimulating discussions on this subject.

References

- [1] M. Bittner, A. Botorabi, A. Poth, M.-O. Reiser, M. Weber: "Managing Variability and Reuse of Features and Requirements for Large and Complex Organizational Structures". In: *Proceedings of the 13th IEEE International Requirements Engineering Conference*, RE 2005, pp. 469, 470, 2005.
- [2] G. Böckle, P. Knauber, K. Pohl, K. Schmidt: "Software Produktlinien". dpunkt Verlag, 2004.
- [3] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, T. J. Grose: "Eclipse Modeling Framework". Addison Wesley, 2003.
- [4] S. Bühne, K. Lauenroth, K. Pohl, M. Weber: "Modeling Features for Multi-Criteria Product-Lines in Automotive Industry". Workshop on Software Engineering for Automotive Systems (SEAS), at ICSE 2004, Edinburgh, 2004.
- [5] S. Bühne, K. Lauenroth, K. Pohl: "Modelling Requirements Variability across Product Lines". In: *Proceedings of the 13th IEEE International Requirements Engineering Conference*, RE 2005, pp. 41-50, 2005.
- [6] P. Clements, L. Northrop: "Software Product Lines: Practices and Patterns". Addison-Wesley, 2002.
- [7] B. Collins-Sussman, B. W. Fitzpatrick, C. M. Pilato: "Version Control with Subversion". O'Reilly, 2004.
- [8] K. Czarnecki, U. W. Eisenecker: "Generative Programming – Methods, Tools and Applications". Addison-Wesley, 2000.
- [9] K. Czarnecki, S. Helsen, U. W. Eisenecker: "Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models". In: *Software Process Improvement and Practice*, 10(2), pp. 143-169, 2005.
- [10] Eclipse Foundation Web-Site: <http://www.eclipse.org>
- [11] J. Greenfield, K. Short, et. al.: "Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools". Wiley, 2004.
- [12] K. Grimm: "Software Technology in an Automotive Company – Major Challenges". In: *Proceedings of the 25th International Conference on Software Engineering, ICSE 2003*, Association for Computing Machinery & IEEE Computer Society, pp. 498-503, 2003.
- [13] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson: "Feature Oriented Domain Analysis (FODA) – Feasibility Study". Technical Report, CMU/SEI-90-TR-21, 1990.
- [14] K. C. Kang, J. Lee, P. Donohoe: "Feature-Oriented Product Line Engineering". In: *IEEE Software*, Vol. 19, pp. 58-65, 2002.
- [15] Object Management Group (OMG): "Meta Object Facility (MOF) – Specification". Version 1.4, 2002. OMG Document formal/02-04-03.
- [16] K. Pohl, G. Böckle, F. van der Linden: "Software Product Line Engineering: Foundations, Principles and Techniques". Springer, Heidelberg, 2005.
- [17] J. M. Thompson, M. P. E. Heimdahl: "Structuring product family requirements for n-dimensional and hierarchical product lines". *Requirements Engineering* 8(1), pp. 42-54, 2003.
- [18] J. Vesperman: "CVS - Concurrent Versions System". O'Reilly, 2004.
- [19] M. Weber, J. Weisbrod: "Requirements Engineering in Automotive Development – Experiences and Challenges". RE 2002, pp. 331-340, 2002.