Feature-oriented Modelling of Product Line Evolution

Andreas Pleuss
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

Goetz Botterweck
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

Deepak Dhungana
Lero – The Irish Software Engineering Research Centre
University of Limerick, Ireland

Andreas Polzer
RWTH Aachen, Germany

Stefan Kowalewski
RWTH Aachen, Germany

26th November 2010

Contact

Address	Lero International Science Centre University of Limerick Ireland
Phone	+353 61 233799
Fax	+353 61 213036
E-Mail	info@lero.ie
Website	http://www.lero.ie/

Copyright 2010 Lero, University of Limerick

This work is partially supported by Science Foundation Ireland
Lero Technical Report Lero-TR-2010-05

Feature-oriented Modelling of Product Line Evolution

Andreas Pleuss[a,], Goetz Botterweck[a], Deepak Dhungana[a], Andreas Polzer[b], Stefan Kowalewski[b]

[a] *Lero, University of Limerick, Ireland*
{*andreas.pleuss, goetz.botterweck, deepak.dhungana*}*@lero.ie*
[b] *RWTH Aachen, Germany*
{*polzer, kowalewski*}*@cs.rwth-aachen.de*

Abstract

Product line engineering (PLE) needs to consider evolution and strategic planning of evolution steps right from the beginning. If evolution is not taken care of, the benefits of model-driven PLE (e.g., productivity gains, strategic reuse, complexity handling) will be difficult to achieve. In this paper we propose an approach for strategic planning and management of feature model evolution. Our approach includes a modelling framework that describes evolutionary changes of product lines using extended feature models (called *EvoFM*), evolution plans, and fragments of models. Different versions of a feature model along the evolution path are considered to be "products" that can be configured using the EvoFM. By using feature models to document the evolution plan of feature models, we demonstrate an elegant solution to dealing with long term planning of changes to feature models.

1. Introduction

The core idea in Software Product Line Engineering is to spend additional effort into creating and implementing a family of product so that later on single products can be derived more efficiently. Concepts from the area of Model-Driven Engineering can be used to further increase efficiency and automation of product derivation which leads to model-driven PLE. The commonality and variability between the products in a product line are represented by variability models, e.g., by feature models which describe the available features in the product line and the dependencies between them in a hierarchical structure. A concrete product is then defined by a feature

configuration specifying which features are selected and which are deselected in the product.

To leverage the benefits offered by the product line over many years and keep products up to date, the product line often has to evolve. However, it is impossible to foresee all the changes necessary in the future and scope the product line accordingly right from the beginning. Apart from that, changes can also be induced by evolving technologies or events in the market place (e.g., customers changing preferences, features introduced by the competition).

There is a general consensus that product line evolution is an important aspect to consider [1, 2, 3], however, only very little literature is available on strategic planning of product line evolution (see Section 5). Product line evolution is often "handled on the fly", i.e., the product line is arbitrarily extended to satisfy the needs of new customers or technological changes. Spontaneous changes to a product line can be helpful and an easy thing to do. But to ensure sustainable success one has to apply a systematic approach to product line evolution planning. In the context of model-driven PLE, this requires the use of defined processes and tools to deal with evolution of feature models (FM) and the reusable artefacts.

Our experience with industry and common engineering practices for product line maintenance has shown that there is a huge need for proactive evolution of product line artefacts (e.g., feature models, reusable assets) [4, 5]. In industrial contexts, decisions about the evolution of the product portfolio (i.e., product line) are often made by the product management team. Such decisions then have to be implemented by the development team. In such situations, a strategic planning is needed to decide on the different changes that need to be made to the feature model.

In this paper, we present a novel approach for proactive evolution management. Initial ideas related to this approach have been published in [4, 5]. Here we present an extension of our previously published papers and present the concepts and tool support in greater detail.

Our approach considers evolution to be a sequence of models with one model for each evolution step. As an example, consider the different versions of the same feature model shown in Figure 1. These models represent a product line of automotive infotainment systems that evolves over four years. In 2009, there is just an optional radio that can be configured. In 2010, a navigation system is added, which uses the user interface of the radio. Due to a management decision, the navigation system is planned to be separated

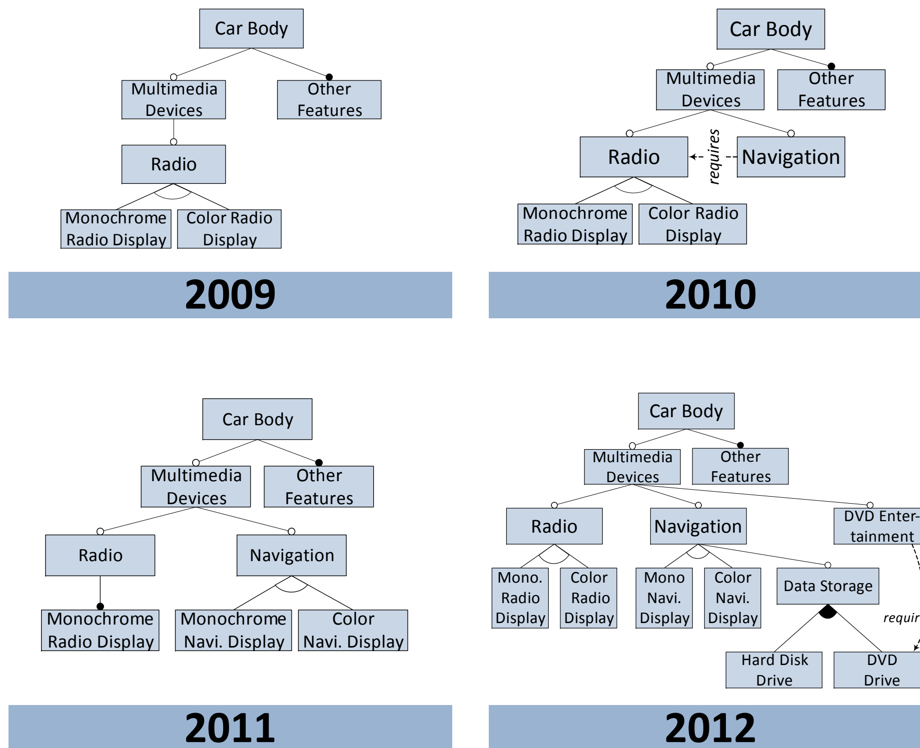


Figure 1: Evolution of a product line, depicted in terms of a sequence of evolved feature models.

from the radio in 2011 having a display of its own while radio is offered with a simple monochrome display only. In 2012, the navigation system will be enhanced with the possibility to store more maps and DVD Entertainment will be introduced, which requires a DVD Drive.

To describe the evolution, we capture the *changes* made to the feature model over time in special separate models and describe the evolution path as a combination of changes. Each change made (or planned) is considered to be either a "model fragment" or a "change operator" on the original model. Such changes are documented in an EvoFM model, which is a special kind of feature model consisting of model fragments and change operators as features.

The remainder of the paper is structured as follows: Section 2 provides a first overview of the EvoFM framework, Section 3 introduces the various models that are used in the framework, and Section 4 shows how to use these

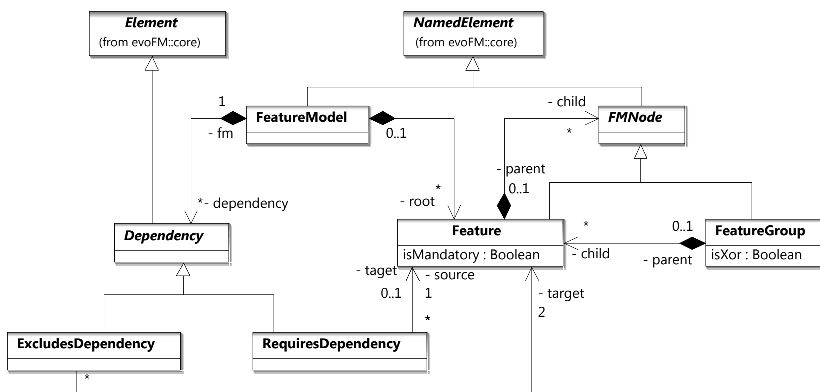


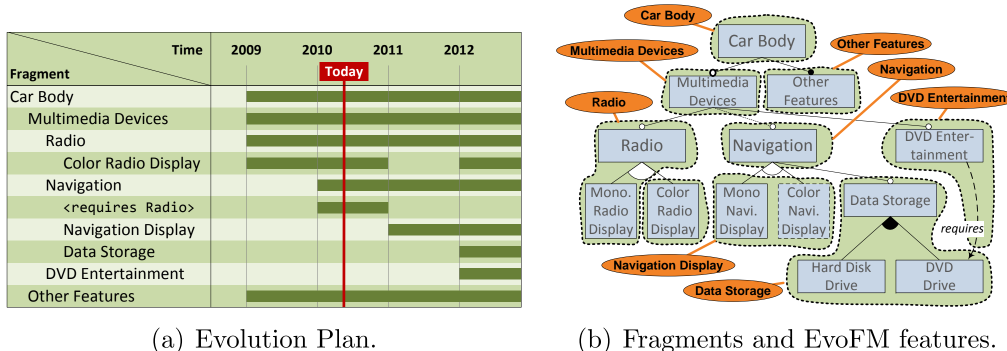
Figure 2: Metamodel for Feature Models used in this article.

models. The paper concludes with an overview of related work in Section 5 and final thoughts in Section 6.

2. EvoFM Framework Overview

This section discusses drawbacks of possible existing solutions and then provides an overview of our approach.

In this article we assume the common notation and semantics of FODA-based feature modelling approaches [6]. Figure 2 shows the metamodel we refer to in the following. We restrict here to feature defined as *mandatory* or *optional* and features groups defined as *or* or *xor* but nothing prevents from extending the approach to further extensions on feature models like arbitrary cardinalities for features and features groups like in [7]. Feature groups are understood as model elements of their own (implying that a feature can own several feature groups) but are usually denoted in diagrams as anonymous arcs on edges between parent and child features (like in Figure 1). Cross-tree dependencies between features are restricted to *requires* and *excludes* dependencies. Other, more implementation-oriented models associated with the feature models are not further discussed in this article. As explained earlier (cf. Section 1), we consider the evolution of a product line as a sequence of models. Hence, the essential information when reasoning about product line evolution are the changes between feature models over time. An apparent solution to expose these changes could be using model comparison tools like *EMFCompare* [8] or Epsilon ECL [9] to determine the differences



(a) Evolution Plan. (b) Fragments and EvoFM features.

Figure 3: Example: the product line is clustered into fragments which are represented as features in EvoFM. The Evolution Plan shows the different features models over time.

("Diff Model") between evolution steps. However, this approach has certain limitations:

- *Scalability in terms of evolution steps:* Model comparison provides only usable results for a small number of models. Comparing a large number of different models usually results in very complex Diff Models.
- *Scalability in terms of features:* When dealing with evolution of large and complex feature models it is desirable to support a suitable level of abstraction. However, Diff Models do not provide such an abstraction since they consider models only on the lowest level of detail.
- *No support for planning:* Model comparison is suitable for a set of given models but not intended for planning and managing future evolution. Proactive evolution is not supported by model comparison tools.

We therefore propose a new way of dealing with proactive evolution of feature models. Feature models themselves are used as an enabling technique to model evolution plans. Our approach consists of the following key elements:

(1) An *Evolution Feature Model* (EvoFM) defines *which* evolutionary changes are planned. An EvoFM can be seen as a product line model (where the different versions of the feature models, which can be instantiated using EvoFM are the "products"). Selecting a "feature" from EvoFM means to apply that particular change to the original feature model.

(2) An *Evolution Plan* describes *when* the changes are applied. An Evolution Plan gives a compact overview of the evolution path and describes it as a sequence of configurations of EvoFM. An example of such evolution plan is depicted in Figure 3(a). Evolution plans can be seen as product maps, where each product of the product line is a feature model at a certain point in time.

(3) *Feature Model Fragments* represent detailed feature model elements on product line level which are associated with features in EvoFM. One could say, that they describe *how* the changes are applied. Figure 3(b) shows how the feature model from the example in 1 can be clustered into fragments. Each fragment corresponds to an EvoFM feature (annotations in Figure 3(b)).

3. Modelling Evolution Plans with EvoFM

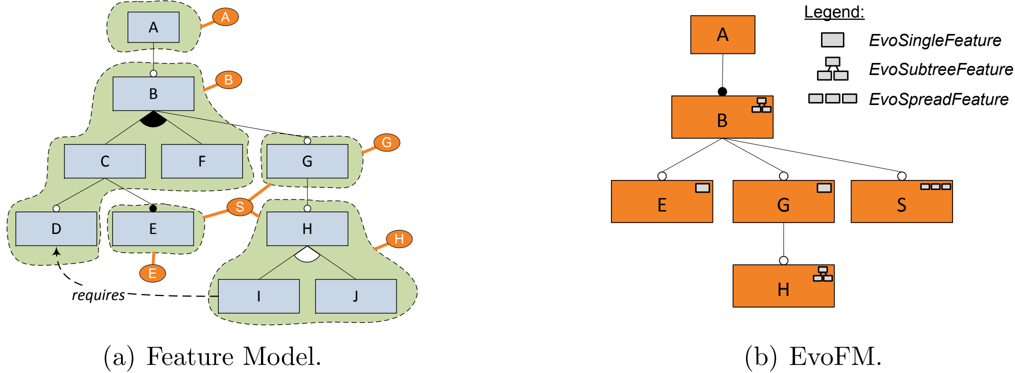
This section describes the different EvoFM elements and their semantics in detail. EvoFM is a special type of FODA-based [6] feature models. More precisely, the configuration of an EvoFM results in a concrete feature model, planned (or existing) along the evolution path. To differentiate the special EvoFM elements from conventional feature models, we name them with the prefix *Evo* (e.g., *EvoFeature*).

To keep an EvoFM model as compact and precise as possible, we make use of techniques such as *abstraction* (details in a feature model are abstracted away in a EvoFM model), *mappings* (EvoFM elements are mapped to features in a conventional feature model), *Delta Models* (set of EvoFM modification steps, e.g., add, remove feature), and *Change Operators* (special features in EvoFM describing how a feature model should change).

3.1. Abstraction

The purpose of EvoFM is to focus on the commonalities and differences between FMs at the different evolution steps. Depending on the kind of change, one single feature, one subtree, or a group of features (across the whole tree) may be involved in the evolution scenario. EvoFM groups these changes into single EvoFeatures.

1. Whenever only one feature is involved in an evolution step (e.g., "feature x will be removed next year"), that single feature is mapped to one single feature in EvoFM. In the further discussion, we call this an



(a) Feature Model. (b) EvoFM.

Figure 4: EvoFeatures can represent single features, subtrees, or spread features.

EvoSingleFeature, which is a feature in EvoFM that maps to a single feature on FM level (details of mappings will be discussed in the next subsection). For instance, consider the features E and G in the example in Figure 4(a).

2. Whenever, a whole subtree is involved in an evolution step, we map the subtree to an *EvoSubtreeFeature* in EvoFM. Thereby, the leaf features of the represented subtree might be leaf nodes in FM (like H in Figure 4) but can also have children, like B, i.e., they may reside at any location in the FMs. Hence, an EvoSubtreeFeature can be understood as a composite node, as used in other tree-based DSLs, e.g., as used in 3D Scene Graphs [10]. Using EvoSubtreeFeatures, one can abstract whole subtrees whose inner structure remains stable during evolution and is irrelevant for the evolution planning. Selecting or deselecting an EvoSubtreeFeature in a configuration corresponds to adding or removing the whole associated subtree to or from the FM, respectively.
3. Finally, there can be situations where, for the purpose of evolution planning, one wants to describe a change, which is associated with features spread all over the FM. An example is S (Figure 4) which is associated with different features spread all over the FM, here E, G, and H. We call such a kind of EvoFeature *EvoSpreadFeature*. An EvoSpreadFeature is not associated with a mapping as then EvoFM would no longer reflect the structure of the feature models. Thus, EvoSpreadFeatures are associated with constraints over EvoFeatures instead. In the example, S is associated with the constraint $E \wedge G \wedge H$.

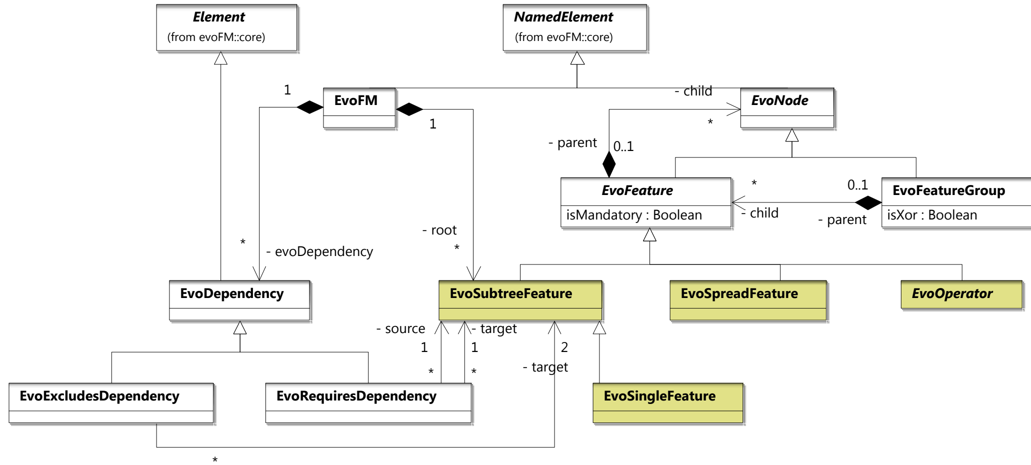


Figure 5: Metamodel for EvoFM models.

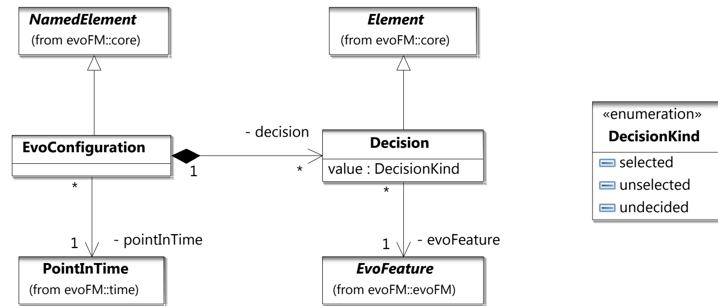


Figure 6: Metamodel for EvoFM configurations.

Consequently, E, G, and H are required by S in the example (which could be expressed explicitly by additional cross-tree constraints – as it is common in any other feature model) and by explicitly modelling such constraints EvoFM benefits from additional tool support (e.g., [11]) to ensure that all constraints are fulfilled during its configuration.

Figure 5 shows the metamodel for EvoFM models. It is structured analogously to the metamodel for feature models in Figure 2 and contains the different types of EvoFeatures (shown in colour). An additional kind of EvoFeatures are *EvoOperators* which are introduced later in Section 3.4.

Figure 5 shows the metamodel for configurations to be visualized in the Evolution Plan (see Figure 3(a)).

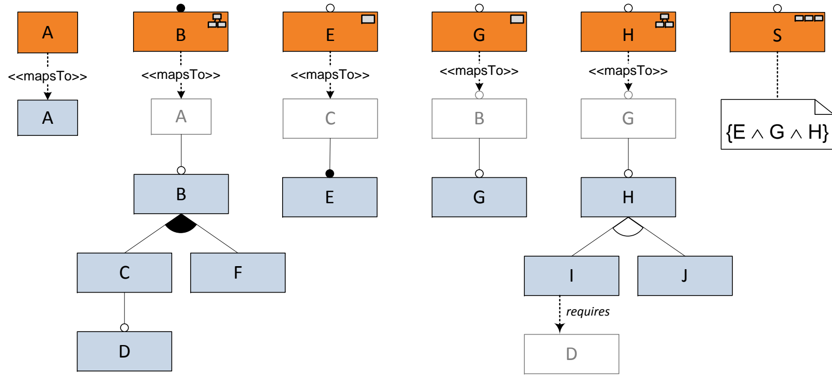


Figure 7: EvoFeatures and associated mappings from the example. Context nodes in fragments are shown in white colour. EvoSpreadNodes (like S) are mapped to constraints.

3.2. Fragments

As EvoFM is more abstract than the FM it does not contain all feature model elements. An EvoFeature describes only *that* a certain evolutionary change is applied (or not applied); it does not describe in detail *how* the change occurs. Such changes are described by the so called feature model fragments. EvoFeatures are mapped to FM fragments. Upon selection of an EvoFeature (e.g., during proactive planning, which is equivalent to product configuration based on EvoFM), the fragments are selected and composed to create the evolved feature model.

As we deal with feature models as target models, we can take advantage of their hierarchical structure to simplify the definition of fragments and the corresponding composition process: In EvoFM, each fragment is a subtree where its root node defines the context of the fragment (thus, called *context node*). This is either a feature or a feature group from the fragment associated with the parent of the fragment's EvoFeature. For instance, in Figure 4, EvoFeature B is parent of EvoFeature G. Thus, the context node of the fragment associated with G is one of the nodes from the fragment associated with B (in this case the feature B itself. Figure 7 shows the mappings for all EvoFeatures from the example above in Figure 4. As explained in Section 3.1, EvoSpreadFeatures are not mapped to a fragment but are associated with a constraint over other fragments instead, like S in the example.

To enable the composition of complete FMs from the defined fragments, each fragment needs to fully specify all FM model elements and their properties contained in the fragment, e.g., feature groups, cross-tree constraints,

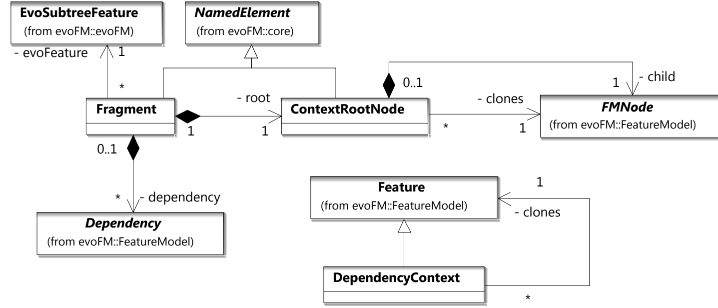


Figure 8: Metamodel for fragments.

and all properties like names and cardinalities.

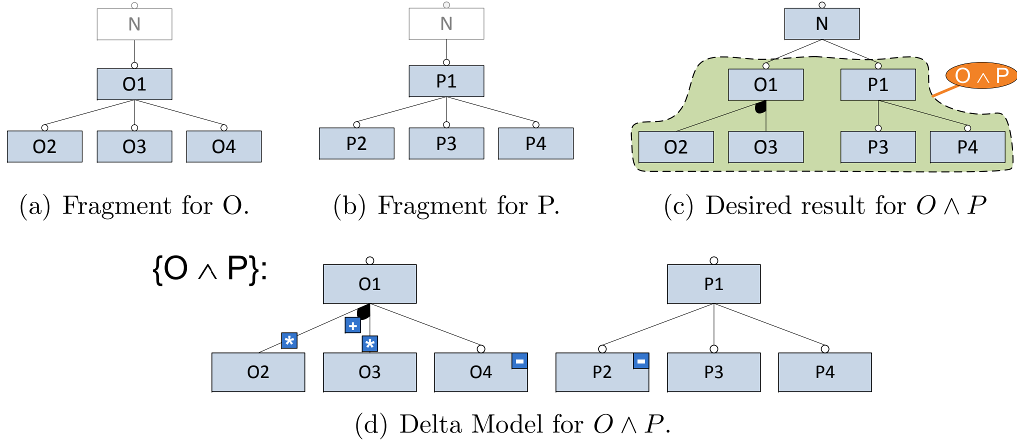
Figure 8 shows the metamodel for fragments. Each fragment is associated with an EvoFeature. A fragment owns a context root node which represents ("clones") a feature or feature group from another fragment. It is a model element of its own as the original feature or feature group in the other fragment remains unchanged. The context root node has at least one feature or feature group as children which can have further children of their own, as defined in the metamodel in Figure 2. If the fragment refers to an EvoSingleFeature then the context rood node has exactly one feature as child which has no further children.

A fragment can also contain dependencies which can either refer to other features in the same fragment or to features from other fragments. In the latter case, again a special context node (DependencyContext) is used which clones the feature from another fragment. In the example in Figure 4, the feature I has a requires relationship to D. Hence, D is used as DependencyContext in the fragment containing I (here for EvoFeature H, see Figure 7).

Using a DependencyContext to refer to other fragments implies a dependency between fragments which could be interpreted as a "requires" relationship between the corresponding EvoFeatures. In the above example the EvoFeature H requires the EvoFeature B. A tool has to keep track on these dependencies (using the same mechanisms like for conventional feature models, e.g. a SAT solver like in [11]) to ensure that only EvoFM configurations are created which lead to the derivation of valid feature models.

3.3. Delta Models

As discussed above, the fragments define the parts for composing an FM according to a given EvoFM configuration. However, sometimes certain con-



(a) Fragment for O. (b) Fragment for P. (c) Desired result for $O \wedge P$

(d) Delta Model for $O \wedge P$.

Figure 9: Example Delta Model to specify dependencies between EvoFeatures.

figurations requires additional changes in the FM, for instance to model "feature interactions" [12] between EvoFeatures.

We use the concept of *Delta Models* [13], (sometimes also called *Change Sets* [14]) to address these issues. A Delta Model defines a set of changes to a given model, here the FM. Possible changes are the basic operations *add*, *delete*, or *modify* on model elements. A Delta Model is associated with an *application condition* which is a constraint over EvoFeatures. A Delta Model is applied to all FMs whose associated EvoFM configuration fulfils the application condition.

A Delta Model can be visually specified by annotating the model elements to be modified with symbols representing the operations *add* (+), *delete* (-), or *modify* (*). If a model element is added or cannot be uniquely identified by its name (like feature groups which are usually anonymous in feature model diagrams) then a sufficient amount of context has to be denoted to specify the element's location in the model. Table 1 shows which operators can be applied to which model element in our approach, including the corresponding notation (i.e. the element to be annotated in the model and the context model elements which are at least required to identify the model element).

Figure 9(a) shows an non-trivial example: Let's consider an extract of an EvoFM with two EvoFeatures O, and P (Figure 9(a)). By default, O and P are mapped to a subtree with three leaf features each (O2, O3, O4 and P2, P3, P4 in Figure 9(b)). However, if both EvoFeatures O and P are selected at the same time, then O4 and P1 should be omitted in the feature model

Operator	Model Element	Notation	
		Annotated Element	**(Minimum) Context**
Add	Feature	Feature to add	Parent Feature
	FeatureGroup	Feature Group to add	Parent Feature, Child Feature(s)
	Dependency	Dependency to add	Source, Target
Remove	Feature	Feature to delete	-
	FeatureGroup	Feature Group to delete	Parent Feature, Child Feature(s)
	Dependency	Dependency to delete	Source, Target
Modify	Feature.isMandatory	Feature	
	Feature.parent	Relationship to new parent	Feature, new parent node
	FeatureGroup.isXor	Feature Group	Parent Feature, Child Feature(s)
	Feature.name	*Not supported (not useful, error-prone)*	
	FeatureGroup.name	*Not allowed (considered as deleting & adding a new Feature Group/Dependency)*	
	FeatureGroup.parent		
	Dependency.source/target		

Table 1: Operations and their notation in Delta Models.

(Figure 9(c)) and, moreover, O2 and O3 should be grouped into an or group. To describe these modifications, a Delta Model is defined for the application condition $O \wedge P$ which deletes O4 and P1, adds a new feature group, and modifies the parent-child relationships of O2 and O3 so that they become children of the new feature group (Figure 9(d)).

Delta Models can be conflicting when multiple Delta Models affect the same model element. To avoid such conflicts we define some rules: In general, add operations have to be executed first, then modifications, and then delete operations (to avoid conflicts like modifying an element which has not been created yet). Deleting a model element means that also dependent elements have to be deleted, i.e., all its child elements as well as cross-tree constraints which involve this model element. Other conflicts (like different modifications of the same model element) have to be solved manually by refining the granularity of Delta Models and their associated constraints [13], which is even easier in our case as Delta Models are used only occasionally to address feature interactions and not to construct the main structure of FMs. An additional solution would be to define additional dependencies between the Delta Models, similar to the approach suggested in [14].

3.4. Change Operators

Sometimes it is desirable to specify changes in a FM without adding or removing features or feature groups. For instance, a feature type is changed from optional to mandatory, or features are grouped into a feature group, or a dependency is added between features. If such changes are relevant on the abstraction level of the evolution plan it is necessary to explicitly model them as configurable elements in EvoFM. For instance, one might want to specify evolution plans such as *"in 2010 the navigation system requires a radio with display but in 2011, the navigation has a display of its own and thus no longer requires a radio display"* (see Figure 1). Such statements can be codified using change operators, which when "activated", change the feature model from 2009 according to the specified plan for 2010.

In general, change operators are used in the area of model co-development where they are used to specify changes, e.g., on a metamodel [15]. We adapt this concept to EvoFM. As we have discussed in previous work [5], it is useful to support not only "atomic' changes operators ("add", "delete", and "modify") but also complex operators frequently required during feature model evolution. For instance, a frequent change is inserting a new feature group to combine several features. While, basically, this could be described by a sequence of simple change operations (e.g., adding a feature group, and successively moving subfeatures to become its children) it seems more appropriate to model this with a single complex operator. We hence defined a catalogue of change operators for feature models [5].

A change operator is considered to be a special kind of feature in EvoFM, which can be configured (i.e., selected or eliminated) and represented as part of visualizations like the Evolution Plan in Figure 3(a). A change operator is specified in EvoFM as a child of a feature or feature group which acts as its context. More precisely, a change operator modifies the FM fragment associated with its parent node.

The supported types of modifications are basically the modification of cardinalities of features and features and moving features or feature groups within the feature models (which corresponds to the modifications in Figure 9). More complex modifications are inserting a feature group and its children (and the inverse operation). In [5] we presented an initial textual syntax to define such changes. During further work with EvoFM we simplified this syntax for change operators, see Table 2. In the Evolution Plan, it is denoted in angle brackets, like requires Radio in Figure 3.

Legend: purple=EvoFM keywords, *blue=features*

Change Operators for Features

Operator	Description
move to *g*	Moves the feature below g
to {mandatory | optional} [move to *g*]	Converts the feature into a mandatory/optional feature (and optionally moves it below g)
subfeatures [*f1, f2, ..., fn*] move to *g*	Moves all subfeatures (or optionally some selected subfeatures f1, f2, ..., fn) below g
subfeatures [*f1, f2, ..., fn*] to {mandatory | optional} [move to *g*]	Converts all subfeatures (or optionally some selected subfeatures f1, f2, ..., fn) into mandatory/optional subfeatures (and optionally moves them below g)
subfeatures [*f1, f2, ..., fn*] to {or group | xor group} [move to *g*]	Groups all subfeatures (or optionally some selected subfeatures f1, f2, ..., fn) into a new or-group/xor-group (and optionally moves the group below g)
requires *g*	Inserts "requires" dependency to g
excludes *g*	Insert "excludes" dependency to g

Change Operators for Feature Groups

Operator	Description
move to *g*	Moves the feature group below g
subfeatures to {or group | xor group} [move to *g*]	Converts feature group into an or-group/xor-group (and optionally moves the group below g)
subfeatures [*f1, f2, ..., fn*] to {or group | xor group} [move to *g*]	Groups subfeatures f1, f2, ..., fn into a new or-group/xor-group (and optionally moves the new group below g)
subfeatures to {mandatory | optional} [move to *g*]	Removes the feature group and converts their subfeatures into mandatory/optional subfeatures of the feature group's parent feature (or optionally moves them below g)
subfeatures [*f1, f2, ..., fn*] to {mandatory | optional} [move to *g*]	Converts subfeatures f1, f2, ..., fn to mandatory /optional subfeatures of the feature group's parent feature (or optionally moves them below g)

Table 2: Syntax and informal semantics of supported change operators.

4. Working with EvoFM

This section illustrates how to work with EvoFM based on the overview in Section 2 and the technical concepts in Section 3. The concepts presented so far have to be complemented by appropriate tool support. Thus, we first present basic concepts and an initial design for such tool support in Section 4.1. On that basis we show the intended practical usage of EvoFM (Section 4.2). Several prototypes for different single aspects of the tool already exist.

4.1. Tool Concepts

EvoFM is presented to the modeller in a Gantt visualisation (cf. Figure 10, *Evolution Plan View*), where each row corresponds to an EvoFeature and the corresponding fragment. Each column in the plan corresponds to an evolution step, showing the EvoFM configuration for this step. The modeller can edit configurations by marking EvoFeatures as *selected*, *eliminated*, or *undecided* and manage the evolution steps, e.g., by inserting a new step to extend the time period covered by the plan. The tool also allows the modeller to edit and extend the EvoFM, e.g., by inserting new EvoFeatures to enrich the plan with more evolution operations. Constraints between EvoFeatures and Delta Models referring to an EvoFeature are indicated by small icons,

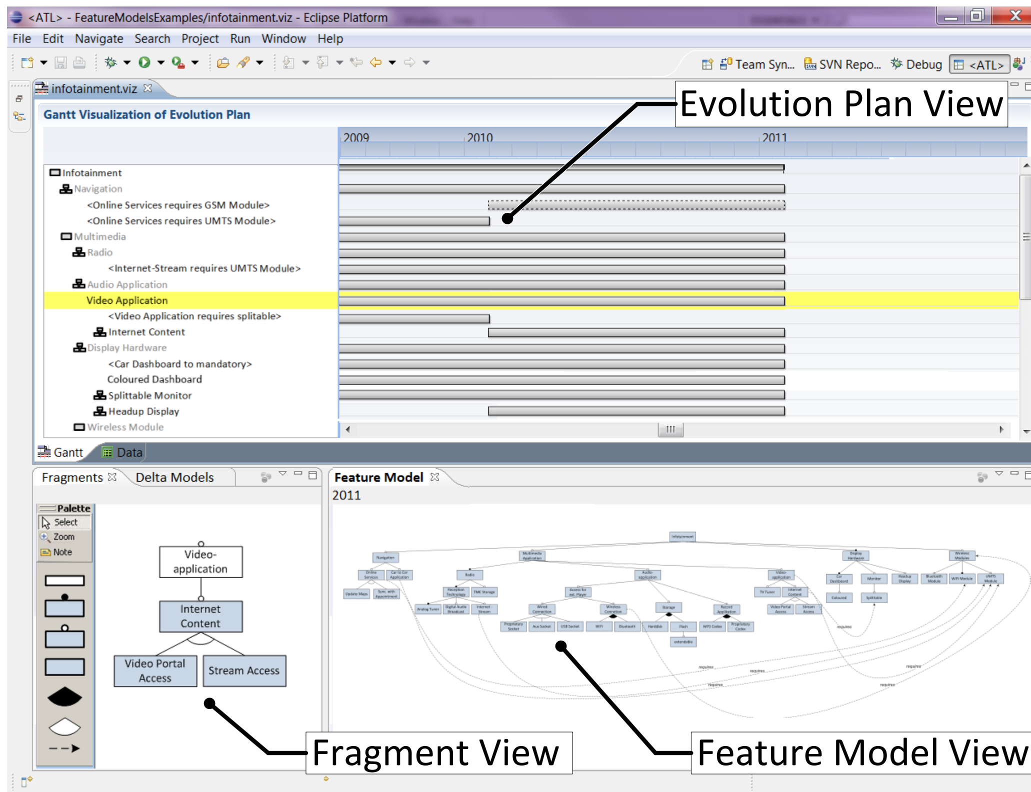


Figure 10: User Interface prototype of the EvoFM Tool.

which can be clicked to gather further information (e.g., to display the Delta Model in the corresponding view).

A visual feature model editor can be used to create and edit fragments (*Fragment View*). The root node for a fragment is found, e.g., by a string-based search or by selecting a node in the Feature Model View. Delta Models are created and edited by selecting existing models elements from the product line and marked with symbols for adding, removing, or modifying elements.

The *Feature Model View* shows the product line feature model resulting from the currently selected EvoFM configuration. As the feature models can become very large rather quickly, the feature model view should support appropriate interaction techniques based on visualisation principles (see, e.g., [16]).

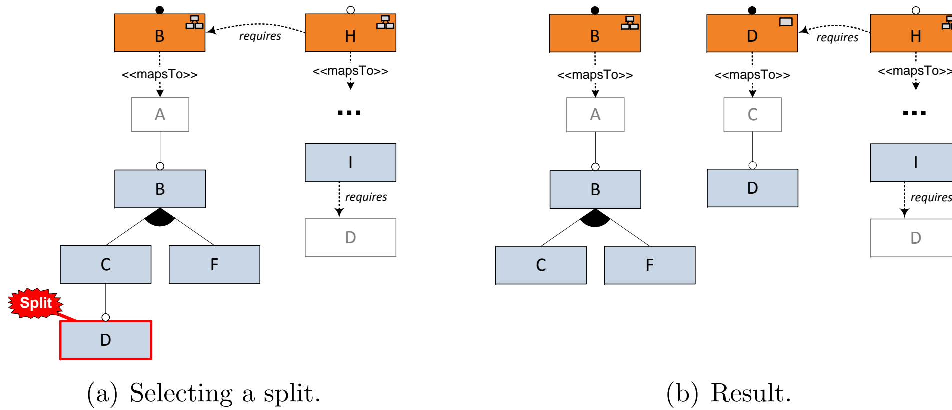
4.2. Typical Workflow

The initial data to be shown in the tool can be derived semi-automatically from a given set of feature models, i.e., already existing feature models from previous version of the product line. The basic idea for the automatic derivation is to perform a conventional model diff and to cluster subtrees that either remain stable or that are modified only together within the same evolution step into an EvoFeature (a simple implementation for deriving EvoFM from two given feature models is described in [4]).

Based on the existing evolution plan, the modeller plans the future evolution by creating new evolution steps. The evolution plan allows to iteratively specify single decisions by selecting or deselecting EvoFM nodes. For instance, if a manager decides that "*in 2013, we will no longer support Radio*" (see Figure 1), the modeller inserts a new evolution step 2013 in the timeline and deselects the EvoFeature Radio there. The tool then has to notify the modeller about potential consequences, if, e.g., other EvoFeatures depend on Radio.

When some features need to be added to the product line, which have not been considered so far (e.g., "*in 2014 the multimedia devices should be extended by an optional video screen with the optional split screen capability*"), the modeller creates a new EvoFeature (e.g., VideoScreen as child of MultimediaDevices) and defines the corresponding model fragment. The fragment is defined as discussed in Section 3.2. Here, e.g., the fragment could consist of a feature VideoScreen with an optional child SplitScreen and the context node MultimediaDevices. It should be mentioned, that fragments do not need to be created immediately when specifying the decision but can be added later by a technical domain expert.

Depending on the evolution plan, sometimes fragments need to be split. For instance, in the example from Figure 4, feature D is part of fragment B. To specify a decision like "*in 2014, D is no longer part of the product line*", one has to split the fragment B and put the feature D in a fragment of its own. A large part of this work can be performed by the tool: The modeller selects the subtree within the fragment B which should be put in a fragment of its own (here, just the feature D) and calls a command "split" (Figure 11(a)). The tool then creates a new EvoFeature (e.g., named D as well) which is in automatically selected in all existing configurations where EvoFeature B is selected so far. If other fragments or Delta Models contain references on D (like fragment H) then the resulting dependencies between EvoFeatures are updated automatically (i.e., EvoFeature H requires the new



(a) Selecting a split.

(b) Result.

Figure 11: Splitting fragments.

EvoFeature D now, (Figure 11(a))). Finally, in the example case from above, the modeller deselects the new EvoFeature D in the configuration for 2014.

Structural changes, like changing a mandatory feature in the product line to optional, requires the insertion of a new Change Operator. The Change Operator is added to EvoFM as a child of an EvoFeature. Its semantics is indicated by a textual label as described in Section 3.4. At any point during the work with EvoFM, the modeller can select a configuration for an evolution step and inspect the resulting feature model. Finally, when a configuration is fully specified (including all the fragments for the features selected), the resulting feature model can be automatically created and, e.g., be exported to other tools.

4.3. Limitations

In this section we discuss the degree of abstraction EvoFM provides and its limitations. Basically, EvoFM provides abstraction from the details of changes in the feature model over the time by clustering them into fragments. As a fragment can be any kind of subtree below a selected context node, all features which are connected by a parent child-relationship (including siblings below the context node) can be clustered into a fragment. In turn, the number of fragments (and thus the complexity of EvoFM) increases, the less features can be clustered together. Moreover, fragments have to be subdivided further if fragments for different evolution steps intersect each other. The number of required EvoFeatures (aside from change operators) results from the number of features/feature groups changed over all evolution

steps minus the number of features which can be clustered together. More precisely:

Let $N = \{n_1, \ldots, n_i\}$ be a set of feature model nodes (features or feature groups) and $R \subseteq N \times N$ be the set of parent-child relationships between them. Furthermore, let $\Delta_t = (N_t, R_t)$ be a tuple of all nodes changed at evolution step t (i.e. added to or removed from the product line at this evolution step) and all relationships between them (including added relationships and existing relationships; if any).

For a sequence $\Delta_{t_1}, \Delta_{t_2}, \ldots, \Delta_{t_k}$ for different evolution steps $t_1, t_2, \ldots, t_k$ the number of required EvoFeatures can be calculated as follows:

$$\text{Number of EvoFeatures} = |\bigcup_{l=t_1}^{t_k} N_l| - |\bigcap_{l=t_1}^{t_k} R_l|$$

The worst case occurs when, over the time, every node on product line level becomes subject to a change (i.e. $|\bigcup_{l=t_1}^{t_k} N_l|$ is equal to the number of nodes) and changes over the time overlap so that no subtrees can be clustered any more (i.e., $|\bigcap_{l=t_1}^{t_k} R_l| = 0$). In this case, the number of EvoFeatures is equal to the number of nodes on product line level which results in a very low degree of abstraction (however, EvoFM then still provides a benefit by enabling to show the different changes over the time in terms of an evolution plan). In addition, change operators have to be considered as well, so that, in theory, EvoFM can contain more EvoFeatures than the feature model on product line level over the time. Nevertheless, in most practical settings there will be at least a set of stable core product line features so that the worst case scenario will not occur.

A second issue is that after a large number of evolution steps a significant part of EvoFeatures might become deprecated, because of features which were only relevant in earlier evolution steps but do not play any role in the current and future product line any more. Keeping these EvoFeatures results in unnecessary complexity of EvoFM while getting rid of them results in loss of information about earlier product line versions. A possible solution to address this issue is to allow the user selecting the time frame for which to display EvoFM while storing all information about previous versions in the background.

5. Related Work

Product line engineering should treat evolution as the normal case and not as the exception [17]. Despite this importance surprisingly few approaches deal with product line evolution (e.g., [18? , 19]). Evolution support is of particular importance in model-driven PLE, e.g., to ensure consistency after changes to meta-models, models, and other artefacts. Several authors [20, 21] stress the importance of approaches for product line evolution to avoid the erosion of a product line. Some existing work deals with implementation issues for evolving product lines like [22, 23], e.g., using aspect-oriented programming. Mende et al. [24] describe tool-support for the evolution of product lines based on the "grow-and-prune" model, i.e., they support refactoring code that has been created by copy & paste and which can potentially be propagated to product line level. Svahnberg and Bosch [19] report on experiences regarding the evolution of products, components, and software architecture.

Deelstra et al. [25] focus on product derivation. Besides other aspects they discuss different types of adaptations of product lines (product specific adaptation, reactive evolution, proactive evolution) and the scope of adjustment (e.g., adding variants or variation points). According to this, our approach addresses proactive evolution, i.e., active planning of future versions on domain level.

Evolution of models in general is addressed in the area of "model co-evolution". As shown in [15], the most common way to provide semantically rich descriptions of changes in a model is the usage of a set of change operators. For instance [26, 27, 28] provide large sets of complex operators for changes in metamodels.

Some work focusses on changes between feature models. Thüm et al. [29] present a tool to classify whether some given changes on a feature model result in a specialization (enlarging the set of products), a generalization (reducing the set of products), or a refactoring of the model, or in an arbitrary change. Other work provides change operators for different purposes, like feature model generalization [30] or feature model specialisation [7]. Follow-up work in [31] also discusses feature model changes for co-evolution, which includes addition and deletion of nodes, moving nodes or subtrees, and changes of cardinalities. As we discussed in [5], existing sets of change operators are not sufficient in the context of proactive evolution of product lines. Thus, we presented catalogue of evolution operators in [5] which was the foundation

for Section 3.4.

Another concept which we reuse and adapt in our work are model fragments and Delta Models. Model fragments (or "model snippets" [32]) can be used in model-driven product line engineering to implement "positive variability", i.e., the composition of a product according to a given feature model configuration [33]. The composition can then be achieved using techniques for model merging [9] or model weaving [34]. In [17], fragments of variability models are used to allow multiple teams parallel evolution of variability models.

Delta Models can be used to specify changes on a given model. Hendrickson and van der Hoek [14] describe a tool for specifying delta models (which they call "change sets") and relationships between them to describe the variability in product line architectures. Schaefer [13] uses Delta Models for incremental model refinement and provides a formalization of the Delta Models.

However, none of the existing work addresses product line evolution in terms of systematic proactive planning. We presented first ideas towards feature-oriented modelling of product line evolution, including an overall conceptual framework and three application scenarios, in earlier work [4]. A catalogue of Change Operators for Feature Models has been presented in [5]. The current report integrates these results and presents a concrete realization, including the resulting metamodels.

6. Summary and Outlook

EvoFM has considerable potential to support proactive evolution of feature models. By using feature modelling technique to model the evolution plan, we have achieved an elegant way of dealing with changes. EvoFM leverages the capabilities of feature models to model commonalities and variability between a large number of assets on an appropriate level of abstraction. Planning is supported by step-wise specification of feature configurations. Also, product line developers are already familiar with feature models, which is an additional advantage.

In summary, an EvoFM abstracts from the details of concrete product line feature models at the different evolution steps by focusing on commonality and variability. The concrete product line feature model elements associated with an EvoFeature are defined in fragments. Thus, the constraints which are

relevant to the evolution can be expressed as dependencies between EvoFeatures itself, while irrelevant details can be hidden in the fragments, which reduces complexity for the modeller. Delta Models and change operators can be used to further adapt the models composed of fragments. Delta Models are used to specify changes on the fragments for specific EvoFM configurations, analogous to feature interactions in conventional product lines. In contrast, change operators specify changes on fragments in terms of the evolution and thus act as a special kind of EvoFeatures, which can be explicitly be applied (depending on the configuration of EvoFM) to evolution steps.

We are currently working on a more comprehensive and complete tool suite for proactive evolution of feature models. Here, feedback from industry partners has helped us a lot in understanding the practical requirements for such tool support.

In this paper, we focused on the product line's feature model only. Future work will include the extension of the approach towards other parts of the product line (e.g., the implementation models associated with the feature model), such that (1) for each evolution step a whole product line can be derived from an EvoFM configuration and (2) the evolution plan can be analysed in more detail, e.g., to discover hidden dependencies or inconsistencies, which can only be detected by taking the implementation into account.

7. Acknowledgements

This work was supported, in part, by Science Foundation Ireland grant 03/CE2/I303_1 to Lero – the Irish Software Engineering Research Centre, http://www.lero.ie/.

References

[1] J. Bosch, Maturity and evolution in software product lines: Approaches, artefacts, and organization, in: G. Chastek (Ed.), Proceedings of the Second Software Product Line Conference, LNCS 2379, Springer, San Diego, CA, 2002, pp. 257–271.

[2] Y. Chen, A process-centric approach for software product line evolution management, in: Proceedings of the First International Software Product Lines Young Researchers Workshop (SPLYR 2004), 2004, pp. 9–18.

[3] S. R. Schach, A. Tomer, Development/maintenance/reuse: Software evolution in product lines, in: P. Donohoe (Ed.), Proceedings of the First Software Product Line Conference, Kluwer Academic Publishers, 2000, pp. 437–450.

[4] G. Botterweck, A. Pleuss, A. Polzer, S. Kowalewski, Towards feature-driven planning of product-line evolution, in: 1st International Workshop on Feature-oriented Software Development (FOSD 2009), collocated with MODELS, Denver, CO, USA, 2009.

[5] G. Botterweck, A. Pleuss, D. Dhungana, A. Polzer, S. Kowalewski, Evofm: Feature-driven planning of product-line evolution, in: 1st International Workshop on Product Line Approaches in Software Engineering (PLEASE 2010) collocated with the 32nd International Conference on Software Engineering (ICSE 2010), Cape Town, South Africa, 2010, http://doi.acm.org/10.1145/1808937.1808941. doi:http://doi.acm.org/10.1145/1808937.1808941.

[6] K. Kang, S. Cohen, J. Hess, W. Novak, S. Peterson, Feature oriented domain analysis (FODA) feasibility study, SEI Technical Report CMU/SEI-90-TR-21, ADA 235785, Software Engineering Institute (1990).

[7] K. Czarnecki, S. Helsen, U. Eisenecker, Formalizing cardinality-based feature models and their specialization, Software Process Improvement and Practice 10 (1) (2005) 7–29.

[8] Eclipse-Foundation, Emf compare, Website, http://www.eclipse.org/modeling/emft/?project=compare.
URL http://www.eclipse.org/modeling/emft/?project=compare

[9] Eclipse-Foundation, Epsilon project, Website, http://www.eclipse.org/gmt/epsilon/.
URL http://www.eclipse.org/gmt/epsilon/

[10] A. Vitzthum, A. Pleuß, Ssiml: designing structure and application integration of 3d scenes, in: N. W. John, S. Ressler, L. Chittaro, D. A. Duce (Eds.), Web3D, ACM, 2005, pp. 9–17.

[11] G. Botterweck, D. Schneeweiss, A. Pleuss, Interactive techniques to support the configuration of complex feature models, in: 1st International

Workshop on Model-Driven Product Line Engineering (MDPLE 2009), held in conjunction with ECMDA 2009, Twente, The Netherlands, 2009.
URL https://feasiple.de/public/proceedings-mdple2009.pdf

[12] K. Lee, G. Botterweck, S. Thiel, Aspectual separation of feature dependencies for flexible feature composition, in: 33rd Annual IEEE International Computer Software and Applications Conference (COMPSAC 2009), Seattle, WA, 2009.
URL http://dx.doi.org/10.1109/COMPSAC.2009.16

[13] I. Schaefer, Variability modelling for model-driven development of software product lines, in: Proceedings of the 4th International Workshop on Variability Modelling of Software-Intensive Systems (VAMOS 2010), Linz, Austria, 2010.
URL http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf

[14] S. A. Hendrickson, A. van der Hoek, Modeling product line architectures through change sets and relationships, in: ICSE '07: Proceedings of the 29th international conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, 2007, pp. 189–198. doi:http://dx.doi.org/10.1109/ICSE.2007.56.

[15] L. M. Rose, R. F. Paige, D. S. Kolovos, F. A. Polack, An analysis of approaches to model migration, in: Joint MODELS 2009 Workshop on Model-Driven Software Evolution (MoDSE) and Model Co-Evolution and Consistency Management (MCCM), 2009.

[16] G. Botterweck, S. Thiel, D. Nestor, S. bin Abid, C. Cawley, Visual tool support for configuring and understanding software product lines, in: 12th International Software Product Line Conference (SPLC 2008), Limerick, Ireland, 2008, iSBN 978-7695-3303-2.
URL http://dx.doi.org/10.1109/SPLC.2008.32

[17] D. Dhungana, T. Neumayer, P. Grünbacher, R. Rabiser, Supporting evolution in model-based product line engineering, in: SPLC, 2008, pp. 319–328.

[18] J. Bosch, Design and Use of Software Architectures, Addison-Wesley, 2000.

[19] M. Svahnberg, J. Bosch, Evolution in software product lines: two cases, Journal of Software Maintenance: Research and Practice 11 (6) (1999) 391–422.

[20] S. Johnsson, J. Bosch, Quantifying software product line ageing, in: P. Knauber, G. Succi (Eds.), 1st ICSE Workshop on Software Product Lines: Economics, Architectures, and Implications, Limerick, Ireland, 2000, pp. 27–32.

[21] H. Siy, D. Perry, Challenges in evolving a large scale software product, in: Principles of Software Evolution Workshop. Collocated with International Software Engineering Conference (ICSE98), Kyoto Japan, 1998, pp. 29–32.

[22] N. Loughran, A. Rashid, Framed aspects: Supporting variability and configurability for aop, in: ICSR, Vol. 3107 of Lecture Notes in Computer Science, Springer, 2004, pp. 127–140.

[23] G. Deng, G. Lenz, D. C. Schmidt, Addressing domain evolution challenges in software product lines, in: Workshop MDD for Product-Lines, Satellite Events at the MoDELS 2005 Conference, 2005. doi:10.1.1.59.6707.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6707

[24] T. Mende, F. Beckwermert, R. Koschke, G. Meier, Supporting the grow-and-prune model in software product lines evolution using clone detection, in: CSMR '08: Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering, IEEE Computer Society, Washington, DC, USA, 2008, pp. 163–172. doi:http://dx.doi.org/10.1109/CSMR.2008.4493311.

[25] S. Deelstra, M. Sinnema, J. Bosch, Product derivation in software product families: a case study, The Journal of Systems and Software 74 (2005) 173–194.
URL http://www.msinnema.nl/journalExperiencesInSPF.pdf

[26] B. S. Lerner, A model for compound type changes encountered in schema evolution, ACM Trans. Database Syst. 25 (1) (2000) 83–127. doi:http://doi.acm.org/10.1145/352958.352983.

[27] G. Wachsmuth, Metamodel adaptation and model co-adaptation, in: E. Ernst (Ed.), ECOOP, Vol. 4609 of Lecture Notes in Computer Science, Springer, 2007, pp. 600–624.

[28] M. Herrmannsdoerfer, S. Benz, E. Juergens, Cope - automating coupled evolution of metamodels and models, in: Genoa: Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 52–76. doi:http://dx.doi.org/10.1007/978-3-642-03013-0_4.

[29] T. Thüm, D. Batory, C. Kästner, Reasoning about edits to feature models, in: ICSE '09: Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, IEEE Computer Society, Washington, DC, USA, 2009, pp. 254–264. doi:http://dx.doi.org/10.1109/ICSE.2009.5070526.

[30] V. Alves, R. Gheyi, T. Massoni, Refactoring product lines, in: GPCE'06, 2006.
URL http://twiki.cin.ufpe.br/twiki/pub/SPG/AspectProductLine/gpce40-alves.pdf

[31] C. H. P. Kim, K. Czarnecki, Synchronizing cardinality-based feature models and their specializations, in: A. Hartman, D. Kreische (Eds.), ECMDA-FA, Vol. 3748 of Lecture Notes in Computer Science, Springer, 2005, pp. 331–348.

[32] R. Ramos, O. Barais, J.-M. Jézéquel, Matching model-snippets, in: G. Engels, B. Opdyke, D. C. Schmidt, F. Weil (Eds.), MoDELS, Vol. 4735 of Lecture Notes in Computer Science, Springer, 2007, pp. 121–135.

[33] M. Voelter, I. Groher, Product line implementation using aspect-oriented and model-driven software development, in: 11th International Software Product Line Conference (SPLC 2007), Kyoto, Japan, 2007.
URL http://www.voelter.de/data/pub/VoelterGroher_SPLEwithAOandMDD.pdf

[34] Eclipse-Foundation, Atlas model weaver, Website, http://www.eclipse.org/gmt/amw/.
URL http://www.eclipse.org/gmt/amw/