

# EvoFM: Feature-driven Planning of Product-line Evolution

Goetz Botterweck  
Lero – The Irish Software  
Engineering Research Centre  
University of Limerick  
Limerick, Ireland  
goetz.botterweck@lero.ie

Andreas Pleuss  
Lero – The Irish Software  
Engineering Research Centre  
University of Limerick  
Limerick, Ireland  
andreas.pleuss@lero.ie

Deepak Dhungana  
Lero – The Irish Software  
Engineering Research Centre  
University of Limerick  
Limerick, Ireland  
deepak.dhungana@lero.ie

Andreas Polzer  
Embedded Software  
Laboratory  
RWTH Aachen  
Aachen, Germany  
polzer@cs.rwth-  
aachen.de

Stefan Kowalewski  
Embedded Software  
Laboratory  
RWTH Aachen  
Aachen, Germany  
kowalewski@cs.rwth-  
aachen.de

## ABSTRACT

Companies successfully applying product line approaches often follow a long-term strategy and need to plan product portfolios years ahead. For instance in the automotive industry, managers constantly make decisions about future product evolution, like “the LED tail lights will be introduced with the next facelift and the LED front lights two years later”. With a raising number of features, feature changes, and evolution steps, a systematic approach for evolution planning becomes essential. However, there is only very little support for such evolution in model-based product line engineering so far.

This paper presents an approach for extending model-driven product line engineering towards automated and tool-supported techniques for product line evolution. We provide a feature-based approach to model the variability over time and a catalogue of change operators for feature models.

## Categories and Subject Descriptors

D.2.13 [Software Engineering]: Reusable Software; D.2.2 [Software Engineering]: Design Tools and Techniques

## General Terms

Design, Algorithms, Management

## Keywords

Model-driven product line engineering, software evolution, feature modelling

## 1. INTRODUCTION

Feature-oriented modelling [7] can help to reduce complexity by clustering concepts into larger chunks that have a meaning for some stakeholder. In Software Product Lines (SPL), feature models can be used to describe the capabilities and configuration options of product lines, thereby defining the range of products.

Companies that successfully apply product lines in industry [13] often take a strategic perspective and practice long-term, feature-oriented planning of changes. For instance, in the automotive industry it is common to hear statements like “In 2 years we introduce the new safety concept with automatic distance control and emergency brake assistant. Hence, next year we need the new distance sensor.” Usually, a large amount of change requests is raised over time and all of them have to be integrated into the existing product line in a consistent and coordinated way.

Despite this long-term planning and the evolution of product lines in industry practice, so far there is little support for such practices in product line approaches. In particular, the authors would argue that *model-driven* product line approaches could gain and provide additional benefits if they would integrate a long-term planning of evolution.

In this paper, we address these challenges with an approach for feature-oriented modelling of product line evolution, based on first ideas described in [3]. We develop these concepts by discussing a sample product line of evolving automotive parking assistants.

Based on the example, we provide (1) an conceptual framework including modelling languages for product line evolution and (2) a catalogue of evolution operators for feature models. The presented initial approach provides foundations for many usage scenarios and extensions, like review of former evolution, planning of future evolution, analysis of evolution plans, and consistency checking (see [3]).

## 2. RESEARCH PROBLEM

For companies in industry it is often necessary to plan their product portfolios over years. This is especially true

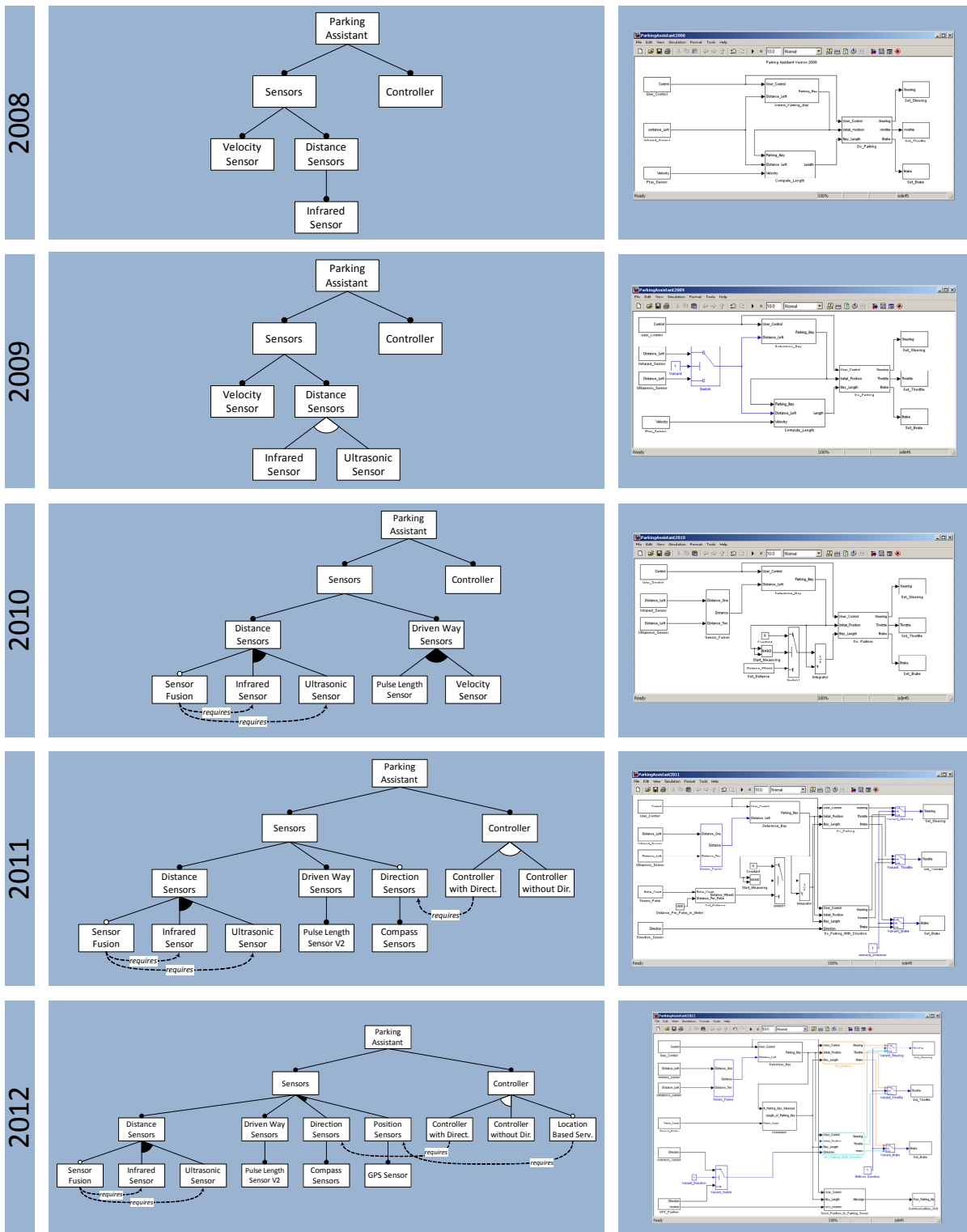


Figure 1: Evolution of the sample product line.

when introducing new features into families of similar products. Here we focus on the integration of (1) such evolution and long-term planning with (2) model-driven product line engineering. In particular, we aim to extend existing *automated* and *tool-supported* approaches for model-driven PLE

to include techniques for product line evolution and portfolio planning. To facilitate such automated and tool-supported techniques we require concepts and languages that are expressive and precise enough to be processed by automatic or semi-automatic tools.

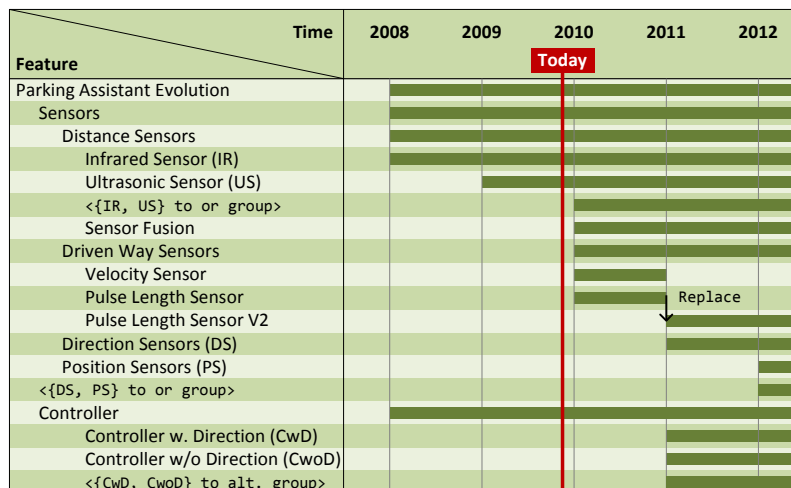


Figure 2: Evolution plan visualising the evolution steps across time.

The research presented here is motivated by requirements of the automotive industry. Hence, we focus on product lines of embedded systems, in particular control systems in an automotive environment. We assume that such systems are created with approaches from Model-based Engineering of Embedded Systems and the implementation is described in a domain-specific (modelling) language for embedded systems, such as Simulink.

As a sample case for product line evolution we use a product line of driving assistant applications, the “Parking Assistant Product Line”, see Figure 1. This product line was implemented in our group by using Simulink models (right hand side of Figure 1). Instances of the product line can be executed and evaluated on relatively complex model cars, that are built to a 1:5 scale and support rapid prototyping of such applications [11, 4].

In our sample case the Parking Assistant product line evolves and future evolution needs to planned several years ahead. For instance, in 2008 it uses only a simple *Infrared Sensor* (IR) for measuring distances. In 2009 this is augmented with the alternative to use an *Ultrasonic Sensor* (US). For 2010, we plan to introduce the choice to use IR, US or both. In the latter case, there is the option to use a “Sensor Fusion” component to combine the measurements by the sensors to improve the range and accuracy of the measured distance. Also *Driven-Way Sensors* will be introduced. In 2011 we will introduce a *Compass*. This implies the option to use a special *Controller* that uses the direction information to perform more precise parking. In 2012 the system will be extended to include *Position Sensors* and *Location Based Services*, e.g., to report detected potential parking spaces to a central service.

Since we are aiming for a *model-driven* approach, we aim to model this evolution in a precise and tool-processable way and integrate that with the existing approaches.

### 3. MODELLING PRODUCT LINE EVOLUTION

In this section we present our approach to feature-oriented modelling of product line evolution. From the viewpoint of a model-driven product line engineering, the evolution of

a (feature model-based) product line can be considered as a sequence of feature models (left column in Figure 1). While the differences between a very small number of models can be visualized using model comparison approaches (like *EMF Compare*<sup>1</sup>), this becomes insufficient for greater numbers of models. Moreover, model comparison approaches do neither provide a sufficient level of abstraction nor are they intended for planning purposes. Thus, we propose to take a different approach by focussing on the commonalities and variability between the models over the time. For this purpose we use a special kind of feature models itself for modelling the product line evolution, which we call *Evolution Feature Model (EvoFM)*. A feature model of the product line at a specific evolution step corresponds to specific configuration of EvoFM. Consequently, a visualization of the different EvoFM configurations over the time (called *Evolution Plan*) provides a very compact and intuitive overview on the product line evolution.

In this section, we first present the Evolution Plan to illustrate the benefits of the approach. Then, we discuss the underlying formalism EvoFM and some of its technical details.

#### 3.1 Evolution Plan

The EvoFM approach [3] can best be introduced by first discussing the *Evolution Plan*, see Figure 2. It provides an overview of the planned evolution of the Parking Assistant product line. The graphical representation is similar to a Gantt chart and inspired by “roadmapping” techniques from software visualisation [10].

The plan corresponds to the sequence of models shown earlier in Figure 1. However, here we abstract details, focussing on the most relevant evolution events. These include the introduction and removal of features, e.g., in 2009 an Ultrasonic Sensor is introduced and continues to be available throughout the following years. The beginning of a bar indicates an **add feature** operation, the end of a bar indicates a **remove feature** operation. Other operators include converting the optional subfeatures of Distance Sensors (Infrared Sensor and Ultrasonic Sensor) into an alterna-

<sup>1</sup>[http://wiki.eclipse.org/index.php/EMF\\_Compare](http://wiki.eclipse.org/index.php/EMF_Compare)

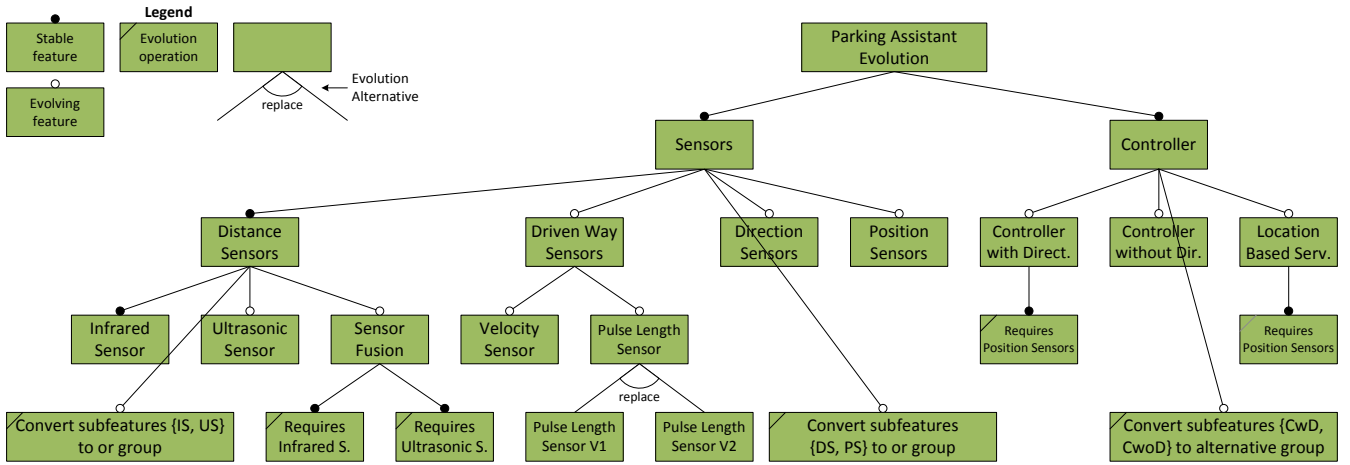


Figure 3: Evolution Feature Model (EvoFM) for the evolution 2009 to 2012.

tive group in 2009, converting that group into an or group in 2010, or replacing the Pulse Length Sensor with an improved version in 2011.

The Evolution Plan provides a very compact and intuitive representation of the evolution of a product line. Technically, it is a visualization of the configurations of the EvoFM features (left column in Figure 2) which model the variability of the product line over the time. In the next section we discuss EvoFM and its design decisions.

### 3.2 EvoFM

The basic idea of our approach is to model the evolution of product lines by focussing on commonalities and variability between the feature models *over time* and describing these variabilities with a special form of feature model. We call this evolution-oriented feature model “EvoFM” (shown in all figures in green color), whereas the ordinary feature model of the product line are referred to as just “feature model” or FM (shown in all figures in blue).

An example for EvoFM corresponding to the earlier sample product line and evolution plan is shown in Figure 3. It is important to note that EvoFM does *not* express the variability within the product line. It rather describes the options for evolution changes. If a feature in EvoFM is specified as mandatory, this means that it has always to be part of the product line feature model. However, this makes no statement about the feature’s constraints in the product line, i.e., whether it is, e.g., mandatory, optional or part of a feature group (this is defined by mappings and will be discussed later, see Section 3.2.2).

An optional feature in EvoFM represent varying structures in the FM. For instance, the feature *Ultrasonic Sensor* is not included in 2008 but present from 2009 to 2012. Thus, *Ultrasonic Sensor* is represented in EvoFM by an optional feature. Other features, which remain stable in the SPL over time, are represented in EvoFM by mandatory features. However, to simplify the model mandatory (sub-) features in EvoFM are abstracted away whenever possible. The more advanced operators shown in EvoFM (“requires”, “convert to ...”) will be discussed later in Section 3.2.3.

Each configuration of EvoFM corresponds to one evolution step, i.e., one row in Figure 1 and one year in the Evolution Plan in Figure 2.

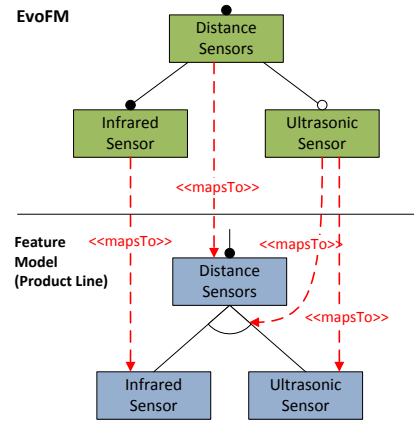


Figure 4: Mappings between EvoFM and the feature model from the product line (extract).

#### 3.2.1 Abstraction levels in EvoFM

When modelling the EvoFM we can chose different abstraction levels, see Figure 5. The bottom row in Figure 5 shows an example feature model evolution. The simplest option would be to take all details of each feature model into account (“No Abstraction”, second row in Fig. 5). The resulting EvoFM would then include all features from the product line, either as mandatory features (all the product line features which are always included in the product line) or as optional features (product line features which are subject to evolution). A second option is to abstract from subtrees which remain stable over the time by presenting them just by their root node (“Abstraction by Subtrees” in Fig. 5). For instance, all subfeatures of *B* remain stable over the time and thus need not to be part of EvoFM. Both options (no abstraction and abstraction of subtrees) can be described by formal rules. Thus, they both allow to automatically compute an EvoFM from a given sequence of feature models [3].

The highest level of abstraction is achieved by allowing the modeller to specify additional manual abstractions using domain-specific knowledge. For instance, in the top row

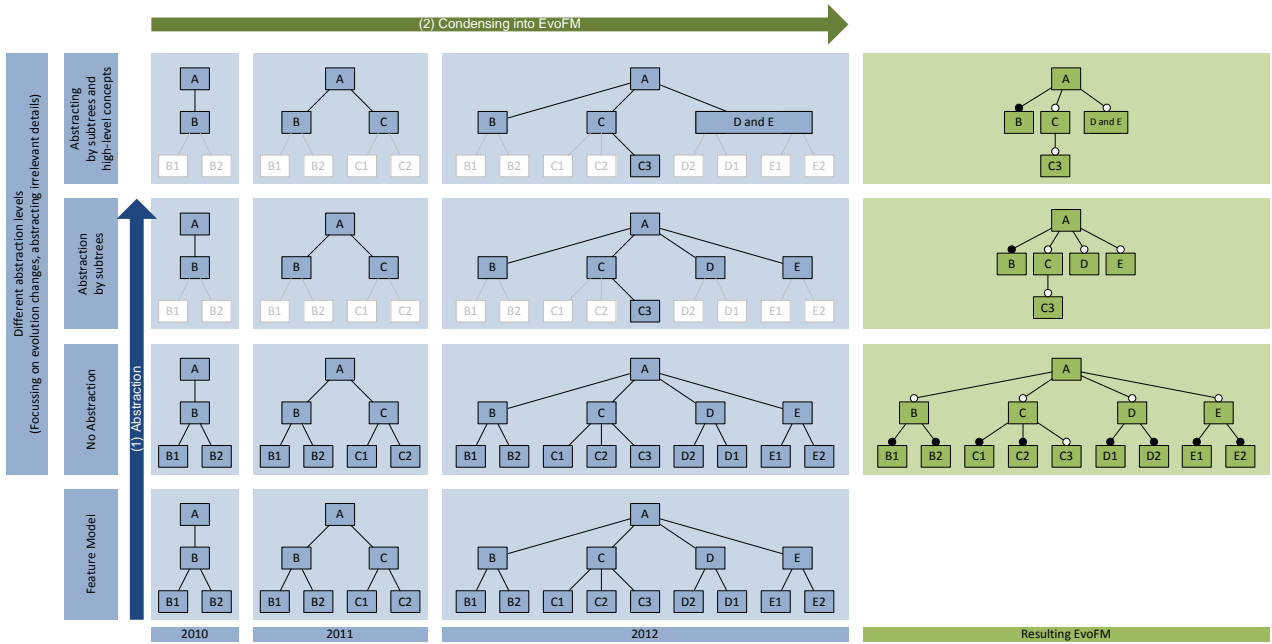


Figure 5: Different abstraction levels leading to different forms of EvoFM.

of Figure 5, *D* and *E* are represented by a more abstract feature *D and E*. In the parking assistant example, such a higher level of abstraction could be, e.g., *HighPrecisionSensors* representing multiple sensors in the product line.

### 3.2.2 Mappings

As EvoFM is more abstract than the FM it does not contain all feature model elements. In particular, EvoFM does not specify the properties of a feature (mandatory/optional) or additional feature model elements like cross-tree constraints.

Thus, to enable derivation of a feature model by configuring EvoFM, the features in EvoFM must be associated with assets from the product line feature model (in the same way as implementation assets are usually associated with features in a feature model so that the implementation can be derived by configuring the feature model). Figure 4 shows these mappings for an extract from the models for the parking assistant example.

For instance, *Distance Sensor* and *Infrared Sensor* in EvoFM are mapped to corresponding features in the FM. *Ultrasonic Sensor* is mapped to a corresponding feature *Ultrasonic Sensor* as well. In addition, if *Ultrasonic Sensor* is selected, it is mapped to an feature group as alternative to *Infrared Sensor*, while *Infrared Sensor* is a mandatory subfeature otherwise (like in 2008 in the example). The example also shows that there is a need for an appropriate mechanism for the mappings which are the basis for deriving and analysing models for the varying evolution steps.

### 3.2.3 Change operators

As described above, an EvoFM in conjunction with its mappings allows to derive a feature model from a given EvoFM configuration. The configuration thus defines the presence or absence of features in the feature model. However, sometimes there are structural changes in the feature

model which are not associated with presence/absence of a feature, like changing a product line feature from mandatory to optional or converting an alternative group into an OR group.

It should be possible to model such changes in EvoFM, if desired, so that they can be selected as part of a configuration and are visible in the Evolution Plan as well. To specify such changes we use the concepts of *Change Operators* similar as used in the area of model co-evolution (see also related work in Section 5).

Each change operator can be added as a special subfeature to features in EvoFM, specifying a corresponding change in the FM. As an example consider the evolution from 2008 to 2009 in Figure 1, in particular the subfeatures of *Distance Sensors*. The conversion of this set of subfeatures into an alternative group is modelled in EvoFM as a special feature *Convert subfeatures {IR, US} to alternative group* located in EvoFM below the relevant node *Distance Sensors*.

In a similar fashion we specify the introduction of constraints, e.g., by the mandatory subfeature *requires Infrared Sensor* below *Sensor Fusion* to indicate that we have to introduce a *SensorFusion-requires-InfraredSensor* whenever an evolution step introduces the feature *Sensor Fusion* into the FM.<sup>2</sup>

It is important to notice that a change operator always refers to the feature model as defined in the mappings. Like for other EvoFM features, selection of a change operator in a EvoFM configuration results in its presence, i.e., its application to the feature model defined by the EvoFM configuration. Hence, change operators either have to be defined in such that they do not conflict with each other or additional constraints have to be defined between them to handle the conflicts (e.g., an exclusion constraint).

<sup>2</sup>As these constraints remain stable in the example, one could alternatively decide to model them implicitly as part of the mappings.

Textual Specification	Evolution Step n	Evolution Step n+1	EvoFM	Semantics
add [(optional mandatory)] feature g [below f]  remove feature g				Adds $g:Feature$ and $c1:HasOptionalSubfeature$ (or $c1:HasMandatorySubfeature$ ) with $c1.parent=f$ and $c1.subfeature=g$ If $f$ is not specified, then $g$ is added below the root.  „Remove“ is expressed as an inverse of „add“, i.e., the effects of add are not activated.
replace $g_1$ with [(mandatory optional) feature] $g_2$				For each of the children the same semantics are executed as for „add“
move g to h				Removes $c1:HasOptionalSubfeature$ (or $c1:HasMandatorySubfeature$ ) with $c1.parent=f$ and $c1.subfeature=g$  Adds $c2:HasOptionalSubfeature$ (or $c2:HasMandatorySubfeature$ ) with $c2.parent=h$ and $c1.subfeature=g$
rename $f_1$ to $f_2$				Sets $f1.name="f2"$
change feature f to mandatory  change feature f to optional				Removes $c1:HasOptionalSubfeature$ (or $c1:HasMandatorySubfeature$ ) with $c1.parent=f$ and $c1.subfeature=g$  Adds $c2:HasMandatorySubfeature$ (or $c2:HasOptionalSubfeature$ ) with $c2.parent=f$ and $c1.subfeature=g$

Figure 6: Catalogue of evolution operators - operators on features.

Basically, the set of 'atomic' change operators as used in our previous work [3] (*add*, *remove*, *modify*) is sufficient to describe all possible changes on a feature model. However, as discussed in [12], it is often useful to provide additional, more complex operators as they (1) simplify the specification of complex changes, and (2) prevent the loss of semantics, e.g., during *replace* operations (instead of simple combinations of *remove* and *add*).

To support the aim of EvoFM as means for communication and proactive planning, it seems desirable to allow the modeller to specify evolution changes on a more abstract level. For instance, in the parking assistant example in 2009, a feature *Ultrasonic Sensor* is added as an alternative to *Infrared Sensor*. Using atomic operators, this has to be expressed as a sequence of *add feature group*, *add Ultrasonic Sensor* (as child of the feature group) and *modify Distance Sensor* (to become a child of the feature group as well). However, from the viewpoint of the modeller, this can be seen as a single change in the model, like *add Ultrasonic Sensor as alternative to Infrared Sensor* and should thus be supported by a respective operator.

As visible in the parking assistant example, such restruc-

turing with addition or deletion of feature groups frequently occurs during the addition or deletion of subfeatures. Thus, the modeller should be provided with appropriate complex change operators. While there is some existing work on change operators for metamodels, change operators for feature models have been defined for specific purposes only and do not cover all change operators, which frequently occur during feature model evolution (see related work in Section 5). Thus, we introduce a catalogue of change operators for feature model evolution in the next section.

#### 4. EVOLUTION OPERATORS

In the preceding section we introduced the Evolution Plan, EvoFM and the design rationale behind these concepts. To complement this we now provide a catalogue of evolution operators. Most of these concepts are presented in the graphical overviews in Figures 6 and 7. The first column (in cyan) show the textual specification of the evolution operation. The next two columns (in blue) show the FM in transition from evolution step  $n$  to  $n + 1$ . The next column (in green) illustrates how the particular evolution operation is represented in EvoFM. Selecting this configura-

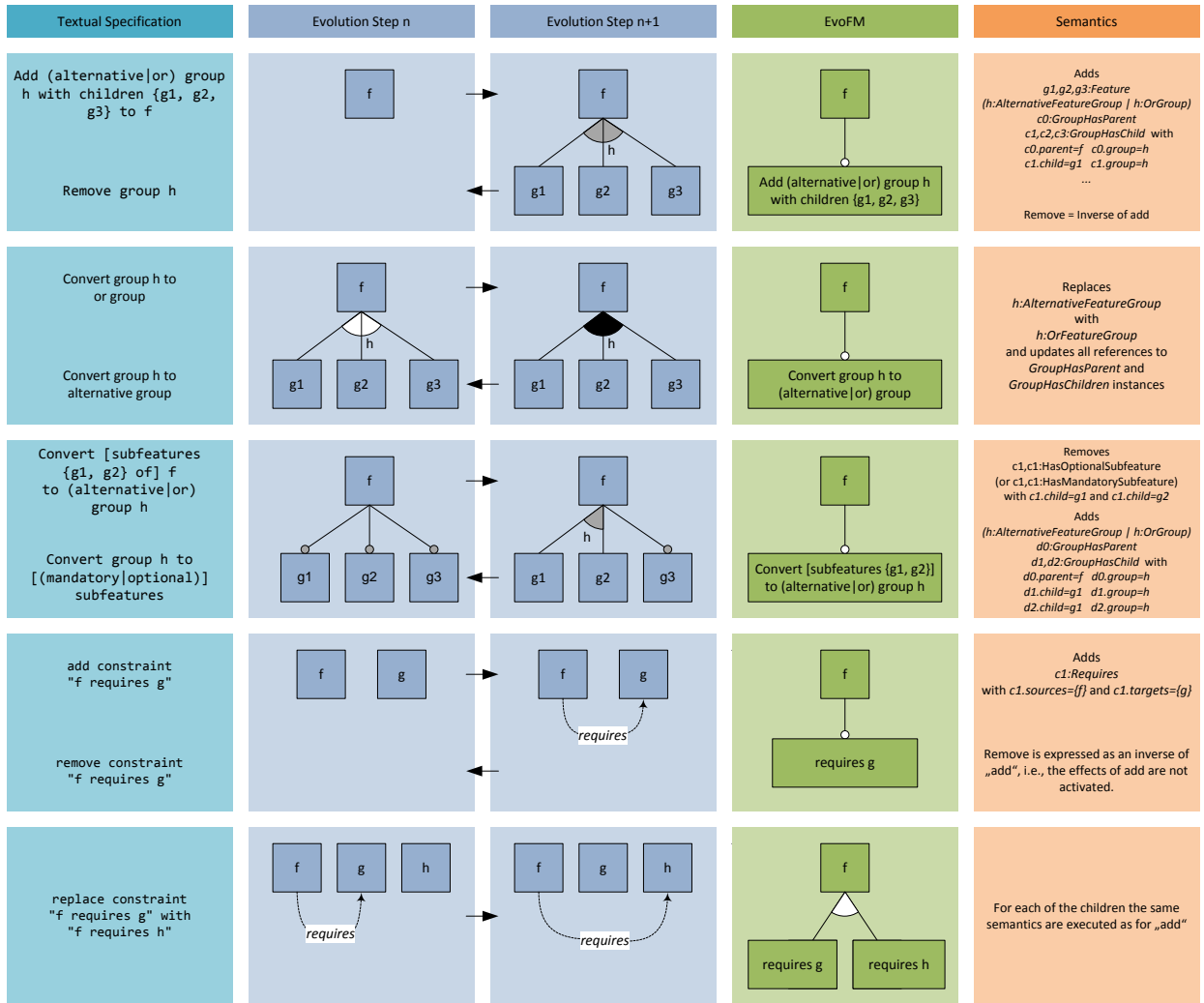


Figure 7: Catalogue of evolution operators - simple feature group operators.

tion in EvoFM triggers the change in FM shown between column two and three. The last column (in orange) specifies the semantics of this operation by defining which changes in terms of features and feature model primitives [2] will be applied. Please note that to save space some cases have been condensed (e.g., mandatory/optional features and or/alternative groups). In the textual specification this is shown like this (mandatory|optional). The corresponding graphical elements are shown in grey to indicate that the element could be either white or black. Optional elements in the textual notation are indicated with [square brackets].

The most simple evolution operators are changes on features (see Figure 6) including add, remove, and replace. In addition, there are modifications such as move, replace, rename or a change of feature type.

Operators involving feature groups are shown in the upper part of Figure 7. These include adding or removing a group and operators to change the type of a group from “alternative” to “or” or vice versa. The rename/replace/-move operators for feature groups and operators that add/remove/replace/move children are omitted here for space reasons.

More advanced operators are the conversion of sets of subfeatures into groups and vice versa.

As in our approach [2] a feature model basically is just a long list of constraints (feature model primitives), we can perform arbitrary evolution operators by adding, removing, or modifying these constraints. For two examples involving “requires” constraints see lower part of Figure 7.

## 5. RELATED WORK

Discovery and specification of changes between two or more models is a task of general importance when dealing with model evolution. For instance, when changes on a metamodel are performed, all corresponding model instances must be updated accordingly as otherwise they no longer conform to the metamodel. This process is often referred to as “model co-evolution”. As shown in the overview in [12], 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, [9, 15, 6] provide large sets of complex operators for changes in metamodels, classifying them, e.g., into three categories *refactoring*, *construction*, and *destruction* [9, 15]. However, while the approaches pro-

vide useful basic techniques for our purpose, they cannot be applied directly to feature models as they address the specifics of metamodels.

Nevertheless, some other work focusses on change operators for feature models. Thüm et al. [14] distinguishes between four basic kind of changes on feature models: *Specialisation* means that the set of possible products is reduced, e.g., by removing variability. *Generalisation* means the set of products is enlarged, e.g., by adding variability. *Refactoring* means that the set of products remains the same, e.g., when cleaning up the feature model without changing its semantics. The fourth category is called *Arbitrary Changes* e.g., when adding and removing products at the same time. In [14], no change operators are defined, instead the authors present an algorithm to automatically classify given changes on a feature model into one of the four categories.

Alves et al. [1] propose a set of change operators for feature model generalisation, i.e., operators which increase the set of possible products. This includes adding features or increasing variability by changing a mandatory node to an optional one or changing an OR group into an XOR group. They also propose basic operators to merge two feature models under a common node and for basic refactorings.

A set of operators for feature model specialisation is proposed in [5]. This includes deletion of features or changing an optional node to a mandatory one. Follow-up work in [8] also discusses feature model changes for co-evolution, which includes addition and deletion of nodes, moving nodes or subtrees, and changes of cardinalities.

In this paper, we integrate these change operators into a consistent catalogue. Moreover, complex operators have been added – like grouping subfeatures into a feature group – as it turned out that they frequently occur during feature model evolution.

We presented first steps towards feature-oriented modelling of product line evolution in [3], including an overall conceptual framework and three application scenarios. The current paper substantiates this ideas by showing concrete concepts for EvoFM and the Evolution Plan (Section 3). Furthermore, we provide a catalogue of catalogue of evolution operators (Section 4).

## 6. CONCLUSIONS

Although initially EvoFM was very much like an ordinary feature model, during the further development (e.g., with the introduction of special subfeatures for evolution operators) it became a special type of feature model. Certainly, it is worth to consider alternative representations. However, in our opinion it is worthwhile to start from a well-established and tool-supported existing modelling formalism as long as the required modelling concepts are still under development.

In our initial experiments with the presented models, it appeared that the Evolution Plan and EvoFM are complementary. The Evolution Plan provides an compact, visual overview with focus on the relevant details. Early discussions with potential users indicate that such plan is a good communication device. In contrast to the overview provided by the Evolution Plan, EvoFM supports the consideration of evolution options on a more technical level. In particular it allows to express evolution steps as EvoFM configuration, which helps (1) to document an evolution history and (2) to explore the space of potential future evolution paths.

Future work includes the investigation of further design

issues which came up during the development of EvoFM: Feature interactions, as frequently discussed in feature modelling, have to be considered for EvoFM as well. In particular, the interaction of evolution operators needs to be investigated further. We intend to implement the presented evolution operators as model transformations. Here, we will to use our existing metamodels and configuration tools as a foundation.

## 7. ACKNOWLEDGMENTS

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/>.

## 8. REFERENCES

- [1] V. Alves, R. Gheyi, and T. Massoni. Refactoring product lines. In *GPCE'06*, 2006.
- [2] G. Botterweck, M. Janota, and D. Schneeweiss. A design of a configurable feature model configurator. In *VAMOS 2009*, 2009.
- [3] G. Botterweck, A. Pleuss, A. Polzer, and S. Kowalewski. Towards feature-driven planning of product-line evolution. In *FOSD 2009*, 2009.
- [4] G. Botterweck, A. Polzer, and S. Kowalewski. Using higher-order transformations to derive variability mechanism for embedded systems. In *ACES-MB 2009*, 2009.
- [5] K. Czarnecki, S. Helsen, and U. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process Improvement and Practice*, 10(1):7–29, 2005.
- [6] M. Herrmannsdoerfer, S. Benz, and E. Juergens. Cope - automating coupled evolution of metamodels and models. In *ECOOP 2009*. Springer, 2009.
- [7] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature oriented domain analysis (FODA) feasibility study. SEI Technical Report CMU/SEI-90-TR-21, 1990.
- [8] C. H. P. Kim and K. Czarnecki. Synchronizing cardinality-based feature models and their specializations. In *ECMDA-FA*, pages 331–348, 2005.
- [9] B. S. Lerner. A model for compound type changes encountered in schema evolution. *ACM Trans. Database Syst.*, 25(1):83–127, 2000.
- [10] R. Phaal, C. J. P. Farrukh, and D. R. Probert. Technology roadmapping—a planning framework for evolution and revolution. *Technological Forecasting and Social Change*, 71(1-2):5 – 26, 2004.
- [11] A. Polzer, S. Kowalewski, and G. Botterweck. Applying software product line techniques in model-based embedded systems engineering. In *MOMPES'09*, 2009.
- [12] L. M. Rose, R. F. Paige, D. S. Kolovos, and F. A. Polack. An analysis of approaches to model migration. In *MoDSE and MCCM*, 2009.
- [13] Software Engineering Institute. SPL Hall of Fame. Web site, 2008. <http://splc.net/fame.html>.
- [14] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to feature models. In *ICSE '09*, 2009.
- [15] G. Wachsmuth. Metamodel adaptation and model co-adaptation. In *ECOOP'07*, 2007.