A Model-Driven Approach to the Development of Heterogeneous Software Product Lines

Thomas Buchmann and Felix Schwägerl
University of Bayreuth
Chair of Applied Computer Science I
Bayreuth, Germany
{thomas.buchmann, felix.schwaegerl}@uni-bayreuth.de

***Abstract*—Software product line engineering is dedicated to planned reuse of software components based upon a common platform, from which single products may be derived. The common platform consists of different types of artefacts like requirements, specifications, architecture definitions, source code, and so forth. Only recently, research projects have been started dealing with model-driven development of software product lines. So far, the resulting tools can only handle one type of artefact at the same time. In this paper, requirements, concepts and limitations of tool support for heterogeneous model-driven software product line engineering are discussed. As a proof of concept, an extension to the toolchain FAMILE is presented, which supports mapping of features to different types of artefacts in heterogeneous model-driven software projects at the same time.**

Keywords–software product lines; model-driven development; negative variability; feature models; heterogeneity.

I. INTRODUCTION

Software Product Line Engineering (SPLE) [1][2] deals with the systematic development of products belonging to a common system family. Rather than developing each instance of a product line from scratch, reusable software artefacts are created such that each product may be composed from a collection of reusable artefacts — the platform. Commonalities and differences among different products may be captured in a *feature model* [3], whereas *feature configurations* describe the characteristics of particular products by selecting or deselecting the respective features. Typical SPLE processes distinguish between *domain engineering*, which deals with the establishment of the platform as well as the feature model, and *application engineering*, which is concerned with the derivation of particular products out of the product line by exploiting and binding the variability provided by the platform.

Two distinct approaches exist to realize variability in SPLE: In approaches based upon *positive variability*, product-specific artefacts are built around a common core [4][5]. *Composition* techniques are used to derive products. In approaches based on *negative variability*, a *superimposition* of all variants is created — a *multi-variant domain model*. The derivation of products is achieved by removing all fragments of artefacts implementing features which are *not* contained in the specific feature configuration [6][7]. The toolchain "Features and Mappings in Lucid Evolution" (FAMILE) [8][9], which is used in this paper, belongs to the latter category.

Model-driven Software Engineering (MDSE) [10] puts strong emphasis on the development of high-level models rather than on the source code. Models are not considered as documentation or as informal guidelines how to program the actual system. In contrast, models have a well-defined syntax and semantics. Moreover, MDSE aims at the development of *executable* models. The Eclipse Modeling Framework (EMF) [11] has been established as an extensible platform for the development of MDSE applications. It is based on the Ecore metamodel which is compatible with the OMG Meta Object Facility (MOF) specification [12]. Ideally, software engineers operate only on the level of models such that there is no need to inspect or edit the actual source code, which is generated from the models automatically. However, practical experiences have shown that language-specific adaptations to the generated source code are frequently necessary. In EMF, for instance, only structure is modeled by means of class diagrams, whereas behavior is described by modifications to the generated source code.

In the past, several approaches have been taken in combining SPLE and MDSE to get the best out of both worlds. Both software engineering techniques consider models as primary artefacts: Feature models [3] are used in SPLE to capture the commonalities and differences of a product line, whereas Unified Modeling Language (UML) models [13] or domain-specific models are used in MDSE to describe the software system at a higher level of abstraction. The resulting integrating discipline, Model-Driven Software Product Line Engineering (MDPLE), operates at a higher level of abstraction. The upcoming MDPLE approach has been successfully applied in several case studies, including MOD2-SCM [14], a model-driven product line for software configuration systems.

In this paper, requirements, concepts and limitations of tool support for *heterogeneous* product lines are discussed. Here, the term 'heterogeneity' means that (a) artefacts are distributed over multiple resources, (b) the underlying data format of artefacts may differ (e.g., text files or XMI files), (c) in the case of models, the metamodel may vary, and (d) variability among different resources may be expressed by a shared variability model that uses a common variability mechanism. Based upon these assumptions, several conceptual extensions to MDPLE frameworks are developed, which are implemented in the form of extensions to the toolchain FAMILE as a proof of concept.

The paper is structured as follows: After clarifying the contribution (Section II), the state of the art of homogeneous SPLE tools is outlined in Section IV. Section III discusses related work, before a brief introduction of the running example is given in Section V. Section VI explains the new concepts

 ISBN: 978-1-61208-367-4

introduced for the support of heterogeneous product lines. In Section VII, the example is revisited in order to demonstrate the heterogeneous extension to the MDPLE toolchain FAMILE on a product line for graphs, which has been modeled using Eclipse Modeling Technology (EMF and the Graphical Modeling Framework (GMF) [15]). Both the toolchain and the running example project may be retrieved via an Eclipse update site (http://btn1x4.inf.uni-bayreuth.de/famile2/update). Section VIII concludes the paper.

II. Challenges and Contribution

Heterogeneous software projects consist of a variety of interconnected resources of different types. Different representations may be used for requirements engineering, analysis and design. The generated source code is typically expressed in a general purpose language, e.g., Java, and extended with language-specific – mostly behavioral – components. Furthermore, a software project contains a set of configuration files such as build scripts, which are typically represented in plain text or XML format. In order to adequately handle variability of the overall software project, all these different artefacts need to be subject to variability management.

In its current state, *tool support* for model-driven product line engineering does not adequately address heterogeneous software projects (see Section III). In particular, the following new challenges arise for SPLE tools:

(a) They should ensure the consistency of *cross-resource links* between different artefacts.

(b) The *level of abstraction* needs to be variable, i.e., the tool should be able to operate both at the modeling and at the source code level.

(c) Different artefacts are based on different formalisms, e.g., metamodels or language grammars. In the special case of models, supporting a mixture of different metamodels requires adequate tool support.

(d) All artefacts must be handled by a *uniform variability mechanism* (e.g., a common feature model) in order to allow for product configuration in a single step.

In this paper, an approach to heterogeneous SPL development is presented, which advances the state of the art by the following conceptual contributions:

(a) **Multi-resource artefacts** Heterogeneous projects consist of inter-related models created for different development tasks such as requirements engineering or testing. The referential integrity among these inter-related models is maintained during product derivation.

(b) **Heterogeneous artefact types** The approach presented here can handle product lines composed from different kinds of artefacts. Technically, an abstraction from different resource types is conducted by representing them as EMF models.

(c) **Variable metamodels** In the special case of models, the approach presented here does not assume a specific metamodel but allows an arbitrary mixture of models which may be instances of any Ecore-based metamodel(s).

(d) **Common variability mechanism** In the original version of FAMILE, the variability mechanism of feature models has been applied to single-resource EMF models. The presented approach allows for an extension of the product space to almost arbitrary resources. All artefacts are managed by a unique feature model.

These conceptual contributions will be demonstrated by the example of a proof-of-concept implementation that provides an extension to the FAMILE toolchain [8][9]. The extended version of FAMILE can deal with plain text files, XML files, Java source code files, arbitrary EMF models, and further types of resources. This way, variability within complete Eclipse projects may be managed. Internally, all artefacts, even plain text and XML files, are represented as EMF models. In Section VI, tool support is discussed in detail.

III. Related Work

Many different tools and approaches have been published in the last few years, which address (model-driven) software product line development. Due to space restrictions, the focus of this comparison lies on support for heterogeneous software projects, using the definition of heterogeneity given in the introduction. Other comparisons of FAMILE and related approaches can be found in [8] and [9].

The tool *fmp2rsm* [16] combines FeaturePlugin [17] with IBM's Rational Software Modeler (RSM), a UML-based modeling tool. The connection of features and domain model elements is realized by embedding the mapping information into the domain model using stereotypes (each feature is represented by its own stereotype), which requires manual extensions to the domain model. While fmp2rsm is limited to the support of RSM models, the approach presented in this paper provides a greater flexibility since the only restriction is that the domain model needs to be Ecore based. Furthermore, the extensions presented in this paper allow to use several domain metamodels within one software product line project.

FeatureMapper [6] is a tool that allows for the mapping of features to Ecore based domain models. Like FAMILE, it follows a very general approach permitting arbitrary Ecore models as domain models. FeatureMapper only allows to map a single (self-contained) domain model, while the work presented in this paper allows to use FAMILE also for software product lines whose multi-variant domain model is composed of artefacts distributed over different resources. Furthermore, the artefacts may be instances of different metamodels.

VML* [4] is a family of languages for variability management in software product lines. It addresses the ability to explicitly express the relationship between feature models and other artefacts of the product line. It can handle any domain model as long as a corresponding VML language exists for it. VML* supports both positive and negative variability as well as any combination thereof, since every action is a small transformation on the core model. As a consequence, the order in which model transformations are executed during product derivation becomes important. So far, VML* is designed to work with text files, provided that a corresponding VML

 ISBN: 978-1-61208-367-4

language exists for it (i.e., a grammar has to be specified). Theoretically, VML languages could be written that work with XMI serializations of the respective models in the example presented in this paper, whereas FAMILE provides generic support for model-driven software development based on Ecore compliant models. In other words, VML* and FAMILE provide similar support for heterogeneous projects, but they operate on different ”technological spaces“. As a consequence, the example provided in Section VII cannot be realized with VML* easily. In fact, significant effort would be required to create VML languages for the different models involved in the graph product line example as presented here.

MATA [5] is another language which also allows to develop model-driven product lines with UML. It is based on positive variability, which means that, around a common core specified in UML, variant models described in the MATA language are composed to a product specific UML model. Graph transformations based on AGG [18] are used to compose the common core with the single MATA specifications. While MATA is limited to UML, the approach presented in this paper provides support for any Ecore based model and furthermore allows the combination of different domain metamodels within one product line project.

CIDE [7] is a tool for source-code based approaches. It provides a product specific view on the source code, where all source code fragments which are not part of the chosen configuration are omitted. The approach is similar to *#ifdef*-preprocessors known from the C programming language [19]. The difference is that it abstracts from plain text files and works on the abstract syntax tree of the target language instead. In its current state, CIDE provides support for a wide range of different programming languages. Unfortunately, it cannot be used for model-driven development. In contrast, FAMILE provides full-fledged support for model-driven development based on Ecore models. Furthermore, it may also deal with regular Java source code by using the MoDisco [20] framework.

Bühne et al. [21] and Dhungana et al. [22] present approaches for heterogeneous variability modeling, i.e., managing commonalities and differences across *multi product lines*. Dhungana et al. aim at unifying multi product lines which rely on different tools and formalisms for modeling variability. Web services are used for a prototypical implementation. In contrast to the approach presented here, in both approaches, the term ‘heterogeneity’ concerns different variability models rather than the product space. While Bühne et al. and Dhungana et al. only address variability modeling, the approach presented in this paper covers a larger part of the software life-cycle. Furthermore, FAMILE does not only allow for variability modeling, but also for mapping the variability information to heterogeneous implementation artefacts.

IV. State of the Art: Homogeneous MDPLE tools

This section provides a brief overview on the state of the art of current tools for model-driven product line engineering. The description is confined to approaches based on negative variability. As one representative, the original version of the FAMILE toolchain [8][9] is presented. Current MDPLE tools assist the user in the following tasks:

1) **Definition of a feature model** At the beginning of the domain engineering phase of the product line life-cycle, the problem domain is analyzed and the commonalities and differences are captured in a *feature model* [3]. For feature models, several extensions such as cardinality-based feature modeling [23] have been proposed.
2) **Creation of the domain model** For the construction of a multi-variant domain model, modelers may use their preferred modeling languages and tools. Most MDPLE approaches only support single-resource models. FAMILE requires that the resulting model is an instance of an Ecore metamodel.
3) **Mapping features to model elements** In order to define which parts of the domain model realize which feature, or a combination thereof, MDPLE tools provide different mechanisms to map features to model elements. For this purpose, FAMILE includes the Feature to Domain Mapping Model (F2DMM) editor which supports the process of assigning *feature expressions* – arbitrary propositional formula on the set of features – to particular model elements.
4) **Ensuring the consistency of the product line** The increasing complexity coming with both the size of the multi-variant domain model and the number of features requires sophisticated mechanisms to detect and repair inconsistencies among the product line. In particular, the consistency between (a) the mapping model and the domain model, (b) the feature model and its corresponding feature configurations, and (c) feature expressions and the feature model, must be ensured. Different approaches are described in [23], [24]. FAMILE introduces the concepts of *surrogates* and *propagation strategies* [9] for this purpose.
5) **Definition of feature configurations** As soon as the mapping is complete, MDPLE tools support the creation of *feature configurations*, each describing the characteristics of a member of the software product line. For each feature defined in the feature model, a *selection state* must be provided that determines whether a feature is present in the corresponding product.
6) **Product derivation** A specific product can be derived by applying its corresponding feature configuration to the product line. During the derivation process, the multi-variant domain model is filtered by elements whose assigned feature expressions evaluate to false, i.e., the corresponding features are deselected in the respective feature configuration. In homogeneous MDPLE tools, the result of this operation is a product-specific single-resource model represented in the (previously fixed) domain metamodel.

V. Example: Homogeneous FAMILE Product Line for Graph Metamodels

The following statements refer to the original version of the tool FAMILE as one representative of homogeneous MDPLE tools. Section VI demonstrates how heterogeneous project support is added to the toolchain.

FAMILE itself has been developed using EMF as its technological foundation. A model-driven software product

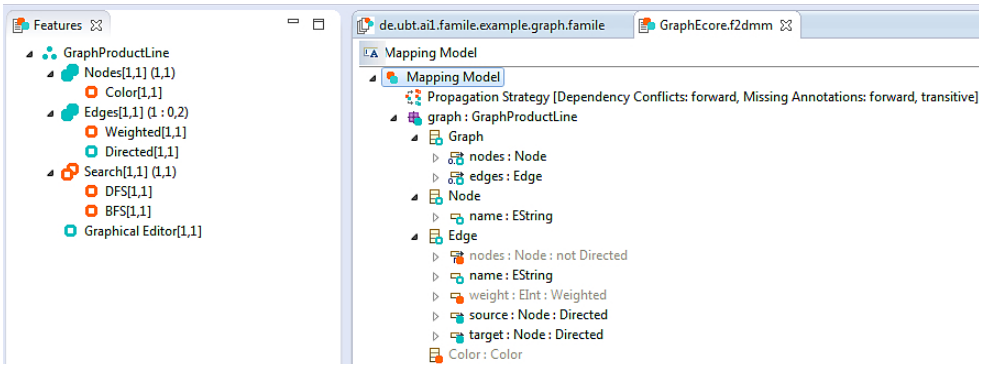

Figure 1. Screenshot of the F2DMM mapping model editor showing the multi-variant domain model of the (homogeneous) graph product line.

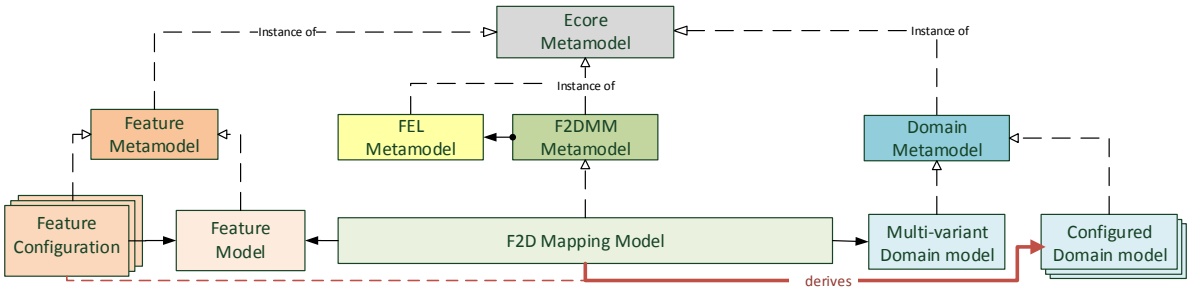

Figure 2. Metamodels and models involved in the original version of FAMILE. Different models are used to map a single-resource multi-variant domain model. All metamodels are based on Ecore.

line developed with FAMILE is spread over multiple EMF resources which are instances of multiple metamodels (cf. Figure 2): Feature models and configurations share a common metamodel which also supports cardinality-based feature modeling. The F2DMM mapping model describes how domain model elements are mapped to features. The domain model is instance of an arbitrary domain metamodel, which is fixed for the mapped resource. It is assumed to be a single-resource entity. The Feature Expression Language (FEL) metamodel describes a textual language for feature expressions [8].

With the F2DMM editor (see Figure 1), the user is assisted in assigning feature expressions to domain model elements. The underlying F2DMM mapping model is constructed automatically and reflects the spanning containment tree structure of the domain model. Using the reflective EMF editing mechanism [11], the F2DMM user interface emulates the reflective EMF tree editor. Optionally, the user may load an example feature configuration already during the mapping process in order to comprehend how feature expressions are evaluated. The screenshot in Figure 1 depicts an example feature configuration in the left pane. Selected features or groups are displayed in cyan, deselected features or groups in orange. The right pane contains the mapping of specific features to artefacts of the multi-variant domain model. Elements are annotated with feature expressions after a colon. The calculated selection states (selected, deselected) are represented in cyan and orange, respectively.

As a demonstrating example within this paper, the *graph product line* example has been adopted, which is frequently used in research papers because it is easy to understand and its size is rather small. In Figure 1, an example feature configuration is loaded that represents a directed graph (with uncolored nodes and unweighted edges) that realizes neither depth-first search nor breadth-first search. The feature "Graphical Editor" will be explained in Section VII, where the example is revisited in the context of heterogeneous product line support.

VI. Support for Heterogeneous Approaches

This section explains how support for heterogeneous model-driven software product lines has been added to the MDPLE tool FAMILE. From a technical point of view, this requires multiple metamodels for the platform and multiple models that describe different artefacts of the product in different stages of the development process (e.g., requirements, static model, implementation). As stated in the introduction, it is assumed that all project artefacts may be expressed using EMF. EMF and its metamodel Ecore are wide-spread in the Eclipse community, thus a large number of potential domain models is addressed. A (non-exhaustive) list may comprise of course Ecore class diagrams, Eclipse UML models [25], Xtext [26] / EMFText [27] grammars and documents, GMF models [15], Acceleo source code generation templates [28], MWE2 Workflow files [29], Xtend specifications [30], domain-specific languages based on Ecore, and many more. Additionally, FAMILE has been applied successfully to Java source code as well. To this end, the MoDisco [20] framework is used, which allows to parse Java source code into a corresponding Java model instance (which is also based on Ecore). MoDisco may be also used to create EMF model instances out of XML files. For plain text files that are not yet mapped by language-specific mechanisms, the new extension framework provides an additional "fall-back" metamodel, which operates on the granularity of text lines. As a consequence, FAMILE may handle arbitrary resource types that may occur within typical

 ISBN: 978-1-61208-367-4

MDSE-related Eclipse projects.

Figure 4 shows the conceptual overview of the new, heterogeneous version of the FAMILE toolchain. A FAMILE model wraps different F2DMM model instances which are used for mapping features to the different (heterogeneous) multi-variant domain model instances. A FAMILE model references a given feature model and one out of an arbitrary number of corresponding feature configurations. Features are mapped to the respective domain artefacts by using a separate mapping model per resource.

A. The FAMILE Metamodel

The specific requirements of heterogeneous modeling projects have been addressed by the FAMILE metamodel and its corresponding instances, which constitute an extension to the F2DMM metamodel, where models have been considered as self-contained single-resource entities [8]. In order to support multiple (EMF-based) resources of different type, the new FAMILE metamodel shown in Figure 3 wraps several instances of the F2DMM metamodel, which still constitutes the core of the extended toolchain.

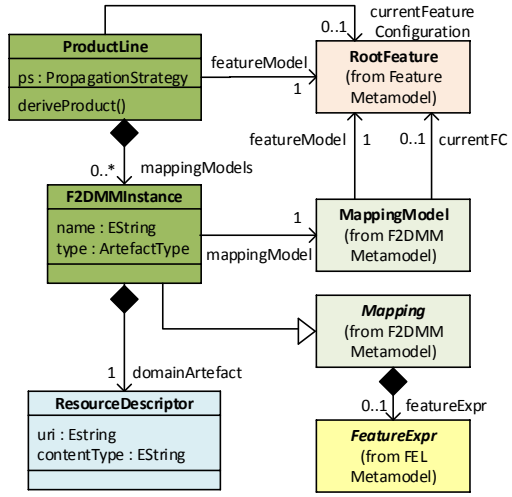

Figure 3. The FAMILE metamodel which is designed to support heterogeneous software product lines.

The FAMILE metamodel defines a logical grouping of inter-related mapping models. The root element – an instance of ProductLine – defines a number of *global project parameters*, being the references to the used feature model and optionally a feature configuration, as well as a *propagation strategy* (used for automatic detection and resolution of inconsistencies; see [9]). FAMILE takes care that global project parameters are kept consistent within different F2DMM resources of the same heterogeneous product line.

A single F2DMM mapping model, which refers to exactly one mapped resource, is represented by F2DMMInstance. This meta-class defines a number of *resource-specific* parameters, such as the name and the *artefact type* (requirements, implementation, test, etc.). Please note that F2DMMInstance extends the abstract meta-class Mapping defined in the F2DMM metamodel, which manages variability by the use of feature expressions and the calculation of selection states [8]. The referenced MappingModel describes the mapping of the specific contents of a mapped resource, e.g., mapped EMF objects in the case of EMF model resources. Furthermore, a contained ResourceDescriptor element describes additional resource-specific parameters, being the relative URI of the mapped resource, as well as its *content type* (plain text, XML, EMF, etc.). The resource containing a multi-variant domain model is referenced by its URI.

Besides the possibility of annotating specific resources of the multi-variant domain model with feature expressions, the presented extension addresses the fact that in heterogeneous projects, *cross-resource links* occur frequently. For instance, in the example in Section VII, elements of an Ecore model are referenced by a corresponding GMF mapping model located in a different resource. During product derivation, these links are detected and resolved automatically in order to meet the requirement of referential integrity across multiple resources. As a result, a derived product will never contain any reference to the multi-variant model.

B. User Interface

The user interface has been extended to support heterogeneous software product lines. A new FAMILE editor manages the mapping for a set of resources rather than single-resource models, which are still covered by the existing F2DMM editor. In addition to the tasks listed in Section IV, the extended FAMILE framework supports the following user interactions (see also example in Section VII):

1) **Adding heterogeneous product line support** An arbitrary Eclipse project containing any kind of resource (e.g., EMF models, source code and documentation) can be provided with the *FAMILE nature*, which adds heterogeneous product line support by automatically creating a FAMILE product line model.
2) **Definition of a global feature model** As soon as the FAMILE nature has been added, the feature model editor is opened automatically and can be used to provide the results of domain analysis. Once a new feature model has been created or an existing feature model has been selected, its contained features may be used in feature expressions annotating corresponding implementation fragments from the multi-variant domain model(s).
3) **Adding variability to resources** Initially, it is assumed that none of the project resources is subject to variability. In order to add variability to a specific resource, the *Add F2DMM Instance* command can be invoked. It will create a new mapping model for the selected resource and append it to the reference mappingModels of the ProductLine instance. Furthermore, global project parameters are transferred to the new F2DMM instance.
4) **Assigning feature expressions to resources** In many cases, variability is achieved at a rather coarse-grained level, having resources rather than objects implement features. The FAMILE editor supports this requirement by the possibility of assigning feature expressions to entire resources.
5) **Applying a feature configuration globally** The command *Set Feature Configuration* allows to change

 ISBN: 978-1-61208-367-4

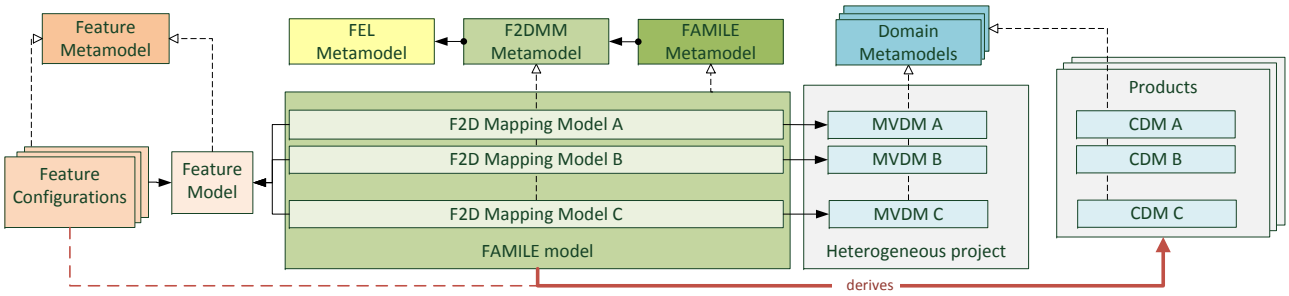

Figure 4. Metamodels and models involved in the extension of FAMILE. Abbreviations: MVDM = multi-variant domain model; CDM = configured domain model.

the current configuration, which will restrict the visible elements/resources in both the F2DMM and the FAMILE editor to elements with a feature expression that satisfies the new configuration. This global project parameter is propagated to all existing F2DMM instances.

6) **Deriving a multi-resource product** After applying a specific feature configuration, a product can be exported. Invoking the *Derive Product* command will prompt the user for a name of the derived Eclipse project. As described above, F2DMM product derivation will be applied to each mapping model covering a resource, keeping cross-resource links consistent. Resources which are not wrapped by any F2DMM instance or which are not annotated with FEL expressions will be copied without any further restriction.

VII. Example Revisited: Heterogeneous Product Line for Graph Metamodels and Editors

To demonstrate FAMILE's support for heterogeneous software projects, the graph product line introduced in Section V has been extended by a graphical editor. To achieve this in a model-driven way, GMF [15] has been used. A screencast demonstrating how to use the extensions for heterogeneous projects provided by FAMILE can be found on the corresponding webpages (http://btn1x4.inf.uni-bayreuth.de/famile/screencasts).

A. GMF Artefacts as a Heterogeneous Set of Multi-Variant Domain Models

Figure 5 depicts the different models involved in the GMF development process. The abstract syntax is defined by an Ecore model, while the editor providing the concrete (graphical) syntax is defined by a graphical definition model, a tooling definition model and a GMF mapping model. The EMF generator model is used to generate Java source code for the abstract syntax while the GMF generator model is responsible for generating the diagram editor's source code. Please note that the screencast does not cover the definition of the models mentioned below. It is assumed that the models describing the abstract and concrete syntax definitions have been created beforehand:

1) **Ecore** The abstract syntax of the graph metamodel has been created in Ecore. As shown in Section V, the F2DMM instance which maps features to the semantic model (abstract syntax).

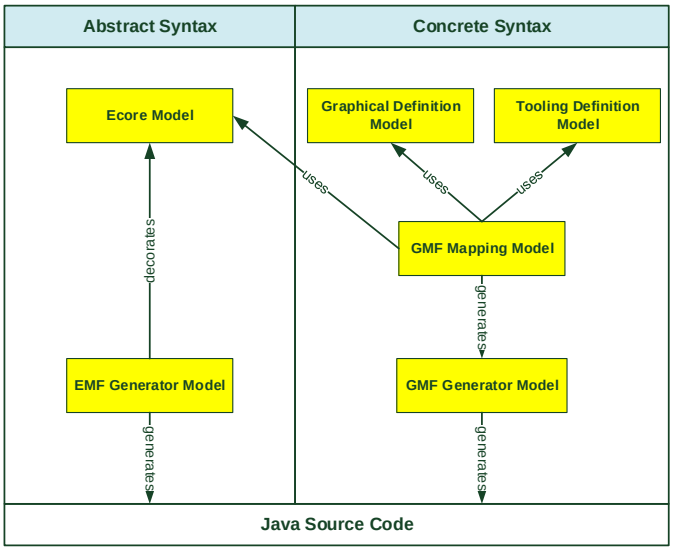

Figure 5. Models involved in the GMF development process.

2) **GMFGraph (Graphical Definition Model)** GMF uses a GMFGraph model to define the graphical representation of the concrete syntax. In case of the example, the visual appearance of nodes and edges of the graph is defined.

3) **GMFTool (Tooling Definition Model)** Every GEF based editor uses a so called palette to drag new elements to the canvas. As GMF is a model-driven extension to GEF, it follows this paradigm. The GMFTooling definition model is used to specify the contents of the editor's tool palette.

4) **GMFMap (GMF Mapping Model)** The models described above (Ecore, GMFGraph and GMFTool) are combined in the GMF mapping model. In this model, a relation between abstract syntax (Ecore) and graphical notation (GMFGraph) is established. Furthermore, the tools (GMFTool) for creating corresponding model elements are linked to those relations. Please note that the GMF mapping model is the central part of the Graphical Modeling Framework. It has nothing in common with the F2DMM mapping model, which is the core of the FAMILE toolchain.

 ISBN: 978-1-61208-367-4

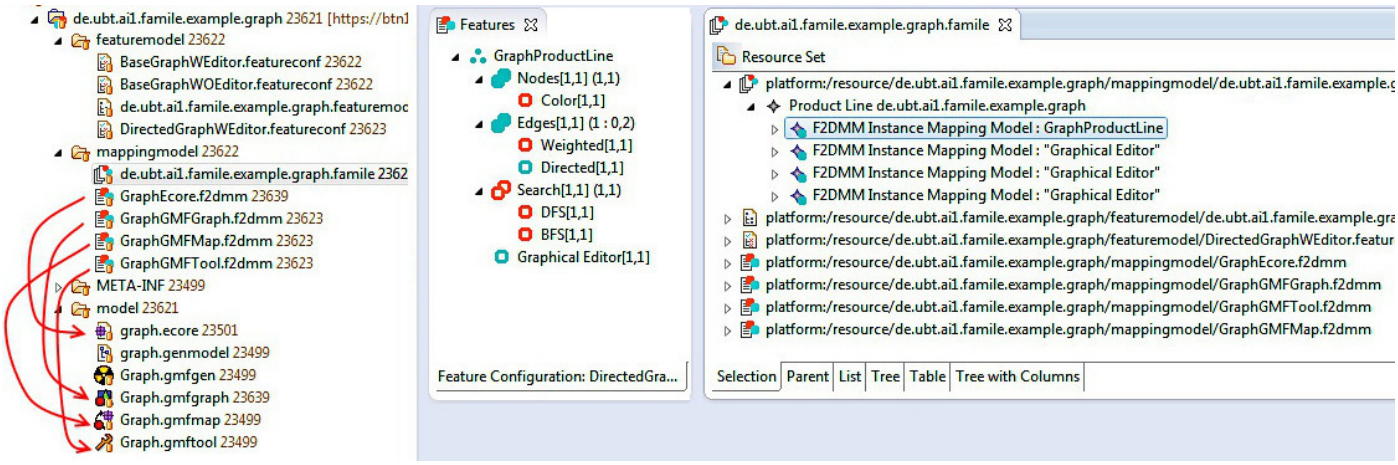

Figure 6. Screenshot of the FAMILE model editor. The left pane shows the feature model and feature configuration. In the main pane, the contents of the FAMILE model are shown.

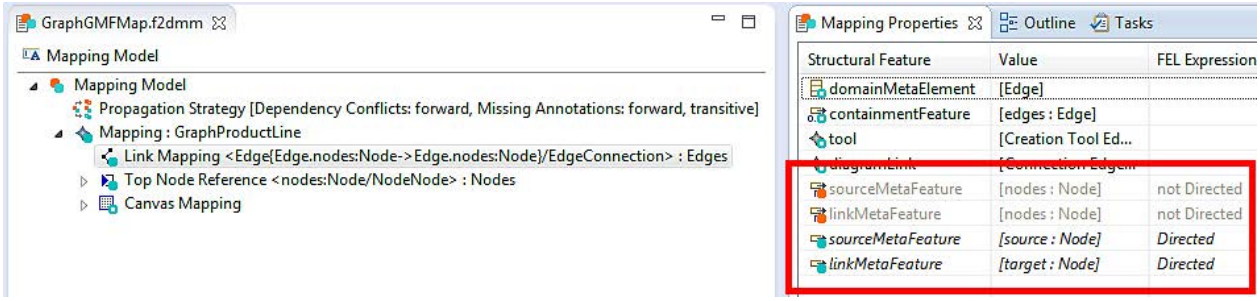

Figure 7. Usage of alternative mappings. The red box depicts where elements of the multi-variant domain model have been virtually extended by alternative mapping values (in italics).

B. *Mapping Heterogeneous Artefacts*

In order to use FAMILE for a (heterogeneous) project, the FAMILE project nature has to be assigned. As a result, an empty feature model and a FAMILE model are created within the project. In the example, the feature model shown in Figure 1 is applied to the entire product line as a global project parameter. In order to map features to corresponding implementation fragments, F2DMM mapping models have to be created for each domain model. In the example, four F2DMM instances have been defined, one for each EMF/GMF resource mentioned above.

Figure 6 depicts the state of the example project after corresponding F2DMM instances have been created. The red arrows in the left part of the figure indicate which domain model resource the corresponding F2DMM models refer to. As one can see, FAMILE model elements may also be annotated with feature expressions. For the example, a feature called *Graphical Editor* has been introduced in order to make the visualization (tree editor vs. graphical editor) of the graph variable. In case this feature is deselected in a respective configuration, it is obvious that the resulting product must not contain the GMF models. As a consequence, the respective F2DMM instances are annotated with the feature expression Graphical Editor, as shown in Figure 6.

Figure 1 has already shown the content of the F2DMM mapping model for the Ecore model which is used to define the abstract syntax of the graph model. Analogously, F2DMM instances for the other required models (GMFGraph, GMFTool and GMFMap) are created. Each model file contains a superimposition of all possible variants. Common approaches using negative variability suffer from restrictions imposed by the respective domain metamodels which usually do not provide adequate support for variability. FAMILE mitigates this restriction by offering the advanced concept of *alternative mappings*. In the example, alternative mappings are used in the Link mapping in the GMFMap model (c.f., Figure 7). In case of an undirected graph, the corresponding graphical editor should just connect two nodes by a solid line. To this end, the underlying semantic model (i.e., the Ecore class model) provides a reference nodes in the class Edge. In contrast, if the feature *Directed* edges is selected, the graphical editor should indicate the direction of the edge connecting two nodes by using an arrow as a target decorator. Furthermore, the semantic model does no longer contain a reference nodes, but instead two single-valued references source and target, which are used to store the corresponding nodes connected by the edge. In GMF, a link mapping requires to specify the corresponding EReferences which are used as the link's source and target. While in the first case, both source and target features in the GMFMap file are set to the EReference nodes, the latter case requires those features to point at the corresponding source and target EReferences.

In this example, FAMILE's alternative mapping capabilities are necessary because the GMF mapping model uses a single-valued EReference to store the sourceMetaFeature

and linkMetaFeature features. In case of undirected edges, the nodes Reference defined in the Ecore model of our graph product line is used. However, in case of directed edges, a distinction between source and target nodes is required. To this end, the Ecore model provides corresponding source and target EReferences in the class Edge (c.f. Figure 1), which have to be used in the GMFMap model instead in case the feature *Directed* is chosen. Figure 7 depicts how this has been solved using FAMILE's alternative mappings [8], which can virtually extend the multi-variant model and thus mitigate the limited variability of the respective domain metamodels.

C. Product Derivation

Once the mapping is completed, specific feature configurations may be used to derive concrete products. In the example, a derived Eclipse project is created which contains the required model files. Please note that the derived project does not include the Ecore and GMF generator models which are required to generate code. Since code generation is always invoked on a *configured* product, this task clearly belongs to the *application engineering* rather than to the *domain engineering* phase.

With the feature configurations provided in the example project, a fully automatic generation of four Eclipse plugin projects may be performed, which differ from each other as follows:

- an EMF tree editor for undirected, unweighted graphs,
- a GMF-based graphical editor for undirected, unweighted graphs,
- a graphical editor for directed, unweighted graphs, and
- a graphical editor for directed, weighted graphs.

Of course this set of feature configurations does not contain all possible combinations of features and it may be extended arbitrarily based on the features defined in the feature model.

D. Outlook: Increasing the Heterogeneity of the Example Project

The example described in this section has been conducted using only EMF-compatible resources as artefacts. All models involved in the GMF development process, i.e., the Ecore domain model, the Graphical Definition Model, the Tooling Definition Model, as well as the GMF Mapping Model, are instances of different Ecore-based metamodels. In the current state of the project, these models constitute the adequate level of abstraction for variability management. However, it might become necessary to define additional F2DMM mapping models for non-EMF resources in addition, for different reasons:

- With Ecore, only *structure* may be modeled in the form of class diagrams. For the *behavioral* part, modifications to the generated Java source code might become necessary. In order to map specific parts of the source code, e.g., specific method bodies, to features, additional F2DMM mapping models may be added for the respective Java files. For this purpose, Java constructs are internally mapped to EMF models using the MoDisco framework, as described in Section VI. Please note that using the current FAMILE extension, the user may annotate Java elements directly in the standard Java text editor.
- The file `plugin.properties` in the Eclipse project contains language-specific UI string constants, each declared in a respective text line. Currently, the generated Editor displays UI elements in English. However, if support for different languages is desired, one may add an additional F2DMM mapping model for the properties file, and corresponding features for each additional language to the feature model. The mapping may be adequately managed by means of a per-line mapping, using the "fall-back" EMF representation for plain text files (see Section VI).
- The file `plugin.xml` defines plugin extensions which are used to integrate the generated editor with the Eclipse platform. By adding an F2DMM mapping model and corresponding features, variability may be added to the plugin's runtime configuration, i.e., in order to make the editor's icon, label, or file extension depend on specific feature configurations. Assuming that no EMF-compatible metamodel for Eclipse plugin files is defined, the "fall-back" EMF representation for XML files (see Section VI) may be used.

VIII. Conclusion and Future Work

In this paper, requirements, concepts and limitations with respect to tool support for heterogeneous model-driven software product lines have been discussed. The approach presented in this paper solves a significant gap in the tool support for model-driven development of software product lines, whose artefacts are heterogeneous in terms of the used metamodels as well as in containing artefacts like text files or XML documents. As a proof of concept, an implementation of an extension to the FAMILE toolchain was shown.

Usually, (model-driven) software projects do not only consist of one single model. In contrast, different models and metamodels are involved. The main challenges of heterogeneous SPLE tool support are (a) to cope with different levels of abstractions (models and source code / plain text files) as well as (b) different forms of representation, (c) to ensure that links between different resources are kept consistent, and (d) to provide a uniform variability mechanism with respect to all project resources.

The approach presented here comes with the assumption that each resource type may be expressed by an EMF model; the new version of FAMILE provides adequate mapping constructs in order to support entire Eclipse projects. Furthermore, the solution to heterogeneous SPLE tooling is to divide a heterogeneous software project into a set of single-resource mapping models, for which adequate SPLE support is already implemented. Links between different models are kept consistent during product derivation. Extensions to the user interface ease the integration of new artefacts into heterogeneous product lines as well as modifications to existing mappings. Furthermore, fallback mechanisms for plain text files and XML files are provided, which also allow to map features to those kinds of artefacts at a lower level of abstraction. A demonstration of the presented approach was given by

 ISBN: 978-1-61208-367-4

applying the heterogeneous FAMILE toolchain to a product line for graph metamodels and editors, which manages an entire Eclipse plug-in project.

Current and future work addresses a case study which is carried out in the field of robotics [31][32]. Although first results produced by the old (homogeneous) version of the FAMILE toolchain are very promising, it is expected that a significant gain in productivity is achieved by exploiting the new, heterogeneous approach. Future work on the tool comprises a better integration of the mapping assistant into the user interface of Xtext-generated textual editors. Furthermore, work in progress addresses extensions to the MoDisco framework in order to provide support for other programming languages like C++ or C#.

REFERENCES

[1] P. Clements and L. Northrop, *Software Product Lines: Practices and Patterns*, Boston, MA, 2001.

[2] K. Pohl, G. Böckle, and F. van der Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer Verlag, 2005.

[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Carnegie-Mellon University, Software Engineering Institute, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[4] S. Zschaler, P. Sánchez, J. Santos, M. Alférez, A. Rashid, L. Fuentes, A. Moreira, J. Araújo, and U. Kulesza, "VML* - A Family of Languages for Variability Management in Software Product Lines," in *Software Language Engineering*, ser. Lecture Notes in Computer Science, M. van den Brand, D. Gaevic, and J. Gray, Eds. Denver, CO, USA: Springer Berlin / Heidelberg, 2010, vol. 5969, pp. 82–102.

[5] J. Whittle, P. Jayaraman, A. Elkhodary, A. Moreira, and J. Arajo, "MATA: A Unified Approach for Composing UML Aspect Models Based on Graph Transformation," in *Transactions on Aspect-Oriented Software Development VI*, ser. Lecture Notes in Computer Science, S. Katz, H. Ossher, R. France, and J.-M. Jzquel, Eds. Springer Berlin / Heidelberg, 2009, vol. 5560, pp. 191–237.

[6] F. Heidenreich, J. Kopcsek, and C. Wende, "FeatureMapper: Mapping features to models," in *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*, Leipzig, Germany, May 2008, pp. 943–944.

[7] C. Kästner, S. Apel, S. Trujillo, M. Kuhlemann, and D. S. Batory, "Guaranteeing syntactic correctness for all product line variants: A language-independent approach," in *TOOLS (47)*, ser. Lecture Notes in Business Information Processing, M. Oriol and B. Meyer, Eds., vol. 33. Springer, 2009, pp. 175–194.

[8] T. Buchmann and F. Schwägerl, "FAMILE: tool support for evolving model-driven product lines," in *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications*, ser. CEUR WS, H. Störrle, G. Botterweck, M. Bourdells, D. Kolovos, R. Paige, E. Roubtsova, J. Rubin, and J.-P. Tolvanen, Eds. Building 321, DK-2800 Kongens Lyngby: Technical University of Denmark (DTU), Jul. 2012, pp. 59–62.

[9] T. Buchmann and F. Schwägerl, "Ensuring well-formedness of configured domain models in model-driven product lines based on negative variability," in *Proceedings of the 4th International Workshop on Feature-Oriented Software Development*, ser. FOSD 2012. New York, NY, USA: ACM, 2012, pp. 37–44.

[10] M. Völter, T. Stahl, J. Bettin, A. Haase, and S. Helsen, *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons, 2006.

[11] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks, *EMF Eclipse Modeling Framework*, 2nd ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[12] OMG, *Meta Object Facility (MOF) Core*, formal/2011-08-07 ed., Object Management Group, Needham, MA, Aug. 2011.

[13] ——, *UML Superstructure*, formal/2011-08-06 ed., Object Management Group, Needham, MA, Aug. 2011.

[14] T. Buchmann, A. Dotor, and B. Westfechtel, "Mod2-scm: A model-driven product line for software configuration management systems," *Information and Software Technology*, 2012, http://dx.doi.org/10.1016/j.infsof.2012.07.010. [Online]. Available: http://dx.doi.org/10.1016/j.infsof.2012.07.010

[15] R. C. Gronback, *Eclipse Modeling Project: A Domain-Specific Language (DSL) Toolkit*, 1st ed., ser. The Eclipse Series. Boston, MA: Addison-Wesley, 2009.

[16] "fmp2rsm project," http://gsd.uwaterloo.ca/fmp2rsm, accessed: 2014-07-15.

[17] M. Antkiewicz and K. Czarnecki, "FeaturePlugin: Feature modeling plug-in for Eclipse," in *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange (eclipse'04)*, New York, NY, 2004, pp. 67–72.

[18] G. Taentzer, "AGG: A Graph Transformation Environment for Modeling and Validation of Software," in *Applications of Graph Transformations with Industrial Relevance*, ser. Lecture Notes in Computer Science, J. Pfaltz, M. Nagl, and B. Böhlen, Eds. Charlottesville, VA, USA: Springer Berlin / Heidelberg, 2004, vol. 3062, pp. 446–453.

[19] B. W. Kernighan, *The C Programming Language*, 2nd ed., D. M. Ritchie, Ed. Prentice Hall Professional Technical Reference, 1988.

[20] H. Bruneliere, J. Cabot, F. Jouault, and F. Madiot, "MoDisco: a generic and extensible framework for model driven reverse engineering," in *Proceedings of the IEEE/ACM International Conference on Automated software engineering (ASE 2010)*, Antwerp, Belgium, 2010, pp. 173–174.

[21] S. Bühne, K. Lauenroth, and K. Pohl, "Modelling requirements variability across product lines," in *RE*. IEEE Computer Society, 2005, pp. 41–52.

[22] D. Dhungana, D. Seichter, G. Botterweck, R. Rabiser, P. Grünbacher, D. Benavides, and J. A. Galindo, "Configuration of multi product lines by bridging heterogeneous variability modeling approaches," in *SPLC*, E. S. de Almeida, T. Kishi, C. Schwanninger, I. John, and K. Schmid, Eds. IEEE, 2011, pp. 120–129.

[23] K. Czarnecki, S. Helsen, and U. W. Eisenecker, "Formalizing cardinality-based feature models and their specialization," *Software Process: Improvement and Practice*, vol. 10, no. 1, pp. 7–29, 2005.

[24] F. Heidenreich, "Towards systematic ensuring well-formedness of software product lines," in *Proceedings of the 1st Workshop on Feature-Oriented Software Development*. Denver, CO, USA: ACM, Oct. 2009, pp. 69–74.

[25] "Eclipse UML2 Project," http://www.eclipse.org/modeling/mdt/?project=uml2, accessed: 2014-07-15.

[26] "Xtext project," http://www.eclipse.org/Xtext, accessed: 2014-07-15.

[27] "EMFText Project," http://www.emftext.org, accessed: 2014-07-15.

[28] "Acceleo project," http://www.eclipse.org/acceleo, accessed: 2014-07-15.

[29] "MWE2 Project," http://www.eclipse.org/modeling/emft/?project=mwe, accessed: 2014-07-15.

[30] "Xtend project," http://www.eclipse.org/xtend, accessed: 2014-07-15.

[31] J. Baumgartl, T. Buchmann, D. Henrich, and B. Westfechtel, "Towards easy robot programming: Using dsls, code generators and software product lines," in *Proceedings of the 8th International Conference on Software Paradigm Trends (ICSOFT 2013)*, J. Cordeiro, D. Marca, and M. van Sinderen, Eds. ScitePress, Jul. 2013, pp. 548–554.

[32] T. Buchmann, J. Baumgartl, D. Henrich, and B. Westfechtel, "Towards a domain-specific language for pick-and-place applications," in *Proceedings of the Fourth International Workshop on Domain-Specific Languages and Models for Robotic Systems (DSLRob 2013).*, U. P. S. Christian Schlegel and S. Stinckwich, Eds. arXiv.org, 2013. [Online]. Available: http://arxiv.org/abs/1401.1376

 ISBN: 978-1-61208-367-4