Developing Heterogeneous Software Product Lines with FAMILE – a Model-Driven Approach

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

***Abstract*—Model-Driven Software Development and Software Product Line Engineering are independent disciplines, which both promise less development effort and increased software quality. While Model-Driven Software Development relies on raising the level of abstraction and automatic code generation, 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 Software Product Line Engineering are discussed. As a proof of concept, an extension to the model-driven tool chain FAMILE is presented, which supports mapping of features to different types of artefacts in heterogeneous software projects at the same time. The added value of the approach is presented by an example product line, which has been developed in a strictly model-driven way using FAMILE.**

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

I. INTRODUCTION

This article is an extended version of an ICSEA 2014 conference paper [1]. It contains a deeper insight into the tool FAMILE and the adaptations, which enable the management of heterogeneous software product lines. It also presents the relevant ideas using a concrete example.

Model-Driven Software Engineering (MDSE) [2] 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) [3] 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 [4]. 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 a posteriori modifications to the generated source code.

Software Product Line Engineering (SPLE) [5][6] 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* [7], whereas *feature configurations* describe the characteristics of particular products by selecting or deselecting 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.

To realize variability in SPLE, two distinct approaches exist: In approaches based upon *positive variability*, product-specific artefacts are built around a common core [8][9]. *Composition* techniques are used to derive products. In approaches based on *negative variability*, a *superimposition* of all variants is created — a *multi-variant product*. The derivation of products is achieved by removing all fragments of artefacts implementing features that are *not* contained in the specific feature configuration [10][11]. While approaches based on positive variability typically require new languages, negative variability can be applied to existing ones by means of using preprocessor like tools. Thus, approaches based on negative variability can easily be applied to already existing software artefacts. The tool chain "Features and Mappings in Lucid Evolution" (FAMILE) [12][13], which is used in this paper, belongs to the latter category.

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 [7] are used in SPLE to capture the commonalities and differences of a product line, whereas Unified Modeling Language (UML) models [14] 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 on a higher level of abstraction compared to traditional software product line approaches operating on the source code level. By this integration, an additional increase in productivity is achieved. In the special case of negative

variability, the platform is provided as a *multi-variant domain model*. The upcoming MDPLE approach has been successfully applied in several case studies, including MOD2-SCM [15], a model-driven product line for software configuration systems.

In this paper, requirements, concepts and limitations of tool support for *heterogeneous* software product lines (HSPLs) 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, (d) artefacts are connected by both explicit and conceptual links, and (e) 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 tool chain FAMILE as a proof of concept. The practical value of the new approach is shown by developing a heterogeneous product line for editors for *graphs*, which are distinguished by properties such as the editor type (tree or graphical) or the types of graphs that may be edited (weighted, directed, and/or colored graphs, etc.).

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 III. A brief introduction of the running example is given in Section IV, while Section V explains the new concepts introduced for the support of heterogeneous product lines. In Section VI, the example is revisited in order to demonstrate the heterogeneous extension to the MDPLE tool chain FAMILE on the graph product line, which has been modeled using Eclipse Modeling Technology (EMF and the Graphical Modeling Framework (GMF) [16]). Related work is discussed in Section VII, while Section VIII concludes the paper and outlines future work.

Both the tool chain and the running example project may be retrieved via an Eclipse update site (http://btn1x4.inf.uni-bayreuth.de/famile2/update).

II. State of the Art, 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 VII). 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) In the case of model-driven engineering, there exist conceptual links between different artefact types. For example, when adding variability to a certain modeling construct, the corresponding generated source code fragment needs to be provided with the same variability information in order to keep the product line consistent.

(e) 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 artefacts 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) **Maintenance of conceptual links** The presented approach recognizes dependencies between heterogeneous artefacts even in case they are not modeled explicitly. This is true, e.g., for the concrete Java syntax, which may be provided with variability information, too. Internally, the corresponding concrete syntax fragment is mapped to an abstract syntax tree, which is invisible to the user.

(e) **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 [12][13]. 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

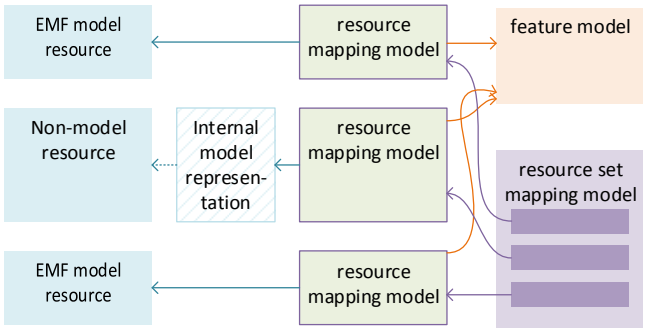

Figure 1. Conceptual mapping of models and non-model artefacts in the presented approach.

projects may be managed. Internally, all artefacts, even plain text and XML files, are represented as EMF models.

For each resource that is subject to variability, a single-resource mapping model (i.e., a model that maintains traceability links between the variability model and the multi-variant domain model) is created, which may be managed by the existing FAMILE core. The consistency between those different mapping models is maintained by an additional *resource set mapping model*. Figure 1 shows how single resource and resource set mapping models are used in order to manage a heterogeneous Eclipse project. In Section VI-D, the interplay between the different resource types is discussed in detail.

III. 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 tool chain [12][13] is presented. FAMILE is tailored towards software product line development processes that distinguish between domain and application engineering [5][6]. *Domain engineering* is dedicated to analyzing the domain and capturing the results in a *feature model*, which describes commonalities and differences thereof. Furthermore, an implementation – the *multi-variant domain model* – is provided as a result of this phase, which is then used during *application engineering* to derive application specific products. Figure 2 depicts the software product line process.

Current MDPLE tools – in particular, FAMILE – support this process by assisting 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* [7]. For feature models, several extensions such as cardinality-based feature modeling [17] 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.

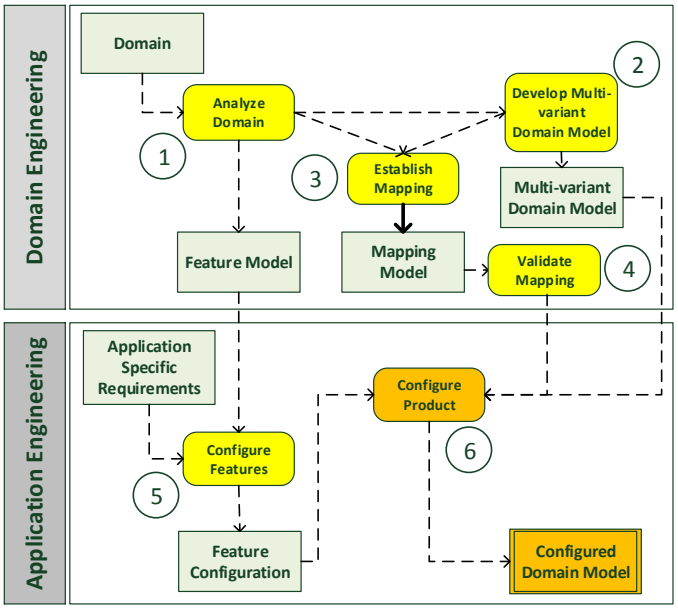

Figure 2. The software product line process supported by the state-of-the-art tool FAMILE.

3) **Mapping features to model elements** In order to define which parts of the domain model realize which feature (or which 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 of a single resource. A feature expression is a logical combination of features, for which FAMILE provides a dedicated textual language (FEL, Feature Expression Language). Modelers can either assign feature expressions by drag-and-drop or by selecting a model element in the editor and textually entering the expression [12].
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 artefacts of 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 [17][18]. FAMILE introduces the concepts of *surrogates* and *propagation strategies* [13] 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

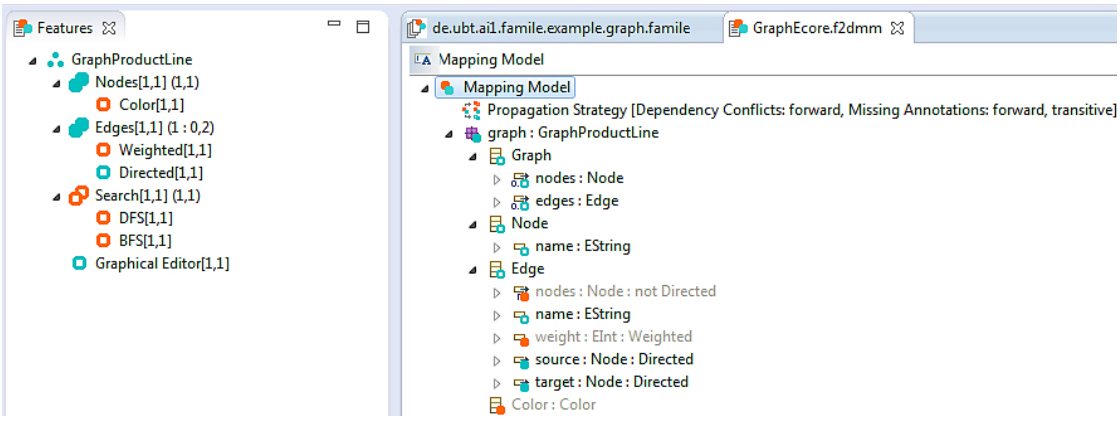

Figure 3. Screenshot of the F2DMM mapping model editor showing the multi-variant domain model of the (homogeneous) graph product line.

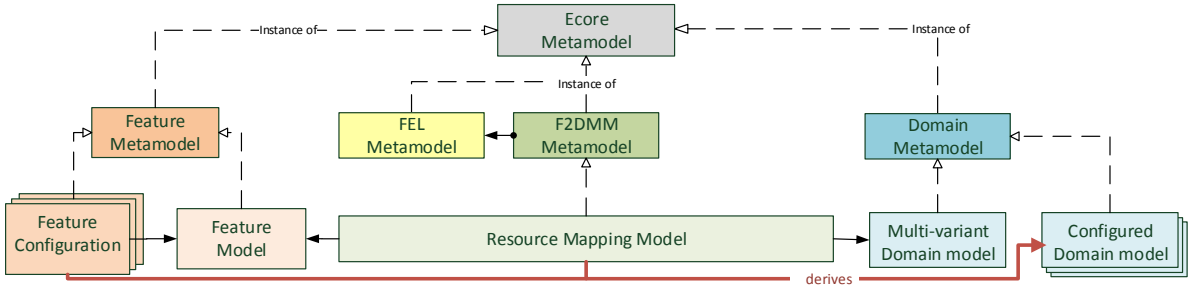

Figure 4. Metamodels and models involved in the original version of FAMILE. Different models are used to map a single-resource multi-variant domain model. All metamodels are based on Ecore.

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 current feature configuration. In homogeneous MDPLE tools, the result of this operation is a product-specific single-resource model represented as an instance of the (previously fixed) domain metamodel.

IV. 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 V demonstrates how heterogeneous project support is added to the tool chain. 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 [19].

FAMILE itself has been developed using EMF as its technological foundation. A model-driven software product line developed with FAMILE is spread over multiple EMF resources, which are instances of multiple metamodels (cf. Figure 4): Feature models and configurations share a common metamodel that supports cardinality-based feature modeling. The (single-resource) F2DMM mapping model describes how domain model elements are mapped to features. The domain model is an 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 [12].

With the F2DMM editor (see Figure 3), 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 (in this case, the domain model is an Ecore class diagram). Using the reflective EMF editing mechanism [3], 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 shown in Figure 3 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* and *deselected* are represented in cyan and orange.

The example feature configuration shown in Figure 3 represents a directed graph (with uncolored nodes and unweighted edges) that realizes neither depth-first search nor breadth-first search.

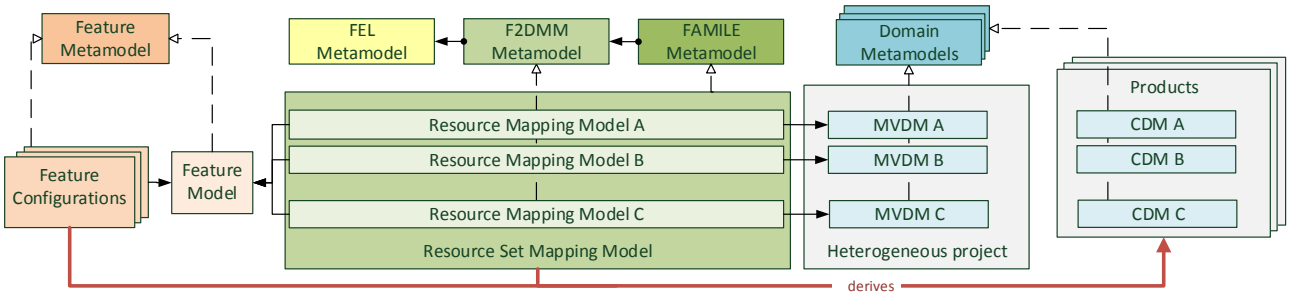

Figure 5. Metamodels and models involved in the extension of FAMILE. Abbreviations: MVDM = multi-variant domain model; CDM = configured domain model.

Considering a (model-driven) software product line as a homogeneous artefact causes a considerable amount of limitations. When referring to the graph example, it is obvious that, although being the core artefact of a model-driven project, the metamodel is not "everything". Using only the metamodel, one is not able to express several behavioral aspects (e.g., the implementation of generated method bodies) or details of representation (e.g., tree or diagram editors), only to name a few. Thus, it is necessary to include further resources into the product line. In the subsequent section, the conceptual and technical prerequisites for heterogeneous SPL support are discussed, before the running example is revisited in Section VI, where the platform will be specified in a greater level of detail. Furthermore, the underlying feature model will be defined more precisely.

V. 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, design, implementation). As stated in the introduction, it is assumed that all project artefacts may be expressed using EMF.

Figure 5 shows the conceptual overview of the new, heterogeneous version of the FAMILE tool chain. A resource set mapping model is an instance of the FAMILE metamodel and wraps different single-resource F2DMM model instances, which are used for mapping features to the different (heterogeneous) multi-variant domain model instances. A resource set mapping model references a given feature model and one out of an arbitrary number of corresponding feature configurations. Features are mapped to the corresponding domain artefacts by using a separate mapping model per resource.

In Subsection V-A, the new FAMILE metamodel will be presented as the core of the heterogeneous extension. Subsection V-B explains how non-model artefacts are mapped to EMF models, which is a technical necessity in order to manage them in a FAMILE product line. Subsection V-C will present additional user interface components that ease heterogeneous SPL development.

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 [12]. While this approach works well for projects with only one domain metamodel, it is obvious that heterogeneous projects, e.g., a GMF project, cannot be handled this way. Furthermore, even in non-heterogeneous projects, a model might be split up into different resources to better cope with size and/or complexity. In order to support multiple (EMF-based) resources of potentially different types, the new FAMILE model shown in Figure 6 wraps several instances of the F2DMM metamodel, which still constitutes the core of the extended tool chain.

The FAMILE metamodel (cf. Figure 6) 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 [13]). FAMILE takes care that global project parameters are kept consistent within different resource mappings of the same heterogeneous product line.

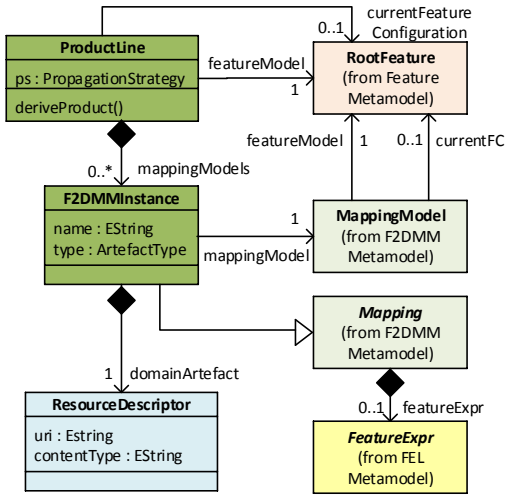

Figure 6. The FAMILE metamodel, which is designed to support heterogeneous software product lines.

A single resource mapping model, which refers to exactly one mapped EMF 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 [12]. Thus, variability on a coarse-grained level (i.e., on the level of resources) is enabled. 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 VI, elements of an Ecore model are referenced by a corresponding GMF mapping model located in a different resource. Please note: from the viewpoint of FAMILE, the GMF mapping model is just an ordinary artefact that is also based on the Ecore metamodel. During product derivation, these links are detected and resolved automatically in order to meet the requirement of referential integrity across multiple resources.

B. Interpreting Non-Model Artifacts as EMF Instances

EMF and its metamodel Ecore are wide-spread in the Eclipse community, thus a large number of potential domain models is addressed by relying on EMF models as SPL artefacts. A (non-exhaustive) list comprises of course Ecore class diagrams, Eclipse UML models [20], Xtext [21] / EMFText [22] grammars and documents, GMF models [16], Acceleo source code generation templates [23], MWE2 Workflow files [24], Xtend specifications [25], 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 [26] 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.

As explained before, the new framework considers all artefacts part of the platform as models. In this subsection, it is explained how particular resource types can be interpreted as EMF instances. The new FAMILE implementation allows for different extensions for specific heterogeneous resource types. For each resource type, different user interface extensions are provided, in order to allow the user to work at an adequate level of abstraction. So far, five resource types have been implemented.

- **XMI-serialized EMF models**. These are ordinary models, which may be mapped using the F2DMM editor, which represents the model as a tree.
- **Xtext models** may be mapped using their abstract syntax tree, since Xtext files implement EMF's `Resource` interface. Currently, it is not possible to add feature expressions to a text selection. It is planned to add this feature in future.
- **Java files**. Invisibly to the user, Java source code files are converted to EMF models using the MoDisco framework. The user may select a source code fragment and invoke the command *Annotate Java element*. Behind the scenes, the mapping is applied to the underlying MoDisco discovery model.
- **XML files**. Similarly to Java files, MoDisco offers a discoverer for XML. Currently, only the concrete syntax may be mapped using the standard F2DMM single resource mapping editor.
- **Unstructured text**. For text files that do not fit any of the categories above, the new FAMILE version includes a fall-back representation. Text files are represented as an instance of a simple text meta-model, which only consists of a sequence of text lines. This way, single text lines may be assigned with feature expressions using the F2DMM editor.

In the running example and in the screencast, the focus lies on two resource types, being XMI-serialized EMF models and Java files.

C. User Interface

The user interface has been extended to support heterogeneous software product lines. An additional 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 III, the extended FAMILE framework supports the following user interactions (see also example in Section VI):

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. Of course, it is also possible to reuse an existing feature model. 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.

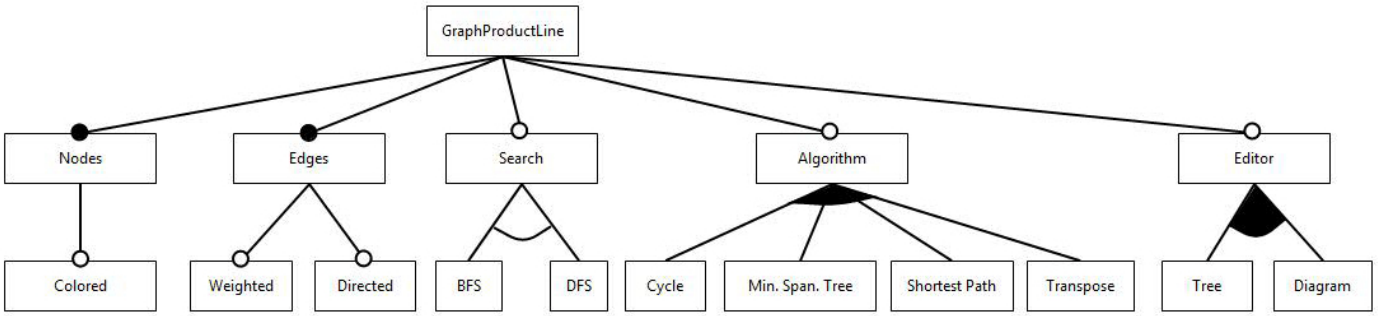

Figure 7. Feature diagram for the graph product line.

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 the current configuration, which will restrict the visible elements/resources in both the resource mapping and the resource set mapping 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, single-resource product derivation will be applied to each mapping model covering a resource, keeping cross-resource links consistent. Resources that are not wrapped by any F2DMM instance or that are not annotated with FEL expressions will be copied without any further restriction.

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

In Section III, the development process that is commonly used in software product line engineering has been briefly sketched. In the following, it is demonstrated how to use FAMILE for model-driven product line engineering with this process, using the running example introduced in Section IV. In order to demonstrate the use of the heterogeneous extensions to FAMILE, the example product line is enriched with editors for the underlying graph data structure. In the domain engineering sub-process, the variability is captured in a variability model and a platform is established, which consists of not only the graph metamodel, but also of resources that control the aspects of representation as well as behavioral components – i.e., graph algorithms – which are described at the level of Java source code.

This section is organized as follows. The (heterogeneous) platform and the variability model are introduced in Subsection VI-A. In Subsection VI-B, the mapping between platform and variability model is described, with a focus on heterogeneous resources. Subsection VI-C refers to the use of two existing FAMILE concepts, being *alternative mappings* and *propagation strategies*. In Subsection VI-D, the created mapping is investigated with a focus on the interplay between heterogeneous artefacts, i.e., different types of links among them. The transition from domain engineering to application engineering – i.e., product derivation – is subject of Subsection VI-E. Subsection VI-F gives an outlook for a more fine-grained specification of variability among the platform.

A. Platform and Variability Model

The platform of the product line contains the following types of artefacts:

- **Ecore Model:** An Ecore model is used to describe the static structure of the product line for graph libraries. EMF allows to generate Java code for the model as well as code for a tree editor from the Ecore specification.
- **Java Code:** As model-driven development using EMF only allows to model the static structure of a software system, it is necessary to supply the corresponding method bodies using hand-written Java code. These method bodies also contain variability, which needs to be managed by FAMILE.
- **GMF Models:** The (optional) graphical editors for the graph product line have been developed using the Eclipse Graphical Modeling Framework (GMF) [16].

The variability of the graph product line is captured in a feature model. The corresponding feature diagram is depicted in Figure 7. A graph consists of *Nodes* and *Edges*. The graphical notation uses filled dots for mandatory and unfilled ones for optional features. Nodes may be *Colored* while edges may be *Weighted* and/or *Directed*. Optional components of the graph product line are the *Search* strategy (e.g., depth-first search or breadth-first search), different *Algorithm*s and *Editor*s. Child elements of feature groups are depicted with arcs. Two different types of group relationships are possible: "inclusive or" (filled arc) and "exclusive or" (unfilled arc) of child elements. While the different search strategies are mutually exclusive, the algorithms and the editor children may be selected in arbitrary combinations. Please note that there are also dependencies between features, which cannot

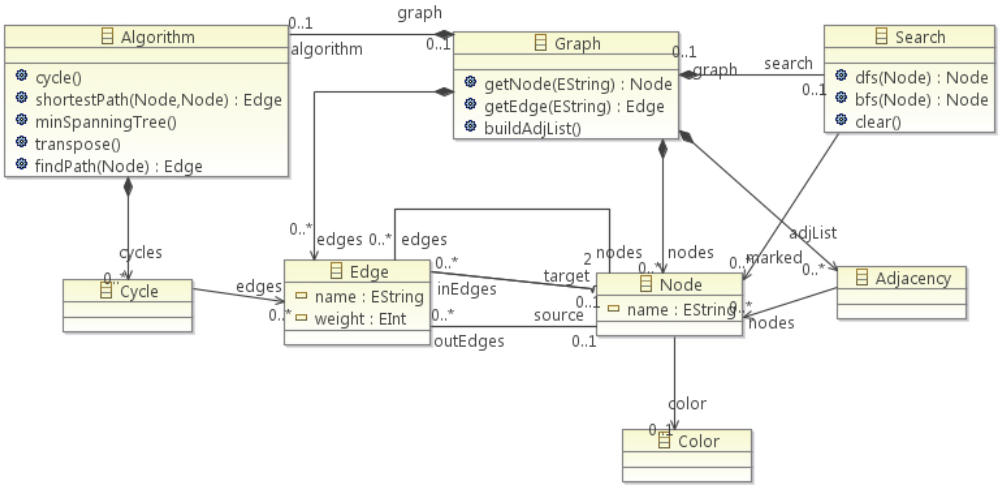

Figure 8. Ecore model for the graph product line.

be displayed in the feature diagram. Features may also define cross-tree constraints indicating feature inclusion or exclusion once a certain feature is selected. E.g., the algorithm to detect cycles in the graph requires *Directed* edges. Furthermore, the calculation of a *Shortest Path* requires *Weighted* edges.

Figure 8 depicts the multi-variant domain model of the graph product line. Following the model-driven approach, an object-oriented decomposition of the underlying data structure is applied: A Graph contains Nodes and Edges. Furthermore, it may contain a Search strategy and Algorithms operating on the graph data structure. For performance reasons, the data structure may be converted into an Adjacency list, to speed up certain algorithms. As the model depicted in Figure 8 is the superimposition of all variants, the relation between nodes and edges is expressed in multiple ways: (1) In case of undirected graphs, an edge is used to simply connect two nodes, expressed by the reference nodes. (2) Directed graphs on the other hand demand for a distinction of the corresponding start and end nodes of an edge. This fact is expressed by two single-valued references named source and target.

As stated above, Ecore only allows for structural modeling, i.e., it does not provide support to model method bodies. Thus, the standard EMF development process [3] demands for a manual specification of an EOperation's body by completing the generated source code. In the example, hand-written Java source code for all operations contained in the class diagram shown in Figure 8 has been supplied. A small cut-out of a method implementation for the class Search is shown in Figure 9. In the corresponding Ecore model (cf. Figure 8), the Search class defines three EOperations. While the EMF code generation only creates Java code for the method head, the body implementation depicted in Figure 9 was supplied manually. In this case, the method implementation also contains variability as the corresponding references between nodes and edges are different depending on the presence or absence of the feature *Directed* in the current feature configuration. Please note that the level of granularity supported by FAMILE's variability annotations is arbitrary, ranging from single Java fragments, over statements, blocks, methods or even classes and packages.

```
246     /**
247      * <!-- begin-user-doc -->
248      * <!-- end-user-doc -->
249      * @generated NOT
250      */
251     public EList<Node> dfs(Node node) {
252         if (getMarked().contains(node))
253             return getMarked();
254         getMarked().add(node);
255
256         // undirected graphs
257         for (Edge e : node.getEdges()) {
258             for (Node n : e.getNodes()) {
259                 if (n != node)
260                     dfs(n);
261             }
262         }
263
264         // directed graphs
265         for (Edge e : node.getOutEdges()) {
266             dfs(e.getTarget());
267         }
268
269         return getMarked();
270     }
```

Figure 9. Example for method bodies written in Java.

The product line for graphs also allows for different types of editors, which may be used to manipulate the graph data structure. As shown in the corresponding feature model in Figure 7, valid product configurations may either have no editor at all, a tree editor, a graphical editor or they may even contain both types of editor. While the tree editor may be automatically generated from the Ecore model, a graphical editor requires additional information. The Graphical Modeling Framework (GMF) [16], which is also part of the Eclipse Modeling Platform, allows for a creation of graphical editors in a model-driven way. Generating graphical editors with GMF requires the definition of three additional models:

1) **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, by specifying the respective

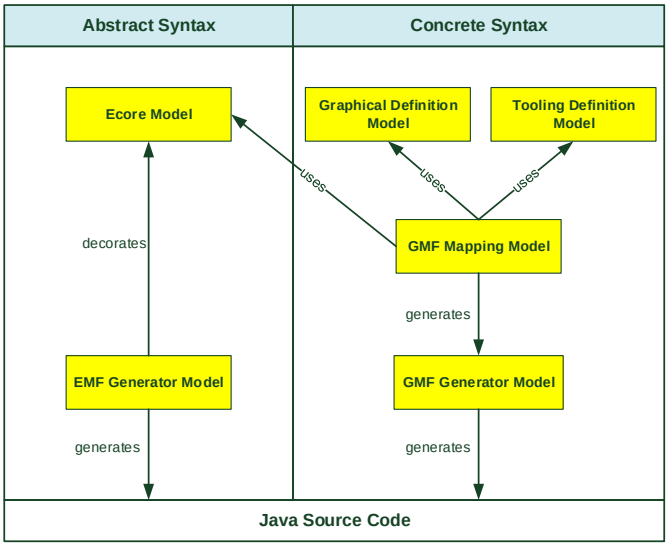

Figure 10. Models involved in the GMF development process.

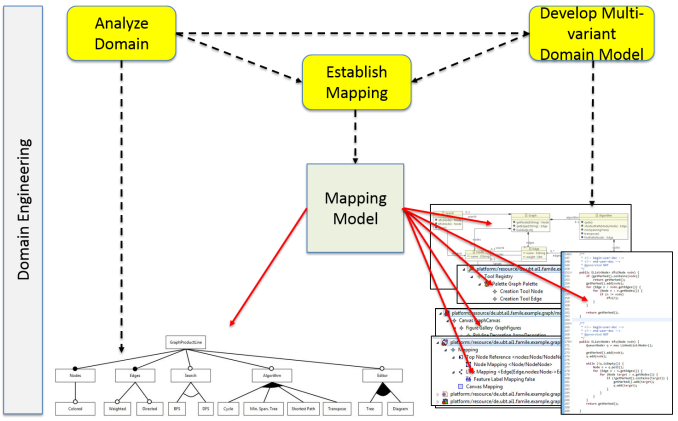

Figure 11. Domain Engineering Process in the graph product line example.

shapes.

2) **GMFTool (Tooling Definition Model)** Every graphical editor in Eclipse, which is based on the Graphical Editing Framework (GEF), uses a so called palette to drag new diagram elements to the drawing 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.
3) **GMFMap (GMF Mapping Model)** The models described above are combined in the GMF mapping model. In this model, a relation between abstract syntax (Ecore), the graphical notation (GMFGraph) and the tooling definition model (GMFTool) is established. The GMF Mapping Model is then automatically transformed into a generator model from which the Eclipse plugin for the graphical editor is generated. 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 tool chain. From the viewpoint of FAMILE, all the models that have been described here are just ordinary artefacts that may contain variability.

Figure 10 depicts the different models involved in the GMF development process. All models are instances of Ecore-based metamodels, and can thus be used easily with the FAMILE tool chain. The abstract syntax of a GMF-based editor is defined by an Ecore model (in this case, the superimposed graph metamodel shown in Figure 8), 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 complementing this paper does not cover the definition of the models mentioned below as it just focuses on how to use FAMILE with these types of artefacts.

B. Mapping Heterogeneous Artefacts

Figure 11 depicts the domain engineering phase in model-driven software product lines developed with FAMILE. First, the variability has to be captured in a feature model. The features are then implemented using the appropriate modeling languages. After that, a mapping between features and their corresponding implementation fragments has to be established. In the following subsection, the necessary steps are described from the tool perspective.

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. Variability modeling, i.e., capturing the commonalities and differences of the products in the product line, is performed during the domain analysis step. The result of this development task is a feature model. In the example, the feature model shown in Figure 7 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, five F2DMM instances have been defined, one for each EMF/GMF resource mentioned above and one for the Java source code — the underlying MoDisco AST representation consists of a single EMF resource although the source code is distributed over multiple packages and compilation units in the physical file system. Please note that for the EMF Generator model and the GMF Generator model (cf. Figure 10) no F2DMM instances have been defined, as they do not contain variability in this example. Furthermore, those models have been automatically derived from the Ecore model and the GMF mapping model and only contain information for the code generators. Thus, they can easily be created again after product derivation.

Since the graph metamodel as well as all GMF artefacts are ordinary EMF models, their mapping is done in a straightforward way using the single resource F2DMM mapping editor (see Section IV). For mapping the source code, the new concrete syntax connector of the FAMILE extension may be used.

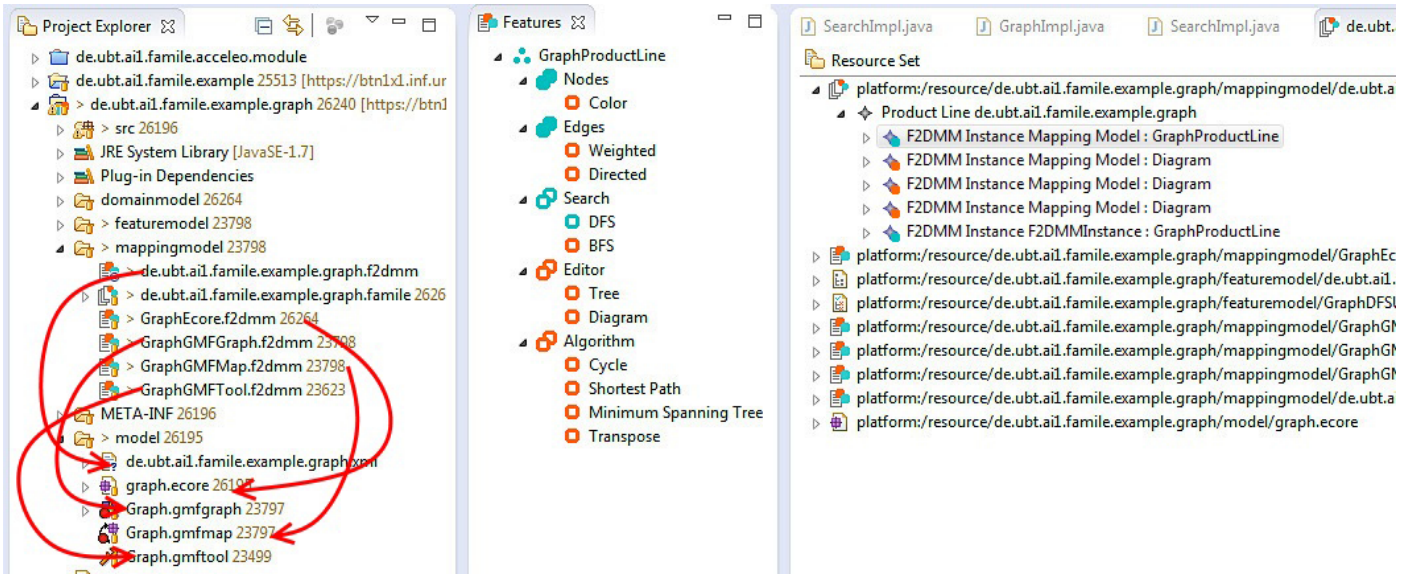

Figure 12. Screenshot of the FAMILE resource set mapping editor. The left pane shows the feature model and feature configuration. In the main pane, the mapping for contents of the resource set are shown.

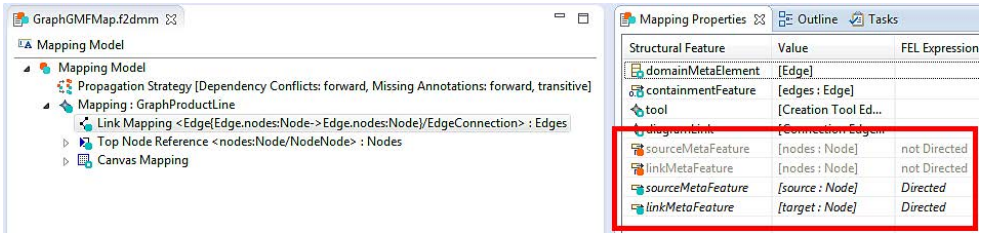

Figure 13. Usage of alternative mappings. The red box depicts where elements of the multi-variant domain model have been virtually extended by alternative mapping values (in italics).

It allows to assign feature expressions to parts of the hand-written method bodies directly in the textual representation of the Java source code. The body of the method `dfs` shown in Figure 9 is annotated as follows (see also screencast) in order to provide an optimized depth-first search for directed graphs. First, the user selects lines `257` until `262` in the editor. Next, he/she invokes the command `Annotate Java element` from the context menu. The entered feature expression "Directed" will be assigned to the corresponding AST element in the MoDisco model in the background. Similarly, the `for`-loop consisting of lines `265` until `267` is restricted by the feature expression "not Directed". As a consequence, derived products will only contain one of the two loops to calculate the successors for a depth-first search, but never both of them.

Figure 12 depicts the state of the example project after corresponding F2DMM instances have been created for the models mentioned above. 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, mapped resources may also be annotated with feature expressions. For the example, a feature called *Editor* has been introduced in order to make the visualization (tree editor vs. diagram editor) of the graph variable. In case the feature *Diagram* is deselected in a feature 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 Diagram, as shown in Figure 12.

C. Usage of Alternative Mappings and Propagation Strategies

Figure 3 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 artefacts (GMFGraph, GMFTool, GMFMap and Java code) are created. Each model file contains a superimposition of all possible variants. Common approaches using negative variability suffer from restrictions imposed by the used 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 (cf. Figure 13). 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 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 the 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 (cf. Figure 8), which have to be used in the GMFMap model instead in case the feature *Directed* is chosen. Please note that the reason why in this example, alternative mappings are necessary in the GMF mapping model but not in the semantic model is the fact that the references `nodes`, `source`, `target` can be defined simultaneously in the Ecore model. The GMF mapping model, however, requires the applied occurrence of exactly one semantic model element here, which cannot be realized by a single multi-variant model. Figure 13 depicts how this has been solved using FAMILE's alternative mappings [12], which can virtually extend the multi-variant model and thus mitigate the limited variability of the used domain metamodels.

It is aimed to keep the annotation effort small for the users of the tool chain. This can be achieved by so called propagation strategies, which avoid the necessity of repeated feature annotations. In the graph product line example, the propagation strategy *forward* is used throughout all mapped resources. As its name says, this strategy propagates selection states of mapped elements *along the direction of dependency*. For instance, each EMF object has an existential dependency to its `eContainer`. As a consequence, in case the presence of an element is restricted by a specific feature expression, this restriction also holds for all contained elements, making repeated annotations unnecessary. For additional details on propagation strategies, the reader is referred to [13].

D. Interplay Between Different Model Types

Through the use of two external frameworks, namely GMF and MoDisco, the resulting example product line is highly heterogeneous. Figure 15 sketches the interplay between different resource types, being the domain model, the GMF models, the generated Java source code and its internal MoDisco representation, which is invisible to the end user. Between different resource types, seven external link types occur, two of which are *conceptual*, i.e., they do not occur explicitly as EMF links.

- **GMF Mapping links** emerge from the GMF mapping model and reference the domain model, the tooling model, and the graphical definition model. These links are created by GMF.
- **Conceptual CS/AS links** virtually link an abstract syntax tree element to its corresponding concrete syntax fragment in the Java source code. This link is restored automatically in case the user selects a concrete syntax fragment and invokes the command *Annotate FEL Expression*.
- **Feature expression links**: Within the feature expression of a mapping, elements of the feature model are referenced. These links are created automatically when feature expressions are specified.
- **Conceptual links between CS and feature expression**: From the user's perspective, source code elements are mapped in concrete syntax. This causes a conceptual dependency between source code fragments and features.
- **Mapping links**: Each element of the mapping model references exactly one element of the multi-variant domain model. These links are created automatically upon creation of a resource mapping model.
- **Cross-Resource Mapping links**: Among different domain models, cross-resource links may occur (see the GMF mapping model). In order to adequately connect these to the variability model, a link between the corresponding resource mappings is established. These links are created automatically during mapping model creation.
- **Resource Mapping links**. The superordinate resource set mapping model references one F2DMM instance per mapped model resource. Links of this kind are created by the user through the *Add F2DMM Instance* command.

By automating sub-tasks such as the synchronization between domain and mapping model, and the discovery of the Java source code, the larger part of the complex relationships shown in Figure 15 is not exposed to the user at all, but managed automatically "behind the curtains".

E. Product Derivation

After the domain engineering phase has been completed, the platform may be used to derive specific products from the product line in the application engineering step (cf. Figure

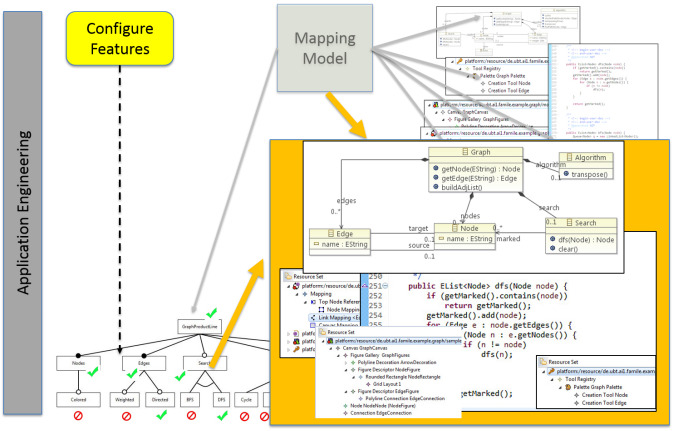

Figure 14. Application Engineering Process in the graph product line example.

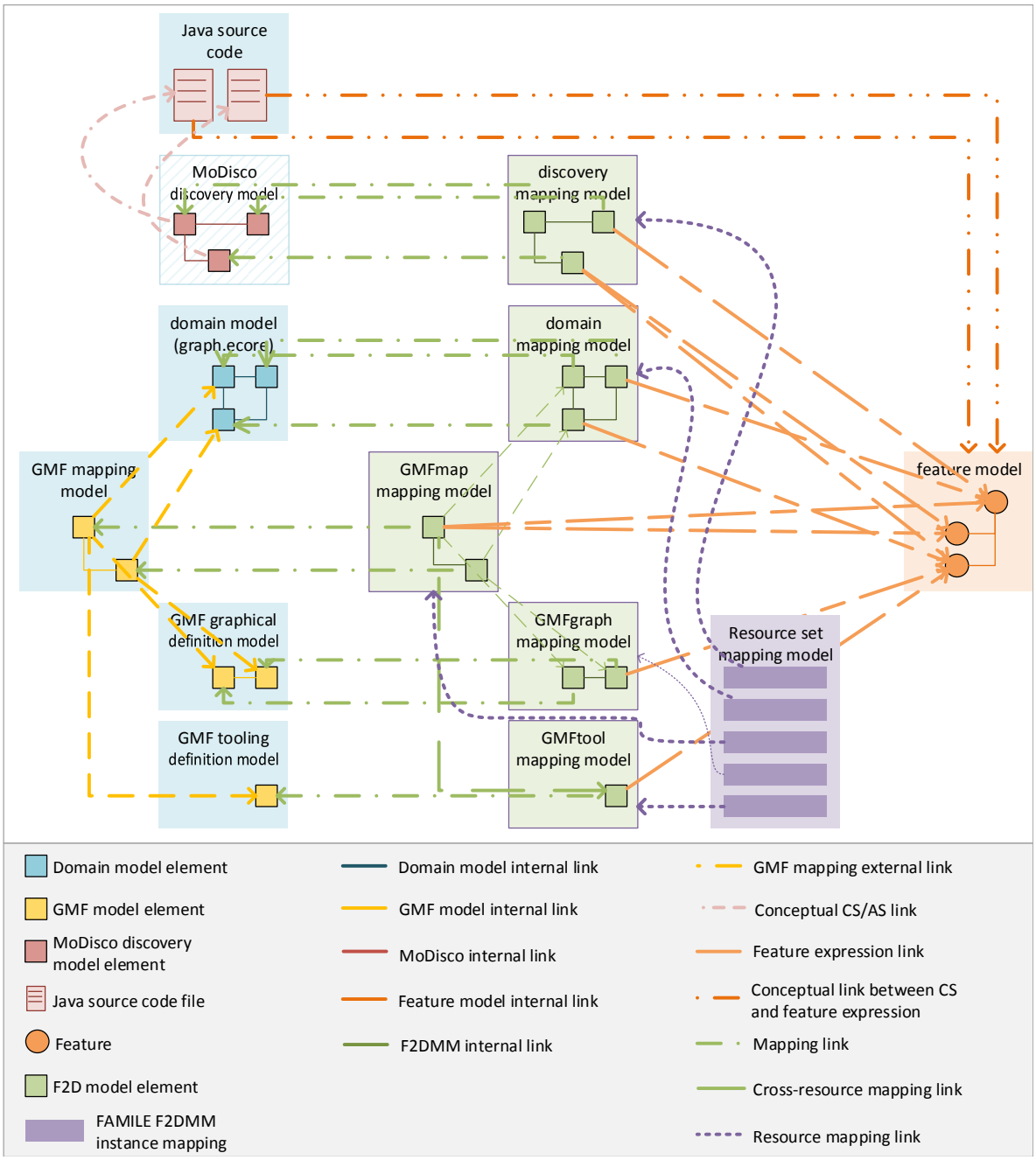

Figure 15. Interplay between resources of different types, which are created in subsequent steps of the *domain engineering* phase.

14). In the presented approach, application engineering is reduced to a simple configuration task. The user only has to specify an appropriate feature configuration, by selecting and deselecting corresponding features in the feature model, while all constraints (e.g., parent-child relationships, requires/excludes relationships) must be satisfied. The subsequent product derivation step is a fully automatic process.

In the example, a derived Eclipse project is created, which contains the required model files. Sample feature configurations are provided, which allow for a fully automatic generation of four Eclipse plugin projects, which differ from each other as follows (cf. Figure 16):

(FC1) An EMF tree editor for undirected, unweighted, uncolored graphs, traversed by depth-first search. No additional algorithms are offered.

(FC2) A GMF graphical editor for undirected, unweighted, uncolored graphs, traversed by depth-first search. No additional algorithms are offered.

(FC3) A GMF graphical editor for directed, unweighted, uncolored graphs, traversed by depth-first search. Deployed algorithms include cycle detection, as well as calculation of a minimum spanning tree and the transpose.

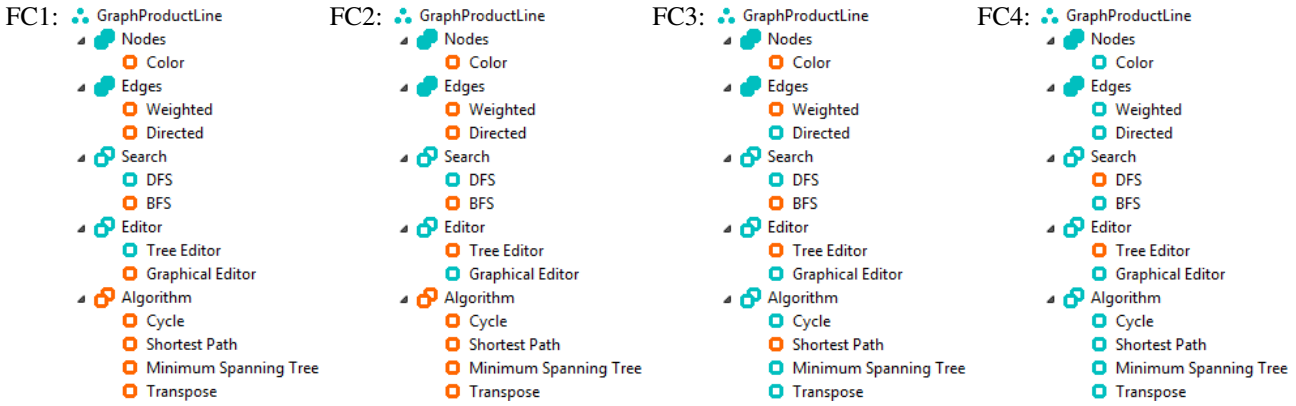

Figure 16. Four example feature configurations for instances of the graph product line.

(FC4) A GMF graphical editor for directed, weighted, colored graphs, traversed by breadth-first search. All available graph algorithms are deployed.

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 and constraints defined in the feature model.

F. Outlook: Increasing the Heterogeneity of the Project

The example described in this section has been conducted using a variety of different 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. Manual adaptations to the Java source code are managed with the help of the MoDisco framework, letting the user operate on concrete textual syntax. 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 additional non-EMF resources, for different reasons:

- The file `plugin.properties` in the Eclipse project contains language-specific UI string constants, each declared in a separate 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 V-B).
- The file `plugin.xml` defines plug-in extensions 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 MoDisco based representation for XML files (see Section V-B) may be used.

VII. 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 [12] and [13].

The tool *fmp2rsm* [27] combines FeaturePlugin [28] 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 [10] 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* [8] 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 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 VI 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 [9] is another language that 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 [29] 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 [11] is a tool for source-code based approaches. It provides a product specific view on the source code, where all source code fragments not part of the chosen configuration are omitted. The approach is similar to *#ifdef*-preprocessors known from the C programming language [30]. 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 [26] framework.

Bühne et al. [31] and Dhungana et al. [32] 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.

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 closes 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 source code or configuration files or XML documents. As a proof of concept, an implementation of an extension to the FAMILE tool chain was shown. A concrete example has been given, demonstrating the benefits of the presented approach on a concrete product line for graphs.

Usually, (model-driven) software projects do not only consist of one single model resource. In contrast, different models and metamodels as well as non-model artefacts are involved. FAMILE is able to map features to model fragments in such heterogeneous projects and also to derive consistent products. Besides the aforementioned heterogeneous support, FAMILE advances the state of the art by allowing to flexibly change the granularity of the mapping between features and the product space (project-wide or resource-wide scope). Furthermore, the tool chain also allows for the usage in model-driven projects, where parts of the software are still realized with manually written Java source code. Of course, FAMILE may be used in regular (non model-driven) Java projects as well. 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, (d) to adequately handle conceptual links between artefacts of different types, e.g., between model elements and source code fragments, and (e) 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 presented solution to heterogeneous SPLE tooling is to divide a heterogeneous software project into a set of single-resource mapping models, for which adequate MDPLE support is already available. 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. Non-model resources such as Java or XML files are automatically interpreted as EMF models, using the MoDisco framework under the hood. Furthermore, a fallback metamodel for text files is provided, which also allows to map features to those kinds of artefacts at a lower level of abstraction. A demonstration of the presented approach was given by applying the heterogeneous FAMILE tool chain to a product line for graph metamodels, including modifications of the generated source code, and editors. The resulting heterogeneous product line manages an entire Eclipse plug-in project.

Current and future work addresses a case study carried out in the field of robotics [33][34]. Although first results produced by the old (homogeneous) version of the FAMILE tool chain are very promising, it is expected that a significant gain in productivity is achieved by exploiting the new, heterogeneous approach. In this case study, the platform of the product line consists of language grammar files, code generation template files and C++ source code files.

Acknowledgments

The authors want to thank Bernhard Westfechtel for his valuable comments on the draft of this paper.

REFERENCES

[1] T. Buchmann and F. Schwägerl, "A model-driven approach to the development of heterogeneous software product lines," in *Proceedings of the Ninth International Conference on Software Engineering Advances (ICSEA 2014)*, H. Mannaert, L. Lavazza, R. Oberhauser, M. Kajko-Mattsson, and M. Gebhart, Eds. Nice, France: IARIA, 2014, pp. 300–308.

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

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

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

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

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

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

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

[15] 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

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

[17] 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.

[18] 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.

[19] R. E. Lopez-Herrejon and D. S. Batory, "A standard problem for evaluating product-line methodologies," in *Proceedings of the Third International Conference on Generative and Component-Based Software Engineering*, ser. GCSE '01. London, UK, UK: Springer-Verlag, 2001, pp. 10–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=645418.652082

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

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

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

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

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

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

[26] 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.

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

[28] 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.

[29] 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.

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

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

[32] 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.

[33] 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.

[34] 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