175% Modeling for Product-Line Evolution of Domain Artifacts

Sascha Lity, Sophia Nahrendorf, Thomas Thüm, Christoph Seidl, and Ina Schaefer
Technische Universität Braunschweig
Braunschweig, Germany
{s.lity,s.nahrendorf,t.thuem,c.seidl,i.schaefer}@tu-braunschweig.de

ABSTRACT

Software evolution is an inevitable process in the development of long-living software systems as, e.g., changes of requirements demand corresponding adaptations. For software product lines, the incorporation of evolution in the development process gets even more complex due to the vast number of potential variants and the set of reusable domain artifacts and their interrelations. To allow for the application of existing analyses also for combined dimensions of variants and versions, recent evolution-aware variability modeling techniques are insufficient for capturing both version and variant information by the same means. In this paper, we propose an extension of annotative variability modeling, also known as 150% modeling, to tackle evolution and variability by the same means. The so called 175% modeling formalism allows for the development and documentation of evolving product lines. A 175% model combines all variant-specific models of all versions of a product line, where elements are mapped to features and versions to specify which version of a variant contains the element. We discuss potential application scenarios for 175% modeling. Furthermore, we propose a bidirectional transformation between 175% and higher-order delta models to exploit the benefits of both modeling formalisms, when solely one type is available.

CCS CONCEPTS

• **Software and its engineering → Software product lines**; *Model-driven software engineering*; *Software evolution*;

KEYWORDS

Software Evolution, Software Product Lines, Variability Modeling

ACM Reference Format:
Sascha Lity, Sophia Nahrendorf, Thomas Thüm, Christoph Seidl, and Ina Schaefer. 2018. 175% Modeling for Product-Line Evolution of Domain Artifacts. In *VAMOS 2018: 12th International Workshop on Variability Modelling of Software-Intensive Systems, February 7–9, 2018, Madrid, Spain.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3168365.3168369

1 INTRODUCTION

Software evolution is an inevitable challenge not only for single-software systems but also for variant-rich software systems, e.g., software product lines [33] (SPLs). Due to changes, e.g., in requirements or standards, software has to be adapted [6, 26]. For SPLs, i.e., a family of similar software systems with explicit commonality and variability, the incorporation of evolution in their development is challenging [6]. Due to the vast number of variants of an SPL, an evolution step and its changes to reusable SPL domain artifacts may introduce inconsistencies caused, e.g., (1) by the loss of information about artifacts and their interactions, or (2) by the addition of incorrect or unintended artifact dependencies. To tackle the challenges, development techniques for evolving SPLs are required incorporating both variability and evolution as first class entities, where the application of changes to artifacts is specified and well-documented. By capturing the evolution history, those techniques facilitate reasoning about already applied evolution steps and further support planning of upcoming changes. In addition, those techniques also enable the analysis of variants and SPL versions to detect induced artifact inconsistencies to be fixed.

For an SPL, various variability modeling techniques exist [35] categorized as annotative, compositional, and transformative already supporting variability-aware analyses [41]. In annotative approaches, all variant-specific models are combined in one super model, also called 150% model, facilitating family-based analyses [41] such as conditioned model slicing [16], SPL test-suite generation [8], or SPL verification [20]. In contrast, compositional techniques specify reusable model fragments to be composed to realize a variant supporting feature-based analysis [41]. Transformational approaches such as delta modeling [9] define transformations to a core model to transform the core into a variant-specific model and facilitate, e.g., incremental model-based SPL testing [10, 25]. However, for exploiting the benefits of those modeling techniques also for SPL evolution, concepts for handling evolution need to be incorporated to enable the documentation and analysis also for SPL versions.

Existing approaches consider SPL evolution on the feature model level [31, 32, 37], by adopting delta modeling [15, 24, 38], by using aspect-oriented programming [2, 3, 13], or specify SPL evolution templates [29]. However, to the best of our knowledge no approach exists in the literature for the annotative category of variability modeling techniques such that version as well as variant information is captured together in a single model, i.e., elements are annotated with both version and variability conditions. Such a modeling concept provides an overview not only of one SPL version in time but also of the complete SPL history, e.g., to understand its past and to plan its future. In addition, the application of family-based analyses [41] also to evolving SPLs is established by taking not only variant information but also the evolution history into account to improve comprehension and development of evolving SPLs.

In this paper, we contribute the following:

(1) We propose *175% modeling* to capture evolution and variability for the development and documentation of evolving

VAMOS 2018, February 7–9, 2018, Madrid, Spain

ACM ISBN 978-1-4503-5398-4/18/02…$15.00
https://doi.org/10.1145/3168365.3168369

SPLs by the same means. 175% modeling represents an extension of 150% modeling, where an instance combines all variant-specific models of all SPL versions. To specify which elements are contained in which version of a variant, elements are not solely annotated with features but also with versions. In this paper, we abstract from feature model evolution [32] but assume that, for an SPL version, a respective feature model exists allowing for the definition of feature annotations of elements.

(2) We discuss *benefits and limitations* of 175% modeling and propose *potential application scenarios*.
(3) 175% modeling does not support, e.g., incremental change-oriented analysis of evolution steps which is crucial for SPL regression testing as, e.g., provided by higher-order delta modeling [24]. To overcome this drawback and to exploit the benefits of both formalisms, we provide a *bidirectional transformation* between 175% and higher-order delta models, where one type is used to derive the opposing type.
(4) We provide a *proof sketch* using complete induction to show the soundness of the transformation, i.e., the transformed models are equivalent w.r.t. the derivable variant-specific models for the respective SPL versions.

2 FOUNDATIONS

An SPL [33] defines a family of similar software product variants $P_{SPL} = \{p_0, \ldots, p_m\}$ sharing common and variable domain artifacts, where the commonality and variability between variants is explicitly specified in terms of *features* $F_{SPL} = \{f_1, \ldots, f_n\}$, i.e., customer-visible system functionality. Each variant $p_i \in P_{SPL}$ is characterized by a certain combination of features called *feature configuration* $F_{p_i} \subseteq F_{SPL}$. The potential combinations are restricted, e.g., by using a feature model FM_{SPL} [17], where every F_{p_i} satisfies the constraints defined by FM_{SPL}. Based on a configuration F_{p_i}, reusable domain artifacts are assembled. For the development of those artifacts and, thus, for SPL development, variability modeling techniques like 150% modeling or delta modeling are applied [35].

150% Modeling. 150% modeling captures the variability of an SPL by merging all variant-specific models in a *super model* M_{150}, where elements $e \in E_{M_{150}}$ are annotated, e.g., with a Boolean expression over features $\varphi_e \in \mathbb{B}(F_{SPL})$ as *presence condition*. By $\mathbb{B}(F_{SPL})$, we refer to the set of all Boolean expressions defined over F_{SPL}. For annotations, an *annotation function* $\alpha_{150} : E_{M_{150}} \rightarrow \mathbb{B}(F_{SPL})$ is defined, i.e., for every $e \in E_{M_{150}}$ a presence condition $\alpha_{150}(e) = \varphi_e$ exists and $\alpha_{150}(e)$ respects the constraints of FM_{SPL}. For elements that are part of all variant-specific models denoting common core functionality, the presence condition is specified as `true`. To this end, a 150% model is defined as tuple (M_{150}, α_{150}) comprising a merged super model M_{150} and the corresponding annotation function α_{150}. In general, M_{150} does not represent a valid variant-specific model. To derive a model for a variant $p_i \in P_{SPL}$, solely elements $e \in E_{M_{150}}$ are selected from M_{150} for which the presence condition φ_e is satisfied by the respective feature configuration F_{p_i}.

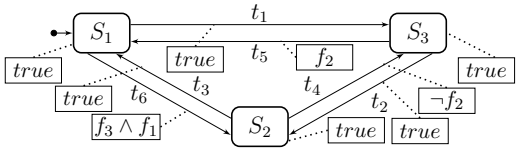


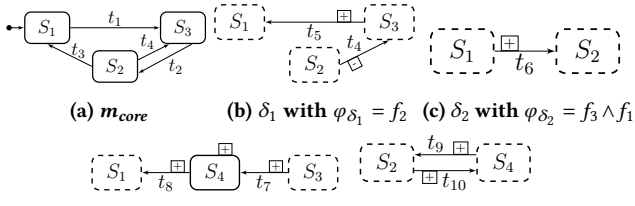
Figure 1: Sample 150% Model (M_{150}, α_{150})

Example 2.1. Consider the sample (M_{150}, α_{150}) shown in Fig. 1 specifying the behavior of a sample SPL as state machine comprising the states S_1, S_2, and S_3 as well as the transitions $t_1 - t_6$. State machine M_{150} merges all variant-specific models, where core elements are annotated with $true$, e.g., transition t_1 is contained in all variants, and otherwise with a feature expression, e.g., transition t_6 is annotated with the conjunction $f_3 \wedge f_1$ of features f_3 and f_1.

A 150% model provides an overview over the commonality and variability of all SPL variants. To also incorporate SPL evolution, we propose an extension of 150% modeling to capture both variant and version information in a so called 175% model in Sect. 3. In previous work [24], we introduced higher-order delta modeling as an extension of delta modeling [9] to also capture SPL evolution, where we focused on the differences between SPL versions. To exploit the modeling benefits of both formalisms, we present a bidirectional transformation in Sect. 4 and describe (higher-order) delta modeling in the following.

Delta Modeling. In delta modeling [9], the differences between variants are specified explicitly in terms of transformations called *deltas*. Based on a predefined *core model* m_{core} implementing a *core variant* $p_{core} \in P_{SPL}$, deltas $\delta \in \Delta_{SPL}$ are defined to transform m_{core} into a model m_i of variant $p_i \in P_{SPL}$. By Δ_{SPL}, we refer to the set of all deltas of an SPL. A *delta* $\delta = (OP_\delta, \varphi_\delta)$ captures *change operations* $OP_\delta = \{op_1, \ldots, op_k\} \subseteq OP_{SPL}$, such as additions (add e) or removals (rem e) of elements. For brevity, we abstract from the modification of elements (mod(e, e')) as they can be encoded as a removal and addition of the (modified) element. The set OP_{SPL} denotes the set of all change operations defined over the set of all elements of the current SPL, where further OP_{SPL} is a subset of the universe of all possible change operations $\mathcal{OP}$. Besides change operations, a delta comprises an *application condition* φ_δ, i.e., a Boolean expression over features. Based on a given configuration F_{p_i} for a variant p_i, for each delta $\delta_j \in \Delta_{SPL}$, its application condition is evaluated if F_{p_i} satisfies φ_{δ_j}. If the application condition of a delta is satisfied by F_{p_i}, it is applied to m_{core} to obtain m_i. To this end, we define a *delta model* $DM_{SPL} = (m_{core}, \Delta_{SPL})$ of an SPL as tuple comprising its core model and the set of applicable deltas.

Higher-Order Delta Modeling. For the evolution of an SPL defined by a DM_{SPL}, we introduced an extension of delta modeling [9] in previous work [24]. *Higher-order delta modeling* specifies the evolution of a DM_{SPL} into its new version DM'_{SPL} via higher-order deltas. A *higher-order delta* $\delta^H = \{op_1^H, \ldots, op_l^H\} \subset OP_{SPL}^H$ transforms a DM_{SPL} by changing its delta set Δ_{SPL} based on the application of evolution operations. An *evolution operation* $op^H \in OP_{SPL}^H$ specifies additions (add δ) and removals (rem δ) of deltas δ from/to a DM_{SPL}. By OP_{SPL}^H, we refer to the set of evolution operations defined over the universe $\mathcal{OP}$. We again abstract from modifications, e.g. an exchange of the application condition of a delta, and also apply the removal-addition encoding. We further abstract from feature model evolution [32] but assume that, for an SPL version, a respective FM_{SPL} exists building the basis for the definition of application conditions of added deltas. The application of a higher-order



(a) m_{core} **(b)** δ_1 **with** $\varphi_{\delta_1} = f_2$ **(c)** δ_2 **with** $\varphi_{\delta_2} = f_3 \wedge f_1$

(d) δ_3 **with** $\varphi_{\delta_3} = f_2$ **(e)** δ_4 **with** $\varphi_{\delta_4} = f_2 \wedge f_3$

Figure 2: Sample Higher-Order Delta Model DM^H_{hist}

delta denotes an *evolution step* of the complete *evolution history* captured in a *higher-order delta model*. A higher-order delta model $DM^H_{hist} = (m_{core}, \Delta^H)$ comprises a core model m_{core} and a sequence of higher-order deltas $\Delta^H = (\delta^H_0, \delta^H_1, \ldots, \delta^H_r)$. Based on the incremental application of Δ^H, the evolved version DM^i_{SPL} of the delta model DM^{i-1}_{SPL} is obtained, where the application of δ^H_0 combined with the m_{core} defines the initial delta model version DM^0_{SPL}.

Example 2.2. Consider the sample DM^H_{hist} in Fig. 2, where we adapted higher-order delta modeling [24] for state machines. Based on the core state machine shown in Fig. 2a, the initial $\delta^H_0 = \{\text{add } \delta_1, \text{add } \delta_2\}$ adds the deltas δ_1 and δ_2 to realize the first $DM^0_{SPL} = (m_{core}, \{\delta_1, \delta_2\})$ of the sample SPL. For version $DM^1_{SPL} = (m_{core}, \{\delta_2, \delta_3, \delta_4\})$, we specify the removal of δ_1 and the addition of δ_3 and δ_4 in δ^H_1, where δ_2 stays constant for both versions. In each delta δ_i, + denotes an addition and − a removal of a model element.

3 175% MODELING FORMALISM

For the development of an evolving SPL, its *versions* $\Theta = \{\theta_0, ..., \theta_n\}$ caused by *evolution steps* are important starting from the initial version θ_0 up to its present version θ_n. Each $\theta_i \in \Theta$ defines version-specific domain artifacts, e.g., a 150% model $(M_{150}, \alpha_{150})^{\theta_i}$ containing valid elements $E^{\theta_i}_{M_{150}}$ and annotations $\alpha^{\theta_i}_{150}$ for θ_i. Existing techniques for the evolution of SPL domain artifacts are mainly categorizable as transformative [15, 22, 24, 38] or compositional [2, 3, 13]. To the best of our knowledge no approach exists that captures variant and version information by the same means in an annotative fashion. Such a modeling formalism (1) provides an overview over all variants of all SPL versions and (2) enables the application of family-based analyses [41] also to evolving SPLs. Therefore, we propose an extension of 150% modeling called 175% modeling.

For 175% modeling, we first map elements $e \in E_{M_{175}}$ and versions $\theta \in \Theta$ to specify in which version an element is contained in the derivable version-specific 150% model $(M_{150}, \alpha_{150})^{\theta}$. By $E_{M_{175}}$, we refer to the set of all elements of a 175% model, i.e., the joined set of all 150% model elements $E^{\theta_i}_{M_{150}}$. Therefore, we define the temporal annotation function $\psi : E_{M_{175}} \rightarrow \mathcal{P}(\Theta)$ such that, for every element $e \in E_{M_{175}}$, the set of versions $\psi(e) = \{\theta_k, \ldots, \theta_l\}$ is returned in which e is contained in at least one model.

Second, we combine the version mapping with feature annotations to capture variability and version information of model elements at the same time. We adapt the annotation function α_{150} of 150% models to incorporate the combined information in a new annotation function $\alpha_{175} : E_{M_{175}} \times \mathcal{P}(\Theta) \rightarrow \mathcal{P}(\Theta \times \mathbb{B}(F^{\Theta}_{SPL}))$, where

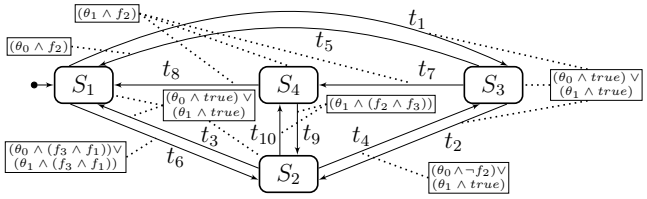


Figure 3: Sample 175% Model (M_{175}, α_{175})

F^{Θ}_{SPL} denotes the set of all features of the evolution history. For each element e, the annotation function $\alpha_{175}(e, \psi(e)) = \{(\theta_k, \varphi_e^{\theta_k}), \ldots, (\theta_l, \varphi_e^{\theta_l})\}$ returns the pairs of versions θ_i and feature conditions $\varphi_e^{\theta_i}$ for which the element is contained in the version-specific 150% model with annotation $\alpha^{\theta_i}_{150}(e) = \varphi_e^{\theta_i}$. As an element e is contained in different SPL versions θ_i, the associated annotations $\varphi_e^{\theta_i}$ potentially differ as well, e.g., due to restrictions in which variants the element should be contained or due to feature model evolution [31, 32, 37]. To allow for valid feature annotations in a 175% model, for each θ_i, we require a respective feature model $FM^{\theta_i}_{SPL}$ to ensure a valid mapping in α_{175}. To this end, we assume the existence of a $FM^{\theta_i}_{SPL}$ and a feature set $F^{\theta_i}_{SPL}$ for each version θ_i to allow for consistent derivations of versions of variant-specific models.

We define a *175% model* (M_{175}, α_{175}) similar to a 150% model, where M_{175} represents the super model merging all variant-specific models of all SPL versions, and α_{175} denotes the annotation function specifying version and feature conditions for all elements $e \in E_{M_{175}}$. We are able to derive (1) a single variant-specific model m^{θ}_p for a given version θ and configuration F^{θ}_p, (2) a 150% model $(M_{150}, \alpha_{150})^{\theta}$ for a given θ, (3) a restricted 175% model $(M_{175}, \alpha_{175})^{\Theta'}$ for a given set of versions $\Theta' \subseteq \Theta$, or (4) a variant-specific 175% model $(M_{175}, \alpha_{175})^{F_p, \Theta'}$ for a given F_p and subset $\Theta' \subseteq \Theta$. The process of the version derivation is similar to the one defined for 150% modeling. For case (1), an element $e \in E_{M_{175}}$ is selected to be contained in the resulting model m^{θ}_p, if a pair $(\theta, \varphi^{\theta}_e) \in \alpha_{175}(e, \psi(e))$ exists for the given version, where further F^{θ}_p satisfies φ^{θ}_e. For brevity, we use the tuple $(\theta, \varphi^{\theta}_e)$ notation, where the version information is lightly adaptable as interval or formula representation. In cases (2), (3), and (4), the selection of elements to obtain a version-restricted model is simpler as we only have to evaluate if the element is mapped to a given version $\theta \in \psi(e)$ or subset of versions $\Theta' \cap \psi(e) \neq \emptyset$. For case (4), we further take the given configuration into account.

A 175% model captures the evolution history Θ up to the present version θ_n, but an SPL may evolve to its next version θ_{n+1} in the future. A 175% model has to be adapted to capture θ_{n+1}. As a 175% model comprises all elements contained in at least one variant-specific model version, we extend the set of all elements $E_{175} \cup E^{\theta_{n+1}}_{150}$ as first step. Then, for all elements, we update their version mapping $\psi(e)$ and their annotation $\alpha_{175}(e, \psi(e))$ to be consistent with the extended version set $\Theta \cup \theta_{n+1}$. Elements that were valid for θ_n, i.e., there exists an annotation $(\theta_n, \varphi^{\theta_n}_e)$, but are not contained in the new version $(M_{150}, \alpha_{150})^{\theta_{n+1}}$, become invalid such that their annotation comprises no new tuple $(\theta_{n+1}, \varphi^{\theta_{n+1}}_e) \notin \alpha_{175}(e, \psi(e))$. For elements

added for the first time to the 175% model, the version mapping $\psi(e) = \{\theta_{n+1}\}$ and the annotation $\alpha_{175}(e, \psi(e)) = \{(\theta_{n+1}, \varphi_e^{\theta_{n+1}})\}$ are initialized, where $\varphi_e^{\theta_{n+1}}$ is defined by $\alpha_{150}^{\theta_{n+1}}(e)$. Elements that are not contained in the last version θ_n, but were already added to the 175% model in prior versions $\psi(e) \neq \emptyset$, become valid again, i.e., their version mapping and annotation are extended similar to newly added elements. As last step, we update the version mapping and the annotation of elements that are contained in the last and also in the new version, i.e., either the feature annotation $\varphi_e^{\theta_n} = \varphi_e^{\theta_{n+1}}$ stays the same or is newly set to $\varphi_e^{\theta_{n+1}}$.

Example 3.1. Consider the sample (M_{175}, α_{175}) shown in Fig. 3 capturing the evolution of our sample SPL. Similar to the 150% model from Ex. 2.1, M_{175} merges all variant-specific state machines, but also all their versions. Elements are annotated with a disjunction of their tuples $(\theta_i, \varphi_e^{\theta_i}) \in \alpha_{175}(e, \psi(e))$, where the version and feature annotation of a tuple are combined via a conjunction.

Benefits, Limitations and Potential Applications. We focus on a generic, artifact-independent definition of 175% modeling. Hence, our approach is adaptable for various artifact types already used for annotative variability modeling [35], e.g., source code [21, 42], feature-annotated state machines [16], or featured transition systems [20]. We presented an adaptation for feature-annotated state machines in Ex. 3.1. Similarly, for source code, where variability is realized, e.g., preprocessor-based [21] or via variability encoding [42], each pair $(\theta_i, \varphi_e^{\theta_i})$ is implemented by representing the version information as (preprocessor) variables, where, again, several pairs of version and feature annotations for a source code block are connected via disjunction. Besides the artifact-independency, the comprehensive as well as unified representation and, hence, the overview of the evolving SPL are further benefits as we are able to identify and understand interdependencies between variable artifacts and also their versions. Due to the captured evolution history, we further enable the support of planning future evolution steps by incorporating the information of previous evolution changes and their impact.

Furthermore, based on 175% modeling, the application of family-based analyses [41] also to evolving SPLs is achievable. For example, variability-aware slicing [16, 18] can be enhanced to also incorporate versions for the analysis, e.g., of changes and their impact on SPL versions improving the comprehension and development of evolving SPLs as we can examine how artifacts or even features and their interdependencies are evolved. In addition, quality assurance of evolving SPLs benefits from extending existing techniques. By using a 175% model as test model, SPL test-case generation [8] can be enhanced, where not only the reusability of test cases between variants but also across SPL versions is derived during generation. For family-based SPL verification [20, 40], a 175% model used as functional specification will enable a verification of SPL behavior and changed behavior between SPL versions.

However, the formalism also has limitations mainly in the size and, thus, in the complexity of the resulting model. This is a general drawback of annotative variability modeling [35] but is increased by the incorporation of SPL versions. To cope with this drawback, the formalism allows for the projection to restricted models w.r.t. given feature configurations and/or versions. The complexity further compromise a manual creation of the model. Hence, tool support for the model creation as well as a systematic process is crucial and much needed. By providing tool support, we either apply and create a 175% model directly when used as front end or apply the formalism as back end in, e.g., variant control systems [23].

Another limitation is that 175% modeling does not allow for, e.g., an analysis about changes to the set of variants of subsequent SPL versions as, e.g., provided by higher-order delta modeling [24]. This information is important for SPL regression testing to reduce the testing redundancy by focusing on changes and their impact caused by an evolution step. To exploit the benefits of both formalisms, a transformation between them is crucial, where one type is used to derive the opposing type. We propose a bidirectional transformation between 175% and higher-order delta modeling in the next section.

4 BIDIRECTIONAL TRANSFORMATION

For using both formalisms during SPL development to benefit from (1) an overview provided by 175% modeling, and (2) change impact analysis provided by higher-order delta modeling, we propose a generic transformation between both model types incorporating version and variant information. Kowal et al. [19] proposed a similar transformation by building a 150% performance-annotated activity diagram from a given delta model to also exploit the benefits of two formalisms, i.e., the application of family-based performance analysis. We describe both transformation directions separately and assume the respective input model to be given.

Transformation Higher-Order Delta Model to 175% Model. For the transformation of a higher-order delta model DM_{hist}^H into an equivalent 175% model (M_{175}, α_{175}), we exploit properties of delta modeling [9, 24]: First, we have a predefined core m_{core} building the basis for the definition of deltas and their evolution via higher-order deltas. We use m_{core} also as core of the 175% model to be generated, where elements are further integrated by incorporating the element operations (add e, rem e). Second, the version and feature mapping of an element is given by (higher-order) deltas, i.e., feature annotations are derived taking the application conditions of deltas affecting the element into account and the version is denoted by the evolution step specified by a higher-order delta. In addition, the application order of higher-order deltas is defined such that no ambiguity for version and feature annotations can be introduced. Those properties allow for a unique and automated transformation shown in Alg. 1. We use the function `updateAnnotation` to abstract from the concrete annotation process but describe each case denoted by a respective tag, e.g., core, in the following.

Based on a given DM_{hist}^H, we integrate m_{core} by adding all elements and annotating them with $(\theta_0, \varphi_e = true)$ as the core represents the commonality of all variants of the initial version θ_0 (cf. l. 1-3). If a core element is not contained in all variants of θ_0, a delta capturing a remove operation is added via the initial higher-order delta $\delta_{\theta_0}^H$ to be handled in the second part of the algorithm (cf. l. 4ff).

We iterate over all $\delta^H \in \Delta^H$ to integrate new elements and to derive their version and feature annotations. For valid annotations, we analyze the captured evolution and change operation combinations of δ^H, split them based on their operation types, and provide a sorted list of the combinations to be incorporated (cf. l. 5). This step is required as the different types of operation combinations

Algorithm 1: Transformation DM^H_{hist} to (M_{175}, α_{175})

```
1  forall e ∈ E_{m_core} do
2      addElement(e, M_175);
3      updateAnnotation(e, θ_0, core, true);
4  forall δ^H ∈ Δ^H do
5      List operations = splitAndSortOperations(δ^H);
6      Set visited = ∅;
7      forall (op^H, op) ∈ operations do
8          if op^H = add δ ∧ op = add e then
9              if e ∉ E_{M_175} then
10                 addElement(e, M_175);
11             visited ∪ updateAnnotation(e, θ_{δ^H}, addadd, φ_δ);
12         else if op^H = add δ ∧ op = rem e then
13             visited ∪ updateAnnotation(e, θ_{δ^H}, addrem, φ_δ);
14         else if op^H = rem δ ∧ op = rem e then
15             visited ∪ updateAnnotation(e, θ_{δ^H}, remrem, φ_δ);
16         else if op^H = rem δ ∧ op = add e then
17             visited ∪ updateAnnotation(e, θ_{δ^H}, remadd, φ_δ);
18     updateAnnotation(E_{M_175} \ visited, θ_{δ^H}, none);
```

have a distinct influence on the resulting feature annotation of the θ_{δ^H} to be handled. For instance, the addition of a delta that adds a new element (add(add(e))) has to be integrated before the addition of a delta which removes the element for a specific feature condition (add(rem(e))) as the removal operation can only update the annotation if the element was added to the 175% model via the first operation combination. The splitting of captured evolution and change operation combinations of δ^H results in the sequence: (1) add(add(e)), (2) add(rem(e)), (3) rem(rem(e)), and (4) rem(add(e)). We iterate over this sequence to update the annotations of elements (cf. l. 7ff), where we add all elements for which an update is applied to the set `visited`.

For the combination addadd (cf. l. 8), we check if the element is already contained in the 175% model and if not it is added and annotated with $(\theta_{\delta^H}, \varphi_\delta)$, i.e., with the version captured by δ^H and the application condition of the newly added delta. Otherwise, we examine two cases. First, the element was removed in a prior version based on a higher-order delta. In this case, there is no annotation in the last version and we annotate the element with $(\theta_{\delta^H}, \varphi_\delta)$ to be valid again for the current θ_{δ^H}. Second, there exists another delta in the last version that adds the element. As an element is only added to a model via one delta in a delta model version, there has to exist another operation combination that removes the obsolete delta to ensure a valid delta model during higher-order delta application. For each annotation, we record which part is specified by a delta $\varphi_e = \varphi_{\delta'} \wedge \varphi_{\delta''} \wedge \ldots$ to update the annotation by connecting the last annotation and the application condition of the new delta $\varphi_e \wedge \varphi_\delta$. We remove the application condition of the obsolete delta, e.g., $\varphi_{\delta''}$, when the respective operation combination is handled by Alg. 1.

For combination addrem (cf. l. 12), i.e., the addition of a delta removing an element, we connect the existing annotation $\varphi_e \wedge \neg\varphi_\delta$ with the negated delta application condition. By using the negation, the updated feature annotation restricts the element to be not contained in a variant of θ_{δ^H} for which F^θ_p satisfies φ_δ. The existing annotation φ_e is (1) used from the last version, if there exists no other operation on the element for this version, (2) introduced by the addadd combination, or (3) is defined by the integration of m_{core}. The updated annotation is defined by $(\theta_{\delta^H}, \varphi_e \wedge \neg\varphi_\delta)$.

For combination remrem (cf. l. 14), i.e., the removal of a delta removing an element, we remove the restriction denoted by the negated application condition from the existing annotation φ_e. The updated annotation results in $(\theta_{\delta^H}, \varphi_e^{-\neg\varphi_\delta})$, where $\varphi_e^{-\neg\varphi_\delta}$ represents the existing annotation without the obsolete part $\neg\varphi_\delta$.

For combination remadd (cf. l. 16), i.e., the removal of a delta adding an element, we remove the delta-specific annotation from the existing annotation φ_e denoted by $\varphi_e^{-\varphi_\delta}$ similar to the combination remrem. In case the removed delta specified the only possibility to add the captured element, i.e., the element is not contained in any variant-specific model of θ_{δ^H}, we define no updated annotation.

As last step (cf. l. 18), we update the version and feature annotations of all elements $E_{M_{175}} \setminus visited$ in the 175% model that are not influenced by the current higher-order delta δ^H such that $(\theta_{\delta^H}, \varphi_e) \in \alpha_{175}(e, \psi(e))$ holds.

As the result of Alg. 1, we obtain an equivalent 175% model in which all versions of variants derivable from the input DM^H_{hist} are merged into one super model. Based on this model, we are able to apply evolution-aware family-based analyses as discussed in Sect. 3 to support the development of evolving SPLs when the evolution is documented by higher-order delta modeling [24].

Example 4.1. By applying Alg. 1 to the DM^H_{hist} from Ex. 2.2, we obtain the (M_{175}, α_{175}) shown in Fig. 3. There is only one difference between the depicted model and the result of the transformation as transition $t4$ should be annotated by $true \wedge \neg f_2$ in the initial version which can be shortened to $\neg f_2$ as defined in the model.

Transformation 175% Model to Higher-Order Delta Model. The transformation of a 175% model (M_{175}, α_{175}) into an equivalent higher-order delta model DM^H_{hist} is not unique as one model has to be chosen as core m_{core} to derive (higher-order) deltas. For the core, several options exist, i.e., either (1) every element is selected that is annotated with $(\theta_0, true)$, (2) a model has to be determined that has the most commonality w.r.t. all variant-specific models of the initial version, where potentially alternatives exist to be chosen from, or (3) we provide a core configuration $F^{\theta_0}_{p_{core}}$. In case (1), the resulting model may not be a valid variant-specific model, i.e., to correspond to a variant, further elements have to be selected. For case (2), an extra analysis is required to determine the most common core model, where further user interaction is necessary to select from potential alternatives. In case (3), m_{core} is explicitly derivable from the 175% model due to $F^{\theta_0}_{p_{core}}$. In this paper, we apply case (3) to realize a semi-automated transformation as we require a valid model m_{core} and want to keep the transformation effort low.

In addition, as a 175% model comprises all elements contained in at least one variant-specific model of an SPL version, we determine, for each element individually, how it is captured for the resulting DM^H_{hist} in terms of evolution and change operations. To this end, we obtain a very fine-grained DM^H_{hist} as each delta only comprises one change operation. In future work, we will apply delta-oriented refactorings [36] to further improve the granularity, the readability as well as the analyzability of the resulting DM^H_{hist}.

In the following, we describe the semi-automated transformation shown in Alg. 2. Based on a 175% model (M_{175}, α_{175}) and a configuration $F^{\theta_0}_{p_{core}}$ for the core of the initial θ_0, we analyze for every element $e \in E_{M_{175}}$ in the 175% model their annotations $\alpha_{175}(e, \psi(e))$

Algorithm 2: Transformation (M_{175}, α_{175}) to DM^H_{hist}

```
1  forall e ∈ E_{M_175} do
2      lastAnno = (⊥_θ, ⊥_φ);
3      forall θ ∈ Θ do
4          if (θ, φ_e^θ) ∈ α_175(e, {θ}) then
5              if θ = θ_0 then
6                  if φ_e^θ = true then
7                      addElement(e, m_core);
8                  else if F^{θ_0}_{p_core} satisfies φ_e^θ then
9                      addElement(e, m_core);
10                     addDelta(θ, rem, e, ¬φ_e^θ);
11                 else
12                     addDelta(θ, add, e, φ_e^θ);
13             else
14                 if lastAnno = (⊥_θ, ⊥_φ) ∧ δ_e ∉ DM^H_hist then
15                     addDelta(θ, add, e, φ_e^θ);
16                 else if lastAnno = (⊥_θ, ⊥_φ) ∧ δ_e ∈ DM^H_hist then
17                     addDelta(θ, δ_e, φ_e^θ);
18                 else if lastAnno ≠ (θ, φ_e^θ) then
19                     remDelta(θ, δ_e);
20                     addDelta(θ, add, e, φ_e^θ);
21             lastAnno = (θ, φ_e^θ);
22         else if (θ, φ_e) ∉ α_175(e, {θ}) ∧ lastAnno ≠ (⊥_θ, ⊥_φ) then
23             if δ_e ∈ DM^H_hist then
24                 remDelta(θ, δ_e);
25             else
26                 addDelta(θ, rem, e, true);
27             lastAnno = (⊥_θ, ⊥_φ);
```

to determine evolution and change operations for the resulting DM^H_{hist}. To identify changes of the annotation between subsequent versions of an element, we record analyzed annotations in the variable `lastAnno` to enable the comparison. The annotation $(\bot_\theta, \bot_\varphi)$ denotes that no annotation for the last element version exists.

For each element e, we iterate over all versions $\theta \in \Theta$ incorporated in the 175% model to ensure the detection of potential changes of e to be captured in (higher-order) deltas (cf. l. 3ff). Hence, we check if the element is annotated or not for the current version θ. For an existing annotation $(\theta, \varphi^\theta_e)$, we further examine if the version corresponds to the initial version θ_0 (cf. l. 5). We focus explicitly on θ_0, as from this version, we determine m_{core} used as basis to derive deltas. Elements that have a feature annotation defined as $true$ or their annotation φ^θ_e is satisfied by the given configuration $F^{\theta_0}_{p_{core}}$ are added to m_{core}. In the latter case, a delta δ_e is created capturing a remove change operation of the element and is also attached to the initial higher-order delta $\delta^H_{\theta=\theta_0}$, where the application condition of δ_e is defined by the negated feature annotation $\neg\varphi^\theta_e$ (cf. l. 10), i.e., the delta removes the element from m_{core} if $F^{\theta_0}_p$ satisfies $\neg\varphi^\theta_e$. For elements that are not contained in the core, we similarly create and attach deltas δ_e adding the element to a model if $F^{\theta_0}_p$ satisfies φ^θ_e.

For the remaining annotations $(\theta, \varphi^\theta_e)$ of an element e, we check if (1) e has no last annotation and is not yet contained in DM^H_{hist} (cf. l. 14), (2) e has no last annotation but is already part of DM^H_{hist} based on a δ_e (cf. l. 16), or (3) there exists a last annotation of the prior version but it differs compared to the current one (cf. l. 18). For case (1), we create a δ_e that adds the element as it is contained in a variant-specific model for the first time in the current version. In case (2), the δ_e of element e was removed in a prior version, but is now added again via a higher-order delta δ^H_θ, where further the application condition is set to φ^θ_e. For case (3), we simply remove the old δ_e and add a new δ'_e with updated application condition.

If an element is not annotated in the θ to be analyzed but was annotated in the former version (cf. l. 22ff), we check if a delta for the element exists in DM^H_{hist} or not. For elements that are added via δ_e but have to be removed from θ, we remove their δ_e with a removal evolution operation. For a core element with $\varphi_e = true$, we realize a workaround as there is no delta $\delta_e \notin DM^H_{hist}$ which is removed to ensure that the element is not contained in a variant-specific model derivable in the current version. Hence, we add to δ^H_θ a new delta δ_e removing the core element for every valid feature configuration, i.e., we define the application condition as $true$.

In the end, we obtain an equivalent higher-order delta model DM^H_{hist}, where all versions of variants derivable from the input 175% model are generatable by transforming m_{core}. Based on DM^H_{hist}, we are able to apply incremental analyses, e.g., change impact analysis for supporting regression testing of evolving SPLs when the evolution is documented by a 175% model.

Example 4.2. By applying Alg. 2 on (M_{175}, α_{175}) from Ex. 3.1, we obtain DM^H_{hist} shown in Fig. 2 and described in Ex. 2.2. In contrast to Ex. 2.2, the transformation results in deltas only comprising one change operation each, where further the derived application conditions may differ to the application conditions defined in Fig. 2.

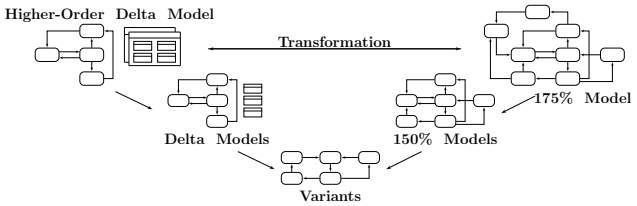


Figure 4: Equivalence of (M_{175}, α_{175}) and DM^H_{hist}

5 SOUNDNESS OF TRANSFORMATIONS

We reason about the soundness of the proposed transformations in Alg. 1 and Alg. 2 w.r.t. the equivalence of the input model to the resulting output model as shown in Fig. 4. A higher-order delta model is equivalent to a 175% model and vice versa if for all versions $\theta_i \in \Theta$ and configurations $F^{\theta_i}_{p_j}$, the same variant-specific model versions are derived. To prove the soundness, we use complete induction and show that the application of the specified transformation rules result in an output model allowing for the derivation of the same variant-specific models w.r.t. the input model. The base case is defined by the transformation of the initial version resulting in a standard delta model or 150% model, respectively, from which the same variant-specific models are derivable. The inductive step shows that the application of the transformation rules again result in version-specific delta or 150% models. We discuss a proof sketch for both directions, whereas the complete proof for the transformation of DM^H_{hist} to (M_{175}, α_{175}) is defined by Nahrendorf [28].

Algorithm 1. To obtain a 175% model, we first integrate m_{core} of the DM^H_{hist} and then incorporate the initial higher-order delta δ^H_0. At this point, elements that are common for all variant-specific models of the initial version are annotated with $(\theta_0, true)$, whereas

elements (1) to be added (add e) are annotated with φ_{δ_e} of delta δ_e, i.e., they are selected for a model for the same condition as added via the delta, or (2) to be removed (rem e) are annotated with the negated application condition. Hence, the base case holds as both models allow for the derivation of the same models for θ_0.

For the inductive step, i.e., the integration of the remaining θ_i defined by δ_i^H, we examine the results of the transformation rules. New elements are added to the model and annotated with φ_{δ_e} similarly as for θ_0. The same holds for elements that are removed via a newly added delta. If the application condition of a delta is changed, the annotation of the respective element for the current version θ_i is also changed such that the element is selected for a model for the same feature configurations as the delta would be applied in the delta model of θ_i. The removal of a delta capturing a remove change operation implies that an element is less restricted to be contained in a model and, thus, Alg.1 does the same by removing the application condition of the delta from the feature annotation such that the annotation for the θ_i is also less restricted. If the delta adding an element is removed via δ_i^H and there is no other delta adding the element as described in Sect. 3, the element is not contained in a model of θ_i and, thus, does not get an annotation tuple for θ_i in the 175% model. To this end, as the element annotations for each version in the 175% model correspond to the application condition of the respective deltas in the higher-order delta model, both models allow for the derivation of the same models for θ_i.

Algorithm 2. To obtain a higher-order delta model, Alg. 2 derives m_{core} from version θ_0 based on the given $F_{p_{core}}^{\theta_0}$. In addition, the initial δ_0^H is determined, where deltas are added capturing either (1) a removal if the element is annotated such that $\varphi_e^{\theta_0} \neq true$ but $F_{p_{core}}^{\theta_0}$ satisfies $\varphi_e^{\theta_0}$, or (2) capturing an addition if the element is not part of the core. Hence, the base case holds as both models allow for the derivation of the same variant-specific models for θ_0.

For the inductive step and, thus, for the remaining θ_i, Alg. 2 takes into account (1) if an annotation $(\theta_i, \varphi_e^{\theta_i})$ exists or not, (2) how it may change compared to its prior annotation, and (3) if the element is already integrated in the DM_{hist}^H to be created. Depending on those checks and further sub-cases to be checked, deltas capturing a change operation (add e, rem e) are added or removed via δ_i^H. As we examine each element individually for each θ_i, the transformation rules are sufficient to detect every change of elements in the input 175% model and to capture them in the output higher-order delta model. The special case, i.e., an core element is not contained in a θ_i, is handled via a newly added delta removing the element for every feature configuration, i.e., $\varphi_e^{\theta_i} = true$. To this end, the application conditions of the deltas that are version-specific added/removed in the resulting DM_{hist}^H correspond to the combined version and feature annotations of all elements in the 175% model and, hence, both models allow for the derivation of the same models for θ_i.

6 RELATED WORK

We discuss related work w.r.t. SPL evolution modeling. We further refer to Botterweck and Pleuss [6] and Montalvillo and Díaz [27] for an overview on SPL evolution. For an overview about SPL reengineering, where our proposed transformations belong to, we refer to Fenske et al. [12]. SPL evolution has been investigated with a focus on various domain artifacts, e.g., requirements, features, source code etc. [4–6, 27]. Svahnberg and Bosch [39] proposed a general categorization of SPL evolution and guidelines to deal with it based on the examination of two industrial case studies.

In literature, SPL evolution is mainly tackled with feature models [7, 14, 31, 32, 37]. Gamez et al. [14] use cardinality-based feature models to capture and reason about evolution, whereas Bürdek et al. [7] apply model differencing. In contrast, Pleuss et al. [32] with EvoFM, Seidl et al. [37] with hyper feature models, and Nieke et al. [31] with temporal feature models propose extended feature models to capture the variability and the evolution by the same formalism. Compared to 175% modeling, those approaches only focus on feature models and are not adaptable for SPL domain artifacts.

SPL evolution is also tackled by adopting existing variability modeling techniques as, e.g., delta modeling [11, 15, 22, 24, 38] or aspect-oriented programming [2, 3, 13]. Haber et al. [15] investigate the evolution of delta-oriented architectures based on the specification of change operations w.r.t. the delta set. In contrast, Seidl et al. [38] and Lity et al. [24] present a generic extension of delta modeling [9]. Seidl et al. [38] define two types of deltas, i.e., configuration and evolution deltas, and apply both types on the same modeling level, whereas Lity et al. [24] specify the evolution of delta models via higher-order deltas (cf. Sect. 2). Compared to our formalism, those approaches capture the evolution on a transformative and not on an annotative basis. Nevertheless, our transformation described in Sect. 4 is extendable to also cope with other delta modeling adaptations. Lima et al. [22] and Dhungana et al. [11] apply deltas to determine differences between SPL versions and use this information to detect and resolve potential evolution conflicts. Both approaches do not build on a modeling formalism coping with variability and evolution by the same means like our novel formalism does. Apel et al. [3] combine feature- and aspect-oriented programming to exploit the respective benefits to support SPL evolution using FeatureC++ as example. Alves et al. [2] and Figueirdo et al. [13] apply aspects to allow for the evolution of the SPL code base. Compared to our approach, the three approaches focus on source code and are hard to adapt for other domain artifacts.

Neves et al. [29] introduce templates for safe SPL evolution. The template application guarantees that applied evolution operations do not influence the behavior of existing variants other than intended. Compared to our technique, the approach provides no evolution-aware variability modeling formalism.

SPL evolution can be further supported using traceability mechanisms between domain artifacts [1]. In contrast to our approach, Ajila and Kaba [1] abstract from an explicit specification of evolution steps w.r.t. combined version and feature annotations.

Comprehensive tool support for SPL evolution coping with feature model as well as artifact evolution is provided by DarwinSPL [30], DeltaEcore [38], and SPLEmma [34]. We are currently working on the tool support for 175% modeling and plan to also incorporate temporal feature models [31] for handling the feature model evolution similar to DarwinSPL.

7 CONCLUSION

We proposed 175% modeling as an annotative modeling formalism to capture the evolution and variability of evolving SPLs by the

same means. Hence, we merge all variant-specific models of all SPL versions into one model, where elements are mapped to combined version and feature annotations to specify in which variant-specific model version the element is contained. We discussed benefits, limitations, and potential application scenarios enabling evolution-aware family-based analyses. In addition, we provided a bidirectional transformation between 175% and higher-order delta models and show its soundness.

As future work, we plan to combine 175% modeling with temporal feature modeling [31] to provide a sophisticated formalism tackling feature model and domain artifact evolution. We want to evaluate the applicability of our formalism by capturing and documenting the evolution history of existing SPLs. We also plan to instantiate our formalism for different artifact types to examine its limitations and benefits in a wider context and investigate further application scenarios. As a last step, we will extend our transformation from 175% models into higher-order delta models by incorporating delta-oriented refactorings [36] to improve the granularity of the resulting higher-order delta model.

Acknowledgment. We thank Malte Lochau and Andy Schürr for discussions on 175% modeling. This work was partially supported by the German Research Foundation within the project IMoTEP (grant agreement LO 2198/2-1) and by the European Commission within the project HyVar (grant agreement H2020-644298).

REFERENCES

[1] Samuel A Ajila and Ali B Kaba. 2008. Evolution Support Mechanisms for Software Product Line Process. *J. Sys. and Soft.* 81, 10 (2008), 1784–1801.
[2] Vander Alves, Pedro Matos, Jr., Leonardo Cole, Alexandre Vasconcelos, Paulo Borba, and Geber Ramalho. 2007. Extracting and Evolving Code in Product Lines with Aspect-oriented Programming. In *Trans. Aspect-Oriented Soft. Dev.* 117–142.
[3] Sven Apel, Thomas Leich, Marko Rosenmüller, and Gunter Saake. 2005. Combining Feature-Oriented and Aspect-Oriented Programming to Support Software Evolution. In *RAM-SE'05-ECOOP'05.*
[4] Jan Bosch. 2000. *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach.* Pearson.
[5] Jan Bosch. 2002. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. In *Proc. Int'l Soft. Product Line Conf.* 257–271.
[6] Goetz Botterweck and Andreas Pleuss. 2014. *Evolution of Software Product Lines.* Springer, 265–295.
[7] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. 2016. Reasoning about Product-Line Evolution using Complex Feature Model Differences. *Adv. Softw. Eng.* 23, 4 (2016), 687–733.
[8] Johannes Bürdek, Malte Lochau, Stefan Bauregger, Andreas Holzer, Alexander von Rhein, Sven Apel, and Dirk Beyer. 2015. Facilitating Reuse in Multi-goal Test-Suite Generation for Software Product Lines. In *Proc. Int'l Conf. Fundamental Approaches to Software Engineering.* Springer, 84–99.
[9] Dave Clarke, Michiel Helvensteijn, and Ina Schaefer. 2015. Abstract Delta Modelling. *Math. Struct. Comp. Sci.* 25, 3 (2015), 482–527.
[10] Ferruccio Damiani, David Faitelson, Christoph Gladisch, and Shmuel Tyszberowicz. 2016. A Novel Model-Based Testing Approach for Software Product Lines. *Software & Systems Modeling* (2016), 1–29.
[11] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. 2010. Structuring the Modeling Space and Supporting Evolution in Software Product Line Engineering. *J. Sys. and Soft.* 83, 7 (2010), 1108–1122.
[12] Wolfram Fenske, Thomas Thüm, and Gunter Saake. 2014. A Taxonomy of Software Product Line Reengineering. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems.* ACM, Article 4, 8 pages.
[13] Eduardo Figueiredo, Nelio Cacho, Claudio Sant'Anna, Mario Monteiro, Uira Kulesza, Alessandro Garcia, Sérgio Soares, Fabiano Ferrari, Safoora Khan, Fernando Castor Filho, et al. 2008. Evolving Software Product Lines with Aspects. In *Proc. Int'l Conf. Software Engineering.* 261–270.
[14] Nadia Gamez and Lidia Fuentes. 2011. Software Product Line Evolution with Cardinality-Based Feature Models. In *Proc. Int'l Conf. Software Reuse.* 102–118.
[15] Arne Haber, Holger Rendel, Bernhard Rumpe, and Ina Schaefer. 2012. Evolving Delta-Oriented Software Product Line Architectures. In *Monterey Workshop.* Springer, 183–208.
[16] Jochen Kamischke, Malte Lochau, and Hauke Baller. 2012. Conditioned Model Slicing of Feature-Annotated State Machines. In *Proc. Int'l Workshop Feature-Oriented Software Development.* 9–16.
[17] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. Feature-Oriented Domain Analysis Feasibility Study. (1990).
[18] Frederik Kanning and Sandro Schulze. 2014. Program Slicing in the Presence of Preprocessor Variability. In *Proc. Int'l Conf. Soft. Maintenance and Evol.* 501–505.
[19] Matthias Kowal, Ina Schaefer, and Mirco Tribastone. 2014. Family-Based Performance Analysis of Variant-Rich Software Systems. In *Proc. Int'l Conf. Fundamental Approaches to Software Engineering.* Springer, 94–108.
[20] Axel Legay, Gilles Perrouin, Xavier Devroey, Maxime Cordy, Pierre-Yves Schobbens, and Patrick Heymans. 2017. On Featured Transition Systems. In *Proc. Int'l Conf. Current Trends in Theory and Practice of Informatics.* Springer, 453–463.
[21] Jörg Liebig, Sven Apel, Christian Lengauer, Christian Kästner, and Michael Schulze. 2010. An Analysis of the Variability in Forty Preprocessor-based Software Product Lines. In *Proc. Int'l Conf. Software Engineering.* ACM, 105–114.
[22] Gleydson Lima, Jadson Santos, Uir Kulesza, Daniel Alencar, and Sergio Vianna Fialho. 2013. A Delta Oriented Approach to the Evolution and Reconciliation of Enterprise Software Products Lines. In *Proc. Int'l Conf. Enterprise Information Systems.* 255–263.
[23] Lukas Linsbauer, Thorsten Berger, and Paul Grünbacher. 2017. A Classification of Variation Control Systems. In *Proc. Int'l Conf. Generative Programming and Component Engineering.* ACM, 49–62.
[24] Sascha Lity, Matthias Kowal, and Ina Schaefer. 2016. Higher-Order Delta Modeling for Software Product Line Evolution. In *Proc. Int'l Workshop Feature-Oriented Software Development.* ACM, 39–48.
[25] Sascha Lity, Thomas Morbach, Thomas Thüm, and Ina Schaefer. 2016. Applying Incremental Model Slicing to Product-Line Regression Testing. In *Proc. Int'l Conf. Software Reuse.* Springer, 3–19.
[26] Tom Mens, Alexander Serebrenik, and Anthony Cleve (Eds.). 2014. *Evolving Software Systems.* Springer.
[27] Leticia Montalvillo and Oscar Díaz. 2016. Requirement-Driven Evolution in Software Product Lines: A Systematic Mapping Study. *J. Sys. and Soft.* 122 (2016), 110 – 143.
[28] Sophia Nahrendorf. 2017. *Evolution-aware Modeling and Analysis of Software Product Lines.* Masterthesis. TU Braunschweig. (in German Language).
[29] L. Neves, P. Borba, V. Alves, L. Turnes, L. Teixeira, D. Sena, and U. Kulesza. 2015. Safe Evolution Templates for Software Product Lines . *J. Sys. and Soft.* 106 (2015), 42 – 58.
[30] Michael Nieke, Gil Engel, and Christoph Seidl. 2017. DarwinSPL: An Integrated Tool Suite for Modeling Evolving Context-aware Software Product Lines. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems.* 92–99.
[31] Michael Nieke, Christoph Seidl, and Sven Schuster. 2016. Guaranteeing Configuration Validity in Evolving Software Product Lines. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems.* ACM, 73–80.
[32] Andreas Pleuss, Goetz Botterweck, Deepak Dhungana, Andreas Polzer, and Stefan Kowalewski. 2012. Model-Driven Support for Product Line Evolution on Feature Level. *J. Sys. and Soft.* 85, 10 (2012), 2261 – 2274.
[33] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques.* Springer.
[34] Daniel Romero, Simon Urli, Clément Quinton, Mireille Blay-Fornarino, Philippe Collet, Laurence Duchien, and Sébastien Mosser. 2013. SPLEMMA: A Generic Framework for Controlled-Evolution of Software Product Lines. In *Proc. Int'l Soft. Product Line Conf.* ACM, 59–66.
[35] Ina Schaefer, Rick Rabiser, Dave Clarke, Lorenzo Bettini, David Benavides, Goetz Botterweck, Animesh Pathak, Salvador Trujillo, and Karina Villela. 2012. Software Diversity: State of the Art and Perspectives. *Int. J. Softw. Tools Technol. Transf.* 14, 5 (2012), 477–495.
[36] Sandro Schulze, Oliver Richers, and Ina Schaefer. 2013. Refactoring Delta-Oriented Software Product Lines. In *Proc. Int'l Conf. Aspect-Oriented Software Development.* ACM, New York, NY, USA, 73–84.
[37] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2013. Capturing Variability in Space and Time with Hyper Feature Models. In *Proc. Int'l Workshop on Variability Modeling of Software-intensive Systems.* ACM, Article 6, 8 pages.
[38] Christoph Seidl, Ina Schaefer, and Uwe Aßmann. 2014. Integrated Management of Variability in Space and Time in Software Families. In *Proc. Int'l Soft. Product Line Conf.* ACM, 22–31.
[39] Mikael Svahnberg and Jan Bosch. 1999. Evolution in Software Product Lines: Two Cases. *J. Softw. Maint.* 11, 6 (1999), 391–422.
[40] Maurice H. ter Beek, Erik P. de Vink, and Tim A. C. Willemse. 2017. Family-Based Model Checking with mCRL2. In *Proc. Int'l Conf. Fundamental Approaches to Software Engineering.* Springer, 387–405.
[41] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Computing Surveys* 47, 1, Article 6 (2014), 45 pages.
[42] Alexander von Rhein, Thomas Thüm, Ina Schaefer, Jörg Liebig, and Sven Apel. 2016. Variability Encoding: From Compile-Time to Load-Time Variability. In *J. Logical and Algebraic Methods in Prog.*, Vol. 85. 125 – 145.