Issues in Model-Driven Behavioural Product Derivation

Paul Istoan
ISC Department, CRP Gabriel Lippmann, Luxembourg
LASSY, University of Luxembourg
Triskell Research Team, IRISA/INRIA Rennes, France
istoan@lippmann.lu

Nicolas Biri
ISC Department, CRP Gabriel Lippmann, Luxembourg
biri@lippmann.lu

Jacques Klein
LASSY and SnT Interdisciplinary Center for Security, Reliability and Trust, University of Luxembourg, Luxembourg
jacques.klein@uni.lu

ABSTRACT

Model Driven Engineering (MDE) was identified as a viable software development paradigm to help improve the product derivation phase of the Software Product Line (SPL) engineering process. Existing model-driven derivation approaches fail to properly address the behavioural derivation part, yielding a frustrating situation. In this paper we first introduce a model-driven derivation approach that combines Feature Diagrams (FD) and model fragments. We then identify and analyse several issues that emerge during the derivation process. We show that the order in which models associated to selected features are composed has a great impact on the end result of the derivation. We also present a particular class of features called *disjoint* and prove that current composition operators do not offer any viable solution to compose them. Finally, we argue that insufficient information available to composition operators leads to derivation results that do not satisfy user requirements.

Keywords

Model Driven Engineering, Software Product Lines, Model composition, Aspect Oriented Modelling.

Categories and Subject Descriptors

D [**2.2**]: Software Engineering: Design Tools and Techniques

1. INTRODUCTION

Constant market evolution triggered an exponential growth in the complexity and variability of modern software solutions. Software Product Lines (SPL), or *software families*, are rapidly emerging as a viable and important software development paradigm [33] to address these issues. They promise to help develop flexible, cost-effective software systems and support a high level of reuse. Success stories [1] from companies like Nokia, Motorola or HP prove that use of SPL approaches generates important quantitative and qualitative gains of productivity, product quality and customer satisfaction. Several definitions of the concept can be found, from Clements and Northrop [33] to Bosch [4].

SPL engineering focuses on capturing existing *commonality* and *variability* between several software products [5]. Instead of describing a single system, a SPL describes a *set of products* in the same domain, by distinguishing between elements common to all SPL members and those that may vary from one product to another. It is highly encouraged to *reuse core assets*, which extend beyond simple code reuse and may include the architecture, reusable software components, domain models, requirements statements, documentation or test cases [43]. The SPL engineering process contains two major intertwined steps, described in the next paragraphs. A graphical representation of the entire process can be found in [33].

Domain Engineering (*development for reuse*): past experience in building systems is collected, organized and stored in the form of reusable core assets [35].

Application Engineering (*development with reuse*): complete process of constructing the final products from core assets. It is a complex process that plays a key role in SPL engineering, characterized as "tedious and error-prone" [11]. Therefore, we could assume it has been exhaustively studied. This is actually not the case, with most SPL research focusing on the domain engineering phase, neglecting or insufficiently addressing *product derivation*.

To address this frustrating situation, SPL engineering has recently turned towards Model-Driven Engineering (MDE) [23], capable to offer viable solutions for improving product derivation. The result of a model-driven derivation process is the model of an individual product. Two types of such models are possible: *structural* and *behavioural*. For a complete product representation, both are required. Most derivation approaches address only the structural part, neglecting the behavioural one. That is why, throughout this paper, we focus on model-driven behavioural product derivation.

Most model-driven product derivation approaches use decision models, with *Feature Diagrams* (FD) [21] extensively used for this purpose. A FD presents a hierarchical structuring of high level product functionalities, but tells very little about how the features are combined into an actual product. An emerging research direction is to associate model fragments to features. Structural and behavioural model frag-

VaMoS '11 January 27-29, 2011 Namur, Belgium

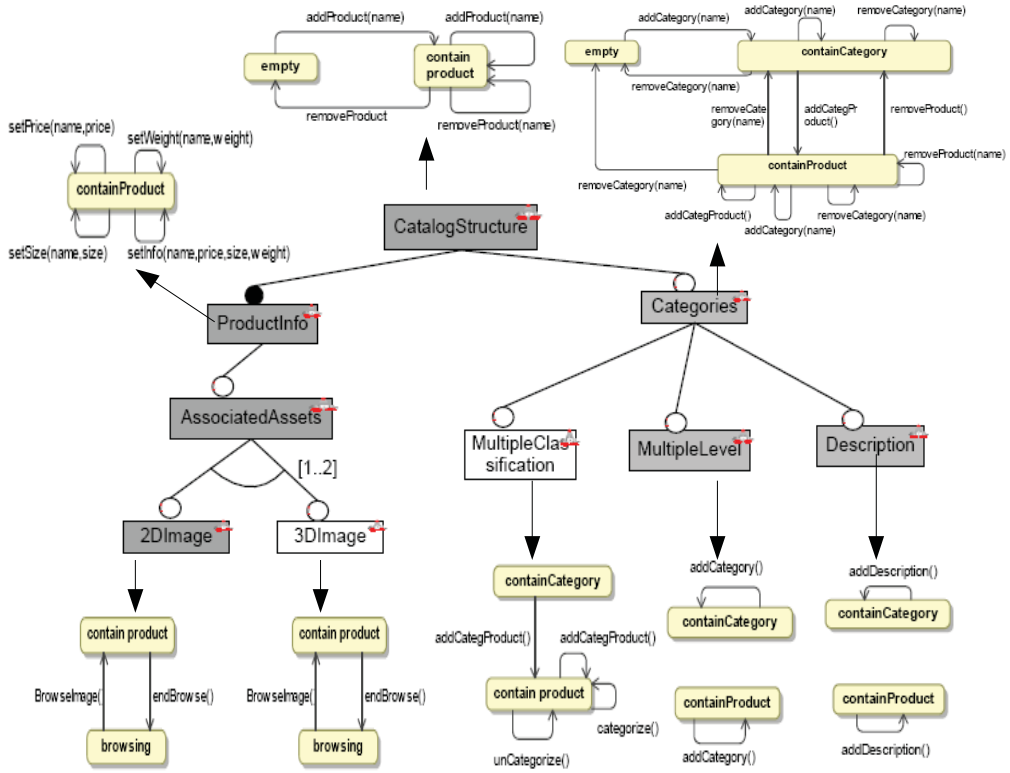

Figure 1: ItemCatalog Feature Diagram and associate models

ments can be attached, thus covering both types of product representations. In this paper, we analyse into more detail this particular derivation approach.

FD offer limited quantitative and qualitative information that only supports the initial step of the derivation process, the *feature selection*. In a MDE derivation approach, it results in a set of unconnected features with model fragments associated to them. In a second step, the actual product representation is obtained by composing these models. We identify this phase of the derivation process as the one that raises the most problems. Therefore, the overall objective of the paper is to identify and present some issues related to model-driven behavioural product derivation. It is not our intention to formally prove properties related to product derivation (composition), but to give an initial intuitive idea of possible problems that might occur. The identified issues are briefly listed in the following, and thoroughly discussed later on in the paper.

Throughout the paper, several model composition operators are used. Due to limitations in space, they are not presented in detail. The first issue identified is the *order of model composition*. Without further guidance provided by the FD, several orders for composing the models are possible. We could assume that all possible orders lead to the same resulting product. We show that this is actually not the case and that the order in which models are composed influences the end result. It is also studied under what circumstances this happens: type of models composed, type of selected features, influence of the composition operators applied. A second identified issue concerns the actual model composition process. We address a particular type of composition that can frequently happen in the SPL field and which poses problems: *composition of models that have no common elements*. After testing several existent composition operators, we conclude that there are no viable solutions for this issue in current research literature. Finally, we point out the fact that current composition operators don't make use of information provided by the FD, which leads to semantically incorrect results.

The paper is structured as follows: Section 2 provides further information on Application Engineering, introduces Feature Diagrams and details a particular product derivation approach, based on model composition, that attaches models fragments to FD. The contribution of the paper is presented in Section 3. Using two examples, we exemplify first the derivation process. Different composition orders and composition operators are used. We then focus on detailing the derivation issues mentioned in the previous paragraph: importance of composition order, composition of models sharing no common element, composition of models attached to alternative features. Section 4 concludes the paper and presents some future research directions.

2. BACKGROUND

This section provides first a brief overview of Application Engineering and mentions several existing product derivation approaches. It then discusses Feature Diagrams as variability modelling method in SPL engineering and explains their role during product derivation. Finally, a particular product derivation approach that combines feature diagrams and model composition is explained and exemplified.

2.1 Application Engineering

Application Engineering, or *product derivation*, consists

of assembling applications from the existing core assets base (already defined during domain engineering), according to specific customer requirements.

In recent years, MDE has played a prominent role in the entire SPL engineering process. According to the derivation technique used, currently available model-driven product derivation approaches can roughly be organized into:

Derivation by configuration: based on the parametrization and/or composition of core assets. Product features are selected according to specific customer requirements, then core assets are automatically assembled. Some relevant approaches are [22, 9, 6, 36, 20, 2, 19, 42].

Derivation by transformation: promotes transforming core assets and relies extensively on MDE concepts. It uses *models* to represent core assets and *model transformations* to manipulate them and generate concrete products. Examples of such approaches are [43, 18, 28, 17, 37, 30].

Two different types of models, providing different views of the derived products, can be obtained as result of model-driven derivation: *structural* - for instance UML class or component diagram; *behavioural* - for instance UML sequence, activity or statechart diagram. For a complete and comprehensive view of the derived products both representations are necessary. Although extremely important, behavioural product derivation did not receive sufficient attention from the SPL community, yielding a frustrating situation. Examples of behavioural product derivation approaches are [43, 6, 36, 17].

2.2 Feature Diagrams

Feature Diagrams (FD) [21] are a popular way to model variability in SPL engineering. They provide a concise and explicit way to: describe allowed variabilities between products, represent feature dependencies, guide the selection of features allowing construction of specific products.

A feature diagram provides a graphical tree-like notation depicting the hierarchical organization of features. The root of the tree refers to the complete system, being progressively decomposed and refined using "alternative", "and", "xor" features. Relations between nodes are materialized by *decomposition edges* and *textual constraints*. The latter describe certain dependencies between features like *require* or *mutex*. Commonality is modelled using mandatory features, while variability using optional, alternative, and or-features.

For the last 20 years, research and industry have developed several FD languages. Important extensions have been added to the initial proposal of Kang et al. [21]: annotate features with cardinalities [8]; introduce attributes for representing choice [7]; feature categories and annotations [22]; extend FD with different kinds of relations [41].

2.3 Associate model fragments to features

Despite their popularity and extensive use, FDs provide only a hierarchical structuring of high level product functionalities. The need arises to combine them with other product representations. An emerging research direction is to associate model fragments to features. In this way, the FD defines the product line variability and acts as a decision model for product derivation, while each feature has an associated implementation. Two existing possibilities for associating models to features briefly explain in the following.

Czarnecki et al. [6] propose a general template-based approach to map FD to concrete representations, using structural or behavioural models. They use a *model template* representing a superimposition of all variants, with elements related to corresponding features through *annotations*. (*presence conditions* and *meta-expressions*). A derived product is specified by creating first a feature configuration, then the model template is automatically instantiated.

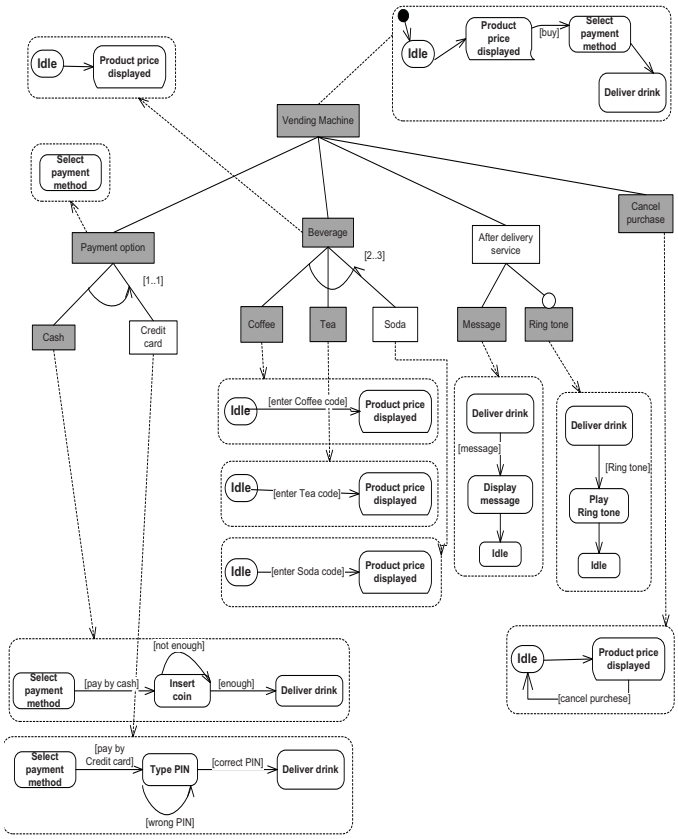

Figure 2: Vending Machine feature diagram and associated statechart fragments

A second approach comes from Perrouin et al. [36]. They use a generic feature meta-model that supports a wide variety of existing FD dialects and an assets meta-model defined using a subset of UML. A composite association between the *Feature* and *Model* meta-classes is introduced, specifying that a feature may be implemented by several model fragments. Initially exploited with class diagrams, the approach allows any kind of assets to be associated to features. We analyse how the product derivation is performed for this approach into more detail throughout the rest of the paper and identify several existing issues. For better understanding, the approach is explained by means of an example.

We consider the *ItemCatalog SPL*, introduced in [10]. It describes a catalogue for an e-commerce application, shown in Figure 1. A catalogue provides items descriptions (*ProductInformation*) and optionally their organization into *Categories*. Each item can only be associated with one media file type, choices being *2DImage* or *3DImage*. Three possible types of categories are available: *MultipleClassification* (allows an item to belong to several categories), *MultipleLevel* (describes support for nested categories) and *Description* (gives additional category information). These features are organized in the FD from Figure 1. As we are interested in behavioural product derivation, we choose to associate UML statechart fragments to each feature.

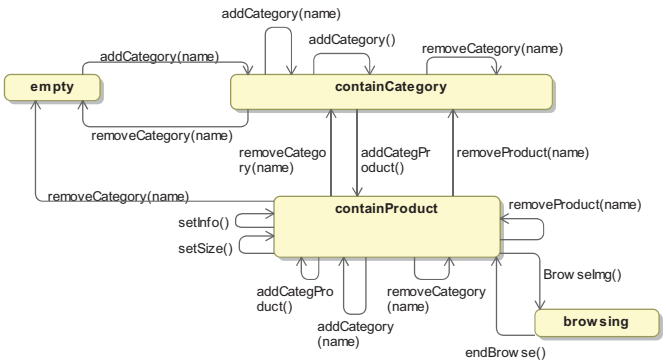

Figure 3: Derived product of the ItemCatalog SPL

Product derivation starts by selecting features satisfying specific user requirements and obtaining a valid product configuration. Features marked in grey in Figure 1 correspond to a particular product of the ItemCatalog SPL. Features selected in the configuration phase are not connected between themselves in any way, any neither are the model fragments associated to them. Therefore, the obvious question that arises is: "what should be done after feature selection?".

3. ISSUES IN BEHAVIOURAL PRODUCT DERIVATION

In order to obtain a concrete representation of the derived product, model fragments associated to the selected features need to be composed into a single model. We begin this section by presenting several successful product derivations, using model composition, on two different examples. Then, we focus on identifying and discussing several problems that arise during the composition process. First of all, without any further information provided by the FD, composition of these models can be performed in any order. This does not pose any problems if the same product is obtained for every possible composition order. We show that this is not the case and that the order in which models are composed has a direct impact on the result of the derivation: different orders imply different derived products. Secondly, once a composition order has been decided upon, models need to actually be composed. Due to the nature of SPL, it is frequently the case that models that are composed have no common elements. We study how several model composition approaches address this situation and prove the lack of viable solutions to solve it. Finally, we show that composing models associated to features connected by different cardinality relations in the FD gives incorrect results that do not satisfy initial user requirements.

3.1 Introduction of examples used

Throughout this paper, we make use of two different examples. The first one is the *ItemCatalog SPL*, introduced in Section II.C and described in Figure 1.

To increase the validity and consistency of our experimentations, a second example is used: a *Vending Machine SPL* inspired from [13]. It offers customers a selection of drinks like coffee, tea or soda. Different payment methods are possible: cash or credit card. Additional services are proposed after product delivery: it is mandatory to display a message signalling the end of the purchase or optionally play a ring tone. Several constraints are imposed. Only one payment option is possible for a particular vending machine. Each machine needs to sell at least two types of beverages. Statechart fragments are attached to each feature and describe their behaviour. The example is presented in Figure 2, features marked in grey representing a valid feature selection.

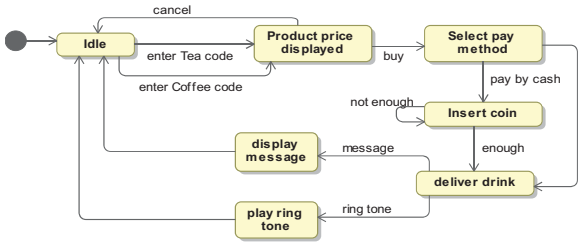

Figure 4: Derived product of the Vending Machine SPL

3.2 Examples of successful derivation

The purpose of this subsection is to detail the composition process on the *ItemCatalog* and *Vending Machine SPL* examples. For each of them, several derivation examples are presented in which different orders of composition are used.

To begin the composition process, a composition operator must be first selected. The MDE field proposes several composition operators that automate and support this process. Throughout this paper, we use several model-driven composition operators. Due to space limitations, we only briefly introduce their main characteristics. For a detailed description of their functioning, we invite the reader to analyse the original papers in which they are presented.

The first composition operator used is the one proposed by France et al. [16],[15]. It is a symmetric, signature-based, generic composition approach, independent from any modelling language. Model elements to be composed must be of the same syntactic type. The approach is based on the *systematic merging of matching elements*. First, elements to be merged are identified by *element matching*. To automate this activity, each element has an associated *signature*, determining its uniqueness. The matching process actually consists of matching signatures of elements. Then, matched model elements are *merged* together. Flexibility is obtained by using *composition directives* [39] (pre-merge, post-merge). They can be used during composition to force matches, disallow merges or override default merge rules. The approach is implemented in the *Kompose* [15] tool.

With several models to be composed, multiple composition orders are possible. In the following, we use the composition operator of France et al. and different orders of composition for the same set of selected features.

The selected features are {*catalogStructure, ProductInformation, 2DImage, Categories, MultipleLevel, Description*}, marked in grey in Figure 1. The first composition order is {*2DImage, MultipleLevel, catalogStructure, ProductInformation, Description, Categories*}. The following composition algorithm is used: apply the matching and merging phases to the first two features, thus composing their statechart fragments. The result is further composed with the statechart of the next feature in the order. This process is applied recursively until all features are composed. Statecharts corresponding to *catalogStructure* and *Categories* features both contain transitions between states *empty* and *containProduct*. To avoid any composition problems, composition directives that remove transitions *addProduct* and *removeProduct* from the statechart of *catalogStructure* feature are used. The final result is shown in Figure 3.

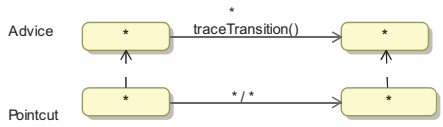

Figure 5: Aspectual feature Trace

We select a new composition order: {*catalogStructure, 2DImage, MultipleLevel, ProductInfo, Description, Categories*}. The same composition algorithm as before is applied. The result obtained is the same as for the previous case, although the composition order was changed. Even for a third different order: {*MultipleLevel, ProductInfo, catalogStructure, Description, Categories, 2DImage*} the end result remains the same as in the first two cases. Therefore, at a first impression, this experiments give an initial intuition that, using statecharts, the order in which models associated to features are composed does not influence the final result obtained, which remains the same in all cases.

To increase the validity and consistency of these results, the same approach and same composition operator are applied on the *Vending Machine* example. The selected features are {*Vending machine, Payment option, Cash, Beverage, Tea, Coffee, Message, Ring tone, Cancel purchase*}. First order chosen is {*Cash, Payment option, Vending machine, Beverage, Tea, Coffee, Message, Ring tone, Cancel*}. The final result is shown in Figure 4. Two other orders are selected: {*Coffee, Message, Beverage, Cancel, Cash, Vending machine, Payment option, Ring tone, Tea*} and {*Message, Cancel, Tea, payment option, Coffee, Vending machine, Cash, Beverage, Ring tone*}. The results obtained are identical to the previous one in Figure 4. Therefore, as before, there is the same intuition that the composition order has no influence on the end result.

Based on these results, one may think that for model-driven product derivation, when statecharts are used, the order of composition has no influence on the end result. Nevertheless, for the first example, with 6 features selected, 6!(720) composition orders are possible. For the second one, 9 features lead to 9!(362.880) possibilities. Due to space limitations, we have only shown here 3 such possibilities for each example, although many more have been performed in practice. Therefore, we do not claim that, simply based on these examples, for all statecharts the order of composition is not important. The results obtained serve only as an initial intuitive idea towards this conclusion, but we are aware that a formal proof is mandatory. This actually represents one of the future research directions for our work.

3.3 Introducing "aspectual" features

Throughout this subsection we identify and give initial details of a particular class of features, called *aspectual*, that prove that the *order of feature composition* is a real problem encountered in model-driven behavioural product derivation. We show, on the previous examples, that inserting them at different places in the composition order changes the end result. This contradicts the initial intuition presented in the conclusions of the previous sub-section.

A new feature called *Trace* is introduced in the *ItemCatalog SPL*. It calls the *traceTransition()* method every time a transition between two states is performed. This behaviour is *cross-cutting*, as it applies at several different places for every derived product containing it. In *Aspect-Oriented Modelling (AOM)*, cross-cutting concerns are called *aspects*. Therefore, "Trace" is an *aspectual feature*. It is presented in

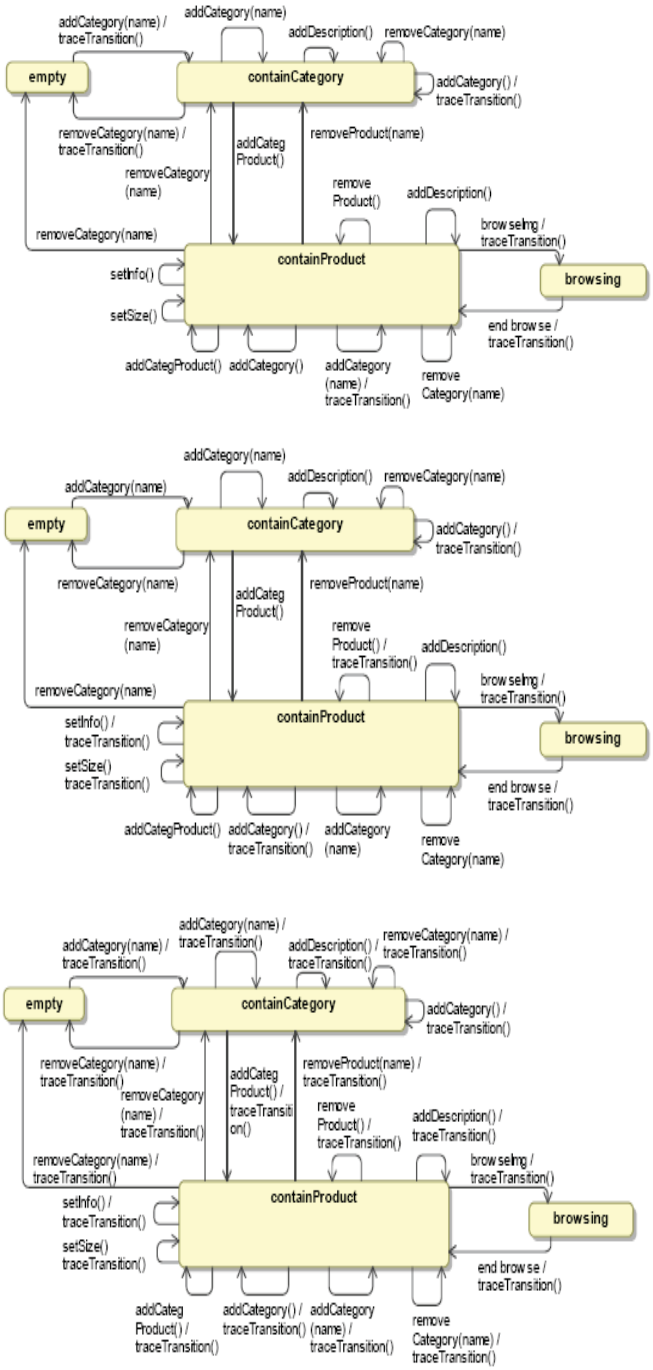

Figure 6: Derived products of ItemCatalog SPL with Trace feature - different composition orders

Figure 5 and contains two parts: a *pointcut* that identifies places in other models where the new behaviour is inserted; an *advice* that describes the new behaviour.

We add *Trace* to the set of selected features. As before, we try to use the signature-based composition operator of France et al. Because an *aspectual feature* applies in more than one place, it requires an additional phase in the composition process: *quantification* (detection of all places where it is applied). The composition operator of France et al. does not offer this functionality, so it is not powerful/expressive

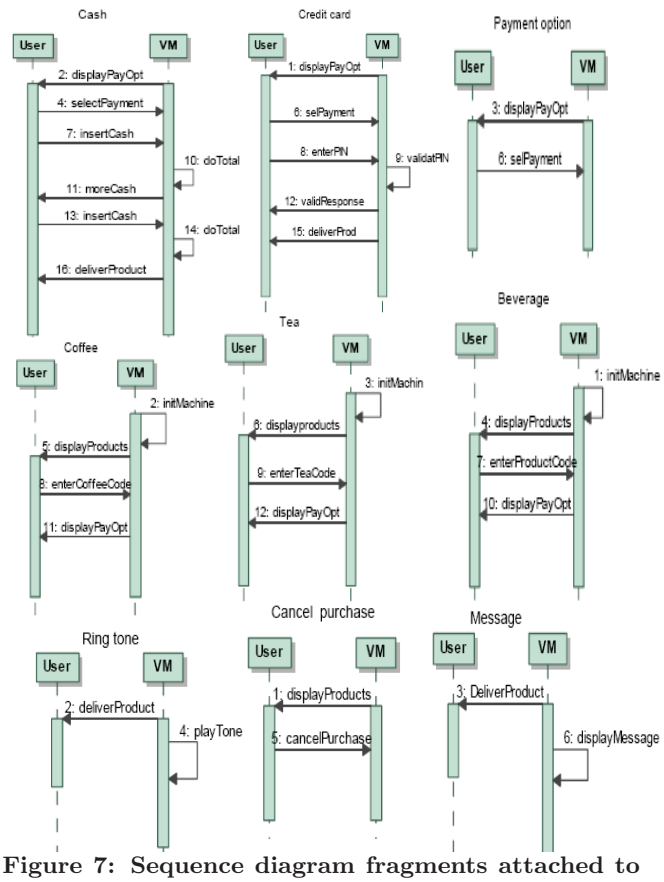

Figure 7: Sequence diagram fragments attached to features of the Vending Machine SPL

enough to properly compose it. Using other "classical" composition operators like *Bernstein's generic merge* [38], *Rational Software Architect combine feature* [27], *Epsilon merging language* [12], *TReMer+* [40] or *UML2 package merge* [34] does not solve the problem, as neither of them offers the required quantification phase.

A possible solution to this issue comes from the AOM field. AOM *model weavers* propose a *quantification* [14] phase which allows them to easily compose *aspectual features*. Due to this, we decide to replace the composition operator of France et al. currently used, with the one proposed by Morin et al. called GeKo [29].

GeKo is a generic, tool-supported, AOM composition and weaving approach, easily adaptable to any domain specific modelling language. It uses two operands for the composition called *aspect* and *base*. It supports the *pointcut-advice* composition mechanism and keeps a graphical representation of the weaving between an aspect and a base model. First, using a Prolog-based pattern matching engine, it detects all the places in the base model where the advice will be applied. These places are called *join-points*. Then, a generic approach of composition of two models at the level of the detected join-points is applied.

Using the same selection of features as before to which we add the "Trace" feature, we apply this new composition operator on the *ItemCatalog SPL*. The first composition order used is {*2DImage, MultipleLevel, Trace, catalogStructure, ProductInfo, Description, Categories*}. The same composition algorithm described before is used but, whenever composing two models, we consider the first one to be the *base*

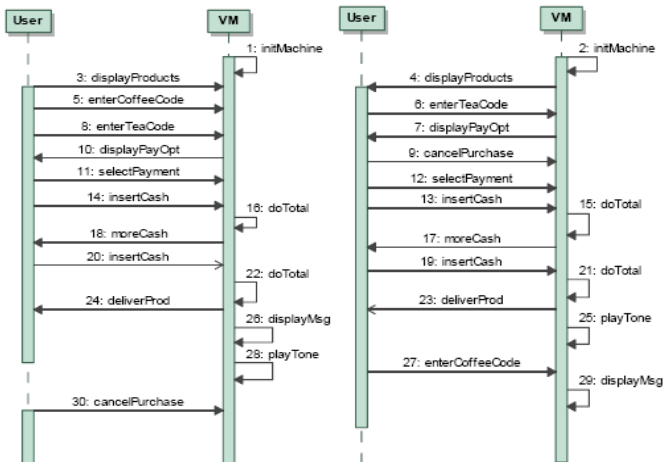

Figure 8: Derived products (sequence diagram) of Vending Machine SPL - method of France et al.

and the second one the *aspect*. The result is shown in the top diagram of Figure 6. Feature "Trace" adds a *traceTransition()* method on several transitions in the resulting model. For clarity reasons, we mark with a special symbol () all the changes caused by the composition of this feature. This is done for all the derivation orders we present. The second composition order is {*catalogStructure, 2DImage, MultipleLevel, ProductInfo, Trace, Description, Categories*}. A close analysis of the result, shown in the middle diagram of Figure 6, reveals several differences compared to the result of the previous composition: as the "Trace" feature has a different place in the composition order, the *traceTransition()* method it adds is applied now to other transitions than in the first case. A new composition order {*MultipleLevel, ProductInfo, catalogStructure, Description, Categories, 2DImage, Trace*} is applied, with result shown in the bottom diagram of Figure 6. It can be easily noticed that the result is different from the first two cases. As the "Trace" feature is composed last, the *traceTransition()* method is added on all transitions of the resulting model.

Different results were obtained for each composition order. Compared to the initial *ItemCatalog SPL* example, the major difference is *the introduction of the "Trace" aspectual feature*. We observe that inserting it at different places in the composition order leads each time to different results. This example leads to the idea that for behavioural product derivation, when *aspectual features* are present, the order of model composition is relevant.

3.4 Importance of composition order

This subsection gives further details on the composition order problem previously identified. We show that the initial intuition presented in Section III.A was false, and explain that by changing the behavioural model used from statechart to sequence diagram, the order in which models are composed is always relevant (different orders leading to different derived products). This result is confirmed even if the composition operator applied is changed.

So far, statecharts were used as behavioural models attached to features. We analyse what happens if the type of behavioural model used is changed. Therefore, we use *sequence diagram*, which describe how processes operate with one another and in what order. We use in the following the *Vending Machine SPL* example from Figure 2, with the as-

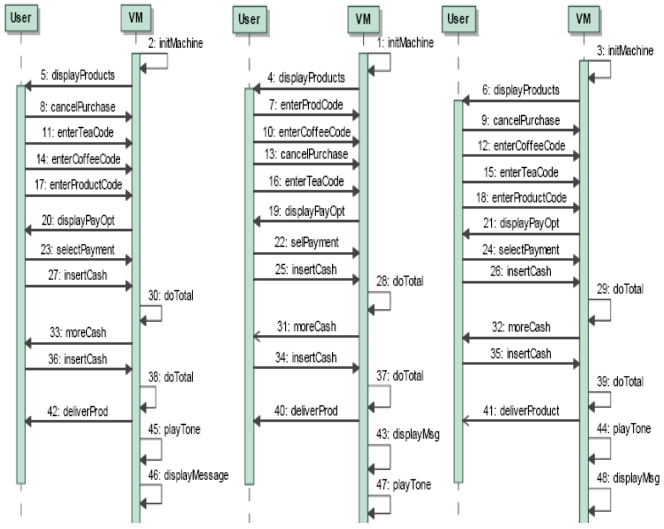

Figure 9: Derived products (sequence diagram) of Vending Machine SPL - method of Klein et al.

sociated sequence diagram fragments described in Figure 7.

Using the same composition algorithm, we apply the operator of France et al. using different composition orders. The first order selected is {*cash, payment option, beverage, coffee, tea, message, ring tone, cancel*}, with the result shown in the left diagram of Figure 8. For the order {*tea, cancel, cash, ring tone, pay option, coffee, message, beverage*} the result is shown in the right diagram of Figure 8. The two composition results are different and thus represent distinct products. This is different from the result obtained in Section III.B on the same example, when statecharts were used. It shows that by simply changing the behavioural model from statechart to sequence diagram, the composition order has an obvious impact on the end result.

One could reason that the result obtained was influenced by the composition operator applied, as the one of France et al. is not specifically designed for composing sequence diagrams. For this reason, we choose a more expressive composition operator designed specifically for this purpose, the one of Klein et al. [25, 26]. This AOM composition approach implements a semantic-based weaving of scenarios and supports the *pointcut-advice* composition mechanism. It proposes a new interpretation for *pointcuts*, expressed as sequence diagrams, to allow them to be matched at the level of *join-points*. With this new way of specifying join-points, the composition of the advice with the detected part cannot any longer be a replacement of the detected part by the advice. The authors thus propose and formally define a new merge operator, called *left amalgamated sum*, tailored for composing behavioural models. The approach is implemented in the Kermeta framework [32].

The first composition order is {*cash, pay option, beverage, coffee, tea, message, ring tone, cancel*}. The result is shown in the left diagram of Figure 9. For the orders {*tea, cancel, cash, ring tone, payment option, coffee, message, beverage*} and {*message, cash, beverage, ring tone, tea, payment option, coffee, cancel*} the results are shown in the middle and right diagrams of Figure 9. As for the operator of France et al., the results obtained are different, corresponding to different derived products. This confirms the previous results.

This results show that, when using sequence diagrams, the order of composition is relevant and influences the products that are derived. This assumption remains valid even if the composition operator used is changed.

To find solutions to the composition order problem, we have analysed research literature from the SPL, AOM and MDE fields. A possible solution comes from Kienzle et al. [24]. They present the *Reusable Aspect Models (RAM)* approach, which allows to express the structure and behaviour of a complex system using class, state and sequence diagrams encapsulated in several aspect models. In their car crash crisis management system (CCCMS) case study, they introduce a reusable concern called *workflow management* that allows modellers to define and execute workflows using concepts like *sequentialExecution, paralleExecution, loopedExecution, Executable or WorkflowEngine*. he problem of this approach is that it assigns the creation of workflows to be executed to the modeller. So it is not automated in any way, and the solution provided is not generic or reusable. Workflows need to be separately defined for all the possible products of a SPL. This is infeasible and almost impossible for large SPLs.

3.5 Composition of disjoint features

Once a particular composition order is determined, the next logical step of the derivation process is the actual model composition. This subsection focuses on a second problem encountered during behavioural product derivation: *composition of disjoint features*. Frequently, models that need to be composed have no common elements. We analyse the manner in which several composition approaches address this situation and prove the lack of viable solutions.

In SPL engineering, features from different branches of a FD are frequently not related between themselves, as they represent different system functionalities. Therefore, the models implementing them might not have any common elements. This can also be the case for features representing different variants of a particular product functionality. We call this type of features *disjoint*. Two features are called *disjoint* if their associated models have no common elements.

We analyse features *Credit card* and *Message* from the *Vending Machine SPL* in Figure 7. They share no element and are a good example of *disjoint features*. Several other examples of pairs of disjoint features can be noticed, like *(message,cancel purchase)*, *(payment option,ring tone)*, *(beverage,message)* and others. In the composition orders used throughout the previous section, we explicitly avoided situations where disjoint features needed to be composed.

In this subsection, we analyse this type of composition. Intuitively, bases on the semantics of the features *Message* and *Credit card*, it is obvious that *Credit card* should come first, followed by *Message*. Nevertheless, composition of such features has to be done automatically during product derivation, by applying a certain model composition approach. In the following, we analyse several model composition approaches to see how they address this issue.

We study first the composition approach of France et al. [16]. As presented in Section 3.2, it implements a two-step composition mechanism: *matching* (identifies model elements that have to be composed) and *merging* (matched model elements are merged). The matching process relies on a *signature-based approach*: two elements match if their signatures are equivalent. When this approach is applied for *disjoint features*, the matching phase does not return any el-

ements to be merged. The problem is briefly addressed in [16]: "if the model signatures are not the same then the models are not composed". We also analysed how this problem was solved in Kompose, the tool implementing the approach: the result returned by the tool in this case is a single model regrouping the initial models that had to be composed, but with no relation or connection between them.

Next approach analysed is the one of Klein et al. [25], introduced in Section 3.4. The authors propose a non commutative composition operator called *left amalgamated sum*, defined in [25] using three models (pointcut M_0, advice M_1, base M_2) and two morphisms (pointcut to base and pointcut to advice). For more details, we invite the reader to refer to [25]. For disjoint features, as the models have no common element, this implies that the pointcut model used by this approach is empty ($E_0 = \emptyset$). With this observation in mind, we analyse the formal definition of the left amalgamated sum presented in [25] on page 23, to determine the result of the composition:

$I = I_1 \cup \{i_2 \in I_2 | \nexists i_0 \in I_0, g_0^{-1}(i_2) = i_0\}$ (1)

But $E_0 = \emptyset$ (2)

$(1), (2) \Longrightarrow I = I_1 \cup I_2$

$E = \{e_1 \in E_1 | \exists e_0 \in E_0, \exists e_2 \in E_2, f_1^{-1}(e_1) = e_0 \wedge g_1(e_2) = e_2\} \cup \{e_1 \in E_1 | \nexists e_0 \in E_0, f_1^{-1}(e_1) = e_0\}\{e_2 \in E_2 | \nexists e_0 \in E_0, g_1^{-1}(e_2) = e_0\}$ (3)

$(2), (3) \Longrightarrow E = \emptyset \cup \{e_1 \in E_1\} \cup \{e_2 \in E_2\} = E_1 \cup E_2$

This means the result contains all the elements of the initial models (base and advice). Analysing the ordering relation $\leq$ imposed by lifelines and messages reveals that, in the final result: elements coming from the same model remain ordered as in the original model; there is no ordering or relation between elements coming from different models. This result is similar to the one proposed by Kompose.

We also study GeKo [29] from Morin et al., introduced in Section 3.3. Models to be composed are called *base* and *aspect*. It is the use of a third model called *pointcut* and two morphisms allowing the identification of base model elements which have to be kept, removed or replaced with those of advice. For disjoint features, the pointcut model is empty, as there are no matching elements between base and advice. Using the composition formalization and notations provided in [29], we obtain:

$R_{keep} = \{obj | obj \in base\}$, $R_- = \emptyset$, $R_\pm = \emptyset$, $R_{ad\pm} = \emptyset$, $R_+ = \{obj | obj \in advice\}$

Using the definition of the *generic composition* in [29] we get: $result = R_{keep} \cup R_+ \cup R_{ad\pm} \setminus R_- \setminus R_\pm$. The final outcome is: $result = \{obj | obj \in base\} \cup \{obj | obj \in advice\}$. The result contains all the elements of the initial models, but they are not related to each other in any way.

Another composition approach we analyse is *SmartAdapters* [31] from Morin et al. For the composition part, common elements in the SmartAdapters models (SAMs) being merged are unified. SmartAdapters allows the user to create his own composition operators. In addition to this, it offers several predefined ones based on the proposals of Barais et al. in [3]. For behavioural models, three different kinds of predefined base-base merge operators are proposed:

Amalgamated sum: merges two sequence diagrams that share common elements. Applied to disjoint features, the same results as the approach of Klein et al. are obtained.

Sequential composition: glue together two models. For disjoint features, two composition orders are always possible. The user must first decide which is the precedence order desired. Then, in the resulting model, certain events might not be ordered or can be concurrent, although this may not be at all the intention.

Inclusion: allows the inclusion of one model into another at points explicitly specified by the user. For disjoint features, this approach requires the user to explicitly define the insertion points. There is no indication of which model is the one to be inserted.

Our analysis reveals that the composition of models sharing no common elements was not explicitly addressed by any approach. All of the methods analysed produced unsatisfactory results, incorrect semantically and which do not capture at all the intention of the user. This is frustrating, as this type of composition can frequently happen in the SPL field.

3.6 Composition of alternative features

This sub-section discusses a last identified problem that can occur during product derivation: composition of *alternative features*.

We analyse the *beverage* feature from Figure 2. It has three possible implementation variants: *coffee*, *tea* and *soda*, connected by a *cardinality relation* indicating that at least two of them must be selected in every derived product.

In Section III.D features *coffee* and *tea* were selected as part of valid feature configurations and used in a total of five derivation examples (see Figure 8 Figure 9). When composed, each feature introduces a specific message (*enterTeaCode()* and *enterCoffeeCode()*). A more thorough analysis of the figures reveals that, in all the models corresponding to the final derivation result, both messages are always present, in different orders, sometimes with other messages intercalated between them. This implies that, in all of these derived products, the user always performs both behaviours defined by these features. This contradicts the semantics of the cardinality relation between the two features: both behaviours specified by these features should be present in every derived product, but the user has the alternative of performing either one or the other or both of them. One possible way to correctly compose these two features with other ones would be to add both *enterTeaCode()* and *enterCoffeeCode()* messages, each of them specified within an *alternative fragment* in the resulting sequence diagram.

None of the composition operators presented in this paper offers such a possibility, or any other alternative solution to this problem. This problem is due to the fact that composition operators do not take into account any information or relations that FDs might provide.

4. CONCLUSIONS

Software Product Lines are receiving a lot of attention from the research community and are more and more applied by industry. Most of the efforts of the SPL community focused on the domain engineering phase, neglecting or insufficiently addressing the application engineering part of the process. The vast majority do not cover the behavioural derivation part, leading to incomplete product representations. Due to this, MDE solutions were found suitable to help improve application engineering.

Throughout this paper, we focused on model-driven product derivation solutions, and in particular on approaches that combine Feature Diagrams and model fragments. For this type of approaches, we noticed that, due to the limited amount of information provided by the FD, the product line

engineer is limited to making a feature selection. To obtain an integrated view of the derived product, models associated to the selected features need to be composed. The article focuses on this part of the derivation process, identifying and giving an initial indication of possible problems that might occur at this stage of the process.

We start by familiarizing the reader with the product derivation process using two different examples, for which we described several successful derivations. Two different composition operators are applied and several different composition orders chosen. Initial experimental results lead to an initial assumption that the order in which models are composed is not relevant. This initial intuition proved to be false, as we identified a particular class of features, called *aspectual*, for which we showed that the composition order has a major influence on the end derivation result.

We then analysed the influence of the type of behavioural model used on the obtained results. Statecharts, used initially, were replaced with sequence diagrams, and we successfully showed that for this type of model the order of composition is always important: different orders resulted in different derived products. This conclusion remains true even if the composition operator used is changed.

A second issue identified and discussed concerns the actual composition process. We analyse a particular type of features that are often encountered in SPLs: *disjoint features*. They are of interest because the associated models have no common elements. We study how their composition is addressed by different existent model composition approaches concluded that there is no viable solution to this issues available. The last identified problem was that current model composition operators do not use any information provided by feature diagrams, leading to derivation results that do not satisfy the initial customer requirements and are semantically incorrect.

As the overall objective of this paper is to present some existing issues in model-driven product derivation, it leaves open a lot of research directions for the future. First of all, we are aware that a formal definition of properties of model composition in the context of product derivation is necessary and would be an important contribution, so this one of our goals for the near future. Secondly, we are currently working on providing solutions to the questions raised in this article. Our current research focuses on proposing a complete methodology to support model-driven behavioural product derivation using model composition and AOM approaches.

Acknowledgment

This work was partially funded by the SPLIT project (FNR and CNRS funding, http://wiki.lassy.uni.lu/Projects/SPLIT).

5. REFERENCES

[1] Software pproduct line conference - hall of fame. `http://splc.net/fame.html`.

[2] S. Apel, C. Kästner, and C. Lengauer. Featurehouse: Language-independent, automatic software composition. In *In Proc. ICSE*, 2009.

[3] O. Barais, J. Klein, B. Baudry, A. Jackson, and S. Clarke. Composing multi-view aspect models. In *Proceedings of the Seventh ICCBSS*, pages 43–52, Washington, DC, USA, 2008. IEEE Computer Society.

[4] J. Bosch. Design and use of software architectures: adopting and evolving a product-line approach. 2000.

[5] J. Coplien, D. Hoffman, and D. Weiss. Commonality and variability in software engineering. *IEEE Softw.*, 15(6):37–45, 1998.

[6] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE*, pages 422–437, 2005.

[7] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. Addison-Wesley Publishing Co., NY, USA, 2000.

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

[9] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Staged configuration through specialization and multilevel configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[10] K. Czarnecki and K. Pietroszek. Verifying feature-based model templates against well-formedness ocl constraints. In *GPCE*, pages 211–220, 2006.

[11] S. Deelstra, M. Sinnema, and J. Bosch. Experiences in software product families: Problems and issues during product derivation. In *SPLC*, pages 165–182, 2004.

[12] K.-D. Engel, R. F. Paige, and D. S. Kolovos. Using a model merging language for reconciling model versions. In *ECMDA-FA*, pages 143–157, 2006.

[13] A. Fantechi and S. Gnesi. Formal modeling for product families engineering. In *Proc. of the 12th SPLC*, pages 193–202, Washington, DC, USA, 2008. IEEE Computer Society.

[14] R. E. Filman and D. P. Friedman. Aspect-oriented programming is quantification and obliviousness. Technical report, 2000.

[15] F. Fleurey, B. Baudry, R. France, and S. Ghosh. Models in software engineering. chapter A Generic Approach for Automatic Model Composition, pages 7–15. Springer-Verlag, Berlin, Heidelberg, 2008.

[16] R. France, F. Fleurey, R. Reddy, B. Baudry, and S. Ghosh. Providing support for model composition in metamodels. In *Proceedings of the 11th IEEE EDOC Conference*, Washington, DC, USA, 2007. IEEE Computer Society.

[17] H. Gomaa. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[18] O. Haugen, B. Moller-pedersen, J. Oldevik, and A. Solberg. An mda-based framework for model-driven product derivation. In *Software Engineering and Applications*, pages 709–714. ACTA Press, 2004.

[19] F. Heidenreich, J. Kopcsek, and C. Wende. Featuremapper: Mapping features to models. In *Companion Proceedings of the 30th ICSE*, pages 943–944, New York, USA, 2008. ACM.

[20] L. Hotz, K. Wolter, and T. Krebs. *Configuration in Industrial Product Families: The ConIPF Methodology*. IOS Press, Inc., 2006.

[21] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. Feature-oriented domain analysis

(foda) feasibility study. Technical report, Carnegie-Mellon University Software Engineering Institute, November 1990.

[22] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[23] S. Kent. Model driven engineering. In *IFM*, pages 286–298, 2002.

[24] J. Kienzle, W. A. Abed, F. Fleurey, J.-M. Jézéquel, and J. Klein. Aspect-oriented design with reusable aspect models. *T. Aspect-Oriented Software Development*, 7:272–320, 2010.

[25] J. Klein, F. Fleurey, and J.-M. Jézéquel. T. aspect-oriented software development. chapter Weaving multiple aspects in sequence diagrams, pages 167–199. Springer-Verlag, 2007.

[26] J. Klein, L. Hélouët, and J.-M. Jézéquel. Semantic-based weaving of scenarios. In *Proceedings of the 5th AOSD Conference*, pages 27–38, New York, NY, USA, 2006. ACM.

[27] K. Letkeman. Comparing and merging uml models in ibm rational software architect: Part 7. ad-hoc modeling Ű fusing two models with diagrams. IBM developerWorks, March 2007.

[28] H. G. Min and J. S. Her. Dream: A practical product line engineering using model driven architecture. In *ICITA '05 Volume 2*, pages 70–75, Washington, DC, USA, 2005. IEEE Computer Society.

[29] B. Morin, J. Klein, O. Barais, and J.-M. Jézéquel. A generic weaver for supporting product lines. In *Proceedings of the 13th EA '08*, pages 11–18, New York, USA, 2008. ACM.

[30] B. Morin, G. Perrouin, P. Lahire, O. Barais, G. Vanwormhoudt, and J.-M. Jézéquel. Weaving variability into domain metamodels. In *MoDELS*, pages 690–705, 2009.

[31] B. Morin, G. Vanwormhoudt, P. Lahire, A. Gaignard, O. Barais, and J.-M. Jézéquel. Managing variability complexity in aspect-oriented modeling. In *Proceedings of MoDELS '08*, pages 797–812, Berlin, Heidelberg.

[32] P.-A. Muller, F. Fleurey, and J. Jean-Marc. Weaving executability into object-oriented meta-languages. In *MoDELS*, pages 264–278. Springer, 2005.

[33] L. M. Northrop. Sei's software product line tenets. *IEEE Softw.*, 19(4):32–40, 2002.

[34] OMG. Uml superstructure specification v2.1.1, February 2007.

[35] G. Perrouin. *Architecting Software Systems using Model Transformation and Architectural Frameworks*. PhD thesis, University of Luxembourg (LASSY) / University of Namur (PReCISE), September 2007.

[36] G. Perrouin, J. Klein, N. Guelfi, and J.-M. Jézéquel. Reconciling automation and flexibility in product derivation. In *SPLC '08*, pages 339–348, Washington, DC, USA. IEEE Computer Society.

[37] K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[38] R. A. Pottinger and P. A. Bernstein. Merging models based on given correspondences. In *Proceedings of VLDB '03 - Volume 29*, pages 862–873.

[39] Y. R. Reddy, S. Ghosh, R. B. France, G. Straw, J. M. Bieman, N. McEachen, E. Song, and G. Georg. Directives for composing aspect-oriented design class models. pages 75–105, 2006.

[40] M. Sabetzadeh and S. Easterbrook. View merging in the presence of incompleteness and inconsistency. *Requir. Eng.*, 11:174–193, June 2006.

[41] J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *WICSA*, pages 45–54, 28-31 August 2001.

[42] M. Volter and I. Groher. Product line implementation using aspect-oriented and model-driven software development. In *11th International Software Product Line Conference*, Kyoto, Japan, September 2007.

[43] T. Ziadi and J.-M. Jézéquel. *Software Product Lines*, chapter Product Line Engineering with the UML: Deriving Products, pages 557–586. 2006.