Feature Unweaving: Efficient Variability Extraction and Specification for Emerging Software Product Lines

Reinhard Stoiber, Martin Glinz
Department of Informatics, University of Zurich, Switzerland
Email: {stoiber, glinz}@ifi.uzh.ch

***Abstract*—Successful software products frequently evolve into software product lines, whether intentionally or not. In such cases, product managers have to be involved in creating and specifying the commonality and variability of the evolving software product line in order to continuously assure a winning business case.**

In this paper we introduce *feature unweaving*, a novel approach that allows a product manager to efficiently evolve an integrated graphical requirements model into a product line model: when he or she has identified variable elements, feature unweaving automatically extracts these elements and refactors them into a feature, using an aspect-oriented approach. Feature unweaving significantly reduces the required effort for variability specification, both on a clerical and intellectual level. Furthermore, variability constraints can be added to capture more knowledge about the features and their interdependencies. We evaluate and validate our approach with two case studies.

***Keywords*-Graphical Requirements Modeling; Variability Extraction; Variability Modeling; Aspect-oriented Modeling; Requirements Engineering; Software Product Lines**

I. Introduction

Software product management is the discipline and role which governs a software product from its inception to the market/customer delivery in order to generate biggest possible value to the business. Its major internal functions are portfolio management, product roadmapping, requirements management and release planning [18].

In product roadmapping, release planning and for customer-oriented tailoring of products, software product management has to deal with the problem of *variability*. Using a software product line approach [10] for this purpose has major advantages: development costs are reduced, products can be developed faster and the software quality will increase because of reuse. However, proactively building a full-fledged software product line is costly and risky. As a consequence, practitioners often employ ad-hoc variability modeling solutions in the form of spreadsheets, AND/OR tables or similar. Such solutions, however, turn out to be insufficient: the variability and, in particular, the variability constraints can't be expressed properly. Moreover, this approach is error-prone because traceability must be done manually. Other strategies for the transition towards a software product line are (i) creating a small product line initially and then evolving it into one that covers the full product scope, and (ii) extracting a product line from already existing product specifications [7].

In this paper we present a new approach, called *feature unweaving*, that supports both of the aforementioned strategies: it provides semi-automatic, incremental extraction of variability from an existing product requirements model. While a software product manager still needs to identify the elements of the model that vary, the extraction and refactoring of these elements into features is fully automated. Such an automated extraction reduces the effort and cost for creating variable features and, equally important, it guarantees that the original model and the new, refactored model are semantically equivalent, thus saving the effort for manual consistency checking.

We have developed the algorithmic concepts of feature unweaving [16] [4] as well as an implementation in the framework of the ADORA [3] language and tool.

The core elements of our approach are the use of integrated requirements modeling and aspect-oriented modeling. An inherently integrated requirements model, such as an ADORA [3] model, avoids information scattering of the specification over multiple diagrams, which happens when using UML, for example [10]. Integrated modeling is a key enabler for our approach, since it allows giving a full selection of elements that constitute a variable feature in a single view. For modeling variability, we use aspect-oriented modeling [8] with slight generalizations [4]. In our approach, variable features are modularized with aspects, while the commonality of a product line remains a plain-vanilla ADORA model [12] [16].

We validate feature unweaving with two case studies. The first one evaluates the efficiency of feature unweaving compared to an AND/OR table-based approach using a real-world governmental decision system exemplar. As a result we found a significant increase in efficiency for all variability-related tasks. The second case study presents a comparative evaluation of our approach and one that combines UML with feature modeling. The requirements of an existing case study from the mobile phone domain [9] have been modeled with both concepts, balanced and compared. The results indicate that specifying requirements and variability in ADORA is more efficient than using UML in combination with feature modeling: the latter required about 60% more graphic modeling elements to specify the given

Proceedings of IWSPM'10: 4th International Workshop on Software Product Management. At RE'10, Sydney, Australia. Sept 27, 2010.

set of requirements.

The remainder of the paper is organized as follows. Section II motivates our approach from a practitioner's viewpoint. Section III describes the details of feature unweaving. Sections IV and V present the case studies. Section VI discusses related work and Section VII concludes.

II. Background, Problem and Motivation

The practitioners that we collaborated with individualized their software products by specifying all requirements in one comprehensive requirements specification and augmenting that specification with tables that described admissible combinations of software features. Such all-inclusive specifications, called reference model henceforth, led to challenges for stakeholders. User representatives with differing, but overlapping interests needed to review the requirements for understanding which requirements were relevant for them. The project board (consisting of application owner, department heads and portfolio manager) needed to make scope decisions with a seemingly never-ending list of requirements whose interrelationships were hard to extract from the specification. The involved requirements engineers had to invest an immense effort for maintaining the consistency and structure of the requirements specification.

Some software or software-intensive products turned out to be very successful and attracted additional customers and markets. Variability inevitably was introduced to focus and tailor the products to their emerging new target environments. In fact, they became software product lines. Other products, however, turned out to be considerably less successful than expected and disappeared again in the short or medium term. Though variability may have been anticipated, it did not step in. Therefore, creating all potentially promising software products as software product lines from the very beginning is rather risky for product managers. As there are high initial costs connected with creating a software product line, the practitioners decided to rely on ad-hoc solutions with spreadsheets, AND/OR tables and similar, e.g. as in [9].

The advantages of using *ad-hoc spreadsheets* to manage the variability of an evolving product line lie in the creation, understanding and maintenance of the variability with only little costs, training and effort. Simple and widely available COTS tooling suffices.

AND/OR tables define a matrix of all variable product features (typically the rows) and all existing or planned products (typically the columns). While this gives a succinct overview of the product portfolio, this approach has two significant shortcomings: Firstly, AND/OR tables don't specify any constraints between variable features explicitly. As a consequence, regular patterns in the table might be due to a constraint, but they might also occur just accidentally. If, for example, every product contains exactly one element from a set of features, it is not clear whether or not this means that these features are mutually exclusive. Secondly, for every feature in the AND/OR table, traceability links to the correponding requirements are necessary. Only when these exist, an efficient reuse of the specification is possible. The practitioners we worked with did not build complete sets of traceability links between product features and requirements for evolving product lines, due to the high effort required for that. Instead, domain experts knew much of this traceability by heart and shared their knowledge in meetings and reviews. The lack of explicit specification may be economic for products with an early phase-out. For sustained software product management, however, this approach is too risky.

Using *feature modeling* [5] or a similar approach (e.g., OVM [10] or decision modeling, e.g. [11]) instead of AND/OR tables solves the first of the two problems: the variability model can be specified intensionally, including the variability constraints, while the product portfolio can still be represented as a list of configurations. However, the second problem remains: traceability links or mappings need to be defined for every feature onto the requirements; see [2] for a typical example. However, creating these mappings is costly, labor-intensive and error-prone.

Existing solutions that automate the extraction and specification of variability all require a given set of product specifications out of which the requirements variability and the feature model are extracted [17] [19]. Currently, to the best of our knowledge, no solution exists that provides considerable automation support in creating a variability specification incrementally from a single reference or product model.

The approach we present in this paper allows a semi-automatic extraction of variability from an existing product requirements model. Using *feature unweaving*, a product manager does not need to create any dedicated variability model or mappings at all. Instead, he or she can directly identify variable requirements in the requirements model and extract them with the feature unweaving function. Feature unweaving then refactors the requirements model and specifies the selected elements with aspect-oriented modeling. This eases the creation of a software product line considerably. Additionally, feature unweaving also guarantees that, after every extraction of a variable feature, the resulting model is semantically equivalent to the original model. Therefore the semi-automatically created variability model will be correct by construction, which eliminates the cost for verifying this equivalence manually.

III. The Feature Unweaving Function

A. Types of Variability

The way a variable feature is unwoven from a requirements model depends on the degree of scattering and tangling of this feature in the original model. Scattering means that the model elements belonging to a feature are scattered over many components of the model. Tangling means that

elements of various features are contained within a single component such that this component lacks cohesiveness.

Feature unweaving rectifies the scattering and tangling of variable features by fully extracting these elements and modeling them with aspects. We have identified three basic types of variability that occur in a requirements model.

1) Local Variability: Local variability means that a variable feature affects only elements within one container (i.e. a node that modularizes a certain concern). A container in ADORA is an object, object set or aspect. Thus, local variability is not scattered over more than one container, but tangling may occur within a container. Local variability can always be modularized with a single aspect.

In our real-world requirements examples, see [14], [4] and [20], local variability was a type of variability that we found frequently.

2) Homogeneously Cross-Cutting Variability: Homogeneously cross-cutting variability means that a variable feature affects requirements at two or more locations in the *same* way. These locations can be somewhere in the behavioral or user interaction specification in the same container, but at different locations, or in different containers. Such a homogeneously cross-cutting variability is scattered over one or more concerns and affects several locations in the specification. Therefore, also one or more other concerns are tangled with such a variable feature. Homogeneously cross-cutting concerns are typically modularized with aspects, as for example logging or authentication [8]. These concerns can also be variable in a software product line.

To modularize homogeneously cross-cutting variability with a single aspect container, the feature unweaving function needs to recognize that all selected elements at the different locations are equal and, thus, merge them. Our current implementation of feature unweaving does not yet automatically perform these tasks, but, instead, unweave such a selection as a heterogeneously cross-cutting variability (see below). Merging then could be performed manually or automated by the tool as well.

In our real-world requirements exemplars we did not find any homogeneously cross-cutting variability.

3) Heterogeneously Cross-Cutting Variability: Heterogeneously cross-cutting variability means that a variable feature affects the requirements in two or more different containers in a *different* way. Thus, such a variable feature is scattered over multiple containers in the model and these containers are tangled with the variable feature.

To modularize such variability appropriately, multiple nested aspect containers have to be created. A base aspect container needs to be created that contains a nested aspect for every other container that is cross-cut by this variability. The selected model elements of every container need to be extracted into one of these aspects. Their respective weaving semantics can then be created analogously to a local variability, see above.

When different model elements in different parent containers in the model vary because of the same variability decision, it is beneficial to unweave them together as a single variable feature. This way only a single variability binding decision is required for selecting or dismissing these requirements and no additional constraint is necessary. This reduces the complexity of the resulting model and therefore elevates the efficiency.

In our real-world examples we found such heterogeneously cross-cutting variability quite frequently. The bigger and more detailed our models were, the more often variable features were heterogeneously cross-cutting. We could simplify the structure of the variability by unweaving them all together into one hierarchically structured base aspect. Our feature unweaving algorithm and implementation fully support an automated refactoring of a selection of heterogeneously cross-cutting variability.

B. The Feature Unweaving Algorithm

1) The ADORA *Language:* ADORA is an integrated requirements and architecture modeling language that is based on abstract objects instead of classes and uses a decomposition hierarchy as its main means for modularizing models. In contrast to UML, which builds on the idea of a set of loosely coupled diagrams, ADORA allows integrating all views of a model into one consistent and coherent structure [3].

As a hierarchy of abstract objects cannot modularize all relevant kinds of concerns, the original version of ADORA described in [3] has been extended by capabilities for aspect-oriented modeling [8]. Aspects are a means for modularizing crosscutting concerns, i.e. information that is scattered and tangled in the model. In product line requirements modeling, ADORA uses aspects for modeling variable features [12] [14]. For an overview of variability modeling in ADORA, including an explanation of the notation see [14]. Feature unweaving builds upon the aspect-oriented extension of ADORA defined in [8] with slight generalizations (see [4]) which turned out to be necessary for supporting aspect modeling also for very abstractly specified requirements.

2) An Overview of the Feature Unweaving Algorithm: In order to apply feature unweaving in the ADORA tool, the user (e.g., a product manager or a requirements engineer) selects the variable elements that s/he wants to unweave and points at the location in the diagram where the extracted aspect(s) shall be positioned. Then ADORA automatically performs the refactoring, using the algorithm explained below. Figure 1 defines the basic terms used. Figures 2 and 3 describe the structure of the algorithm. The details are omitted. Instead, we demonstrate the algorithm in action for a typical example in Figure 4 .

The unweaving algorithm consists of five main steps (Figure 2). As a first step, the algorithm creates a base aspect container (AC). This base AC and the base ADORA model are then handed over to the recursive extraction function to

PC ... *parent container; a model element that directly contains one or more selected element(s); it can be an object, an object set, a state (statecharts can be hierarchical), an aspect container, or the* ADORA *model itself*
BasePC ... *the lowest-level PC in the hierarchy that contains all the selected elements to unweave*
AC ... *aspect container; a container element where variable elements for an unweaving are extracted into; it must include at least one outgoing join relationship (JR); it can hierarchically contain other ACs (nested)*
BaseAC ... *the lowest-level AC created that contains all elements of one variable feature; for cross-cutting features it typically has other nested ACs*

Figure 1. Definitions

1. *Create the BaseAC*
2. *Call the recursive extraction function (parameters: the* ADORA *model, the BaseAC)*
3. *Remove all redundant ACs in the BaseAC*
4. *Fine-tune the graphic layout*
5. *Set the BaseAC as a variability (this adds a unique decision item and annotates the outgoing join relationships)*

Figure 2. The feature unweaving algorithm

perform the extraction. This recursion works as described in Figure 3. The base parent container (BasePC) needs to be found first. This is the lowest-level container (typically an object, but can also be an aspect) in the model that contains all selected elements (see the definitions in Figure 1). Having the BasePC found, all selected elements within this container are extracted into the BaseAC. Next, the necessary join relationships are computed and created. These are defined in such a way that a weaving of all the extracted requirements always results in a semantically equivalent model to the original one. As a last step of this recursion, if further nested elements exist, a new AC is created inside the BaseAC and the next recursive call is performed, using the BasePC and the newly created AC as parameters. Thus, the recursion incrementally descends into the next lower part of the model and extracts the selected elements. Eventually, when all selected elements have been extracted, no further BasePC is found and the recursion terminates.

Input-parameters: a PC, an AC
2.1 *Find the BasePC*
2.2 *Extract all directly nested selected elements of the BasePC into this AC*
2.3 *Calculate and set the join relationship(s)*
2.4 *For every nested PC that contains any selected element*
  2.4.1 *Create a new AC inside this AC*
  2.4.2 *Call the recursive extraction function (parameters: this BasePC, the new AC)*

Figure 3. The recursive extraction function in the unweaving algorithm

The recursion may create redundant ACs (i.e., ACs that only contain other ACs). These are removed in step 3 of the algorithm. At this point, the modeling of the extracted elements is completed and the layout will be fine-tuned in step 4. This concerns mostly reducing white space and aligning the extracted elements coherently. Finally, in step 5, the extracted aspects will be specified as a variability for the evolving product line model. Setting the attribute 'variability' to 'true' in the BaseAC will recursively also set this property for all child ACs and produce a unique decision item that gets annotated onto all join relationships created for one variable feature extraction. This decision item is included in the decision table henceforth, as a means for variability management [14] and product derivation [15].

3) Feature Unweaving Illustrated: The algorithm is illustrated using an example and a detailed execution trace of *feature unweaving* applied to it (Figure 4). This example also includes the semantic details that were omitted in the general description of the algorithm. Diagram *a* of Figure 4 shows a non-trival reference model in ADORA and a selection of elements that a product manager considers constituting a variable feature. The mouse pointer indicates the location where the extracted feature shall be positioned. The selection represents a heterogenously cross-cutting variable feature. It involves a component that includes some behavior description and two steps of a hierarchic scenario chart orthogonally nested in the hierarchic base structure. Starting from this diagram, the subsequent ones below show various intermediate steps of the extraction process. The last diagram shows the final refactored model. The white and grey coloring is used to better visualize containment hierarchies.

Diagram *b* of Figure 4 shows the situation after the first execution of the recursive extraction function (i.e. after step 2.4.1 of the algorithm, see Figure 3). The BasePC in this execution is the *System* object, which does not directly contain any selected objects. Thus, nothing is extracted and the BaseAC is left empty in step 2.2. A new nested aspect container has been created in step 2.4.1. The second execution of the recursive extraction function processes *Component B* as PC. This object contains a couple of selected elements to extract. First, the sub-scenario *sc2* is extracted. This requires removing the connection of *sc2* to its parent scenario and relocating *sc2* into the new AC. Next, the object *Component B.1* is relocated. As *Component B.1* has no connections, this is straightforward. The intermediate result (after step 2.2) is shown in Diagram *c*. To complete this extraction, an appropriate join relationship needs to be computed and created in step 2.3. In this case, a simple join relationship with the weaving type *'before'* between the extracted scenario and scenario *sc3* suffices. To keep the core model consistent, the sequence number of scenario *sc3* is changed from 3 to 2. The result after step 2.3 is

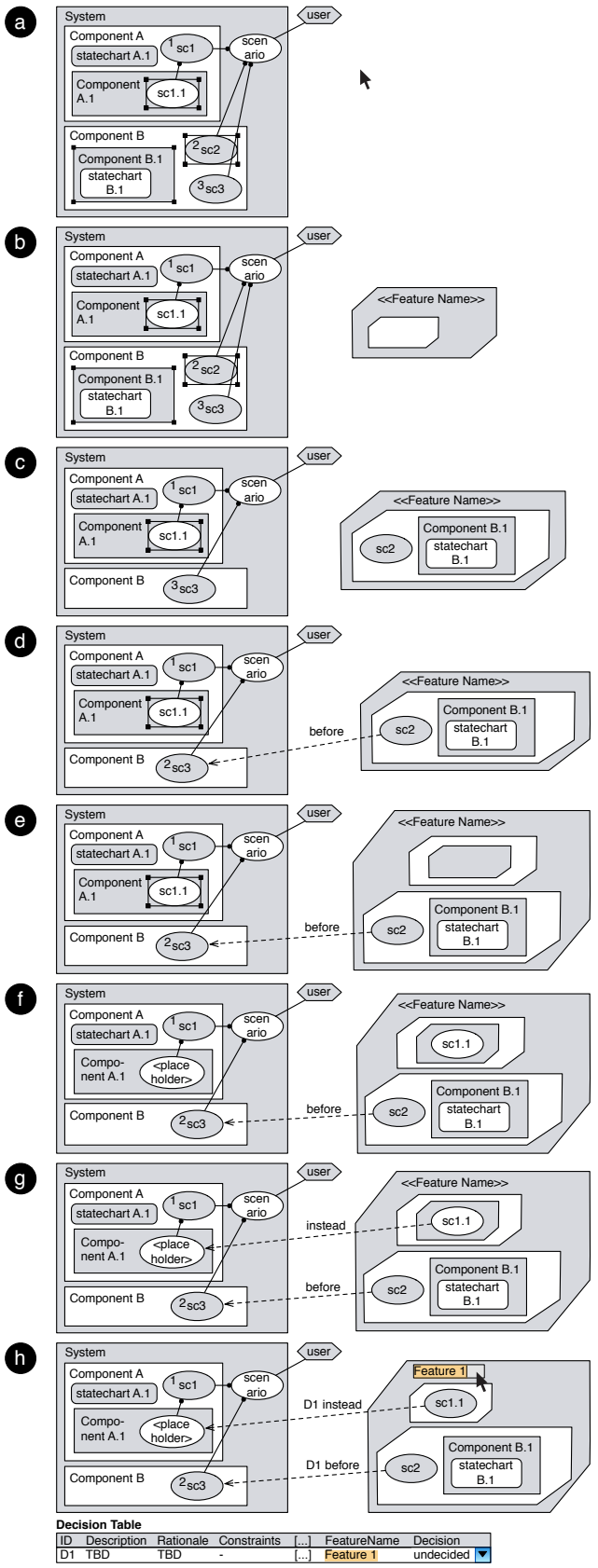

Figure 4. Feature unweaving illustrated by example. a) shows a reference model. The selected elements are those that shall be extracted. b)-g) show intermediate steps, and h) shows the resulting refactored model.

shown in Diagram *d*. This model and the reference model given in Diagram *a* are semantically equivalent: the join relationship specifies that if scenario *sc2* would be woven, it would become a sub-scenario of *scenario* and would be inserted just before the sub-scenario *sc3* [8].

In diagram *e*, another AC within the BaseAC was created when component *A* was traversed, but left empty because there were no selected elements contained directly. Prior to the current recursive call, another nested AC was created into which the nested selected elements of the object *Component A.1* will be extracted. Diagram *f* shows the results of this extraction. In this case, the scenario to extract is the only sibling node of its parent scenario in this container. Therefore, it cannot be relocated directly because the join point would get lost. Consequently, a *<placeholder>* scenario is created and an *'instead'* weaving semantics selected to be applied here [4]. Diagram *g* shows the result. After this execution, no BasePC is found anymore and thus the recursion terminates.

Diagram *h* finally shows the resulting refactored model, including all remaining steps of the feature unweaving algorithm. In particular, the corresponding decision item and an entry in the decision table [12] have been created. This model is syntactically correct and semantically equivalent to the one in Diagram *a* by construction. The originally selected variable elements are now explicitly modularized as a variable feature with aspect-oriented modeling.

IV. Case Study 1: Efficiency Evaluation

Feature unweaving was validated by comparing it with the widely established AND/OR table-based approach in the typical contexts where product line models are used: product variant definition, release planning, and product line evolution. The following questions were asked: *is the approach efficient for defining a variability model and for defining product variants? Is the approach efficient for supporting feature evaluation and selection? Is the approach efficient for maintaining consistency?* Answers to these questions support decision making for whether and when to adopt feature unweaving and for steering its further development.

A. Case Description

A decision support system for a government was studied as a case for answering these questions. In this case three departments desired to use the same system to decide upon customer requests. The departmental business processes resembled each other. Another department was responsible for accounting of the financial transactions managed by the system. The reference model covered all possible authorizations, business transactions, and entities to be managed. AND/OR tables described the desired provision of information and functionality of the system to the departments.

B. Research Design

Table I summarizes the followed process. Five macro steps were executed for *Product Variant Definition*: an initial

Table I
VALIDATION OPERATIONS.

Steps	Results
Product Variant Definition	
1. Specify requirements model for product 1.	**Prod1**: requirements model of product 1.
2. Extend Prod1 with differences towards products 2.	**Var12**: product line model for requirements of products 1 and 2.
3. Derive requirements models for products 1 and 2 from Var12.	**Prod1**: requirements model of product 1. **Prod2**: requirements model of product 2.
4. Extend Var12 with differences towards product 3.	**Var123**: product line model for requirements of products 1, 2, and 3.
5. Derive requirements models for products 1, 2, and 3 from Var123.	**Prod1**: requirements model of product 1. **Prod2**: requirements model of product 2. **Prod3**: requirements model of product 3.
Feature Evaluation and Selection	
1. Evaluate effort for selecting a subset of requirements for implementation.	Total number of possible products Number of elements to compare
Feature Evaluation and Selection	
1. Change a requirement in product 1.	Change to Prod1.
2. Ensure that the altered requirement is changed in all affected products.	Changes to Prod2 and Prod3 if necessary.

product requirements model was created (*ProdSpec1*), the variability was specified/extracted when the second product got introduced (*VarSpec12*), the two product specifications were derived again (*ProdSpec12*), further variability was specified/extracted when a third product was introduced (*VarSpec123*) and the product specifications created again for all three products (*ProdSpec123*). Further, for *Feature Evaluation and Selection* the required effort for reasoning about new product configurations was studied and for *Consistency Maintenance* the required effort for introducing a change in an existing product configuration was evaluated. AND/OR tables were directly defined on the requirements and modified ad-hoc. Requirements were not grouped into features in the AND/OR tables in this real-world exemplar. Feature unweaving grouped requirements into features and was performed with a prototypical implementation of the ADORA tool[1].

Simple quantitative measurements were taken to answer the validation questions. The *numbers of required edit manipulations* were measured to evaluate the efficiency for product variant definition and for maintaining consistency. The *numbers of possible products and features to compare* were used to measure the efficiency for feature evaluation and selection. Smaller numbers were regarded to indicate a higher efficiency. Sub-section D presents these results.

C. Limitations and Threats to Validity

The study validates feature unweaving in a laboratory situation. The use of real-world exemplars increases the confidence that the results apply in general. Implementation of feature unweaving in the ADORA tool ensures that the constructs of interest are evaluated in the way they have been presented. The sequence of activities used to answer the research questions is described and discussed with lessons learned.

[1]The ADORA tool is available under an open source license from http://www.ifi.uzh.ch/rerg/research/projects/adora/

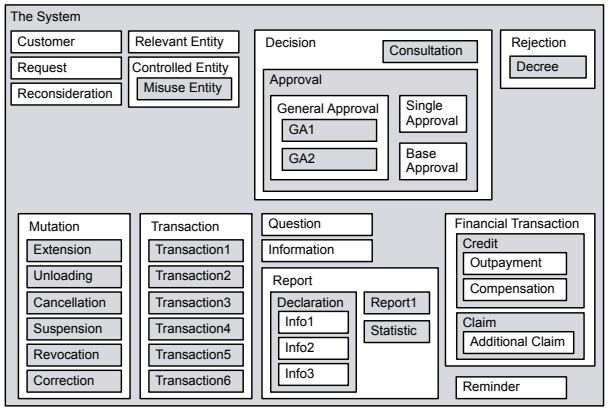

Figure 5. Requirements model *Prod1* of product 1.

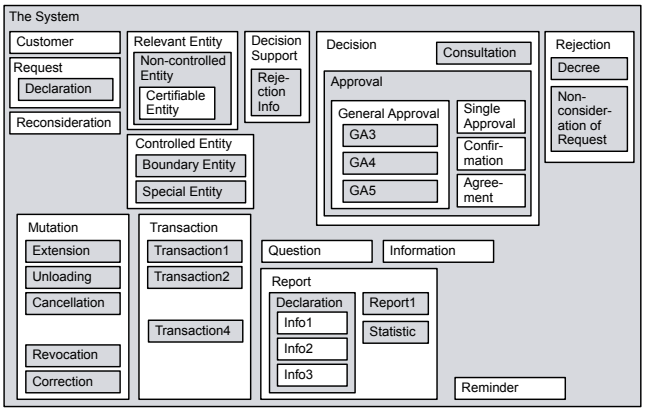

Figure 6. Requirements model *Prod2* of product 2.

Feature unweaving was conceptualized and implemented with ADORA. For a transfer of the results to other modeling languages, the impact of ADORA-specific language features needs to be investigated. Efficiency measurements were based on numbers of edit manipulations and not on measurements of time. This is due to the fact that the implementation of ADORA is still on a prototype level and that AND/OR tables were used ad-hoc, without tool support.

This study measures only the effort for specifying the variability and not the required effort for identifying variable features in the first place. By minimizing the required effort for extracting and for removing the specification of variability by refactoring this may also improve efficiency in variability identification. Such correlation, however, was not measured and is subject to future work.

While in this case study the AND/OR table did not aggregate requirements towards features, such grouping is frequently done in practice, e.g., see [9]. Using an AND/OR table with feature grouping would yield different and probably better results, but also require additional traceability. Section V presents a complementary second case study where such feature grouping was applied.

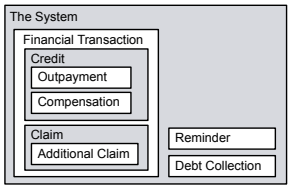

Figure 7. Requirements model *Prod3* of product 3.

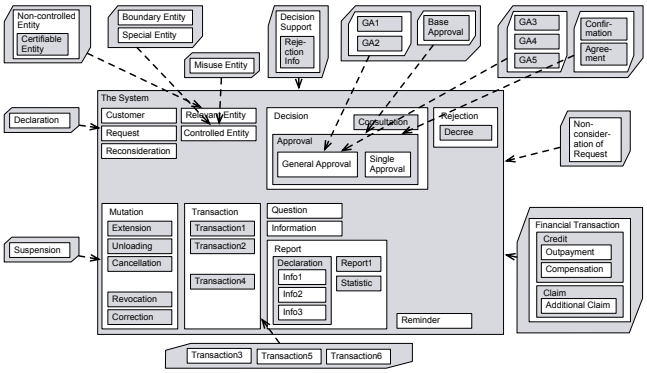

Figure 8. Product line model *Var12* covering products 1 and 2.

D. Operations and Results

1) Product Variant Definition: Figures 5-7 show the product requirements models *Prod1*, *Prod2*, and *Prod3* and Figures 8-9 the product line models *Var12* and *Var123* that have been developed according to the process described in Table I. Table II summarizes the results of all validation operations. It presents the measured effort for specifying the product and product line models by using feature unweaving (left hand side) and AND/OR tables (right hand side).

2) Feature Evaluation and Selection: The relative effort of release planning was estimated by comparing the number of elements to consider for variability resolution and the number of possible products, based on the latest product line model *Var123*. In ADORA *13* variable features need to be considered for release planning. Among these features hierarchical constraints (sub-features can only be chosen if their parent-feature is selected; otherwise the configuration is invalid) already exist, see Figure 9. Taking these into account, an overall of *4'100* products are allowed in ADORA. This number is calculated automatically [15]. In the AND/OR table approach, *62* requirements need to be considered and no constraints could be defined. Thus, all combinations of requirements are possible, leading to 2^{62} ($4.61{*}10^{18}$) possible products.

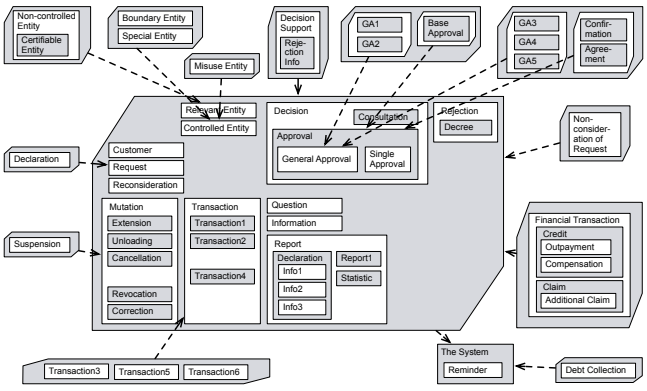

Figure 9. Product line model *Var123* covering products 1, 2, and 3.

Table II
COMPARISON OF SPECIFICATION EFFORT.

Work Result	Feature Extraction	AND/OR Table
Model requirements of Prod1		
Prod1	Specify Prod1: add 47 reqs.	Specify Prod1: add 47 reqs.
Add variability		
Var12	Extract Prod1: add 5 aspects (15 reqs.) Add Prod2: add 6 aspects (14 reqs.)	Add Prod2: add 14 reqs. Expand table: add 72 cells
Derive products from Var12		
Prod1 Prod2	Weave 5, remove 6 aspects Weave 6, remove 5 aspects	Remove 14 requirements Remove 15 requirements
Add variability		
Var123	Extract Prod1+2: add 1 aspect (31 reqs.) Add Prod3: add 1 aspect (1 req.)	Add Prod3: add 1 req. Expand table: add 75 cells
Derive products from Var123		
Prod1 Prod2 Prod3	Weave 6, remove 7 aspects Weave 7, remove 6 aspects Weave 2, remove 1 aspect	Remove 15 requirements Remove 16 requirements Remove 53 requirements
Total	75 specification operations 29 product derivation operations	209 spec. operations 84 prod. deriv. operations

3) Consistency Maintenance: The relative effort for maintaining consistency was estimated by the number of operations necessary to introduce a requirements change and the number of operations needed to re-derive all product models from the updated product line or reference model, based on the latest product line model. For both approaches, only one operation is necessary to introduce the requirements change. In the feature unweaving approach, *29* weaving operations are necessary when all product specifications are re-derived manually (see Table II). If the previous product configurations were saved in ADORA, such consistency maintenance is even reduced to only two operations (one for the actual change plus one for re-weaving all previously saved product configurations). In the AND/OR table approach, the change is performed in the reference model and *84* operations are necessary to re-derive all previously created product models (see Table II). Alternatively, the change could be applied to all products and the reference model directly, which would require only five operations. However, this is risky: consistency requires that the change is introduced equally in *all* affected models, otherwise the reference and product models become inconsistent.

E. Lessons Learned

1) Strengths: Feature unweaving has successfully been implemented and supported the product line engineering tasks efficiently. Feature unweaving required 2.8 times less manipulations for specifying variability, 2.9 times less manipulations for deriving product variants, and 2.8 time less

manipulations for propagating requirement changes than the AND/OR-table approach in the presented case. The number of elements to be considered for release planning was reduced by a factor of 4.8.

Knowledge related to product line architecture and application domain was encoded. Feature unweaving allowed grouping requirements that needed to be implemented together into aspects and defining dependencies. The resulting aspect tree *Var123* can be compared with feature trees [5] and provides aggregated information regarding allowed combinations of requirements. This information is left implicit in the mind of the modeler when using the AND/OR-table approach.

The requirements modeler developed a mental map of the product line requirements in a form that can be conveyed to the reader of the model. The geometric layout of requirements and aspects in *Var123* provided cues regarding product line properties that could not be codified with an AND/OR-table approach.

Feature unweaving simplified planning for new products and increased the quality of such planning. The product line model *Var12* acted as a template and checklist that allowed incremental definition of variability to accommodate for the subsequently added product *Prod3*. The product could be added by locally changing commonality and aspects, hence reusing parts of the specification, instead of adding an entire new definition of the product in terms of a column in the AND/OR-table.

2) Limitations: Feature unweaving and weaving changed the model layout. These operations required rearranging model elements by placing them into aspects or abstract objects respectively. The negative effects of such rearrangements were reduced by compacting nodes and saving a layout after manual arrangement. The modeler's mental map related to the concerned model elements was still disturbed, however.

AND/OR tables are a generic concept that can be used with any type of requirements description and modeling. Feature unweaving in contrast is built specifically for ADORA and designed for extracting variability in integrated requirements models. Therefore, if a requirements specification already exists in a textual or UML format, then an ADORA specification needs to be created first, before feature unweaving can be used. Depending on the amount of effort spent on transitioning towards an ADORA requirements specification, the overall improved efficiency in requirements variability modeling will be lower or even negative, accordingly.

3) Other issues: The approach introduced a considerable amount of aspect containers and join relationships. One could argue that using that many aspects may be problematic. Large numbers of aspects, however, did not cause any problems or obstacles. Feature unweaving and removing of aspect-oriented variability modeling are performed fully automatic by the tool as refactorings, see also [16], for example. Engineers don't need to deal with weaving semantics and can use visualization to hide irrelevant aspect details [13]. Section V confirms these results and shows that the use of aspects can even reduce the amount of model elements required for specifying the variability.

Table III
STATISTICS ON THE REQUIREMENTS AND VARIABILITY MODELS OF THE GOPHONE CASE STUDY, DERIVED FROM [20].

	1. Requirements		2. Requirements and Variability		
	ADORA	**EA**	**ADORA**	**EA**	**pure::variants**
# diagrams	1	17	1	17	1
# elements	174	254	261	331	68
# connections	~ 200	338	318	482	38
# cross-tree constraints	-	-	14	-	12
Sub-total	-	-	-	*830*	*119*
Total	**375**	**609**	**594**	**949**	

V. CASE STUDY 2: A COMPARATIVE EVALUATION

Zoller [20] has performed a comparative evaluation of three state-of-the-art requirements and product line variability modeling concepts: (1) Sparx-Systems Enterprise Architect (EA) in combination with pure-systems pure::variants, (2) IBM Rational Focal Point, and (3) ADORA. He modeled the GoPhone software product line, a hypothetical case from the mobile phone domain [9], for all three concepts. He carefully assured that the three specifications he created were equivalent in content. Then he performed a quantitative and qualitative comparison of the three concepts and derived strengths and weaknesses for each of them. This section presents Zoller's results on ADORA (integrated requirements modeling using aspects to modularize variable features) and EA & pure::variants (requirements modeling with UML and mappings to a feature diagram).

Table III presents some quantitative figures for the created requirements and variability models. The figures in the left part of the table characterize the specification before the variability was specified. The right part characterizes the situation with the variability modeling included. In ADORA, feature unweaving was used to create the variability specification. For EA, a feature model was created with pure::variants and all variable UML model elements in EA were annotated with pure::variants feature restrictions.

For specifying the software requirements given in the GoPhone case study report [9], the UML-based requirements specification required an overall of 61.5% more graphical elements (sum of all nodes and edges of all graphic diagrams) than required in the ADORA specification (609 vs. 375). The main reason for this significant difference is the fact that ADORA does not need explicit traceability links between different diagrams, because this traceability is implicit in the hierarchical nesting of elements. Another reason is that the user interaction modeling in EA required significantly more

graphic modeling elements than the integrated scenario chart modeling in ADORA.

When including also the variability modeling (see the right part of Table III), we get equally strong results. The specification of an additional feature diagram and of all necessary annotations in the requirements specification required overall 59,8% more model elements than the variability specification with aspect-oriented modeling in ADORA (949 vs. 594). For EA a new dedicated feature model had to be created with pure::variants. 68 features (38 of them mandatory and 30 variable or group features), 38 hierarchy connections between features and 12 cross-tree constraints were specified. *Or*-constraints between features were modeled with group features and respective types of sub-features and therefore did not require additional cross-tree constraints. In EA, feature restriction and links to all the affected model elements had to be created in every diagram where variability occurred. This explains the significant increase in model elements and connections. In ADORA, a new aspect was created for every variable feature, including nested sub-aspects when required. As an important result of this study, we have actually found that 15 out of 25 variable features (66.7%) were heterogeneously crosscutting, i.e. required nested sub-aspects. However, the strong increase of model elements was a result of the many pseudo-elements needed because of limitations/requirements of ADORA's aspect weaving semantics [4] [8]. This was especially true for statecharts: 71 additional new pseudo-states and end states were introduced (an increase of the number of states by 88.7%) while only 4 pseudo-scenarios had to be added (an increase of only 6.9%; an example of a pseudo-scenario node can be seen in Figure 4 in the diagrams *e-f*, labeled as *<placeholder>*). The additional states also introduced additional state transitions (connections), consequently. The other added elements and connections were the required aspect containers and join relationships. Further, 14 explicit constraints were specified on the resulting variability model, including the cardinality-based constraints.

Taking these results and complementing them with other quantitative and qualitative results, Zoller has concluded that the integrated modeling, the comprehensive model overview and the reduced modeling effort are among the benefits of ADORA. The need for that many pseudo-elements was credited as a negative point and also that when deriving products, aspects are woven and the "aspect-oriented" modeling of cross-cutting concerns gets lost. For EA the use of modeling standards (UML, feature model) and the reduced need of redundancy (recurring charts were put into separate diagrams and simply referenced) was credited as a plus, while the need of that many diagrams and the big effort needed for creating and maintaining all the feature annotations were assessed as disadvantages. In summary, every of the two concepts still has advantages. The statistics as presented in Table III, however, provides a clear indication towards a more efficient specification of software product lines in ADORA.

VI. Related Work

Existing approaches to product line modeling build on a dedicated variability model, typically a feature model [5], an orthogonal variability model (OVM) [10], or decision models, e.g., [11]. Variability is then mapped onto the corresponding engineering artifacts, for example onto UML models [2]. All these approaches basically possess the same advantages and weaknesses as the 'UML-with-feature-modeling' concept we evaluated in Section V. Feature unweaving does not require an additional dedicated variability model, but models variability directly in the requirements specification using aspects.

Recently, also other aspect-oriented approaches for software product line requirements engineering have been introduced [1] [21]. These, however, still require a dedicated additional variability model and manually defined constraints and traceability links to keep the product line model consistent. Bonifácio and Borba [1] require engineers to write the codified configuration knowledge (i.e., mappings between feature expressions and their related engineering artifacts) manually. In the approach of Zschaler et al. [21], all variability information (variants and their orderings, actions, pointcuts, etc.) has to be defined by hand. In contrast, feature unweaving only requires manual identification of variable model elements. The extraction and creation of the necessary aspect structure in ADORA is fully automatic. Variability constraints can be specified later based on this refactoring and projections can be generated that show, for example, only the pure variability model [13], yielding a view similar to a feature diagram. As Sections IV and V have shown, such an alternative paradigm of variability modeling reduces the effort for variability specification and decreases the size of the overall product line specification.

Aspect model unweaving as presented by Klein et al. [6] also deals with extracting aspects from models. However, this work deals with removing elements of aspects that had been woven previously. Feature unweaving in contrast does not revert any previous weaving operation, but extracts a set of selected model elements and *creates* all the required aspects and weaving semantics in the first place.

There is also other work on automation support for creating variability models in requirements engineering. Weston, Chitchyan, and Rashid [19] use an early aspects mining tool to find candidate crosscutting concerns, according to their semantic similarity, over textual requirements documents of different products. Therewith their approach allows to automatically construct feature models from diverse and heterogeneous requirements documents. Wang et al. [17] introduce a formal model of use cases based on which application feature models are constructed after some preprocessing. These models are then adjusted and finally merged into a domain feature model semi-automatically, after all

conflicts have been removed. Both approaches assume that specifications of concrete products already exist beforehand. The feature models are constructed (semi-)automatically in a reverse engineering way. With feature unweaving, variability extraction is already supported when a second or further product requirements specification is created. The variability is extracted and specified as soon as it surfaces or becomes relevant. A later reverse engineering of the variability model hence becomes obsolete. This enables an automation-supported variability modeling already in an early stage of the software product line life cycle. In fact, as evaluated in Section IV, the efficiency for creating a concrete product specification is already increased when a second software product gets introduced and a product line is created for these two products.

VII. Conclusion

The paper has presented a new approach, called feature unweaving, that enables automated extraction and explicit modularization of a set of selected variable elements in a requirements model based on aspect-oriented modeling. The technical concepts were explained and an algorithmic implementation was introduced. Two real-world case studies were performed. The first case study has shown significant improvements in efficiency of product line specification, product variant definition, release planning, and consistency maintenance. The second one has provided evidence of a significant reduction of the complexity and necessary size of the graphical product line requirements specification.

Future conceptual research will address the graphical layout of model fragments when aspects are extracted or woven and a full formalization of the feature unweaving semantics. Future empirical work will focus on validating the approach in real organizational contexts where product line architects and domain experts collaborate with stakeholders.

Acknowledgment

The authors would like to thank Dr. Samuel Fricker for his enabling and active support that has made the case studies presented in this paper possible.

References

[1] R. Bonifácio and P. Borba, "Modeling scenario variability as crosscutting mechanisms," in *Proceedings of AOSD '09*. ACM, 2009.

[2] K. Czarnecki and M. Antkiewicz, "Mapping features to models: A template approach based on superimposed variants," in *Proc. of GPCE'05*. Springer, 2005.

[3] M. Glinz, S. Berner, and S. Joos, "Object-oriented modeling with ADORA," *Inf. Syst.*, vol. 27, no. 6, pp. 425–444, Elsevier, 2002.

[4] M. Jehle, "Feature unweaving: Semi-automated aspect extraction in product line requirements engineering," Master's thesis, University of Zurich, 2010.

[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Tech. Rep., CMU/SEI-90-TR-021, November 1990.

[6] J. Klein, J. Kienzle, B. Morin, and J.-M. Jézéquel, "Aspect model unweaving," *Proc. of MoDELS'09*. Springer, 2009.

[7] C. W. Krueger, "Easing the transition to software mass customization," in *Proc. of PFE '01*. Springer, 2002.

[8] S. Meier, *Aspect-Oriented Requirements Modeling*. Ph.D. dissertation. University of Zurich, 2009.

[9] D. Muthig, I. John, M. Anastasopoulos, T. Forster, J. Dörr, and K. Schmid, "GoPhone - a software product line in the mobile phone domain," Fraunhofer IESE, Tech. Rep. No. 025.04/E, March 2004.

[10] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[11] K. Schmid and I. John, "A customizable approach to full lifecycle variability management," *Sci. Comput. Program.*, vol. 53, no. 3, pp. 259–284, Elsevier, 2004.

[12] R. Stoiber, S. Meier, and M. Glinz, "Visualizing product line domain variability by aspect-oriented modeling," *Proc. of REV'07 Workshop* at RE'07. IEEE CS, 2007.

[13] R. Stoiber, T. Reinhard, M. Glinz. "Visualization support for software product line modeling," *Proc. of ViSPLE'08 Workshop* at SPLC'08. Lero, Univ. of Limerick, 2008.

[14] R. Stoiber and M. Glinz, "Modeling and managing tacit product line requirements knowledge." *Proc. of MaRK'09 Workshop* at RE'09. IEEE CS, 2009.

[15] R. Stoiber and M. Glinz, "Supporting stepwise, incremental product derivation in product line requirements engineering," *Proc. of VaMoS'10*, ICB-Research Report 37, SSE, Univ. Duisburg-Essen, 2010.

[16] R. Stoiber, S. Fricker, M. Jehle, and M. Glinz, "Feature unweaving: Refactoring software requirements specifications into software product lines." *Proc of RE'10 (posters track)*. IEEE CS, 2010.

[17] B. Wang, W. Zhang, H. Zhao, Z. Jin, H. Mei, "A use case based approach to feature models' construction," in *Proc. of RE'09*. IEEE CS, 2009.

[18] I. v. d. Weerd, S. Brinkkemper, et al., "On the creation of a reference framework for software product management: Validation and tool support," *Proc. of IWSPM'06 Workshop* at RE'06. IEEE CS, 2006.

[19] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in *Proc. of SPLC'09*. ACM, 2009.

[20] U. Zoller, "A comparison of three different concepts for requirements modeling of software product lines – A case study-based investigation [in German]," Bachelor's Thesis, University of Zurich, 2010.

[21] S. Zschaler, P. Sánchez, J. Santos, M. Alférez, A. Rashid, L. Fuentes, A. Moreira, J. Araújo, and U. Kulesza, "VML* - a family of languages for variability management in software product lines," in *Proc. of SLE'09*. Springer, 2009.