Derivation of Subset Product Lines in FeatureIDE

Lukas Linsbauer
TU Braunschweig
Braunschweig, Germany
l.linsbauer@tu-braunschweig.de

Paul Westphal
TU Braunschweig
Braunschweig, Germany
paul.westphal@tu-braunschweig.de

Paul Maximilian Bittner
University of Ulm
Ulm, Germany
paul.bittner@uni-ulm.de

Sebastian Krieter
University of Ulm
Ulm, Germany
sebastian.krieter@uni-ulm.de

Thomas Thüm
University of Ulm
Ulm, Germany
thomas.thuem@uni-ulm.de

Ina Schaefer
Karlsruhe Institute of Technology
Karlsruhe, Germany
ina.schaefer@kit.edu

ABSTRACT

The development and configuration of software product lines can be challenging tasks. During development, engineers often need to focus on a particular subset of features that is relevant for them. In such cases, it would be beneficial to hide other features and their implementation. During product configuration, requirements of potentially multiple stakeholders need to be considered. Therefore, configuration often happens in stages, in which different people contribute configuration decisions for different features. Moreover, in some cases, stakeholders want to share a set of products rather than a specific one. In all these cases, the necessary operation is the same: some features from the product line are assigned a value (e.g., via a partial configuration) while other features remain configurable. In this work, we propose a *subset operation* that takes a product line and a partial configuration to derive a *subset product line* comprising only the desired subset of features and implementation artifacts. Furthermore, we present, evaluate, and publish our implementation of the proposed subset operation within the FeatureIDE framework.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

software product line, partial configuration, subset product line

ACM Reference Format:
Lukas Linsbauer, Paul Westphal, Paul Maximilian Bittner, Sebastian Krieter, Thomas Thüm, and Ina Schaefer. 2022. Derivation of Subset Product Lines in FeatureIDE. In *26th ACM International Systems and Software Product Line Conference - Volume B (SPLC '22), September 12–16, 2022, Graz, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3503229.3547033

This work is licensed under a Creative Commons Attribution International 4.0 License.
SPLC '22, September 12–16, 2022, Graz, Austria
© 2022 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-9206-8/22/09.
https://doi.org/10.1145/3503229.3547033

1 INTRODUCTION

The development of software product lines is challenging for many reasons. In addition to the usual source code, developers need to deal with peculiarities of the respectively used variability mechanism (e.g., a preprocessor), which adds complexity and makes code more difficult to comprehend (e.g., due to annotations scattered across the source code). Additionally, source code of features (and their interactions) that are not relevant for a certain development task further clutter the source code and make it yet more difficult to comprehend and focus on a specific task. In such cases, it would be beneficial to only have to deal with a subset of the entire product line. A similar challenge concern the configuration of product variants that meet a given set of requirements, potentially from multiple stakeholders. Therefore, configuration often happens in stages. This is referred to as staged configuration [8], where the feature selection is gradually refined until it is eventually completed. In other words, the variability of the product line is gradually restricted by assigning values to an increasing number of features until no variability remains and the product line has been reduced to a concrete product variant. Finally, in cases where not a single product variant, but a range of product variants is intended to be shared (e.g., with a subcontractor that only licensed a subset of the original features), it is makes more sense to share a subset of the product line instead of a set of derived product variants.

In all the above scenarios, the fundamental problem is the same: a product line shall (gradually) be reduced in complexity by making certain configuration decisions, be it to simplify a development task, to only leave a subset of configuration choices open for others, or to hide the existence and implementation of certain features. While this general topic has already been researched [4], actual tool support is lacking as there are not many tools that actually provide a concrete and practical implementation of such functionality for existing and commonly used variability mechanisms.

This work is based on a bachelor's thesis [17] and proposes a subset operation for software product lines that takes a product line and a partial configuration to derive a subset product line comprising only the desired subset of features and implementation artifacts. This is achieved by only assigning a specific value to a subset of the features of the product line via a partial configuration while the other features have no value assigned and remain variable. Specifically, this work contributes i) the concept of a subset product line, ii) an operation for the derivation of a subset product line from a product line given a partial configuration, and iii) a practical and publicly available implementation of the above concept

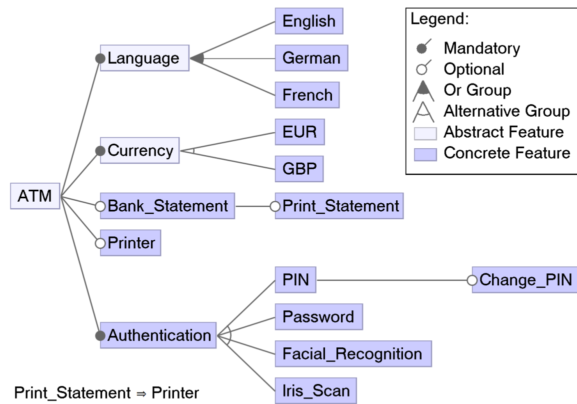

Figure 1: ATM example feature model in FeatureIDE

```
1  public class ATM {
2      private Account acc;
3      ...
4      private void initialize(boolean fullReset) {
5          ...
6          // #if Password
7          auth = new PasswordAuthentication(scan);
8          // #elif PIN
9          auth = new PINAuthentication(scan);
10         // #elif Iris_Scan
11         auth = new IrisScan();
12         // #elif Facial_Recognition
13         auth = new FacialRecognition();
14         // #endif
15         acc = new Account();
16     }
17 }
```

Listing 1: Excerpt of the ATM example implementation, annotated with Antenna preprocessor directives

and operation in the tool FeatureIDE[1] [14] for the Antenna[2] and FeatureHouse[3] [5] composers.

2 MOTIVATING EXAMPLE

As a motivating example, we use an Automated Teller Machine (ATM) that is implemented in Java and also included in the example software product lines included with FeatureIDE. Figure 1 shows the slightly simplified feature model of the ATM. It comprises 17 features in total, of which three are abstract (light gray) and 14 are concrete (blue). An ATM requires a `Currency`, an `Authentication` method, and at least one `Language`. Additionally, developers may choose to include any of the optional features (e.g., `Printer`). The feature model includes a cross-tree constraint that states that the feature `Print_Statement` requires the feature `Printer`. The root feature and its three mandatory features `Language`, `Currency`, and `Authentication` are included in every valid configuration and referred to as core features. An excerpt of the implementation of the ATM product line is shown in Listing 1. It uses the Antenna preprocessor as variability mechanism.

Let us now assume that a company A is producing this ATM but wants to delegate parts of the development to a subcontractor company B. However, the software product line contains source code involving secret technology for the features `Iris_Scan` and `Facial_Recognition` that are not needed by company B for their assignment. Sharing these secret features with company B could increase the risk of a leak, which company A wants to avoid and thus only share the features really needed by company B.

3 TECHNICAL CONCEPT

In this section, we first provide definitions of a software product line and a configuration, then explain partial configurations and subset product lines, and finally introduce operations for the generation of products and the derivation of subset product lines.

Definition 3.1 (Software Product Line). A *software product line* $S = (M, A)$ consists of a feature model M and a set of assets A.

Definition 3.2 (Feature Model). A *feature model* $M = (F, P)$ consists of a set of features F and a propositional formula P over the features F that constrains the valid combinations of features. We assume P to be in conjunctive normal form (i.e., a conjunction of clauses), where every clause $p \in P$ is a disjunction of literals (i.e., features). We treat P as a set of clauses, and a clause $p \in P$ as a set of features.

The example feature model in Figure 1 has the set of features $F = \{\texttt{ATM}, \texttt{Language}, \texttt{Currency}, \texttt{Authentication}, \dots\}$ and the formula $P = (ATM \land (Language \Leftrightarrow ATM) \land (Currency \Leftrightarrow ATM) \land (Printer \Rightarrow ATM) \land \dots)$.

Definition 3.3 (Asset). An *asset* $a \in A$ is a tuple (l, G), where l is the payload of the artifact (e.g., a line of source code) and G is a guard (i.e., a presence condition) which is a propositional formula over features.

The example implementation in Listing 1 has the set of assets $A = \{(\texttt{Line 1}, \top), \dots, (\texttt{Line 11}, \neg\texttt{Password} \land \neg\texttt{PIN} \land \texttt{Iris_Scan}), \dots\}$.

Definition 3.4 (Configuration). A *configuration* C is a set of tuples $c = (f, b)$, where $f \in F$ is a feature and $b \in \{+, -, ?\}$ determines whether feature f is selected (+), deselected (−), or undecided (?) in configuration C. Every feature $f \in F$ must appear exactly once in C, i.e., $\forall f \in F : |\{(f, b) \mid \exists b : (f, b) \in C\}| = 1$. We treat a configuration C as a propositional formula

$$\Big(\bigwedge_{f \in \{f \mid (f,+) \in C\}} f\Big) \land \Big(\bigwedge_{f \in \{f \mid (f,-) \in C\}} \neg f\Big)$$

Definition 3.5 (Full Configuration). A configuration C is a *full* (or *complete* or *total*) configuration if every feature is either selected or deselected (i.e., $\nexists (f, b) \in C : b = ?$).

Definition 3.6 (Partial Configuration). A configuration C is a *partial* configuration if at least one feature is undecided (i.e., $\exists (f, b) \in C : b = ?$).

Figure 2 shows a partial configuration of the feature model in Figure 1 that explicitly selects and deselects some of the features and leaves the remaining features undecided: $\{(\texttt{ATM}, +), (\texttt{Language}, +), (\texttt{English}, ?), (\texttt{German}, ?), (\texttt{French}, -), \dots\}$. The dark gray feature

[1] https://featureide.github.io/, https://github.com/FeatureIDE/FeatureIDE
[2] http://antenna.sourceforge.net/
[3] https://www.se.cs.uni-saarland.de/apel/fh/

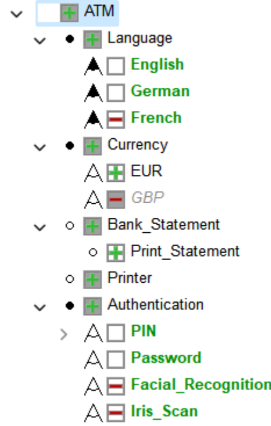

Figure 2: ATM example partial configuration in FeatureIDE

(de)selections were automatically performed by FeatureIDE via decision propagation.

Definition 3.7 (Valid Configuration). A configuration C is *valid* in a feature model $M = (F, P)$, expressed by the predicate valid(M, C), iff the conjunction of C and P is satisfiable (i.e., $SAT(C \wedge P)$) and invalid otherwise.

The example feature model in Figure 1 has 350 full valid configurations. The partial configuration in Figure 2 is also a valid configuration as it does not violate the feature model. However, it cannot be used to generate a product. Instead, it reduces the configurable space to nine remaining full valid configurations.

Definition 3.8 (Product Generation). Given a software product line $S = (M, A)$ with $M = (F, P)$ and a full valid configuration C, a product can be derived as a set of payloads

$$L = \text{generate}(S, C) = \{l \mid (l, G) \in A \wedge C \models G\}$$

Definition 3.9 (Subset Product Line). A software product line $S' = (M', A')$ with $M' = (F', P')$ is a subset of another product line $S = (M, A)$ with $M = (F, P)$, denoted as $S' \subseteq S$, iff $F' \subseteq F$ and $\forall C \in \{C \mid \text{valid}(M', C)\} : \text{valid}(M, C) \wedge \text{generate}(S, C) = \text{generate}(S', C)$.

Definition 3.10 (Subset Product Line Derivation). The operation $subset(S, C) = S'$ derives a subset product line S' from a product line S and a partial configuration C such that:

Features $f \in F$ that are deselected in C (i.e., $(f, -) \in C$) are not included in the set of features F' in S' (i.e., $F' = \{f \mid f \in F \wedge (f, -) \notin C\}$. Clauses $p \in P$ in the feature model M of S that are implied by the configuration are not included in the set of clauses P' of feature model M' of S' (i.e., $P' = \{p \mid p \in P \wedge \neg(C \models p)\}$).

Features $f \in F$ that are selected in C (i.e., $(f, +) \in C$) become core features in S', i.e., for every selected feature f a clause containing only that feature is added to the set of clauses P' of feature model M' of S'. We deliberately decided not to remove selected features in order to maintain feature traceability for these features, even though they were not variable anymore.

Assets $a \in A$ whose presence condition G cannot be satisfied with configuration C (i.e., $\neg\text{SAT}(C \wedge G)$) are not included in the set of assets A' of S' (i.e., $A' = \{(l, G) \mid (l, G) \in A \wedge SAT(G \wedge C)\}$). Every included asset $a' \in A'$ of S' has its presence condition G simplified by removing deselected features from all its clauses.

Table 1: Data Set of Five Software Product Lines

#F Number of Features, #C Number of Constraints, #PPD Number of Preprocessor Directives, #FF Number of Feature Folders

Name	#F	#C	Mechanism	#PPD	#FF
ATM	22	2	Antenna	93	-
Elevator v1.4	21	3	Antenna	109	-
Elevator v1.1	21	3	FeatureHouse	-	9
GPL	38	16	FeatureHouse	-	27
BerkeleyDB	119	68	FeatureHouse	-	99

In summary, the subset operation i) excludes deselected features, ii) includes selected features as core features, iii) removes assets whose presence condition is contradicted, and iv) substitutes the value false for every deselected feature in the propositional formulas of the feature model and the assets and then simplifies them.

4 TOOL IMPLEMENTATION

We implemented the described concept of partial configurations and the derivation of subset product lines within the tool FeatureIDE and released it with version 3.7.0[4]. FeatureIDE provides a wide range of sophisticated feature model analyses and configuration support, as well as various different variability mechanisms (i.e., composers). Using our extensions, the same analysis techniques that FeatureIDE provides for full configurations, such as checking their validity, explaining their invalidity, and propagating configuration decisions, can also be applied to partial configurations. Further, we extended the composer interface of FeatureIDE and the two existing composers Antenna and FeatureHouse with the corresponding implementations of the subset functionality. We demonstrate the new functionality in a publicly available online video[5].

5 EVALUATION

We evaluated our implementation regarding its *correctness* and *scalability* by applying it to a corpus of five software product lines.

Data Set. Table 1 shows for each product line its number of features (#F), number of constraints (#C), used variability mechanism (either the Antenna preprocessor or FeatureHouse), number of preprocessor directives (#PPD) in case of Antenna, and number of feature folders (#FF) in case of FeatureHouse. Our data set thus covers annotative (or subtractive) variability by means of a preprocessor and compositional (or additive) variability by means of feature-oriented programming.

Goals. The *correctness* of the implementation was evaluated by verifying that the products generated from subset product lines are equal to the corresponding products with the same configuration generated from the original product line. The *scalability* of the implementation was evaluated by measuring the runtime of the derivation of subset product lines.

Process. For every software product line in the data set the following steps were performed:

[4]https://github.com/FeatureIDE/FeatureIDE/releases/tag/v3.7.0
[5]https://youtu.be/g1LGIAezvHg

(1) Generate five random valid configurations with increasing number of undecided features.
(2) Derive a subset software product line for each configuration and measure the runtime.
(3) Generate all valid products of each subset software product line.
(4) Generate the same products (i.e., with same configuration) from the original software product line.
(5) Compare each product of the subset product line to the corresponding product of the original product line.

Results. In all cases, the products of the derived subset product line were 100% identical to the products of the original product line. On average, the derivation of a subset product line took between one and two seconds for the first four product lines and 24 seconds for BerkeleyDB, the largest product line in the data set. Note that most of the time is spent on modifying the assets and only very little time on modifying the feature model.

6 RELATED WORK

Acher et al. [1] initially introduced feature model slicing, which aims to remove features from feature models while preserving implicit dependencies between the remaining features. While feature model slicing also removes a feature literal from the feature model formula without assigning a truth value, the subset operation introduced in this work binds a feature literal to a truth value, which ultimately results in different formulas. Bürdek et al. [7] specify and reason about edits to feature models but do not cover the source code and other artifacts of the product line.

Similar to our work, refactorings [2, 10, 13, 15], generalizations [2], and refinements [6] to product lines ensure that certain edits retain valid configurations and the external behaviour of products. The difference is that our approach specifically reduces the set of valid configurations instead of keeping or growing it.

Multi software product lines [12] exhibit the complexity of multiple individual software product lines. Our subset operator might prove useful to reduce complexity, when applied to the individual product lines. Analyses of software product lines, which analyze the possibly exponentially many products [9], might benefit from our subset operator by reducing the variability before performing analyses (e.g., when only a subset of variants is of interest).

Ananieva et al. [3, 4] study tools related to managing variability in space and time, specifically their concepts [3] and operations [4]. FeatureIDE is among the studied tools and the subset product line operation presented in this work has already been considered in their study. They found that a conceptually similar operation is supported by the tools VTS [16] and ECCO [11]. The tool VTS by Stanciulescu et al. [16] provides get and put operations for product lines. The get operation is conceptually similar to the subset operation but does not support a feature model and thus realizes the subset operation only for textual implementation assets annotated with preprocessor directives. The put operation allows for the reintegration of a (modified) subset product line into the original product line. Our implementation of the subset operation in FeatureIDE considers the feature model and supports different composers (i.e., variability mechanisms) beyond preprocessors. However, the reintegration of a subset product line into the original product line is not yet supported by our extension of FeatureIDE.

7 CONCLUSION AND FUTURE WORK

In this paper, we presented the derivation of a subset product line from a product line, given a partial configuration. This operation is useful, for example, for performing staged configuration, applying licensing restrictions, or reducing complexity during development. We first presented the general concept and illustrated it on a motivating example and then showed a concrete implementation and application in the tool FeatureIDE.

To complete the cycle and support even more application scenarios, the next step is the conception and implementation of a *union* operation for product lines that enables the reintegration of a previously derived subset product line into the original product line. This would, for example, enable the evolution of a full product line via simpler subset product lines.

ACKNOWLEDGMENTS

This work has been partially supported by the German Research Foundation within the project VariantSync (TH 2387/1-1).

REFERENCES

[1] Mathieu Acher, Philippe Collet, Philippe Lahire, and Robert B. France. 2011. Slicing Feature Models. In *ASE*. IEEE, 424–427.
[2] Vander Alves, Rohit Gheyi, Tiago Massoni, Uirá Kulesza, Paulo Borba, and Carlos José Pereira de Lucena. 2006. Refactoring Product Lines. In *GPCE*. ACM, 201–210.
[3] Sofia Ananieva, Sandra Greiner, Timo Kehrer, Jacob Krüger, Thomas Kühn, Lukas Linsbauer, Sten Grüner, Anne Koziolek, Henrik Lönn, S. Ramesh, and Ralf H. Reussner. 2022. A Conceptual Model for Unifying Variability in Space and Time: Rationale, Validation, and Illustrative Applications. *EMSE* 27, 5 (2022), 101.
[4] Sofia Ananieva, Sandra Greiner, Jacob Krüger, Lukas Linsbauer, Sten Grüner, Timo Kehrer, Thomas Kühn, Christoph Seidl, and Ralf H. Reussner. 2022. Unified Operations for Variability in Space and Time. In *VaMoS*. ACM, 7:1–7:10.
[5] Sven Apel, Christian Kästner, and Christian Lengauer. 2013. Language-Independent and Automated Software Composition: The FeatureHouse Experience. *TSE* 39, 1 (2013), 63–79.
[6] Paulo Borba, Leopoldo Teixeira, and Rohit Gheyi. 2012. A Theory of Software Product Line Refinement. *TCS* 455, 0 (2012), 2–30.
[7] Johannes Bürdek, Timo Kehrer, Malte Lochau, Dennis Reuling, Udo Kelter, and Andy Schürr. 2015. Reasoning About Product-Line Evolution Using Complex Feature Model Differences. *AUSE* 23, 4 (2015), 687–733.
[8] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. 2005. Staged Configuration through Specialization and Multi-Level Configuration of Feature Models. *SPIP* 10, 2 (2005), 143–169.
[9] Aleksandar S. Dimovski, Claus Brabrand, and Andrzej Wąsowski. 2018. Variability Abstractions for Lifted Analyses. *SCP* 159 (2018), 1–27.
[10] Wolfram Fenske, Thomas Thüm, and Gunter Saake. 2014. A Taxonomy of Software Product Line Reengineering. In *VaMoS*. ACM, 4:1–4:8.
[11] Daniel Hinterreiter, Lukas Linsbauer, Herbert Prähofer, and Paul Grünbacher. 2021. Feature-Oriented Clone and Pull for Distributed Development and Evolution. In *QUATIC (Communications in Computer and Information Science, Vol. 1439)*. Springer, 67–81.
[12] Gerald Holl, Paul Grünbacher, and Rick Rabiser. 2012. A Systematic Review and an Expert Survey on Capabilities Supporting Multi Product Lines. *IST* 54, 8 (2012), 828–852.
[13] Flávio Medeiros, Márcio Ribeiro, Rohit Gheyi, Sven Apel, Christian Kästner, Bruno Ferreira, Luiz Carvalho, and Baldoino Fonseca. 2018. Discipline Matters: Refactoring of Preprocessor Directives in the #ifdef Hell. *TSE* 44, 5 (2018), 453–469.
[14] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.
[15] Sandro Schulze, Thomas Thüm, Martin Kuhlemann, and Gunter Saake. 2012. Variant-Preserving Refactoring in Feature-Oriented Software Product Lines. In *VaMoS*. ACM, 73–81.
[16] Stefan Stanciulescu, Thorsten Berger, Eric Walkingshaw, and Andrzej Wasowski. 2016. Concepts, Operations, and Feasibility of a Projection-Based Variation Control System. In *ICSME*. IEEE Computer Society, 323–333.
[17] Paul Westphal. 2020. *Deriving Subset Software Product Lines Using Partial Configurations with FeatureIDE*. Bachelor's Thesis. TU Braunschweig.