FeatureIDE: Scalable Product Configuration of Variable Systems

Juliana Alves Pereira,[1] Sebastian Krieter,[1] Jens Meinicke,[1,2]
Reimar Schröter,[1] Gunter Saake,[1] Thomas Leich[2]

[1] University of Magdeburg, Germany, [2] METOP GmbH, Germany

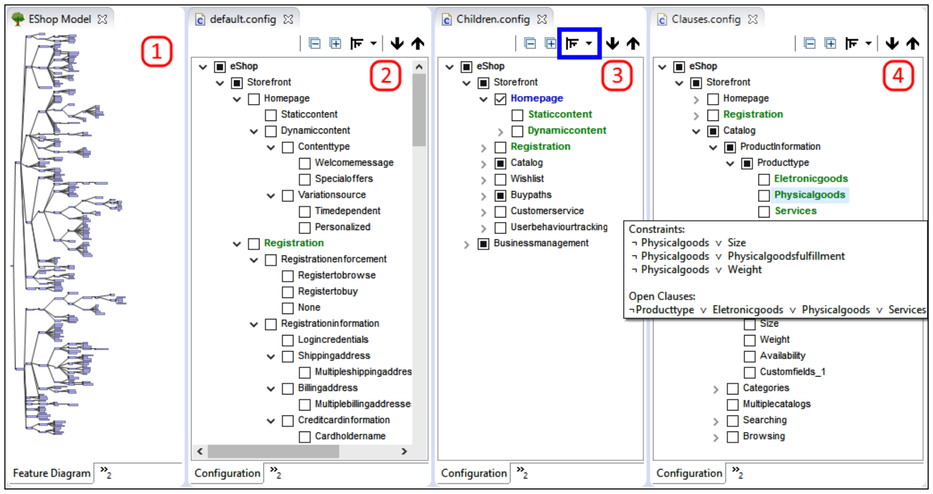
Abstract. In the last decades, variability management for similar products is one of the main challenges in software systems. In this context, feature models are used to describe the dependencies between reusable common and variable artifacts, called features. However, for large feature models it is a complex task to find a valid feature combination as product configuration. Our Eclipse plug-in FeatureIDE provides several mechanisms, such as information hiding and decision propagation, which support the configuration process to combine the reusable artifacts in various manners. We illustrate the applications of these mechanisms from a user's point of view.
Demo Video. https://youtu.be/zM9K3wqUiVE

1 Introduction

Variable software systems are essential to fulfill the individual requirements of several users. Such systems are commonly based on reusable but interdependent artifacts represented by a set of features that can be combined to form custom products [8]. Feature models are a common notation to define features and their interdependencies [4]. As feature models specify the set of valid products (i.e. a selection of features that fulfills all interdependencies), they form the basis of the product configuration process.

In industry feature models often define several thousand features. Hence, it is impractical for the user to keep track of all features and their dependencies during the configuration process. On the one hand, it may be difficult for a user to specify a valid configuration, especially since also features of no interest need to fulfill their dependencies. On the other hand, the user can unintentionally introduce conflicts by specifying mutually exclusive features. However, the user can be guided to configure valid products using specialized tool support.

In this paper, we present the configuration support of our tool FeatureIDE [5,9]. With a close connection to FeatureIDE's feature-model editor, the configuration editor can provide several mechanisms that guide the user. With automated decision propagation, we ensure that any partially configured product is in accordance to the feature model so that the result only describes valid combination of reusable artifacts. Furthermore, we help the user with information hiding mechanisms that let them focus on the parts of the configuration that are of interest. Finally, we present how we guide the user to a valid configuration.



```
eShop ▣
  Storefront ▣
    Homepage ☐
      Staticcontent ☐
      Dynamiccontent ☐
        Contenttype ☐
          Welcomemessage ☐
          Specialoffers ☐
        Variationsource ☐
          Timedependent ☐
          Personalized ☐
    Registration ☐
      Registrationenforcement ☐
        Registertobrowse ☐
        Registertobuy ☐
        None ☐
      Registrationinformation ☐
        Logincredentials ☐
        Shippingaddress ☐
          Multipleshippingaddres
        Billingaddress ☐
          Multiplebillingaddresse
        Creditcardinformation ☐
          Cardholdername ☐
```

```
eShop ▣
  Storefront ▣
    Homepage ✓
      Staticcontent ☐
      Dynamiccontent ☐
    Registration ☐
    Catalog ▣
    Wishlist ☐
    Buypaths ▣
    Customerservice ☐
    Userbehaviourtracking ☐
  Businessmanagement ▣
```

```
eShop ▣
  Storefront ▣
    Homepage ☐
    Registration ☐
    Catalog ▣
      ProductInformation ▣
        Producttype ▣
          Eletronicgoods ☐
          Physicalgoods ☐
          Services ☐
          Size ☐
          Weight ☐
          Availability ☐
          Customfields_1 ☐
      Categories ☐
      Multiplecatalogs ☐
      Searching ☐
      Browsing ☐
```

```
Constraints:
¬ Physicalgoods ∨ Size
¬ Physicalgoods ∨ Physicalgoodsfulfillment
¬ Physicalgoods ∨ Weight

Open Clauses:
¬Producttype ∨ Eletronicgoods ∨ Physicalgoods ∨ Services
```

Fig. 1. An overview of FeatureIDE's configuration support: ① feature model editor, ②-④ configuration editor (② showing all features, ③ showing direct children, ④ finalizing configuration).

2 Preventing Conflicting Feature Combinations

Product configuration is a decision process to form a valid feature combination, where the interdependencies of all features are considered [8]. Especially when dealing with large feature models with complex feature dependencies, a configuration process without tool support is an error-prone and tedious task. Completely configuring products and checking validity afterwards is henceforth not advisable as at least one feature dependency is probably violated.

To ease the configuration process, FeatureIDE provides an iterative strategy, which only allows feature selections that comply with the feature model's dependencies. Thus, similar to the tools SPLOT [6] and fmp [1], FeatureIDE prevents the user to introduce conflicts in their configuration. This functional characteristic of FeatureIDE is based on two concepts: (a) a close coupling between configurations and their feature models and (b) decision propagation.

Close Connection of Feature Models and Configurations. The feature model and the configuration editor of FeatureIDE are closely connected and influence each other. On the one hand, the configuration editor of FeatureIDE uses the same hierarchical structure as the corresponding feature model. Furthermore, the feature model influences configurations so that, for instance, a renaming of a feature also renames the feature in each configuration. On the other hand, each selection in a configuration forces a validity check considering the corresponding feature model. In addition, all implied and excluded features are automatically (de)selected and a change of their selection is forbidden. In Figure 1.①-②, we depict this functionality for the product line *EShop*. In Figure 1.①, the dependencies of the feature model are hard to resolve. However, the representation in the configuration editor (see Figure 1.②) allows an iterative selection of features according to the feature model.

Decision Propagation. Based on the close connection between feature models and configurations, FeatureIDE's configuration editor prevents conflicts in each iteration of the configuration process using *decision propagation*. In detail, if a (de)selection of a feature forces the (de)selection of another feature, FeatureIDE automatically adopts the implied configuration changes. For instance, if we select the feature *Welcomemessage* in the product line *EShop* (see Figure 1.②), all parent feature will be also selected.

3 Information Hiding

Configuring a product can be a difficult process as users usually do not know all features and their dependencies, especially for large feature models [3]. Consequently, showing all features (see in Figure 1.②) is impractical as a user can only focus on one part of the configuration at once. However, a user may already know their features of interest. To ease the configuration process, we provide information hiding mechanisms that focus the user's view on the relevant configuration space. The user can select one of these mechanisms via the configuration editor's menu bar (see the blue rectangle in Figure 1.③).

Focused View. FeatureIDE aims to focus on the part of the configuration that is currently modified. Thus, it initially does not expand all features. When the user selects a feature, they are probably interested in its sub-features (e.g., fine-grained features of the same area). We provide a specialized expand algorithm that automatically expands and shows the sub-features after a feature is selected. This behavior is exemplary illustrated in Figure 1.③. Initially, only the feature *Storefront* is expanded. After the user selects the feature *Homepage*, the expand algorithm shows the sub-features *Staticcontent* and *Dynamiccontent*. With the focus on direct children only, we reduce the number of presented configuration options significantly and present only features that are of interest at the moment.

Finalize Partial Configurations. Decision propagation and specialized expand algorithms can only help to configure partial configurations. Still, a configuration needs to fulfill all dependencies defined in the feature model. Automatic selection of features is an efficient way to create a valid configuration based on the given partial configuration (e.g., the auto-completing mechanism presented by SPLOT [6]). However, such algorithms arbitrarily select features without considering the user's intentions. Thus, undesired features might be selected as well. In order to address this challenge, the tools VISIT-FC [7] and FaMa [10] introduce dependency visualization mechanisms to support the user in configuring products, but both tools present all features to the user. In contrast, FeatureIDE provides a mechanism that guides the user to a valid configuration, reasoning from a smaller number of features. Based on unsatisfied clauses of the feature model's CNF-representation [2], its mechanism shows the user which decisions are necessary to finish the configuration process by highlighting the corresponding features. As each clause needs to be satisfied, the user can focus on one clause at a time. Thus, again the number of configuration options presented to the user is reduced to a minimum. We exemplary show this behavior in Figure 1.④. As shown, only the current open clause (displayed in the tooltip of *Physicalgoods*) is expanded. The feature *Producttype* was automatically selected by decision propagation. Thus, at least one of its children (highlighted with green) has to be selected to satisfy the open clause. A deselection of a feature might also satisfy a clause as shown in Figure 1.③ with a blue highlighting of the feature *Homepage*. After a clause is satisfied by the user's (de)selection, the focus will automatically change to the next unsatisfied clause. Using this mechanism, the user can efficiently finish the configuration process and simultaneously prevent undesired feature selections.

4 Conclusion

Feature models describe the dependencies between features in order to specify valid product configurations. However, the actual process of configuring products for large feature models is an error-prone and tedious task. In this paper, we illustrate FeatureIDE's facilities to support this process by providing advanced configuration support, such as decision propagation and information hiding. This approach ensures a valid and complete configuration while simultaneously maintaining efficiency as the user can focus on their features of interest.

References

1. M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature Modeling Plug-In for Eclipse. In *Eclipse*, pp. 67–72. ACM, 2004.
2. D. Benavides, S. Segura, P. Trinidad, and A. Ruiz-Cortés. A First Step Towards a Framework for the Automated Analysis of Feature Models. In *SPLC*, pp. 39–47. IEEE, 2006.
3. J. Bosch, R. Capilla, and R. Hilliard. Trends in Systems and Software Variability. *IEEE Software*, (3):44–51, 2015.
4. 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 CMU/SEI-90-TR-21, Software Engineering Institute, 1990.
5. J. Meinicke, T. Thüm, R. Schöter, S. Krieter, F. Benduhn, G. Saake, and T. Leich. FeatureIDE: Taming the Preprocessor Wilderness. In *ICSE*. ACM, 2016. to appear.
6. M. Mendonça, M. Branco, and D. Cowan. S.P.L.O.T.: Software Product Lines Online Tools. In *OOPSLA*, pp. 761–762. ACM, 2009.
7. D. Nestor, S. Thiel, G. Botterweck, C. Cawley, and P. Healy. Applying Visualisation Techniques in Software Product Lines. In *SoftVis*, pp. 175–184. ACM, 2008.
8. K. Pohl, G. Böckle, and F. J. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
9. T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. FeatureIDE: An Extensible Framework for Feature-Oriented Software Development. *SCP*, 79(0):70–85, 2014.
10. P. Trinidad, A. R. Cortés, D. Benavides, and S. Segura. Three-dimensional feature diagrams visualization. In *SPLC*, pp. 295–302, 2008.