Questionnaire-Based Configuration of Product Lines in FeatureIDE

Jens Wiemann, Stephan Dörfler,
Otto-von-Guericke-Universität Magdeburg,
{jens.wiemann, stephan.doerfler}@st.ovgu.de

***Abstract*—Variability management is an essential part of working on product lines. Feature models are an established tool to describe the set of features and constraints contained in a given software product-line, thus showing the variability. Feature models also facilitate configuration of product lines. However, for most software product lines there exists no feature model. This paper proposes a method for automatically generating feature models out of the structure of given source code and naming conventions.**

Furthermore, an alternative method of configuration based on questionnaires is developed in this work. It enables users or customers to configure a product on their own. Also, it allows experts to design the questionnaires according to their domain knowledge during implementation of the product line.

***Index Terms*—FeatureIDE, Feature Model, Extraction, Configuration, Questionnaire.**

I. Introduction

Developing feature oriented fashion facilitates the creation of software when it is planned on custom tailoring it for a multiple number of clients [1]. Feature oriented means developing and maintaining single features that can be combined with others to create a whole product. All the individual features make up a product line, whereas a subset of those features create a variant of this product line. Developing software product lines can result in a large amount of variants, when customizing the software to each customers needs. By developing with a feature-oriented approach the configuration of a single variant can be done by selecting the features a customer needs. Often features have dependencies to one another, which can be added to a feature model to verify a configuration. A feature model documents the features of a product line and their relationships [2].

There are projects being developed in a feature-oriented manner, but don't have a feature model yet. Implementing it on top of a given project can result in a complex task due to the amount of its features and the constraints between them. Nevertheless, the benefits of a correct feature model justify the effort to extract one out of a live project. To simplify a big part of the feature model creation, this work aims at automatically generating it out of descriptive files and naming conventions.

Although the feature configuration gives developers the ability to create custom variants, there still has to be a consultant explaining the features to the customer, trying to figure out his current and future needs. As the software grows and gets more features, this process gets difficult, as one can no longer explain all the features, but still has to figure out if the customer needs them or not. This paper tries to come up with a better alternative based on questionnaires to enable users or customers to configure a product on their own and to allow experts to design the questionnaires according to their domain knowledge.

For the implementation of the feature extraction and the questionnaire, we used the eclipse plugin FeatureIDE. FeatureIDE is an open source IDE for feature-oriented-software development. It provides all the functionality needed to programmatically generate and work with feature models. This includes the logic behind configurations, data structures and the visualization and processing of feature models.

In the second part of this work, we demonstrate the feature-model extraction and questionnaire creation based on a real life project. The project, an open source ERP system called *Odoo*[1] is predestined to be developed feature-oriented due to a huge amount of features and complex constraints. It is currently being developed and structured feature oriented, although it doesn't contain a feature model yet. Furthermore, *Odoo* reached a critical amount of features where a salesman cannot consult a customer by going through all the features anymore, but has to come up with a more effective way.

Contributions:

- Automatic feature-model generation based on descriptive files and naming conventions as a foundation for the thereon based questionnaire.
- We introduce an Eclipse plugin, that enables the creation of a questionnaire based on a configuration file and the associated feature model. The liberties of the configuration file specification allows very personalized questionnaires.
- The questionnaire reacts to the choices made by reacting differently on each questions outcome. This enables a responsive questionnaire that prevents unnecessary questions. In addition, every choice made creates a valid partial configuration, even if the questionnaire is exited early.
- Illustration of the described workflow by applying it to a real world project.

II. Basics for simplification of the configuration process

In the scope of supporting the configuration of a given product line some specific tools and techniques are used. This section gives an overview about the used techniques.

[1] https://www.odoo.com/

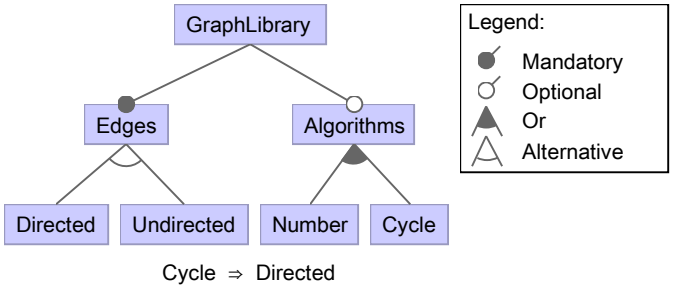

Fig. 1. A simple example of a feature model

A. Feature Models

Feature models are a multi-purpose structure for product lines. One of their benefits is the visualization as a feature diagram, providing an overview of the containing features, and their hierarchy and dependencies. In addition, it classifies the features and their dependencies by ordinality, logical operator and whether it is abstract or concrete. Furthermore, it contains all information needed to provide a configuration of features and its validation. A configuration, or chosen set of features, is valid if there are no contradictions in the models dependencies or logical operators.

There are two main representations of feature models. The first approach is a propositional expression, which is a type of syntactic formula often used in propositional logic. However feature models are typically visualized in form of a feature diagram, which is structured as a tree [3]. In that manner feature models map the hierarchy of features. The possible relationships of features with a common parent feature are *or*, *alternative* and *and*. All child features beneath a parent feature have the same relationship and therefore form a *group*. In the case of an *or*-group, at least one of the child features has to be selected if the parent feature is selected. Within an *alternative*-group exactly one of the child feature has to be selected if the parent feature is selected. When features are grouped with an *and*-relation, they are marked as either mandatory or optional. If the parent feature of an *and*-group is selected, any number of optional marked features can be selected. Features marked as mandatory have to be selected. As features' relations may be of higher complexity than just parent-child relations additional constraints can be noted within a feature model. Constraints can contain any propositional expression.

In Figure 1 we demonstrate an example of a simple feature model. The node *GraphLibrary* is the root feature. It's childs are the node *Edges*, that has to be selected due to the mandatory property, and *Algorithms* that is marked optional. The implemented edge types are *Directed* and *Undirected* from which exactly one has to be chosen as they are mutually exclusive. From the algorithms at least one feature has to be chosen whenever *Algorithms* is selected. The constraint at the bottom of Figure 1 implies that if the feature *Cycle* is selected, the edge type needs to be directed.

B. Product Configuration

The variability of a product line is represented by it's feature model, which itself is a set of features with specific interrelations. The configuration process of a product line describes the steps to derive a product from the product line. To archive this, a user has to select a subset of all the possible features within the product line to meet his requirements [4]. However, not all subsets of features result in a valid product. The interrelations of the features restrict the possible combinations of features. Thus, not all arbitrary cobinations of features result in a *valid* configuration. If only one of the requirements from the interrelations between the features is validated, no product can be created and the configuration as such is considered *invalid*. For an automatic validity check, see the next section.

A non-final configuration, that is a configuration that still has some feature choices left to be made, is called a *partial configuration*. A *valid* partial configuration can be useful, as it narrows down the number of possible resulting variations and choices left to make. As the aim is to achieve a valid final configuration, there has to be a valid partial configuration after every choice made.

C. Satisfiability

During creation of the feature model as well as during the configuration of a product, validity must always be taken care of. Validity in this context is described as the satisfiability of the corresponding propositional expression.

The formalism of propositional expressions (see Section II-A) allows feature models and configurations to be checked for validity. Each selected feature is appended with a logical *AND* and each specifically unselected feature is also appended with a logical *AND* but gets negated. The resulting expression is then evaluated by a SAT-solver to check for satisfiability [5]. If the expression is satisfiable the selected features make up a valid configuration.

Even during configuration this process can be applied to check for invalid partial configurations after each decision. If a partial configuration reduces the valid choices in a group to a single one, this choice can be made automatically. This is called *propagation*. The automatic selection or deselection of a features due to propagation of a made selection always results in a valid configuration [6].

III. Workflow based on an existing software product line

This chapter describes the general workflow we propose within this work. This can be seen in Figure 2. To start off, some kind of software product line is required. It's source code has to be structured in a way that allows for an algorithm to recognize the individual features as well as their interrelations, as these can be very costly to implement by hand.

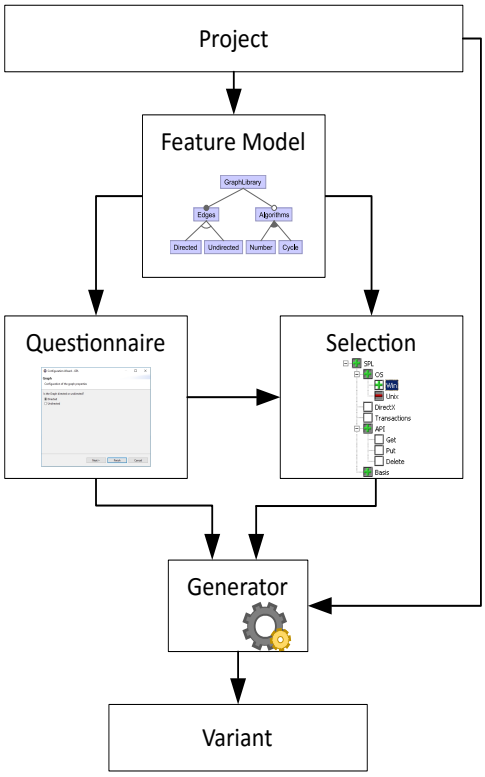

Fig. 2. Diagram showing the general workflow

The two major steps of extraction of the feature model and the creation and usage of the questionnaire are described in detail in the following sections. As a final result of these steps stands a product targeting the users needs.

A. Extraction of a Feature Model

Out of the given structure of the source code base we extract the hierarchies of the features as well as additional dependencies. These are the input for the automated generation of a feature model. This allows for overview over the product line and it's variability. During our efforts to implement an automated generation of a feature model we encounter some problems for which we have to find solutions or workarounds.

The first challenge consists of finding the structures of the given source code and processing them programmatically. Most software complies to conventions regarding the naming and structuring of directories. We try to make use of these conventions. However, in most source code there are violations within their own conventions. Instead of interrupting the whole process, we implement error handling. The result is a notification for the user to alert the developers about the error. Also, the remaining information, which aren't immediately affected by the error, are still extracted and used for the generation of the feature model, if possible.

Another challenge lies within the order of processing the features during the creation of the feature model. To correctly reproduce the hierarchy of the features, features have to be declared as child features of their parents. If the declared parent feature doesn't already exist, the creation of a feature model fails. So we have to ensure for each feature, that it gets processed after it's parent feature.

Another error we found is one or more features having a parent feature which doesn't exist as a code artifact itself. Thus, the parent feature doesn't exists in the feature model and it's child features can't be placed in the hierarchy in the correct place. To still include these features in the feature model we create an abstract feature [2] for the parent feature. The abstract feature is a direct child of the root feature. As it doesn't have any describing code, it only has a name and no additional descriptive details.

In each step of the creation of the feature model, e.g. after adding a feature or constraint to the feature model, it is checked by a SAT-solver to avoid creating an invalid feature model (represented by a void object) which has no valid configurations (see section II-C). The extracted feature model is also used in the following step of configuration. There, it visualizes the interrelations of the features and supports the understanding of the configuration steps.

B. Questionnaire Approach

Configuration is a challenging tasks within the scope of software product lines. The resulting feature model from the automated generation yields a better overview over the possible features and their interrelationships. Although the formalism of a feature model allows for tool support for the process of configuration, still domain knowledge is required to be able to find the right combination of features for a given use-case.

This work introduces a method allowing experts to apply their knowledge and understanding to a whole product line during the development. Whenever a variation is configured using the questionnaire, the user benefits from the invested domain knowledge.

In this work, we made the decision to use a questionnaire-based approach [7]. In the progress of implementation, experts also design a questionnaire. They do so in such a way, that a user has to answer a given amount of questions to perform the configuration of a product. Depending on the implementation of the questionnaire, a partial or even complete configuration can be archived by a user through just answering the questions of the questionnaire.

The selection of features gets lifted to a higher level of abstraction. The user only has to decide between the possible answers presented to him in the questionnaire to best fit to his use-case. Internally the selected answers are mapped to a specified (de-)selection of features. This way the user avoids the hassle of considering implementation-details of each feature. This effectively redesigns the process of configuration in such a way, that a user is independent of the knowledge about the details of the implementation and can focus on tailoring the product line to his specific use-case.

C. Questionnaire Implementation

The quality of the resulting variation highly depends on the implementation of the questionnaire during development. Our work therefore introduces a set of tools to easily integrate such a questionnaire. The following paragraphs will give an overview over the possible definitions of a questionnaire.

Each page of the questionnaire is defined independently. It always contains at least a question and more than one possible answer. The answers can be grouped analogous to the grouping of features in a feature model: *or* (at least one answer has to be selected), *alternative* (exactly one answer has to be selected) and *and* (any number of answers can be selected).

Each answer internally has a mapping to a set of features. In the case a specific answer is chosen, the corresponding features are selected or deselected as specified.

Each answer can also have an indicator to define which page of the questionnaire should be displayed next. An answer can also indicate the end of the questionnaire. If no next page is defined the questionnaire will continue with the next page within it's definition. This allows a dynamic conditional design of the questionnaire so the user is only confronted with the exact set of questions needed to configure the variant for his specific use-case. This also allows the user to skip questions or cancel the configuration before finishing it and thus creating a partial configuration.

We also introduce a data structure to hold the definition of a questionnaire. To archive easy integration we decided for a definition in XML. We defined the necessary tags to create a questionnaire which are displayed in Figure 4.

Tag	Attributes	Child tags
configura-tionSurvey		projectName, section, page
section	id	name, description
page	id, sectionId	question, answers
answers	type	answer
answer	nextPageId	label, description, dependencies
dependencies		feature
feature	selection	

Fig. 3. Table with the XML element structure

The individual tags are explained as follows:

- `configurationSurvey`: The root tag to contain all other tags for the questionnaire.
- `section`: Enables grouping of question-pages. Also displays the name and description at the top of every page.
- `page`: Contains a question and the corresponding answers. Also has an indicator for a section
- `answers`: Contains the individual possible answers and groups them in the specified manner.
- `answer`: Defines the displayed text of an answer as well as the corresponding features. Can also have an indicator for the next page.
- `dependencies`: Defines the (un-)selection of features, if the corresponding answer gets selected.

```
<configurationSurvey>
  <projectName>Name</projectName>
  <section id="0">
    <name>Section Name</name>
    <description>Section description</description>
  </section>
  <page id="0" sectionId="0">
    <question>Question for the user</question>
    <answers type="alternative">
      <answer nextPageId="1">
        <label>Answer label</label>
        <description>
          Answer description
        </description>
        <dependencies>
          <feature selection="false">
            Unselected feature
          </feature>
          <feature>selected feature</feature>
        </dependencies>
      </answer>
    </answers>
  </page>
</configurationSurvey>
```

Fig. 4. Examplary XML file for a questionnaire

This allows for experts to design a questionnaire in such a way, that it can guide a user without specific domain knowledge throughout the whole process of configuration. Through just answering questions concerning his specific use-case the user takes all necessary decisions. After he completes the questionnaire, our tool maps his answers to a complete configuration, which is then applied to the existing code artifacts. Finally, a working product is the result.

IV. Examplary scenario

In the fallowing chapter the workflow described in chapter III is carried out based on the current project status of *Odoo*[2]. *Odoo* (formerly known as OpenERP) is a suite of *open-source*[3] enterprise applications, targeting companies of all sizes. As each costumer needs different functionalities from *Odoo*, the product variations can be very different in size and target use. Besides them researching in the field of easing their product configuration process, this project suits our workflows requirements.

The current version of *Odoo* (version 9.0, released October 1, 2015) has 465 contributors and over 101.200 commits. They have developed 266 features which are all inside of a top hierarchy folder named "addons". Each of them contain code as well as a configuration file named "__openerp__.py". Inside this file there are all the relevant informations for this feature, like dependencies, summaries , categories, versions and more descriptive informations. Inside the *Odoo* repository there are also other directories, like "doc" for documentary files, "setup" for deployment scripts on various platforms and "openerp" with the core code. However, as the variability is only contained within the features, the only relevant directory for this work is "addons".

At first, a feature model is generated based on the naming conventions of the project folders and the configuration files

[2]https://www.odoo.com/
[3]https://github.com/odoo/odoo

contained therein. Afterwards the feature model is corrected from any errors that occurred during the generation process. The resulting model combined with domain knowledge allows for the creation of a questionnaire that takes a minimum of choices to configure a custom tailored application.

All the *Odoo* features lie inside of a folder named ”addons”. The code of each feature is contained in its own folder which is named like the parent feature with a suffix of its own name. According to this naming convention the feature ”website_sale” is a child of the feature ”website”, and parent to ”website_sale_stock”. In addition to the naming convention, there is a configuration file, written in python in each of the feature folders. These files contain a description of the feature, a name, that doesn't always match with the folder name and the dependencies that are required for this feature to work. By first creating a list of all features based on the folder names, we can match them to the names inside of the configuration files. At this point we realize, that there are some conflicts in the naming of the 261 features. For example the feature ”sale” is a standalone feature and has nothing to do with the sale from the feature ”event_sale”. As the default FeatureIDE doesn't support multiple features with the same name, we decided to leave the full names with the underscores. In addition, there are some folder names where the underscore doesn't separate features, like ”point_of_sale”. For those we implemented naming exceptions where the user can add strings that represent a single feature.

We applied these steps to a total of 261 folders, containing in total more than 18.800 files, extracting 306 features. The surplus features result from abstract features, that do not have their own folder, but are implizit in the naming. The resulting feature model is arranged in a total of six hierarchical levels. Additionally, 196 constraints limit the possible valid configurations of *Odoo*.

The choices made by the user have to be evaluated. This happens after each single question got answered. Especially the validity needs to be ensured at every time. Optimally, the questionnaire gets designed in such a way, that it is impossible to create an invalid configuration by just answering it. However, as the questionnaire itself does not get checked for validity, checking the answers adds that needed reassurance for validity. To check for validity, the dependencies assigned to a selected answer are checked. If a dependency-feature is previously unselected (undefined), it get (de-) selected according to the definition in the questionnaire. But if is already is in a state of (de-) selection, that doesn't match the state assigned in the questionnaire, an error occurs and warns the user of an invalid configuration.

As stated in Section III-C, it is possible to define a dynamic ordering of the questions within the questionnaire according to the given answer. The next step in evaluating the user input is looking for the correct question to be asked next. If there is a next question explicitly stated in the selected answer, that question gets displayed next. Else, if there are questions left unanswered in the questionnaire, that occur after the current questions, the question displayed next is the question following the current question in the definition. If there is no next question to be displayed, the questionnaire is considered finished. Optimally, this also results in a final configuration of a product. However, our design also allows for partial configurations as a result of an executed questionnaire. In the case of a finished configuration, this configuration gets passed to FeatureIDE, which then compiles the configured product.

TODO: Fragebogen

V. Related Work

Several techniques for synthesising feature models from a set of configurations, constraints, or even publicly available product descriptions have been proposed [8]–[10]. These techniques assume and require very few informations of the source, but therefore can only create limited feature models. Furthermore they cannot be applied in our context, as they assume the availability of formal and complete descriptions of configurations and constraints. Therefore we developed a new extraction method that relies on set project structure and conventions. This leads to a feature model with a correct hierarchy, original feature names, descriptions and complete dependencies.

Davril et al. [10] presented an approach to generate a feature model out of a product description. The results from this approach equal to the information given by the folder names we are provided with, although Davril et al. lacks the hierarchy of those features. They achieve the hierarchy by mining associations. As we get the hierarchy out of the configuration files, we already have all information needed with less effort and higher probability of a correct result. However, this approach could be used in later work to handle naming conflicts or missing information.

The approach from La Rosa et al. [7] presents some techniques to configure a variable system based on questionnaire models and domain constraints. The main parts can be applied to feature models, although this was not the focus of their work. Therefore they only included and-groups and did not include or-groups and alternative-groups, simplifying it in a way. Nevertheless, they did present a technique to detect circular dependencies and contradictory constraints, which was not part of this work, but could be a focus for further improvements.

VI. Conclusion

In this work we propose a questionnaire-based approach to support the process of configuring an existing software product line to generate an actual product. To do so, we use feature models as formalism to structure the existing code artifacts. The feature model automatically gets filled with the information we get from analyzing the structure and conventions of the existing source code. The resulting feature model contains every code artifact as a feature and also includes the constraints and restrictions.

We also propose a questionnaire to support the configuration of a product from the given product line. We introduce definitions in XML, which allow for creation of questionnaires specifically designed for configuration. A questionnaire needs

to be designed carefully by the developers or experts on the domain of the product line. It allows a mapping between answers to simple questions and actual features of the product line. With this method, a user doesn't have to undergo the process of configuration in the sense of (de-)selecting a specific set of features. Instead, the user only has to answer questions according to its use case to do the configuration of his specific product. As the tool uses the underlying FeatureIDE, it also makes use of its functions, such as validity checks. This additionally supports the usability and correctness of the whole process.

For the exapmle of *Odoo*, we were able to extract a feature model from the code. We also designed an examplary questionnaire and ran it with a simulated test user. This test user had a certain use case, which we were able to apply to the questionnaire through just answering the presented questions. As a result we got a custom tailored version of *Odoo* for our test user.

REFERENCES

[1] P. Clemens and L. Northtop, *Software Product Lines - Practices and Patterns*. SEI: Addision-Wesley, 2002.

[2] S. Apel, D. Batory, C. Kästner, and G. Saake, *Feature-oriented software product lines: concepts and implementation*. Springer Science & Business Media, 2013.

[3] D. Batory, "Feature models, grammars, and propositional formulas," *H. Obbink and K. Pohl (Eds.): SPLC 2005, LNCS 3714, pp. 7 20*, 2005.

[4] M. Svahnberg, J. van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," *Software Practice & Experience*, vol. 35, 2005.

[5] Y. Vizel, G. Weissenbacher, and S. Malik, "Boolean satisfiability solvers and their applications in model checking," *Proceedings of the IEEE*, vol. 103, no. 11, pp. 2021–2035, Nov 2015.

[6] R. Schröter, S. Krieter, T. Thüm, F. Benduhn, and G. Saake, "Feature-model interfaces: The highway to compositional analyses of highly-configurable systems." ICSE, 2016.

[7] M. La Rosa, W. van der Aalst, M. Dumas, and A. Hofstede, "Questionnaire-based variability modeling for system configuration," *BPM Group, Queensland University of Technology, Australia*, 2008.

[8] B. Zhang, *Mining Complex Feature Correlations from Large Software Product Line Configurations*. Technische Universität Kaiserslautern, Fachbereich Informatik, 2013.

[9] G. Bécan, S. Ben Nasr, M. Acher, and B. Baudry, "Webfml: synthesizing feature models everywhere," in *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*. ACM, 2014, pp. 112–116.

[10] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 290–300.