Questionnaire based configuration of product-lines in FeatureIDE

Jens Wiemann, Otto-von-Guericke-Universität Magdeburg,
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. As an established way to simplify the process of product configuration out of software product-lines, feature models are used to describe the set of features and constraints contained in a given software product-line. This paper proposes a method for automatically generating feature models out of descriptive files and naming conventions. Furthermore existing methods of configuration are considered in this work and to develop an 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.**

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

I. Introduction

DEVELOPING software product lines can result in a great 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, automatically including its dependencies. The configuration is based on a Feature Model, that defines the available Features and its relations to one another.

Feature Models are essential structures for the feature-oriented development and later configuration of software product lines (SPL). In such a way that they give a complete and easily understandable overview of the given features and constraints of a product-line. There are projects being developed feature-oriented, but don't have a feature model yet. Implementing it on top of a given Project can result in a very complex task due to the amount of its features and the constraints between them. Additionally those informations are often already kept inside of readme- or configuration files, what predesignates it to be analyzed programmatically. This work aims at automatically generating it out of descriptive files and naming conventions, to simplify a big part of the feature model creation.

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 more difficult, as you can no longer explain all the features, but you still have to figure out if the customer needs them or not. This paper considers existing methods of configuration and 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.

FeatureIDE being the leading open source IDE for feature-oriented-software development, it provides us with all the functionality needed to programatically generate and work with Feature Models. The underlying Eclipse enables us to implement the feature extraction and the questionnaire as a plugin.

This work will demonstrate the named simplifications based on a real life Project. This software, an open source ERP System called "Odoo" [] is predestined to be developed feature-oriented due to a huge amount of features and complex constraints. The Project is currently being developed and structured feature oriented, although it doest'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 of the features anymore, but has to come up with a more effective way.

The paper is structured as follows. In section 2 we ... . Followed by section 3 where we ... . Then in section 4, ... Finally we conclude and give an outlook for future works.

II. Basics for simplification of the configuration process

In the scope of simplifying the configuration of a given product line some specific tools and techniques were used. This section will give an overview of what was used in this work to archive the simplification.

A. Feature Models

Feature models are multi-purpose structure for product lines. Amongst their benefits are:

- Can be visualized as a feature diagram, showing the possible features and their hierarchy
- Classification of features and their dependencies (alternative/or; optional/mandatory/abstract)
- Formal Representation of the whole product line as a feature diagram $\Rightarrow$ computationally processable
- Enables configuration and variant validation

Although feature models can be represented as a boolean formula in CNF or grammer (GUIDSL), it is typically represented in form of a feature diagram and constraints. The feature diagram is basically structured as a tree: There is a root node, an arbitrary number of levels of nodes and finally

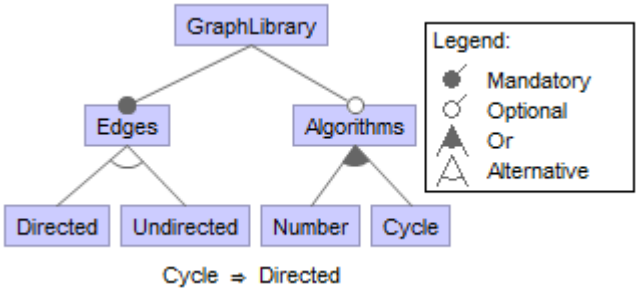

Fig. 1. A simple example of a feature model

leaf nodes without child nodes of their own. In that manner feature models map the hierarchy of features. Non-leaf features are often marked as abstract, as they do not have to be implemented. In addition features can be marked as either mandatory or optional. The possible relationships of multiple features with a mutual parent feature are *OR* (at least one feature has to be selected), *ALTERNATIVE* (exactly one feature has to be selected) and *AND* (any number of features can 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 boolean expression.

Figure 1 shows an example of a simple feature model diagram. The node labeled "GraphLibrary" is the root feature and represents the whole Project. It contains the nodes "Edges" that has to be selected doe to the mandatory marker and "Algorithms" that is marked optional. The implemented Edge types are "Directed" and "Undirected" from which exactly one has to be chosen. From the Algorithms at least one has to be chosen. The Constraint beneath the figure 1 implies that if the feature "Cycle" is selected, the Edge type is set to directed.

B. Product configuration

The whole product line is represented by it's feature model, which itself is a set of features with specific interrelations. The process of configuration of a product line describes the steps to derive a product from the product line. To archive this, one has to select a subset of all the possible features within the product line to meet one's requirements. But not all subsets of features result in an actual product. The interrelations of the features restrict the possible combinations of features resulting in a *valid* configuration. If only one of the requirements from the interrelations between the features is not met, no product can be created and the configuration as such is considered *invalid*.

During configuration a user selects or unselects features to his needs. This process requires a lot of domain knowledge on the one hand and detailed information about each single feature on the other. With growing numbers of possible features (and thus growing numbers of possible products) configuration requires more and more effort. The sheer amount of time needed to iterate over all the features grows to be significant. But the even greater problem arises from the possible interactions and interrelations of features, which one has to keep track of during configuration. The overhead of configuration might even outright negate the benefit gained from using a product line instead of multiple stand-alone products.

TODO: introduce partial configuration

C. Constraints, contradictions, SAT-solver

A feature-tree (from a feature model) can be expressed as a boolean statement. The mapping between the individual groups (sub-trees) and the corresponding boolean expression is shown in the folloing list:

TODO: Rules for

- Or
- Alternative
- And

These particular groups are joined via a boolean *AND*. For the example feature tree shown in figure 1 the corresponding boolean expression is as follows:

$$\begin{aligned} &GraphLibrary \ \wedge \ Edges \\ \wedge \ &((Directed \ \wedge \neg Undirected) \\ \vee &(\neg Directed \ \wedge \ Undirected)) \end{aligned} \tag{1}$$

As the feature model not only contains the feature tree, but also additional constraints, these constraints have to be considered in the boolean expression representing the complete feature model. They can be linked to one another via boolean *AND*s. The connection to the previously created boolean expression of the feature tree is also made through a logical *AND*. Thus, the complete resulting boolean expression for the feature model shown in figure 1 is:

$$\begin{aligned} &GraphLibrary \ \wedge \ Edges \\ \wedge \ &((Directed \ \wedge \neg Undirected) \\ \vee &(\neg Directed \ \wedge \ Undirected)) \\ \wedge \ &(Algorithms \ \Rightarrow \ (Number \ \vee \ Cycle)) \\ \wedge \ &(Cycle \ \Rightarrow \ Directed) \end{aligned} \tag{2}$$

This formalism allows a configuration to be checked for validity. To do so 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. Even during configuration this process can be applied to check for invalid partial configurations after each decision.

III. Workflow based on an existing software product line

[Diagram showing the general workflow]

unstructured product line → generated feature model → optimizing (via SAT-solver?) → domain knowledge of an expert ⇒ Questionnaire → Feature model + Questionnaire ⇒ Product (variant)

This work was build around the usage for Odoo[1], an open-source tool for enterprise-resource-planing written mainly in python[2]. In this chapter, we will explain the workflow shown in figure **??** on the basis of the experiences we made with Odoo.

Odoo itself is programmed as a product line. It has a lot of modular code artifacts in their specific subdirectories of the project. These code artifacts follow certain conventions to describe their integration in the context of the whole project.

Using these conventions we were able to extract a feature model in which the hierarchies and dependencies of the code artifacts are included. This allows for much better overview over the product line and it's capabilities. During our efforts to implement a automated generation of a feature model we encountered some problems for which we had to find solutions/workarounds.

TODO: State the problems:

- Error-Handling on failure to comply the conventions
- Missing Parent Features → Abstract features (can be configured beforehand)

The extracted feature model will also be used in the following step of configuration.

Configuration of product lines generally requires a good understanding of the given problems and the possible solutions to these problems as well as the specific implementations of these solutions. This domain knowledge is the most challenging part of configurations and restricts most of the possible users to do the configuration on their own. The same goes of course for Odoo and the various components contained in the project.

To empower possible users to configure Odoo on their own, we had the idea to use a questionnaire. In the progress of implementation, experts also design a questionnaire in such a way that a possible user has to answer a given amount of questions to perform the configuration of a product meeting his personal needs without him having to know all the details of the implementation of the product line or even the individual features. 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.

The two major steps of extraction of the feature model and the creation and usage of the questionnaire are described in detail on the following sections.

A. Extraction of a feature model

To automatically extract a feature model out of a given software project the structure of that project must be algorithmically processable. We had a thorough look at the source code of Odoo and found it to be structured according to the following rules:

- All features are stored within a specified directory
- Each feature is stored in a separate sub-directory
- A feature's parent features are stated in the name of the corresponding directory
  - The complete hierarchy is displayed, except for the root feature
  - Individual parent features are delimited with an unique symbol
- Each feature directory contains a descriptive file with a specified name containing:
  - A descriptive name of the feature
  - A description text for the feature
  - All dependencies including the parent features

Following these rules we were able to automatically extract the information needed to generate a feature model. For this procedure we implemented a plugin for FeatureIDE which processes the extracted information to a feature model. The procedure to generate a feature model is stated following.

The most obvious correspondence lies within the hierarchy. The parent names are parsed from the directory name and -if existing- looked up in the existing partial feature model from the previously parsed features. As the directories are parsed in alphabetical order and this way of ordering places e.g. "Feature1" in front of "Feature1_Feature", a parent feature will always be processed before it's child features.

After placing it in the correct position within the existing partial feature model each feature's descriptive file gets processed. Each feature gets enriched with it's descriptive details and it's dependencies. The list of dependencies firstly gets reduced by the parent features as the feature models hierarchy already implements this kind of dependencies. The remaining dependencies are stated as constraints. To shorten the list of constraints shared dependencies between multiple features are combined to a single constraint.

B. Questionnaire Approach

As stated above configuration is one of the most challenging tasks within the scope of software product lines. In the previous section the automatic extraction of a feature model out of existing source code was explained. The resulting feature model yields a much better overview over the possible features and their interrelationships. Although the formalism of a feature model allows for tool support to simplify the process of configuration, still a lot of domain knowledge is required to be able to find the right combination of features for a given use-case.

This work therefore introduces a method to allow experts to apply their knowledge and understanding to a whole product line during the development and thus enabling end users to draw on this knowledge whenever a configuration is taking place.

In this work we made the decision to use a questionnaire based approach. Depending on the implementation of the questionnaire during development a partial or event complete configuration can be archived by a user through just answering the questions of the questionnaire. The concrete 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 (un-)selection

[1] https://www.odoo.com/
[2] https://www.python.org/

of features so the user avoids the hassle of considering implementation-details of each feature.

This 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 the possible definitions of a questionnaire.

Each page of the questionnaire is defined independently. It always contains at least a Question and more then 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. This set of features defines which features are selected or specifically unselected in the case of that answer being chosen by the user.

Each answer can also have an indicator to define which page of the questionnaire is to 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.

In this work 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 the following code snippet:

```
<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>
```

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	

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.

IV. Examplary scenarios

FeatureIDE

Explain the used environment

State a few usage scenarios and show why the questionnaire is better or as good as the existing solutions in the given situations. Also point out in which scenario this might be the wrong tool.

V. Conclusion and Future Work

This is where the work is concluded. In this section there will be a description of the way we did things and the experiences we made during it. An emphasis will be on the insights and the findings from the scenarios will get outlined.

Here will be a summary of the new questions that were raised in this work. Also there will be topics for further research. Particularly the problems we encountered and couldn't solve with our concept and why will be pointed out and first approaches will be suggested.

References

[1] M. Antkiewicz, "Featureplugin: Feature modeling plug-in for eclipse," *OOPSLA04 Eclipse Technology eXchange (ETX) Workshop, Oct. 24-28, Vancouver*, 2004.

[2] M. La Rosa, "Questionnaire-driven configuration of reference process models," *BPM Group, Queensland University of Technology, Australia*, 2006.

[3] M. La Rosa, "Questionnaire-based variability modeling for system configuration," *BPM Group, Queensland University of Technology, Australia*, 2008.

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