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. 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.

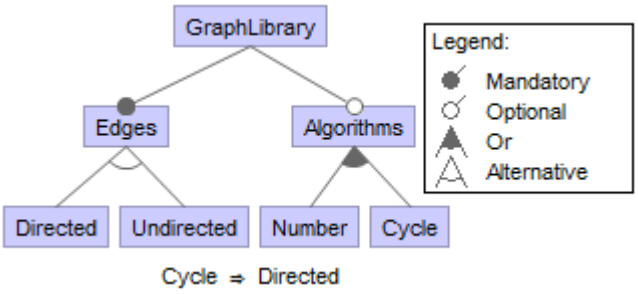

Fig. 1. A simple example of a feature model

Problem Statement

Very high complexity of configuration due to many features and constraints. Domain knowledge is highly required to understand the given software product-line and being able to combine it's features to a valid configuration which satisfies the users needs.

Contribution

Simplifying the process of configuration of product lines through extracting a feature model out of naming conventions and configuration files from an existing product line in FeatureIDE and applying a configuration wizard which guides the user though the creation of a specific product (variant) in the style of a questionnaire, thus applying the domain knowledge of an expert.

II. Basics for simplification of variant configuration

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 tool for product lines. Figure 1 shows a simplified example of a feature model. Amongst their benefits are:

- Visualization of the possible features and their hierarchy
- Classification of features and their dependencies (alternative/or; optional/mandatory/abstract)
- Formal Representation of the whole product line ⇒ computationally processable

- Assistance/foundation for configuration and variant validation

They are basically structured like trees: There is a root node, an arbitrary number of levels of nodes and finally leaf nodes without child nodes of their own. In that manner feature models map the hierarchy of features. In addition they mark each feature as either mandatory or optional as well as either abstract or concrete. 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 are boolean expressions, the example in figure 1 shows an implication.

B. Product configuration

Through configuration a concrete product can be derived from a product line. Each valid configuration represents a specific *variant* of the possible products.

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 variants) configuration get more and more complex and turns out to be not trivial.

One of the purposes of the feature-oriented approach is saving the effort of creating a whole new product and deriving that product from a product line instead. The sheer overhead of configuration might even negate that benefit if configuration grows too complex.

C. Constraints, contradictions, SAT-solver

As stated above a feature model may contain constraints in the form of boolean expressions. Also, the feature-tree can be expressed as a boolean statement. The model shown in figure 1 can be expressed as follows:

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

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 unstructured 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)

Figure **??** shows the general process aimed to archive within this work. At the beginning there is only a given product line with the 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 a feature model gets extracted in which the hierarchies and dependencies of the code artifacts are included. This allows for much better overview or automated optimization, for example scanning for dead features via SAT-solver. 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.

To transfer the domain knowledge of experts to the product line a questionnaire is introduced. In the progress of implementing the product line 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 variant meeting his personal needs without him having to know all the details of the implementation of the product line or even the individual features. The two major steps of extraction 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 understandable. To archive this, conventions are used for the project of which an overview follows:

- All features are stored within a specified directory
- Each feature is stored in a separate 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 configuration file with a specified name containing:
  - A descriptive name of the feature
  - A description text for the feature
  - All dependencies including the parent features

All of these information are included in the resulting feature model. 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 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 configuration 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.

TODO: State the problems:

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

B. Questionnaire Approach

Use questions to guide the user through a conditional configuration process. The questions, their order and the influences of the answers are pre-defined by a developer/an expert who uses his domain knowledge to design the questionnaire. Therefore, a data structure is introduced and mapped to a specific set of XML-tags.

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.