WebFML: Synthesizing Feature Models Everywhere

Guillaume Bécan, Sana Ben Nasr, Mathieu Acher, Benoit Baudry

WebFML: Synthesizing Feature Models Everywhere

Guillaume Bécan, Sana Ben Nasr, Mathieu Acher and Benoit Baudry
Inria / IRISA, University of Rennes 1, France
firstname.lastname@inria.fr

ABSTRACT

Feature Models (FMs) are the de-facto standard for documenting, model checking, and reasoning about the configurations of a software system. This paper introduces WebFML a comprehensive environment for synthesizing FMs from various kinds of artefacts (e.g. propositional formula, dependency graph, FMs or product comparison matrices). A key feature of WebFML is an interactive support (through ranking lists, clusters, and logical heuristics) for choosing a sound and meaningful hierarchy. WebFML opens avenues for numerous practical applications (e.g., merging multiple product lines, slicing a configuration process, reverse engineering configurable systems).

Categories and Subject Descriptors

D.2.13 [**Software Engineering**]: Reusable Software; D.2.9 [**Software Engineering**]: Management—*Software configuration management*

General Terms

Design, Management

Keywords

Ontologic-Aware Synthesis, Feature Modeling Environment, Reverse Engineering Feature Models

1. INTRODUCTION

Feature Models (FMs) are by far the most popular notation for representing and reasoning about common and variable properties (features) of a system [9, 13]. FMs offer a simple yet expressive way to define a set of legal *configurations* (i.e., combinations of features) [4, 15, 31, 38]. FMs also define an *ontological semantics* [15]: a tree-like hierarchy and feature groups organize features into multiple levels of increasing detail.

Both configuration and ontological aspects of FMs are important. First, the proper handling of the configuration set is unquestionable: it should not be too large (otherwise some unsafe compositions of the features are allowed) or too narrow (otherwise it translates as a lack of flexibility in the software project). Second, a doubtful feature hierarchy may pose severe problems for a further exploitation by automated transformation tools or by stakeholders that need to understand, maintain and exploit an FM. For instance, the FM fm_u of Figure 1a characterizes the same set of configurations than the FM fm_g of Figure 1b, but with an inappropriate ontological semantics (e.g. Java is located below Storage rather than being child feature of Programming Language).

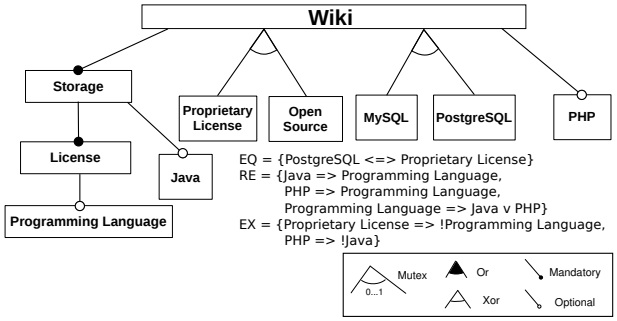
As an FM should both conform to a configuration semantics and exhibit an appropriate hierarchy, a manual elaboration or maintenance of an FM is usually time-consuming and error-prone – even for a small number of features and dependencies [6, 8, 10, 36].

WebFML provides automated techniques for speeding up and supervising the building process of FMs. WebFML also aims to ease the mechanical translation of artefacts with variability into FMs. It enables practitioners to reduce their effort and avoid accidental complexity when (re)-engineering or maintaining their variability models. In a nutshell, WebFML is a comprehensive environment for synthesizing FMs from various kinds of artefacts:

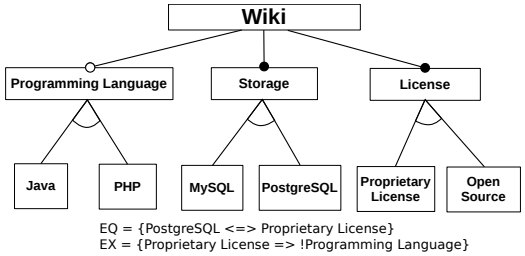
- **product comparison matrices**: a specific form of spreadsheets documenting the features of products under comparison. By giving a specific interpretation of their semantics, these matrices can be interpreted as FMs [34].
- **dependency graphs** which can be extracted from configuration files of build systems (e.g. *pom.xml* files in Maven) [1, 3].
- **compilation directives and source code** in which features and their dependencies can be mined [28, 36];
- **propositional formulas** in conjunctive or disjunctive normal form [6, 8, 35]
- **a (set of) FM(s)**: slicing, merging or refactoring existing FMs are instances of the synthesis problem we are addressing with WebFML [5, 6, 23].

In addition, some outputs of mining tools (e.g., for extracting features and their constraints from source code [19, 28, 33, 36], requirements [29, 40] or product descriptions [11, 18, 21]) are amenable to an input format (e.g., formula) of WebFML.

An unique feature of WebFML is its interactive support for choosing a *sound* and *meaningful* hierarchy. Ranking lists and clusters guide users in the choice of a relevant hierarchy among the thousands of possibilities. For a given feature, siblings or parent candidates are presented to users and sorted according to the syntactic or semantic similarities between feature names. We develop generic, state-of-the-art heuristics [10] (e.g., based on Wordnet or Wikipedia) applicable without prior knowledge to the artefacts exposed above. WebFML also offers extensible mechanisms (plugins) to integrate specialized heuristics. Despite the availability



(a) A FM fm_u with a doubtful ontological semantics



(b) A FM fm_g with a more appropriate ontological semantics

Figure 1: For a same set of configurations, two possible yet different FMs

of a wide range of academic or industrial feature modeling tools [12, 25, 32, 39], we are unaware of an environment that considers both configuration and ontological aspects when synthesizing an FM.

Another strength of WebFML is that numerous operations have been developed on top the synthesis support. Likewise practitioners can envision the use of WebFML in practical scenarios: merging of multiple product lines [5]; slicing of a configuration process into different steps or tasks [23]; sound refactoring of FMs [6], especially when fully automated techniques produce incorrect FMs; reverse engineering of configurable systems through the combined use of composition, decomposition and refactoring operators [1, 3].

2. BACKGROUND AND MOTIVATION

The formalism of FMs offers syntactic constructions to attach variability information to features organized in the hierarchy. When decomposing a feature into subfeatures, the subfeatures may be optional, mandatory or may form Mutex, Xor, or Or groups.

The variability restricts the possible combinations of features (i.e., configurations): features Java and PHP cannot be both selected since the two features belong to a Xor-group in Figure 1b; every configuration of Figure 1b necessarily includes the two mandatory features Storage, License and the root Wiki, etc. Importantly, the feature hierarchy also contributes to the definition of the configuration semantics: all features, except the root, logically imply their parent.

The FM *synthesis problem* [6, 8] is at the heart of many *reverse engineering* procedures (see, e.g. [1, 3, 18, 22, 36]). FM management operations are also impacted by the problem [8, 35]. It can be formulated as follows: given a set of features' names and Boolean dependencies among features typically encoded as a propositional formula ϕ in conjunctive or disjunctive normal form, the problem is to synthesize an FM with a sound configuration semantics. We also expect the FM to be maximal, i.e. as much information as possible is represented in the diagrammatic part of the FM (see a formalization in [8]).

WebFML tackles a generalization of the FM synthesis problem by providing an interactive support in order to synthesize maximal FMs both *i)* conformant to the configuration semantics and *ii)* exhibiting an appropriate ontological semantics. WebFML also targets different kinds of inputs amenable to the synthesis problem.

3. ENVIRONMENT

3.1 Features of the environment

Our procedure is based on the idea that a feature tends to share the same context as its parent and siblings. Thus, we rely on a series of heuristics to *i)* rank parent candidates of each feature according to their semantic similarity and *ii)* compute clusters of conceptually similar features. These types of information can be interactively complemented or refined by a user, and if needed, an optimum branching algorithm can synthesize a complete FM. Specifically, our tool offers an interactive mode where the user can import a formula (e.g., in DIMACS format), synthesizes a complete FM and export the result in different formats. During the FM synthesis, the graphical user interface displays a ranking list of parent candidates for every feature, a list of clusters, a list of cliques (features that are always present together) and a graphical preview of the FM under construction (see Figure 2, Ⓔ, Ⓕ, Ⓖ and Ⓗ). In addition, WebFML is built on top of FAMILIAR [4] which provides numerous reasoning and scripting capabilities for FMs. Overall, the features available are the following:

- select or ignore a parent candidate in the ranking lists (see Figure 2, Ⓕ);
- select a parent for a cluster (or a clique) within the cluster's features or any potential parent feature outside the cluster (see Figure 2, Ⓖ and Ⓗ). The user can also consider a subset of a cluster when selecting the parent;
- undo a previous choice (see Figure 2, Ⓒ);
- define the different heuristics and parameters of the synthesis (see Figure 2, Ⓑ);
- automatically generate a complete FM according to previous choices and selected heuristics (see Figure 2, Ⓒ);
- use FAMILIAR capabilities within an integrated console (see Figure 2, Ⓘ);
- manage FAMILIAR script files and previously computed variables (see Figure 2, Ⓐ and Ⓓ).

A typical usage is to perform some choices, generate a complete FM with the heuristics and the optimum branching algorithm and reiterate until having a satisfactory model.

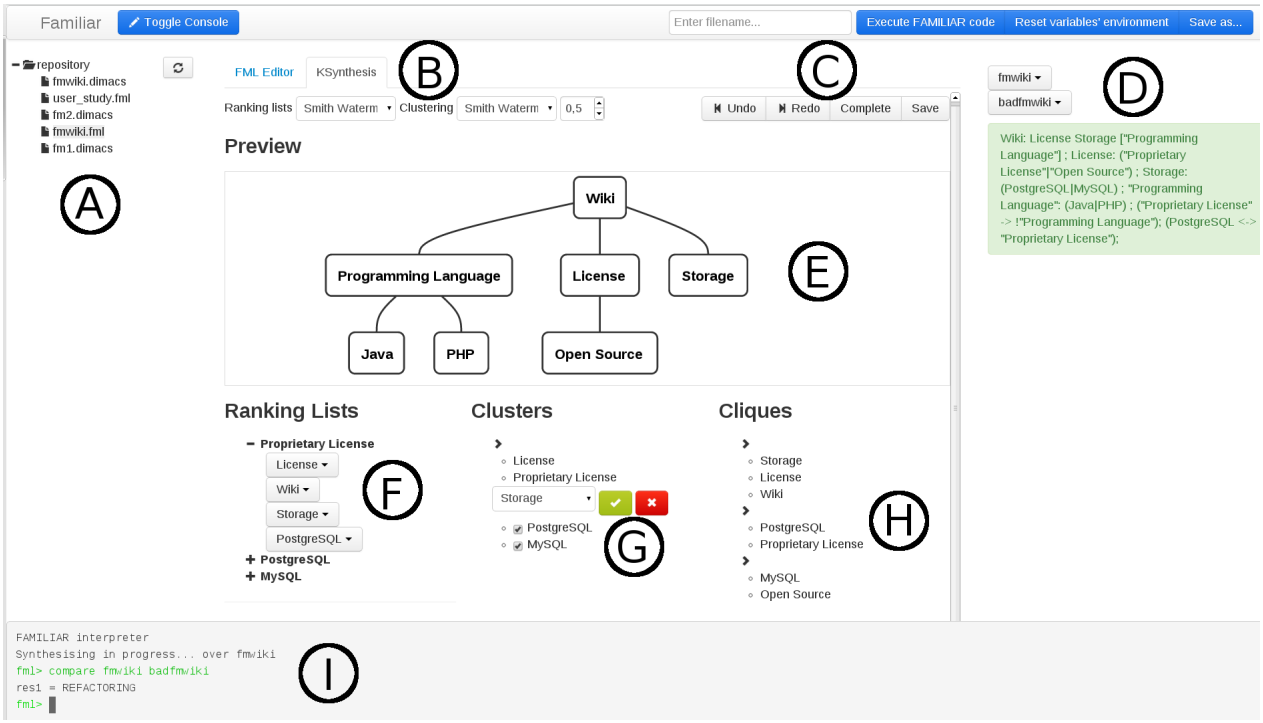


Figure 2: Interface of the environment during synthesis

3.2 Interactive edits of parent candidates and clusters

The ranking lists of parent candidates and the clusters form the main information during the synthesis. As the amount of information can be overwhelming, we propose a tree explorer view for manipulating and visualizing both parent candidates and clusters (see Figure 2, (F), (G) and (H)). A tree explorer view is scalable: it allows to focus on specific features or clusters while the other information in the explorer can be hidden. During the synthesis, the user interacts almost exclusively by clicking on the elements of these explorers. Clicking on a feature in parent candidate lists allows to select or ignore a parent candidate (see Figure 3a). Users can also click on a cluster to choose a parent for all the cluster's features among their common parent candidates (see Figure 3b). If the cluster contains the desired parent, the user can also click on this feature and select the children within the cluster. In both cases, the user can deselect some features to consider only a subset of the cluster. The same behaviour applies to cliques (see Figure 3c). Finally, we propose a *graphical preview* of the FM based on Dagre[1] and D3[2] javascript libraries. It allows to summarize the previous choices and have a global and familiar view of the current result (see Figure 2, (E)).

3.3 Implementation of heuristics

Hierarchical clustering is performed by the Hac[3] library. The optimum branching algorithm for computing a complete FM is based on Tarjan's algorithm [37]. We developed six heuristics based on specialized libraries. *SmithWaterman* and *Levenshtein* compute syntactical similarity based on words' morphology. They come from the Simmetrics[4] library. *PathLength* and *Wu&Palmer* heuristics rely on extJWNL[5] which handles the communication between the lexical database WordNet [26] and our tool. Wikipedia Miner[6] offers an API to browse Wikipedia's articles offline and compute their relatedness [27]. We used this tool on the english version of Wikipedia and Wiktionary which form the last two heuristics.

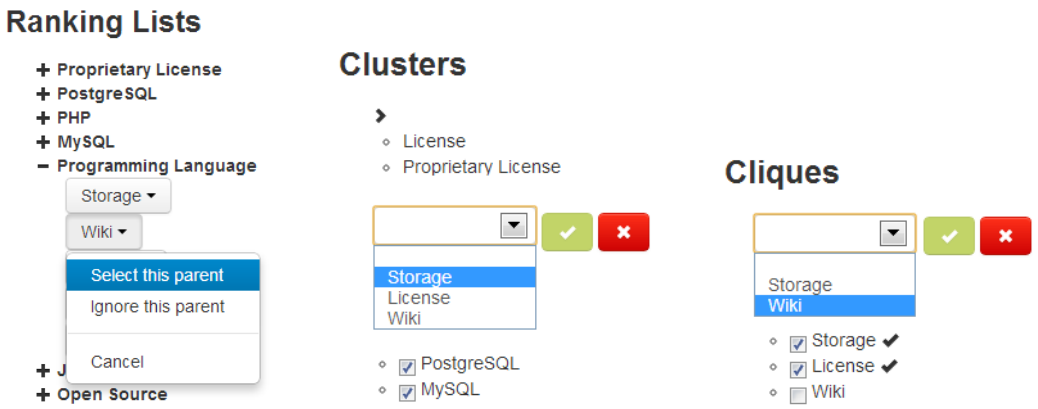
4. RELATED WORK

There are numerous existing academic or industrial tools for specifying and reasoning about FMs. *FeatureIDE* [38,39] is an Eclipse-based IDE that supports all phases of feature-oriented software development for the development of product lines (domain analysis, domain implementation, requirements analysis and software generation). *FAMA* [12] is a framework for the automated analysis of FMs integrating some of the most commonly used logic representations and solvers. Our environment is compatible with most of these tools. However none of the existing tools propose support for synthesizing, merging, slicing, or refactoring FMs.

Techniques for synthesising an FM from a set of dependencies (e.g., encoded as a propositional formula) or from a set of configurations (e.g., encoded in a product comparison matrix) have been proposed [2, 8, 16, 17, 22, 24, 36, 41]. An important limitation of prior works is that they neglected ontological aspects of an FM, i.e., the user support is either absent or limited for identifying an appropriate feature hierarchy. Without adequate guidance for users, the resulting hierarchy is likely to be far from an ideal result and/or the synthesis process can be daunting and time-consuming.

She *et al.* proposed specific heuristics based on textual feature descriptions for the operating system domain [36].

[1] `https://github.com/cpettitt/dagre`
[2] `http://d3js.org/`
[3] `http://sape.inf.usi.ch/hac`
[4] `http://sourceforge.net/projects/simmetrics`
[5] `http://extjwnl.sourceforge.net`
[6] `http://sourceforge.net/projects/wikipedia-miner`



(a) Select or ignore a parent candidate

(b) Select a parent for a cluster

(c) Select a parent for a clique

Figure 3: User interaction during FM synthesis

Davril et al. [18] presented a fully automated approach, based on prior work [21], for constructing FMs from publicly available product descriptions found in online product repositories and marketing websites such as SoftPedia and CNET. The proposal is evaluated in the anti-virus domain. *Our techniques operate over a predefined set of features and dependencies. We do not assume any additional inputs for selecting the feature hierarchy.*

Weston *et al.* [40] provided a tool framework *ArborCraft* which automatically processes natural-language requirements documents into a candidate FM, which can be refined by the requirements engineer. Alves *et al.* [7], Niu *et al.* [30], and Chen *et al.* [14] applied information retrieval techniques to abstract requirements from existing specifications, typically expressed in natural language. These works do not consider precise logical dependencies and solely focus on ontological semantics. As a result users have to manually set the variability information. Moreover, a risk is to build an FM in contradiction with the actual dependencies of a system. *As a summary, our environment addresses both configuration and ontological aspects of an FM.*

5. CONCLUSION

We presented WebFML an environment with an interactive support for synthesizing feature models from various kinds of artefacts (e.g. propositional formula, dependency graph, FMs or product comparison matrices). A key feature of WebFML is to address both configuration and ontological aspects of feature models. Ranking lists, clusters and logical heuristics guide users throughout the interactive process to select a sound and meaningful hierarchy. WebFML not only open avenues for reverse engineering scenarios; slicing of a configuration process, merging of multiple product lines, or refactoring of an existing system are also made possible.

Our recent evaluation of the synthesis techniques showed that the interactive support of WebFML is crucial, otherwise the resulting hierarchy of the FM is far from the expectation. We also demonstrated that WebFML provides state-of-the-art user support – either for fully synthesizing FMs or for assisting users through ranking lists and clusters [10].

Interestingly enough, a distributed effort involving several collaborators can even be considered to synthesize very large feature models. We believe the publicly available provision of the environment (see `http://tinyurl.com/WebFMLDemo`) is an important step towards effective reverse engineering or maintenance of highly configurable systems more and more reported in the industry or in the open source community.

Acknowledgements

This work is partially supported by the French BGLE Project CONNEXION.

6. REFERENCES

[1] E. K. Abbasi, M. Acher, P. Heymans, and A. Cleve. Reverse engineering web configurators. In *CSMR/WRCE'14*, 2014.

[2] M. Acher, B. Baudry, P. Heymans, A. Cleve, and J.-L. Hainaut. Support for reverse engineering and maintaining feature models. In Gnesi et al. [20], page 20.

[3] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. Extraction and evolution of architectural variability models in plugin-based systems. *Software and Systems Modeling (SoSyM)*, 2013.

[4] M. Acher, P. Collet, P. Lahire, and R. B. France. Familiar: A domain-specific language for large scale management of feature models. *Sci. Comput. Program.*, 78(6):657–681, 2013.

[5] M. Acher, B. Combemale, P. Collet, O. Barais, P. Lahire, and R. B. France. Composing your compositions of variability models. In *MoDELS'13*, pages 352–369, 2013.

[6] M. Acher, P. Heymans, A. Cleve, J.-L. Hainaut, and B. Baudry. Support for reverse engineering and maintaining feature models. In *VaMoS'13*. ACM, 2013.

[7] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. An exploratory study of information retrieval techniques in domain analysis. In *SPLC*, pages 67–76. IEEE Computer Society, 2008.

[8] N. Andersen, K. Czarnecki, S. She, and A. Wasowski. Efficient synthesis of feature models. In *SPLC'12*, pages 97–106, 2012.

[9] S. Apel and C. Kästner. An overview of feature-oriented software development. *Journal of Object Technology (JOT)*, 8(5):49–84, July/August 2009.

[10] G. Bécan, M. Acher, B. Baudry, and S. Ben Nasr. Breathing Ontological Knowledge Into Feature Model Management. Rapport Technique RT-0441, INRIA, Oct. 2013.

[11] G. Bécan, N. Sannier, M. Acher, O. Barais, A. Blouin, and B. Baudry. Automating the formalization of product comparison matrices. In *ASE'14*, 2014.

[12] D. Benavides, S. Segura, and A. Ruiz-Cortes. Automated analysis of feature models 20 years later: a literature review. *Information Systems*, 35(6), 2010.

[13] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wasowski. A survey of variability modeling in industrial practice. In *VaMoS'13*. ACM, 2013.

[14] K. Chen, W. Zhang, H. Zhao, and H. Mei. An approach to constructing feature models based on requirements clustering. In *RE'05*, pages 31–40, 2005.

[15] K. Czarnecki, C. H. P. Kim, and K. T. Kalleberg. Feature models are views on ontologies. In *SPLC '06*, pages 41–51. IEEE, 2006.

[16] K. Czarnecki, S. She, and A. Wasowski. Sample spaces and feature models: There and back again. In *SPLC'08*, pages 22–31, 2008.

[17] K. Czarnecki and A. Wasowski. Feature diagrams and logics: There and back again. In *SPLC'07*, pages 23–34. IEEE, 2007.

[18] 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 *ESEC/FSE'13*, 2013.

[19] C. Dietrich, R. Tartler, W. Schröder-Preikschat, and D. Lohmann. A robust approach for variability extraction from the linux build system. In E. S. de Almeida, C. Schwanninger, and D. Benavides, editors, *SPLC (1)*, pages 21–30. ACM, 2012.

[20] S. Gnesi, P. Collet, and K. Schmid, editors. *The Seventh International Workshop on Variability Modelling of Software-intensive Systems, VaMoS '13, Pisa , Italy, January 23 - 25, 2013*. ACM, 2013.

[21] N. Hariri, C. Castro-Herrera, M. Mirakhorli, J. Cleland-Huang, and B. Mobasher. Supporting domain analysis through mining and recommending features from online product listings. *IEEE Transactions on Software Engineering*, 99(PrePrints):1, 2013.

[22] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. On extracting feature models from sets of valid feature combinations. In *FASE'13*, pages 53–67, 2013.

[23] A. Hubaux, M. Acher, T. T. Tun, P. Heymans, P. Collet, and P. Lahire. *Domain Engineering: Product Lines, Conceptual Models, and Languages*, chapter Separating Concerns in Feature Models: Retrospective and Multi-View Support. Springer, 2013.

[24] M. Janota, V. Kuzina, and A. Wasowski. Model construction with external constraints: An interactive journey from semantics to syntax. In *MODELS'08*, volume 5301 of *LNCS*, pages 431–445, 2008.

[25] C. W. Krueger. Biglever software Gears and the 3-tiered spl methodology. In *OOPSLA'07*, pages 844–845. ACM, 2007.

[26] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[27] D. N. Milne and I. H. Witten. An open-source toolkit for mining wikipedia. *Artif. Intell.*, 194:222–239, 2013.

[28] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki. Mining configuration constraints: Static analyses and empirical results. In *Proceedings of the 36th International Conference on Software Engineering (ICSE)*, 6 2014.

[29] N. Niu and S. M. Easterbrook. Extracting and modeling product line functional requirements. *RE*, 8:155–164, 2008.

[30] N. Niu and S. M. Easterbrook. Concept analysis for product line requirements. In K. J. Sullivan, A. Moreira, C. Schwanninger, and J. Gray, editors, *AOSD*, pages 137–148. ACM, 2009.

[31] R. Pohl, K. Lauenroth, and K. Pohl. A performance comparison of contemporary algorithmic approaches for automated analysis operations on feature models. In *ASE'11*, pages 313–322, 2011.

[32] pure::variants. `http://www.pure-systems.com/pure_variants.49.0.html`.

[33] A. Rabkin and R. Katz. Static extraction of program configuration options. In *ICSE'11*, pages 131–140. ACM, 2011.

[34] N. Sannier, M. Acher, and B. Baudry. From comparison matrix to variability model: The wikipedia case study. In *ASE*, pages 580–585, 2013.

[35] S. She. *Feature Model Synthesis*. PhD thesis, University of Waterloo, 2013.

[36] S. She, R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki. Reverse engineering feature models. In *ICSE'11*, pages 461–470. ACM, 2011.

[37] R. E. Tarjan. Finding optimum branchings. *Networks*, 7(1):25–35, 1977.

[38] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to feature models. In *ICSE'09*, pages 254–264. ACM, 2009.

[39] T. Thüm, C. Kstner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 2012.

[40] N. Weston, R. Chitchyan, and A. Rashid. A framework for constructing semantically composable feature models from natural language requirements. In *SPLC'09*, pages 211–220. ACM, 2009.

[41] B. Zhang and M. Becker. Mining complex feature correlations from software product line configurations. In Gnesi et al. [20], page 19.