Special Issue on Software Product Lines
December 2006, *Communications of the ACM*

Automated Analyses of Feature Models: Challenges Ahead

Don Batory
Dept. Computer Sciences
University of Texas at Austin
Austin, Texas, U.S.A.
batory@cs.utexas.edu

David Benavides and Antonio Ruiz-Cortés
Dpto. de Lenguajes y Sistemas Informáticos
University of Seville
Av. de la Reina Mercedes S/N, 41012 Seville, Spain
{benavides,aruiz}@tdg.lsi.us.es

1 Introduction

A *feature* is an increment in product functionality. Features are commonly used to specify and distinguish products in product lines. They communicate product functions in an easy-to-understand way, they capture functionalities concisely, and help delineate the commonalities and variabilities of a domain.

Features can have attributes (much like GUI components can be customized by property lists), where the values of certain attributes are computed from the properties of other features (e.g., the cost of a product is the sum of the costs of its constituent features). Also features often have constraints on their usage: the selection of one feature may preclude or require the selection of others.

Current tool support for feature models is ad hoc, offering little or no support for debugging feature models or optimizing feature selections. Recent work shows how feature models can be reduced to propositional formulas or to constraint satisfaction problems, for which off-the-shelf tools can validate properties of models (e.g., confirming that a given set of features are incompatible or compatible) or to optimize the selection of features (e.g., performance) [1][2][4]. This opens up new possibilities for next-generation tools for specifying products in software product lines.

Of course, feature models are a front-end to a back-end synthesis technology that takes the output of a feature model (i.e., a program specification) and converts it into the program itself. There are many technologies for doing this, and reviewing them is beyond the scope of this paper. The contribution of this paper is to alert readers to recent advances in formalizing feature models and to the challenges ahead in automating product specification and design.

2 Open Issues and a Research Agenda

Model Consistency. The automotive industry has feature models with up to ten thousand features. It is well-known that these models are riddled with inconsistencies that are hard to detect. As an elementary illustration, suppose a feature model requires that (1) if feature `A` is used then `B` must also be used, and (2) if feature `B` is used, feature `A` cannot be used. This is expressed by the formula:

```
(A implies B) and (B implies not A)
```

Clearly there is an inconsistency: if `A` is true, we can conclude `A` is false. Such inconsistencies are rarely this simple in practice. The way they are discovered today is by accidentally stumbling over them: the right set of features must be selected to expose the error. Unfortunately, the number of subformulas to examine is $O(2^n)$, where n is the number of variables in a formula. Are there automated ways to find model inconsistencies?

Explanations. Features can be automatically deselected by numerical constraints (e.g., performance). It is possible for users to specify constraints that are unsatisfiable (e.g., the memory requirements of a program cannot exceed `x` and the program must have feature `Y`, where `memoryRequirements(Y)>x`). Explaining why there is no product for a given set of constraints, and perhaps more importantly, how the situation can be rectified is key. Finding a minimal number of violated constraints, which is vital to understandable explanations, is a difficult problem. Model diagnosis research may be relevant [10].

Model Driven Development (MDD). Mapping feature selections in a feature model into other development artifacts (requirements, architecture, code modules, test cases, documentation, etc.) is fundamental to MDD. As an example of model and code integration, suppose the implementation of feature `F` makes a reference to a variable or method that is part of feature `G`. This means that if `F` is selected, then `G` must also be selected. It should not be possible to specify a product `P` where `F` is selected and `G` is not. That a feature model satisfies this constraint can be verified by a SAT solver. More generally, verifying that other program representations are consistent with their feature model is a significant research challenge [6][3][11].

Artificial Intelligence (AI) Configurators. Consider a product line of aircraft carriers. Each carrier may contain several different kinds of aircraft (short-range and long-range fighters), and each plane may itself be a member of a product line. The planes on a carrier impose constraints on

the carrier's design. A web of customizable objects would be needed to describe a carrier (or other complex products) [5]. Feature models must be generalized to describe these "mega" products, and so too must tools that analyze and visualize these models. *AI configurators*, tools that configure constellations of objects, may be important for the analysis task [8][1].

Performance Scalability. How well do SAT solvers, BDD tools, CSP solvers, and AI configurators perform with large models? (We can even imagine description logic-based reasoners being used to analyze feature models). Even though there has been an enormous increase in computing power in the last decade, the problems of feature combinatorics remain NP-hard and can take a long time to solve. Not all tools and approaches will perform equally well. Which tools should be used and when? Can the choice of which tools to use be made automatically to minimize the time to analyze feature models? Will it be necessary to integrate different solvers?

3 Conclusions

Validating and analyzing product specifications will have significant practical payoffs. The benefits are tools that propagate constraints (so that incorrect specifications can be automatically detected), that provide explanations when design dead-ends are reached (and how to fix such designs), and that automatically optimize configurations for specific needs (to simplify program designs). Exposing the theory that underlines feature models is central to this goal. Answering these challenges will require close cooperation between product-line engineers and researchers.

Acknowledgments. This work was support in part by NSF's Science of Design Project #CCF-0438786 and the Spanish Science and Education Ministry (MEC) under contract TIC2003-02737-C02-01 (AgilWeb). We thank K. Pohl, P. Clements, J. McGregor, K. Czarnecki, O. Diaz, T. Männistö, V. Lifschitz and D. Beuche for their helpful comments on an earlier draft of this paper.

4 References

[1] T. Asikainen, T. Männistö, and T. Soininen. "Using a Configurator for Modelling and Configuring Software Product Lines based on Feature Models". *Workshop on Software Variability Management for Product Derivation*, *Software Product Line Conference (SPLC3)*, 2004.

[2] D. Batory, "Feature Models, Grammars, and Propositional Formulas", *Software Product Line Conference*, 2005.

[3] D. Batory and S. Thaker, "Towards Safe Composition of Product-Lines", Dept. Computer Sciences, University of Texas, TR-06-33, 2006.

[4] D. Benavides, P. Trinidad, and A. Ruiz-Cortes, "Automated Reasoning on Feature Models", *Conference on Advanced Information Systems Engineering (CAISE)*, July 2005.

[5] K. Czarnecki and C.H. Peter Kim, "Cardinality-Based Feature Modeling and Constraints: A Progress Report", *OOPSLA Workshop on Software Factories*, 2005.

[6] K. Czarnecki and K. Pietroszek. "Verifying Feature-Based Model Templates Against Well-Formed OCL Constraints", *Generative Programming and Component Engineering*, 2006.

[7] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. "Feature-Oriented Domain Analysis (FODA) Feasibility Study". Technical Report, CMU/SEI-90TR-21, Nov. 1990.

[8] S. Mittal and F. Frayman, "Towards a Generic Model of Configuration Tasks", 11th *Int. Conference on Artificial Intelligence*, 1989, 1391-1401.

[9] S. Neema, J. Sztipanovits, and G. Karsai, "Constraint-Based Design Space Exploration and Model Synthesis", *EMSOFT 2003*, LNCS 2855, p. 290- 305.

[10] R. Reiter, "A Theory of Diagnosis from First Principles", *Artificial Intelligence* 32, Vol. 1, 1987, 57-96.

[11] K. Pohl, G. Bockle, F v.d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*, Springer 2005.