Reasoning Algorithms on Feature Modeling

Subjects: Computer Science, Software Engineering

Contributor: Samuel Sepulveda, Ania Cravero

Software product lines (SPLs) have reached a considerable level of adoption in the software industry. The most commonly used models for managing the variability of SPLs are feature models (FMs). The analysis of FMs is an error-prone, tedious task, and it is not feasible to accomplish this task manually with large-scale FMs.

Keywords: reasoning algorithms ; automated analysis ; feature modeling ; software product lines

1. Introduction

Today software customers are demanding new and better products and services, which has forced the software industry to devise new approaches that increase the productivity of their processes and the quality of their products. For some time now, researchers in the field of *software engineering* have studied various alternatives for software development; among these is the software product lines (SPLs) approach. There are several differences between traditional single-system development and SPLs. The main difference is a paradigm change from individual software systems to a product line (also known as a family product) approach. According to [1], adopting this new paradigm implies a change in strategy: from the ad hoc next-contract vision to a strategic view of a field of business.

SPLs are defined as a set of characteristics to satisfy the specific needs of a particular market segment [2]. The use of SPLs as a software development methodology provides a set of benefits, including a reduction in development times and increases in productivity, among others [2][3]. Furthermore, Van der Linden et al. argue that these improvements significantly affect the development process, particularly in relation to costs and time to market, but it is at the level of software reuse that it is possible to achieve unprecedented levels of reuse [1].

SPL development is based on a common set of fundamental elements: an architecture, a collection of software components, and a set of products [4]. One of the key concepts is variability, which provides SPLs with the flexibility required for product diversification and differentiation [5]. Variability refers to combining the different functionalities that each component gives to the LPS, and this can be represented graphically using variability models. Nowadays, there are various methods of representing the variability in an LPS; however, feature models (FMs) are the most widely used method [6].

SPLs and variability are currently a fully active research area, showing their validity and relevance in the software engineering community. This relevance can be seen in a series of tertiary studies, i.e., studies that identify how variability is modeled [7]. Additionally, here can see how SPL engineering and variability management has been applied along with the Internet of Things [8].

Building and maintaining an FM is considered an expensive and error-prone task [9][10]. Moreover, the evolution and changes in the FM can introduce redundancy into the models, leading to the information being modeled in a contradictory way, resulting in modeling errors [11][12].

Throughout the SPL framework, starting from the creation of the FMs, the validation of the SPLs, the derivation of products, and even the modification or extension of the product family, it is essential to consult the FMs to obtain relevant information on the processes mentioned above. However, providing answers to these queries is not trivial because, given the structure of FMs, this process requires algorithms that support a set of rules and constraints that tend to be more complex depending on the model’s size. This process of securing information is known as the *automated analysis* of *feature models*(AAFM), and it has been identified as one of the most critical areas in the SPL community [13]. According to [14], it is possible to propose ad hoc algorithms to perform AAFM.

2. Software Product Lines

SPLs are defined as a set of similar software products created from reusable artifacts in the context of a specific application domain [15]. SPLs are developed in two stages: *domain engineering* and *application engineering* [16]. In the *domain engineering* stage, the common and variant elements are described. The *application engineering* stage is where the individual products of the SPL are built by reusing domain devices and exploiting the variability of the SPLs. **Figure 1** shows the SPL framework, including both stages and their interactions.

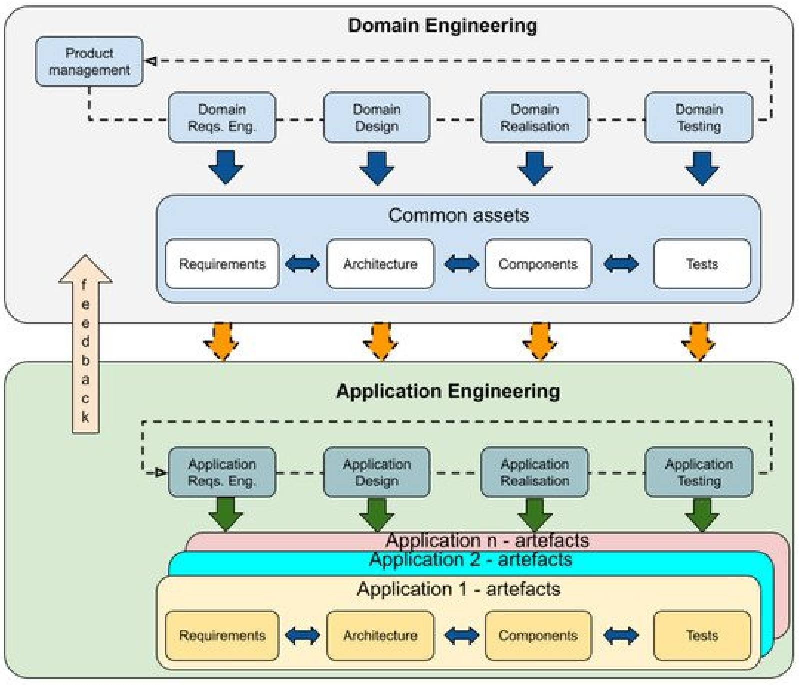

Figure 1. SPL framework, stages and their relationships.

Software development based on SPLs has brought about benefits such as the reuse of components, increases in productivity, reductions in development times, relatively fewer major errors, improvements in product quality, and lower costs, among others [2][3][15][16]. Contrary to what one may initially believe, the successful implementation of SPLs is not a phenomenon exclusive to large development companies but is also feasible in small and medium-sized companies, as demonstrated in [17].

3. Variability

One of the main concepts in SPLs development is *variability*, which gives SPLs the flexibility required to diversify and differentiate products [18].

Variability is introduced by defining reusable artifacts, such as architectures or components. These artifacts are included in the definition of a product family, depending on their inclusion or exclusion in each final product, giving rise to particular products [19]. Several authors have proposed models to manage the variability of SPLs. Most of these proposals are based on the FODA model [20]. This FODA model consists of characteristics and relationships that are graphically extended in the form of a tree.

For example, a software product must be able to adapt to the needs of each client or allow options for some specific configuration so that the products can reach different market segments [3]. In *domain engineering*, it is common to describe SPL and manage its *variability* with the aid of FMs [6].

4. Feature Models

The origin of FMs can be traced to the FODA method [20]. This model is still present but with slight variations and adaptions in some SPL methods based on visual representations of the product's features.

The structure of an FM is a type of tree of which the root node represents the product family, and the features are organized throughout the tree. These features can be assembled to give rise to particular software products [21]. FMs have been a relevant topic for SPLs in recent years, showing the best evolution behavior in terms of the number of published papers and references [13].

To illustrate the concepts present in an FM, consider the following scenario. A mobile phone must have the possibility of making a call and have a screen, but not all mobile phones must have a GPS. Furthermore, some of these features can depend on others for their inclusion or exclusion. For example, if a mobile phone has a GPS, it cannot have a basic screen. See details in **Figure 2**.

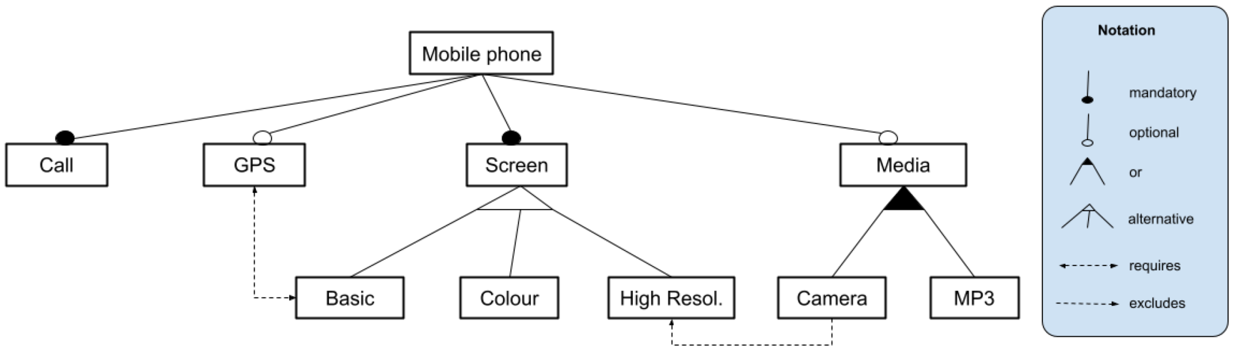

Figure 2. Example of an FM for a Mobile Phone SPL.

5. Automated Analysis of FMs–Reasoning Algorithms

Automatic analysis of FMs (AAFM) extracts information from such models using automated mechanisms [22]. This information includes verifying whether a given product represents a valid combination of features or checking the similarity between FMs. The analysis of FMs is an error-prone, tedious task, and it is not feasible to achieve this task manually with large-scale FMs. AAFM is an active area of research and is gaining importance for both practitioners and researchers in the SPL community [23].

Benavides et al. mention that AAFM can be defined as the computer-assisted extraction of information from FMs [24]. Different proposals for extracting this information have been made, based on specific algorithms or binary decision diagrams, such as BDD, SAT, and CSP [25]. **Table 1** presents a summary of these proposals.

Table 1. Proposals to extract information from FMs.

Proposal	Characteristics
Constraint Satisfaction Problem (CSP)	**This consists of a set of variables, finite domains for those variables, and a set of constraints that restrict the values of the variables. It can perform most of the operations currently identified in feature models [23].**
Boolean Satisfiability Problem (SAT)	**This consists of a set of Boolean variables connected by logical operators. The SAT problem consists of deciding whether a given propositional formula satisfies whether logical values can be assigned to its variables so that the formula is true [26].**
Binary Decision Diagrams (BDD)	**A data structure is used to represent a boolean function. A BDD is an acyclic, directed, rooted graph composed of a group of decision nodes and two terminal nodes called 0-terminal and 1-terminal. Each node of the graph represents a variable in a Boolean function and has two child nodes representing an assignment of the variable to 0 and 1 [27].**

The process of extracting information from an FM starts with the translation of the features and relationships encoded in the FMs and any additional information into a knowledge base described in a logical paradigm [28]. Subsequently, queries to the knowledge base can be performed using solvers.These operations are performed automatically using different approaches. Most of them translate FMs into specific logical paradigms, such as propositional logic, constraint programming, and description logic [14].

A classification of different proposals related to automatic or semi-automatic FM construction is presented in [29]. The authors of that study conceptualized an analysis framework for work in the field of automated FM construction. The framework considers four dimensions (proposal, input, tasks, and output) and fifteen sub-dimensions.

References

1. Van der Linden, F.J.; Schmid, K.; Rommes, E. Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2007.

2. Clements, P.; Northrop, L. Software Product Lines, Course Notes of Product Line Systems Program; Software Engineering Institute, Carnegie Mellon University: Pittsburgh, PA, USA, 2003.
3. Pohl, K.; Böckle, G.; van Der Linden, F.J. Software Product Line Engineering: Foundations, Principles and Techniques; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2005.
4. Ahmed, F.; Capretz, L.F. The software product line architecture: An empirical investigation of key process activities. Inf. Softw. Technol. 2008, 50, 1098–1113.
5. Chen, L.; Babar, M.A. A systematic review of evaluation of variability management approaches in software product lines. Inf. Softw. Technol. 2011, 53, 344–362.
6. Asikainen, T.; Mannisto, T.; Soininen, T. A unified conceptual foundation for feature modelling. In Proceedings of the 10th International Software Product Line Conference (SPLC’06), Baltimore, MD, USA, 21–24 August 2006; pp. 31–40.
7. Raatikainen, M.; Tiihonen, J.; Männistö, T. Software product lines and variability modeling. J. Syst. Softw. 2019, 149, 485–510.
8. Geraldi, R.T.; Reinehr, S.; Malucelli, A. Software product line applied to the Internet of Things: A systematic literature review. Inf. Softw. Technol. 2020, 124, 106293.
9. Ji, W.; Berger, T.; Antkiewicz, M.; Czarnecki, K. Maintaining feature traceability with embedded annotations. In Proceedings of the 19th International Conference on Software Product Line, Nashville, TN, USA, 20–24 July 2015; pp. 61–70.
10. Krüger, J.; Gu, W.; Shen, H.; Mukelabai, M.; Hebig, R.; Berger, T. Towards a better understanding of software features and their characteristics: A case study of marlin. In Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems, Madrid, Spain, 7–9 February 2018; pp. 105–112.
11. Bhushan, M.; Goel, S. Improving software product line using an ontological approach. Sādhanā 2016, 41, 1381–1391.
12. Hähnle, R.; Schaefer, I. The quest for formal methods in software product line engineering. In Software Technology: 10 Years of Innovation in IEEE Computer; John Wiley & Sons: Hoboken, NJ, USA, 2018.
13. Heradio, R.; Perez-Morago, H.; Fernandez-Amoros, D.; Cabrerizo, F.J.; Herrera-Viedma, E. A bibliometric analysis of 20 years of research on software product lines. Inf. Softw. Technol. 2016, 72, 1–15.
14. Benavides, D.; Segura, S.; Ruiz-Cortés, A. Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 2010, 35, 615–636.
15. Arboleda, H.; Royer, J.C. Model-Driven and Software Product Line Engineering; John Wiley & Sons: Hoboken, NJ, USA, 2012.
16. Apel, S.; Batory, D.; Kästner, C.; Saake, G. Software product lines. In Feature-Oriented Software Product Lines; Springer: New York, NY, USA, 2013; pp. 3–15.
17. Camacho, M.C.; Álvarez, F.; Collazos, C.; Leger, P.; Bermúdez, J.D.; Hurtado, J.A. A Collaborative Method for Scoping Software Product Lines: A Case Study in a Small Software Company. Appl. Sci. 2021, 11, 6820.
18. Galster, M.; Weyns, D.; Tofan, D.; Michalik, B.; Avgeriou, P. Variability in software systems—A systematic literature review. IEEE Trans. Softw. Eng. 2013, 40, 282–306.
19. Sinnema, M.; Deelstra, S. Industrial validation of COVAMOF. J. Syst. Softw. 2008, 81, 584–600.
20. Kang, K.C.; Cohen, S.G.; Hess, J.A.; Novak, W.E.; Peterson, A.S. Feature-Oriented Domain Analysis (FODA) Feasibility Study; Technical Report CMU/SEI-90-TR-021; Carnegie-Mellon University: Pittsburgh, PA, USA; Software Engineering Institute: Pittsburgh, PA, USA, 1990.
21. Czarnecki, K.; Wasowski, A. Feature diagrams and logics: There and back again. In Proceedings of the 11th International Software Product Line Conference (SPLC 2007), Kyoto, Japan, 10–14 September 2007; pp. 23–34.
22. Batory, D. Feature models, grammars, and propositional formulas. In Proceedings of the International Conference on Software Product Lines, Rennes, France, 26–29 September 2005; pp. 7–20.
23. Benavides, D.; Trinidad, P.; Ruiz-Cortes, A. Automated reasoning on feature models. In Proceedings of the International Conference on Advanced Information Systems Engineering, Porto, Portugal, 13–17 June 2005; pp. 491–503.
24. Benavides, D.; Galindo, J.A. Automated analysis of feature models: Current state and practices. In Proceedings of the 22nd International Systems and Software Product Line Conference, Gothenburg, Sweden, 10–14 September 2018; Volume 1, p. 298.
25. Lettner, M.; Rodas, J.; Galindo, J.A.; Benavides, D. Automated analysis of two-layered feature models with feature attributes. J. Comput. Lang. 2019, 51, 154–172.

26. Cook, S.A. The complexity of theorem-proving procedures. In Proceedings of the Third Annual ACM Symposium on Theory of Computing, Shaker Heights, OH, USA, 3–5 May 1971; pp. 151–158.
27. Bryant, R.E. Graph-based algorithms for boolean function manipulation. IEEE Trans. Comput. 1986, 100, 677–691.
28. Galindo, J.; Benavides, D.; Trinidad Martín Arroyo, P.; Gutiérrez, A.; Ruiz, A. Automated analysis of feature models: Quo vadis? Computing 2019, 101, 387–433.
29. Gacitúa, R.; Sepúlveda, S.; Mazo, R. FM-CF: A framework for classifying feature model building approaches. J. Syst. Softw. 2019, 154, 1–21.