SPLConfig: Product Configuration in Software Product Line

Lucas Machado, Juliana Pereira, Lucas Garcia, Eduardo Figueiredo

Department of Computer Science, Federal University of Minas Gerais (UFMG), Brazil

{lucasmdo, juliana.pereira, lucas.sg, figueiredo}@dcc.ufmg.br

Abstract. *Software product line (SPL) is a set of software systems that share a common set of features satisfying the specific needs of a particular market segment. A feature represents an increment in functionality relevant to some stakeholders. SPLs commonly use a feature model to capture and document common and varying features. The key challenge of using feature models is to derive a product configuration that satisfies all business and customer requirements. To address this challenge, this paper presents a tool, called SPLConfig, to support business during product configuration in SPL. Based on feature models, SPLConfig automatically finds an optimal product configuration that maximizes the customer satisfaction.*

Demo Video. *https://www.youtube.com/watch?v=QLHtIY8oHT8*

1. Introduction

The growing need for developing larger and more complex software systems demands better support for reusable software artifacts [Pohl et al., 2005]. In order to address these demands, software product line (SPL) has been increasingly adopted in software industry [Clements and Northrop, 2001; Apel et al., 2013]. SPL is a set of software systems that share a common set of features satisfying the specific needs of a particular market segment [Pohl et al., 2005]. It is built around a set of common software components with points of variability that allow product configuration [Clements and Northrop, 2001]. Large companies, such as Hewlett-Packard, Nokia, Motorola, and Dell, have adopted SPL practices[1].

The potential benefits of SPLs are achieved through a software architecture designed to increase reuse of features in several SPL products. An important concept of an SPL is the feature model. Feature models are used to represent the common features found on all products of the product line (known as mandatory features) and variable features that allow distinguishing between products in a product line (generally represented by optional or alternative features) [Czarnecki and Eisenecker, 2000; Kang et al., 1990]. Variable features define points of variation and their role is to permit the instantiation of different products by enabling or disabling specific SPL functionality. In practice, developing an SPL involves modeling features to represent different viewpoints, sub-systems, or concerns of the software system [Batory, 2005].

A fundamental challenge in SPL is the process of enabling and disabling features in a feature model for a new software product configuration [Pohl et al., 2005]. As the number of features increase in a feature model, so does the number of product options in

[1] http://splc.net/fame.html

an SPL [Benavides et al., 2005]. For instance, an SPL where all features are optional can instantiate 2^n different products where n is the number of features. Moreover, once a feature is selected, it must be verified to conform to the myriad constraints in the feature model, turning this process into a complex, time-consuming, and error-prone task. Industrial-sized feature models with hundreds or thousands of features make this process impractical. Guidance and automatic support are needed to increase business efficiency when dealing with many possible combinations in an SPL.

This paper presents a tool, called SPLConfig[2], to support automatic product configuration in SPL. The main goal of the SPLConfig tool is to derive an optimized features set that satisfies the customer requirements. The primary contribution of this tool is to assist businesses during product configuration, answering the following question: What is the set of features that balances cost and customer satisfaction, based on available budget? By resolving this problem, industries can more effectively achieve greater customer satisfaction.

The rest of this paper is organized as follow. Section 2 describes the problem. Section 3 presents the SPLConfig architecture. Section 4 discusses the design and implementation of this tool. Examples are used to validate the tool main functionalities in Section 5. Section 6 presents some related work. Finally, Section 7 concludes and points out directions for future work.

2. Problem Description

Feature models represent the common and variable features in SPL using a feature tree [Kang et al., 1990]. In feature models, nodes represent features and edges show relationships between parent and child features [Batory, 2005]. These relationships constraint how the features can be combined. As an example, the mobile phone industry uses features to specify and build software for configurable phones. The software product deployed into a phone is determined by the phone characteristics. Figure 1 depicts a simplified feature model of an SPL, called MobileMedia [Figueiredo et al., 2008], inspired by the mobile phone industry. It was developed for a family of 4 brands of devices, namely Nokia, Motorola, Siemens, and RIM.

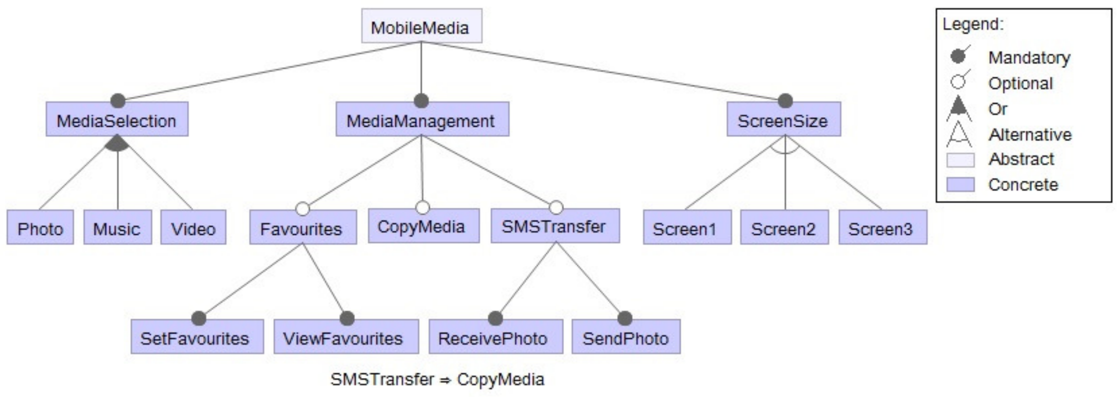


Figure 1. Example of a Feature Model for a Mobile Phone Product Line

[2] SPLConfig is available at https://sourceforge.net/p/splconfig/

A key need with SPL is determining how to configure an optimized feature set that satisfies the customer requirements. In Figure 1, features refer to functional requirements of the MobileMedia SPL. However, feature may also be associated with non-functional requirements. Kang et al. [1990] suggest the need to take into account non-functional requirements since 1990. For instance, considering the MobileMedia SPL, it is possible to identify non-functional requirements related to each feature, such as cost and benefit of each feature to the customer. It means that every product differs not only because of its functional features, but also because of its non-functional requirements. Therefore, we propose to extend feature models with non-functional requirements by adapting the proposed notation in Benavides et al. [2005] to our problem.

Figure 2 illustrates the non-functional requirements in feature model of Figure 1. In Figure 2, all features (mandatory and optional features) have the cost attribute, and only optional features have the benefit attribute. For example, the optional feature *Favourites* has cost and benefit attributes, while the mandatory feature *MediaSelection* has only the cost attribute. Note that the benefit attribute is classified into six qualitative levels: none, very low, low, medium, high and very high. Our goal is to find an optimal solution by means of an objective function that maximizes customer satisfaction (benefit/cost) without exceeding the available budget (in the lower right corner of the Figure 2). As a motivating example, given a mobile phone product line that includes a variety of varying features, what is the product that best meets the customer requirements limited by a given budget? The challenge is that with hundreds or thousands of features, it is hard to analyze all different product configurations to find an optimal configuration.

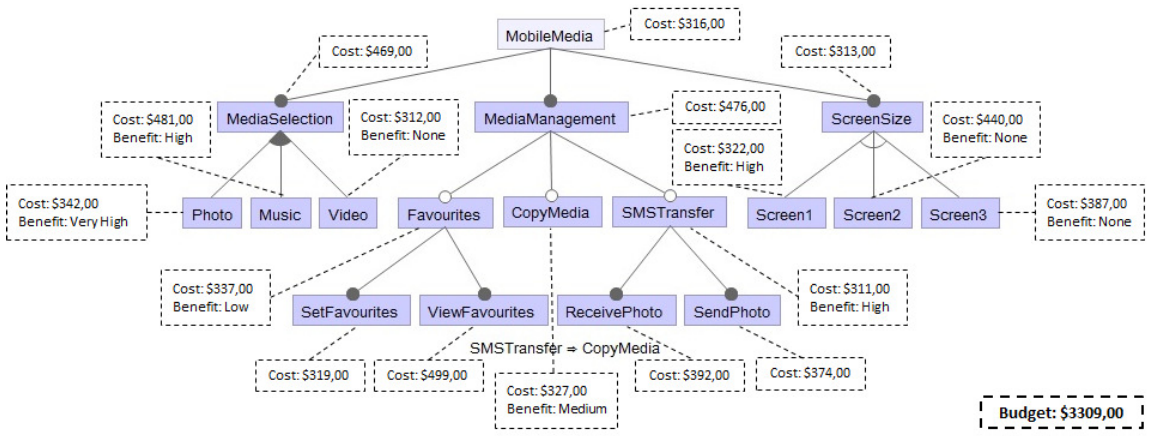


Figure 2. A Feature Model Decorate with Non-Functional Requirements

Therefore, the main goal of the SPLConfig tool is to propose an automatic product configuration method based on search-based software engineering (SBSE) techniques [Harman & Jones, 2001]. Figure 3 presents an abstract overview of our method. As shown in Figure 3, from a feature model we use search-based algorithms to derive an optimized product configuration that maximizes the customer satisfaction, subject to business requirements (cost), customer requirements (benefit and budget), composition constraints, and cross-tree constraints. It is noteworthy that these algorithms are minutely detailed in Pereira (2014).

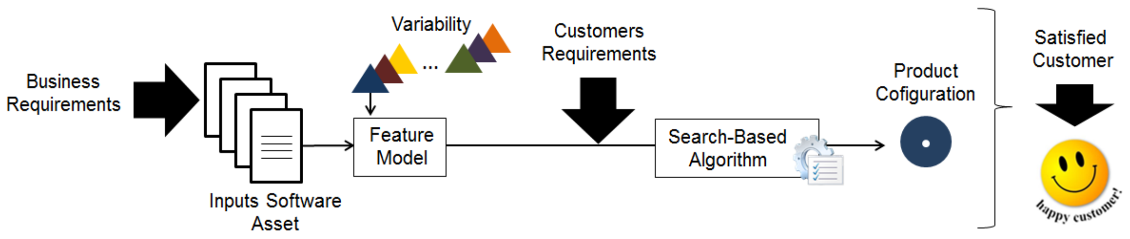


Figure 3. Overview of the Method to Automatic Product Configuration

3. SPLConfig Architecture

SPLConfig is an Eclipse plugin implemented in Java. It requires one additional plugin, named FeatureIDE, in order to support product configuration. Figure 4 presents the architecture of SPLConfig and its relationships with FeatureIDE and Eclipse platform. The decision of extending FeatureIDE is particularly attractive for two reasons. First, FeatureIDE is an open source, extensible, modular, and easy to understand tool. Second, our motivation in extending FeatureIDE occurred because we could reuse the key functionalities of typical feature modeling tools, such as to create and edit a feature model, to automatically analyze the feature model, product configuration basic infrastructure, code generation, and import/export feature model. FeatureIDE is a development tool for SPL largely used and it supports all phases of feature-oriented software development of SPLs: analysis and application domains and code generation.

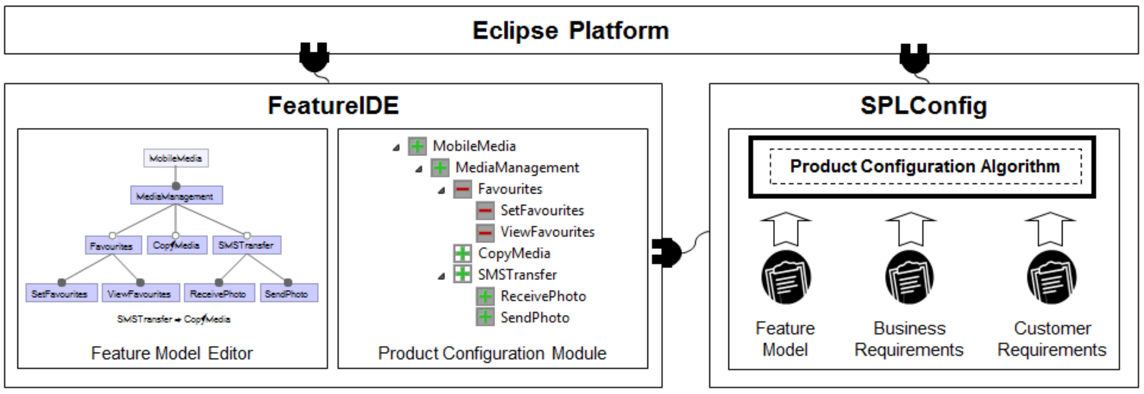


Figure 4. SPLConfig's Architecture

Automatic product configuration allows the customization of different products according to the business and customer requirements, composition, and cross-tree constraints of the feature model (SPLConfig in Figure 4). Two important activities are performed in the SPL lifecycle, domain engineering and application engineering, described below:

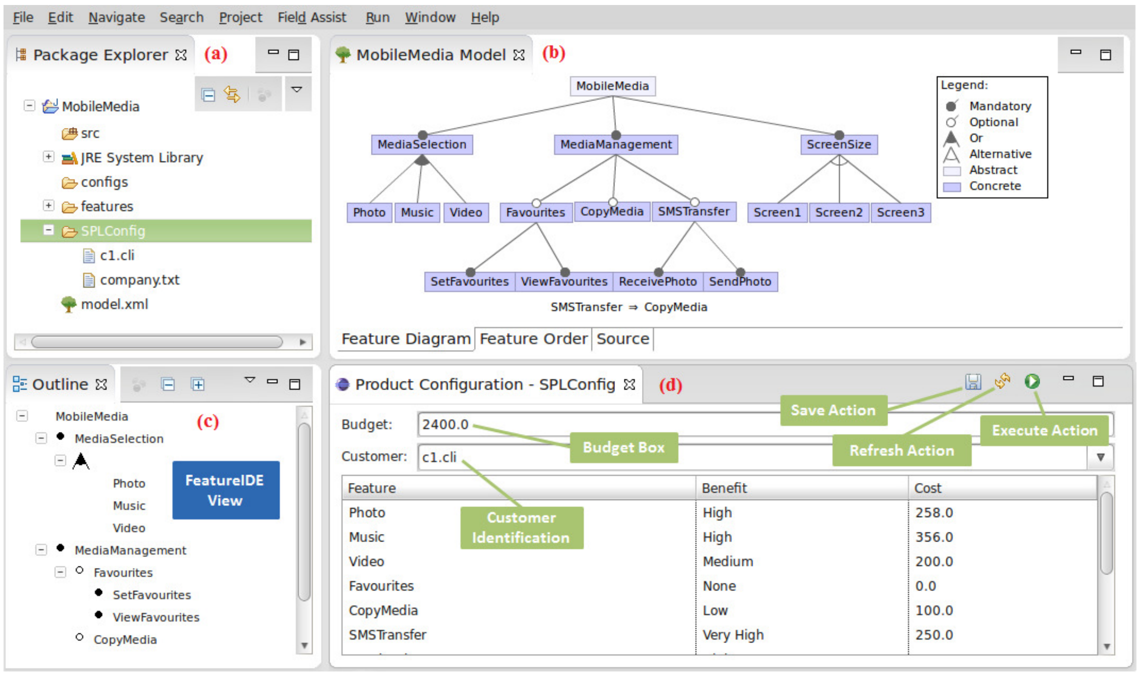
Domain Engineering. This process is represented by a feature model and is supported by FeatureIDE (feature model editor in Figure 4). Common and variable requirements of the product line are elicited and documented. The developers create reusable artifacts of a product line in such a way that these artifacts are sufficiently adaptable (variable) to efficiently derive individual applications from them.

Application Engineering. From meetings, the developers identify and describe the requirements prioritization of customer (SPLConfig in Figure 4). SPLConfig prioritize and create reports to aid the product builder when defining a product configuration for a specific customer. Our product configuration algorithm uses an optimization scheme and provides a valuable decision support to product configuration within the SPL. The product configuration result is visualized in FeatureIDE (product configuration module in Figure 4). It is important to observe that the tool can be easily extended - by means of new algorithms - to support additional non-functional features.

As show in Figure 4, feature modeling in early stages enables to decide which features should be supported by an SPL and which should not. This result is a feature model represented in FeatureIDE. In a second stage, during the product configuration, it is necessary to choose the features that appropriately fulfill the business and customer requirements. This stage allows a single product configuration to be created through search-based algorithms in SPLConfig. This product configuration is presented by FeatureIDE in the product configuration module. Note that the current implementation of SPLConfig randomly picks up one of the optimal solutions. However, we are working on a new version that presents a sorted list of the top best solutions. Therefore, the customer is going to have several options of products to choose from. In a third stage, SPLConfig allows manual configuration of features if necessary. In a fourth stage, it also allows compiling and building the product.

4. Design and Implementation Decisions

This section discusses some design and implementation decisions we made in the development of SPLConfig. Figure 5 shows a screenshot of SPLConfig view in the Eclipse IDE. Figure 5 shows the package explorer view typical of Eclipse IDE (a). It also illustrates the FeatureIDE model editor (b) and outline view (c). Figure 5 (d) show details of the SPLConfig view.



Budget: 2400.0

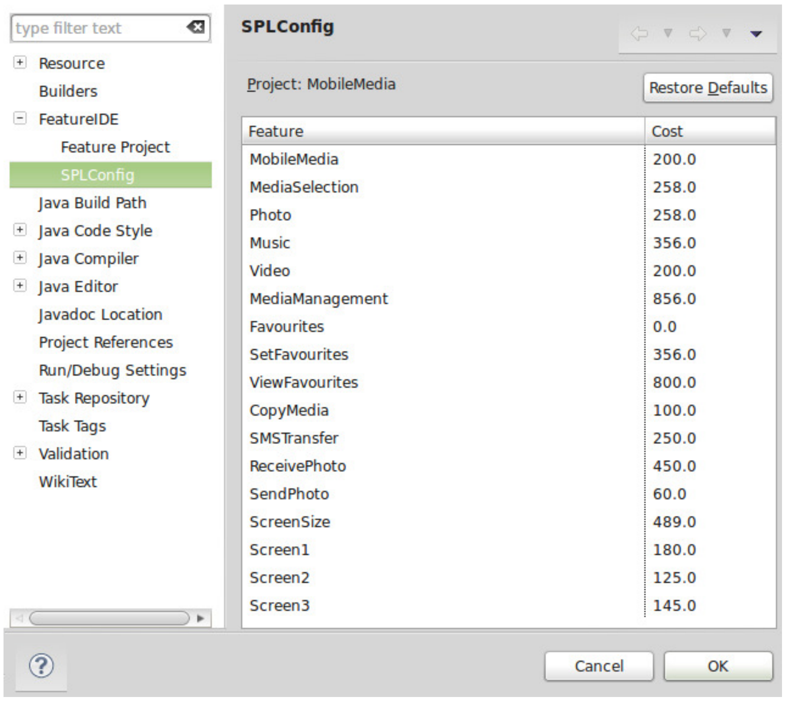
Customer: c1.cli

Feature	Benefit	Cost
Photo	High	258.0
Music	High	356.0
Video	Medium	200.0
Favourites	None	0.0
CopyMedia	Low	100.0
SMSTransfer	Very High	250.0

Figure 5. SPLConfig View in the Eclipse IDE

Figure 5 (d) presents the main view of SPLConfig integrated with the Eclipse IDE. At the top of the view, we have two fields named *Budget* and *Customer* that should be filled by the customer with the available budget and with the customer identification, respectively. Each feature in the feature model is presented in a row of this view. Columns give additional information, such as the feature name, the level of importance of the feature to the customer (*Benefit*), and the cost of development of each feature (*Cost*). The tool is supposed to be used by developers who are expected to translate qualitative information from customers into quantitative ones. Moreover, this view includes typical buttons, such as *Refresh* and *Execute*, which trigger their respective actions. *Refresh* is used to update data being presented in this view, while *Execute* selects the most appropriate product configuration that best satisfies the customer requirements.

In addition to this view, we extend Eclipse with a preference page presented in Figure 6. Industries can set specific preferences, as the cost of development of each feature that makes up the SPL. Therefore, for the same SPL, several products can be generated according to needs and constraints specific of each customer, but keeping the cost of each feature fixed. Note that the FeatureIDE view can be used to show the product configuration that best satisfies the customer requirements, but it does not prevent other features from being included or excluded in the final product (i.e., manual tuning).



Project: MobileMedia

Feature	Cost
MobileMedia	200.0
MediaSelection	258.0
Photo	258.0
Music	356.0
Video	200.0
MediaManagement	856.0
Favourites	0.0
SetFavourites	356.0
ViewFavourites	800.0
CopyMedia	100.0
SMSTransfer	250.0
ReceivePhoto	450.0
SendPhoto	60.0
ScreenSize	489.0
Screen1	180.0
Screen2	125.0
Screen3	145.0

Figure 6. SPLConfig Preference Page

5. Preliminary Evaluation

We reviewed a large number of research works in the field of SPL and selected a set of ten feature models that were used as benchmark instances to evaluate SPLConfig: MobileMedia [Figueiredo et al., 2008], Email System [Thüm et al., 2014], Smart Home [Alferez et al., 2009], Devolution [Thüm et al., 2014], Gasparc [Aranega et al., 2012], Web Portal [Mendonça et al., 2008], FraSCAti [Seinturier et al., 2012], Model Transformation [Czarnecki & Helsen, 2003], Battle of Tanks [Thüm & Benduhn, 2011], and e-Shop [Lau, 2006]. We found that the time spent in the configuration of a feature model with 213 features (e-Shop) is about 6 milliseconds. However, so far, SPLConfig has scaled well for all evaluated feature models (up to 3 hundreds features). It is noteworthy that evaluation details are available in Pereira (2014).

6. Related Tools

Automated reasoning is a challenging field in SPL engineering. Mendonça et al. (2009) and Thüm et al. (2014) have proposed visualization techniques for representing staged feature configuration in SPLs. However, industrial-sized feature models with hundreds or thousands of features make a manual feature selection process hard. Moreover, these approaches focus on functional features of a product and their dependencies; neglecting non-functional requirements. To the best of our knowledge, we still lack tools to deal with non-functional features.

7. Conclusions and Future Work

This paper presented SPLConfig, a tool developed for product configuration in SPL by the use of search-based algorithms. We described the problem handled by the tool (Section 2) and summarized the method behind the tool and its main functionalities (Sections 3 and 4). Our results so far are, in general, satisfactory. Nevertheless, our goals for future work include the improvement of SPLConfig tool in order to (i) include others non-functional requirements [Kang et al. 1998] and (ii) consider the difficulties of developing and maintaining of the product for the industries, in order to minimize the development effort integrating features to compose a product. Further work should also address case studies in industries in order to ensure that realistic instances of requirements are being used, through direct contact with the business and customers.

Acknowledgment

This work was partially supported by CNPq (grant Universal 485907/2013-5) and FAPEMIG (grants APQ-02532-12 and PPM-00382-14).

References

Apel, S., Batory, D., Kästner, C., and Saake, G. (2013). Feature-Oriented Software Product Lines: Concepts and Implementation. Springer-Verlag.

Alferez, M., Santos, J., Moreira, A.and Garcia, A., Kulesza, U., Araujo, J., and Amaral, V. (2009). Multi-view composition language for software product line requirements. In International Conference on Software Language Engineering (SLE), p. 136–154.

Aranega, V., Etien, A., and Mosser, S. (2012). Using feature model to build model transformation chains. In 15th international conference on Model Driven Engineering Languages and Systems (MODELS), pages 562–578.

Batory, D. S. (2005). Feature models, grammars and propositional formulas. In 9th International Software Product Lines Conference (SPLC), pages 7–20.

Benavides, D., Martin-Arroyo, P. T., and Cortes, A. R. (2005). Automated reasoning on feature models. In 17th Conference on Advanced Information Systems Engineering (CAiSE), pages 491–503.

Clements, P. and Northrop, L. (2001). Software product lines: Practices and patterns. Addison-Wesley.

Czarnecki, K. and Eisenecker, U. W. (2000). Generative programming: Methods, tools, and applications. Addison-Wesley.

Czarnecki, K. and Helsen, S. (2003). Classification of model transformation approaches. Available at: http://www.ptidej.net/course/ift6251/fall05/presentations/050914/Czarnecki_Helsen.pdf/.

Figueiredo, E. et al. (2008). Evolving software product lines with aspects: An empirical study. In 30th International Conference on Software Engineering (ICSE), p. 261-270.

Harman, M. and Jones, B. F. (2001). Search based software engineering. Journal Information and Software Technology, 43(0):833–839.

Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, S. (1990). Feature-oriented domain analysis (FODA) feasibility study. Technical Report. CMU/SEI-90-TR-21. ESD-90-TR-222.

Kang, K., Kim, S., Lee, J., Kim, K., Shin, E., and Huh, M. (1998). Form: A feature-oriented reuse method with domain-specific reference architectures. Annals of Software Engineering, 5(1):143–168.

Lau, S. Q. (2006). Domain analysis of e-commerce systems using feature-based model templates. Master's thesis. University of Waterloo, Canada.

Mendonça, M., Bartolomei, T., and Donald, C. (2008). Decision-making coordination in collaborative product configuration. In ACM Symposium on Applied Computing (SAC), p. 108–113.

Mendonça, M., Branco, M., and Cowan, D. (2009). S.p.l.o.t.: Software product lines online tools. In 24th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pages 761–762.

Pereira, Juliana A. Search-Based Product Configuration in Software Product Lines. April 2014. Federal University of Minas Gerais. Belo Horizonte - April 2014.

Pohl, K., Böckle, G., and Van der Linden, F. J. (2005). Software product line engineering: Foundations, principles and techniques. Springer-Verlag.

Seinturier, L., Merle, P., Rouvoy, R., Romero, D.and Schiavoni, V., and Stefani, J. (2012). A component-based middleware platform for reconfigurable service-oriented architectures. Software Practice and Experience, 42(5):559–583.

Thüm, T. and Benduhn, F. (2011). Spl2go: An online repository for open-source software product lines. Available at: http://spl2go.cs.ovgu.de/projects. [Online; accessed 10-December-2013].

Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., and Leich, T. (2014). Featureide: An extensible framework for feature-oriented software development. Journal Science of Computer Programming, 79(0):70–85.