International Journal of Computer Science and Software Engineering (IJCSSE), Volume 5, Issue 12, December 2016
ISSN (Online): 2409-4285 www.IJCSSE.org Page: 265-272

Towards an Architecture for Integration of Feature Models

Vinicius Bischoff[1], Kleinner Farias[2], Lucian Gonçales[3] and Vanessa Weber[4]

[1, 2, 3, 4] Interdisciplinary Postgraduate Program on Applied Computing (PIPCA), University of Vale do Rio dos Sinos (UNISINOS), Av. Unisinos, 950, Cristo Rei, São Leopoldo\RS, Brazil, CEP 93022-750

[1]viniciusbischoff@gmail.com, [2]kleinnerfarias@unisinos.br, [3]lucianjosegoncales@gmail.com, [4]weber.nessa@gmail.com

ABSTRACT

The integration of variability models (e.g., feature models) is considered an error-prone activity, which can consume a lot of effort from development teams working in parallel, thereby compromising developers' productivity as well as the quality of software developed. For this, many integration techniques have been proposed in last decades to support developers to integrate feature models. However, there is a lack of a generic architecture that helps developers to produce integration tools in the current literature. To overcome this shortcoming, this work proposes a flexible, component-based architecture for supporting the integration of feature models. In addition, a model integration workflow for helping developers to improve the understanding of the crucial composition activities and their relationships is also presented.

Keywords: *Feature Model, Model Integration, Tool Support, Architecture.*

1. INTRODUCTION

The software industry in recent years has been increasing its production capacity, followed by a demand for increasingly complex systems with the objective of constant improvements in quality. In this context, companies are faced with the challenge of meeting the individual expectations of each customer while at the same time efficiently executing the required software engineering processes. Component reuse is one of the strategies employed by engineering to reduce costs and effort in system development.

The product lines emerged with this motivation, i.e., to create families of products with characteristics common to each other and systematize the reuse. The combination of the concepts of product families and customization originated the Software Product Line (SPL), a set of software systems defined on a common architecture that share the same set of features [1, 2].

The SPL is represented by several techniques, such as the methods, FODA [3], FORM [4], CBMF [5], FeatuRSEB [6], PLUSS [7]. The techniques propose or use a notation or model to represent the variability of the domain or architecture. The model of variability consists of demonstrating the functionalities of a domain through its characteristics as well as their respective relationships and interdependencies through a hierarchical structure [5, 8, 9, 10]. The way to represent the variability of a SPL is through the *Feature Model* (FM).

The adoption of feature models has become common in mainstream software development projects in industry [11, 14, 15]. In fact, researchers and practitioners have widely used feature models for different purposes, e.g., to manage variability in the context of SPLs, helping describe domain concepts in terms of their commonalities and differences within a family of software systems [12], specifying features (and their dependencies) of product lines [8], deriving automatically products from SPL [16], describing variability in SPL by documenting features and their valid combinations [40], or even help developers to integrate the features of a family of software systems [13].

Given that feature models can be created collaboratively by different software-development teams [27], at some time the models created in parallel must be integrated to form a "big picture" view of the SPL as a whole. For this reason, techniques of feature model integration have been proposed, e.g., [9, 13, 17, 18]. The term *integration of feature models* can be defined as a set of activities that should be performed over two (or more) input feature models, i.e., Feature Model A (FM_A) and a Feature Model B (FM_B), in order to produce an output-desired Feature Model AB (FM_{AB}). In practice, developers make use of integration techniques to accommodate upcoming changes, i.e., typically found in FM_B, into the FM_A.

The integration of large feature models in software industry has been an ever-present concern of researchers [9, 19, 32], which have sought to elaborate precise and efficient techniques to support the integration of heterogeneous feature models. Without this technique support, the production of desired feature models becomes an error-prone and effort-consuming task [29, 30, 31]. Because of this, it is often the case that

developers end up examining all parts of the two input feature models instead of prioritizing the overlapping ones, i.e., those that often give rise to problems of integration conflicts.

In addition, the integration of feature models has been widely investigated in practice, given its pivotal role for supporting the evolution of SPLs. For this reason, both academia and industry have proposed several works in recent years to support the feature model integration practices [8, 9, 14].

Unfortunately, the techniques proposed in the literature have demonstrated to be ineffective to support the integration of feature models in real-world environments. The limitations arise from the inflexibility in the production of a new feature model; the tools are proposed to automate their output [8, 9, 10, 12, 14], and an incorrect implementation may compromise the model architecture by providing rework for development teams with a direct impact on effort, production costs, and, mainly, on the quality of the product generated.

To overcome these shortcomings, this paper, therefore, proposes a flexible, component-based architecture for aiding the development of feature model integration tools, hereafter called FMIt-Arch. In addition, a model integration workflow for helping developers to improve the understanding of the crucial composition activities and their relationships is presented. Our preliminary evaluation indicated that the proposed architecture might support the development of tools for the integration of feature models.

The remainder of the paper is organized as follows. Section 2 contrasts this work with the current literature. Section 3 presents the FMIt architecture. Section 4 describes the integration tool developed using the FMIt-Arch. Finally, Section 5 presents some concluding remarks and future work.

2. RELATED WORK

In the literature, several papers suggest the design and implementation of merge operations (e.g., [10, 17, 18, 19]), in which separate FMs are used to model decision taken by different development teams and the need for integration. In Acher *et al*. [12], the authors compared strengths and weaknesses of different implementation approaches of composition operators. The study provides some evidence that using generic model composition frameworks have not helped. It proposes the use of Boolean logic turns out to fulfill most of the criteria expected from a merge operator. The use of CSP solvers can also be considered in addition to SAT and BDD techniques. A longer-term perspective is to consider the implementation of diff and refactoring operations for FM.

In [17], an algorithm is designed to automatically determine the kind of relations between two FMs in terms of sets of configuration. This work presents little or no tool support to help product line designers' measure the impact of their feature model modifications. The authors presume that a compact representation of all added and removed products would be an extension of this proposed in improving tools for managing the evolution of feature models.

In Berger *et al.* [12], the authors aimed at positioning feature models in the field of software product lines. This work highlights the attention that enterprises have invested on features as a way for supporting the development of their products. The authors then explored the environments of these enterprises to investigate the different aspects of the use of features in real-world settings. They concluded enterprises do not have common practices and guidelines to maintain and manage features throughout the product life cycle. Typical operations of inclusion and exclusion have been overlooked, for example. Therefore, understanding the field of integration of feature models is necessary to pinpoint research gaps and develop a better support to overcome these limitations.

In [13], the authors proposed syntactic and semantic operators for integrating two input feature models. Furthermore, other authors proposed syntactic and semantic operators but with the purpose of differentiating feature models [10]. In contrast, Segura *et al*. [20] proposed using graph data structures to automate the composition of feature models.

Integration of feature models is an important task since the parallel manipulation of these artifacts has become more often. Thus, industry will demand precise and effective integration techniques. To this end, researchers and practitioners need guidelines, and tools support the development of the tasks of feature model integration. Moreover, a precise technique for model composition is needed due to developers demanding more effort when using an inappropriate integration technique [21].

3. FMIt ARCHITECTURE

We present the FMIt's built-in model integration process by identifying the phases, the artifacts generated, and the main activities required to transform the input models, FM_A and FM_B, into an intended output integrated model, FM_{AB}. Moreover, it details the most relevant characteristics related to designing and implementation issues, including feature model elicited, components that implement such features, architectural design, and finally the derived tool for the integration of feature models.

3.1 Model Integration Process

Figure 1 shows the model integration process proposed. It is represented as an intelligible workflow, thus allowing development teams to understand the activities inherent in a process of composition in terms of phases, their artifacts, activities and the flow between them. This workflow is based on the previous studies [22, 24, 28]

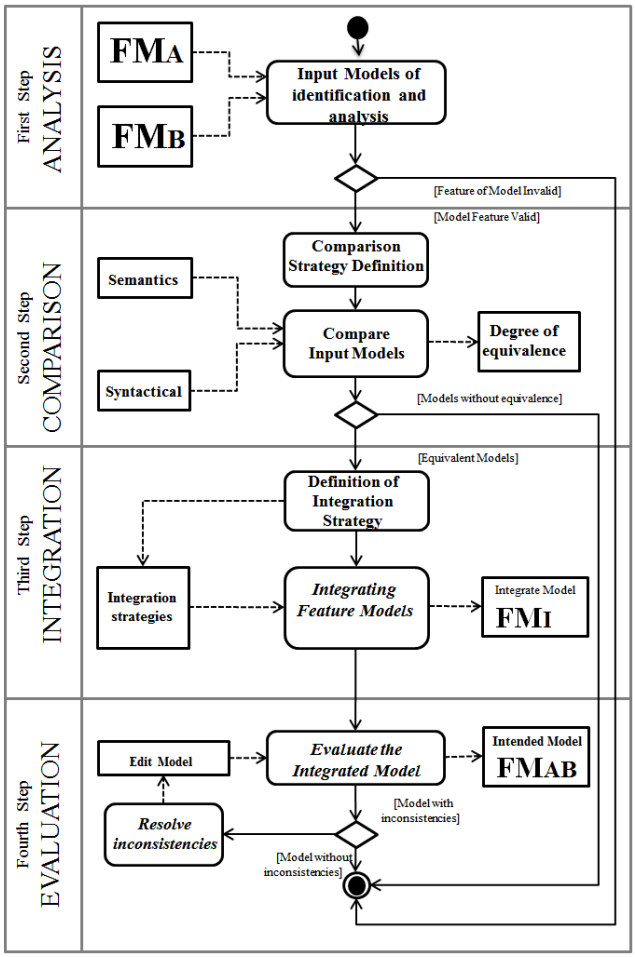

Fig. 1. The proposed model integration process.

First Step: Analysis. The prime goal is to analyze the input models adequately as a basis for assuring the integration of compatible input models as well as preventing input models with inconsistencies [8, 9, 10]. This step should attend to the Analysis Feature Model criteria answering: are the input models of the same type? Do the input models have conflicting? If the input models do not attend to this milestone, the integration process can be cancelled or repeated after the input models are redesigned to comply with milestone criteria.

Second Step: Comparison. The chief goal is to systematically compare the input models for determining the degree of similarity between their elements, thereby mitigating conflicts. The FMIt architecture supports a range the strategies to reduce risks. The inputs of this phase are: syntactic and semantic. Since a natural language is acceptable different interpretations causing ambiguity in the comparison, trying to avoid equity between the meanings to the words, we apply a thesaurus in conjunction with a technique for comparison of string, the algorithm of Jaro Distance. The comparison strategies as well as a threshold will determine the rules to be applied (automatic or semi-automatic).

Hence, producing the following outputs: (i) the similarity matrix, specifying the degree of equivalence (ranging from 0 to 1) between the input model elements; (ii) the matching elements, a description of the elements of FM_A and FM_B being considered equivalent; (iii) the no-matching elements, a description of the elements of FM_A and FM_B being considered no equivalent. Two input feature models are considered distinct when the degree of similarity between them is equal or lower than 0.95, the threshold used as an inference for decision making, i.e. semi-automatic. If the threshold is higher or identical to 1.00 or equal to 0.00, it applies the integration strategies automatically according to the established rules. If it is not possible to identify the equivalence between feature models, the process will be finished.

Third Step: Integration. The master goal is to carefully bring together the matching and no-matching features model for producing an output intended model, FM_{AB}. For this, the proposed integration technique takes into account the similarity matrix, as well as the description of the matching elements of the input models. In addition, it uses a range of established integration strategies, including union, intersection, difference and complementary [12, 13, 17], to accommodate the model from FM_B into FM_A, thereby alleviating the more severe risks. The FMIt's built integration strategies compose the matching elements while the no matching ones are just inserted into the FM_I. Thus, FM_I represents the matching and no-matching feature models, all blended systematically.

Fourth step: Evaluation. The key goal is to evaluate if the output model produced in the previous step matches the output intended one, i.e. $FM_I = FM_{AB}$. If $FM_I \neq M_{AB}$, then FM_I needs to be manipulated so that the inconsistencies can be resolved. For this, the tool enters semi-automatic mode checks if the output model is in compliance with defined strategies to assist development teams, more specifically developers in the face of decision-making. If the model has inconsistencies, then some transformation rules can be applied to transform FM_I into FM_{AB}. This step ends producing the output intended model. After detailing the integration process,

the next Section focuses on describing the design and implementation issues required to put the process in practice.

3.2 FMIt Architecture Feature Model

The FMIt architecture was proposed due to several reasons and requirements identified in previous works [21, 24]. Our experience with model integration has indicated the need for reusable architecture to support and guide the development of new integration tools. It is representative of the model integration domain, since its design decomposes the key concerns into well modularized features.

Lastly, it allows evaluating the models generated and persisting the results. Thus, the proposed architecture provides a set of pivotal features, including analysis of the input models, comparison of the input models, integration of the equivalent input model, persistence of the output model generated, and evaluation of the output model

.

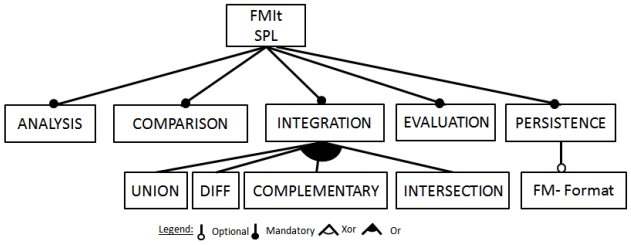

Fig. 2. A simplified FMIt-Arch.

Figure 2 shows a simplified view of FMIt-Arch's feature model. Thus, to develop integration tools developers should firstly implement the mandatory features, including analysis, comparison, integration, persistence, and evaluation. Besides identifying a set of core functionalities, the mandatory features seamlessly specify their dependencies in an easy-to-understand manner. An ever-present concern throughout the FMit architecture was to assure the mandatory features comply with the model integration process described in Figure 2, for example, the analysis feature implements the first step and the persistence feature provides the functionality required to persist the output-integrated model generated at the end of the model integration process. The optional features are the types of file format that the output-integrated model. The *or* features are represented by the integration strategies, and the comparison strategies, the latter are not shown in the feature model for space constraints. Thus, one (or more) comparison and composition strategy should be selected when a integration tool is derived from the FMIt-Arch.

3.3 FMIt Architectural Components

Figure 3 shows the components that are responsible for implementing the feature model as well as relates them with the features depicted in Figure 2. The small squares located on the left or bottom sides of the components represent this feature component mapping. For instance, the (I) on the top of the Integration component (Figure 3) indicates that this component contributes to the implementation of the integration feature. This *design-for-features* is supported by the component-based development, a systematic feature component mapping and aspect-oriented programming.

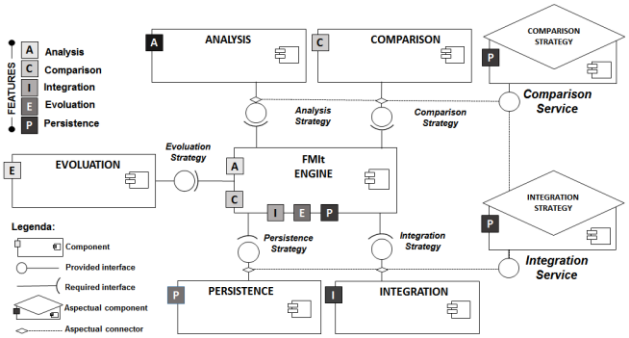

Fig. 3. The FMIt architectural components.

This method of decomposing components based on the features allows creating autonomous, well-modularized design elements within a model integration tool, thereby promoting the reuse of previously elicited feature and constructed components. Each component was designed to: (1) be a self-contained module that encapsulates the state and behavior of a set of executable elements, which are responsible for the implementation of one (or more) feature; (2) present emergent behaviors resulting from the interaction of its executable elements, i.e., one or more classes that realize the expected functionalities of the features; and (3) have well-defined interfaces, including the provided and required ones. For example, to provide the behavior of matching two input models, the Comparison component implements the provided interface, Comparison Strategy. If new components are inserted, then they should implement this interface only. Moreover, Figure 3 focuses on presenting the components as a coherent group of elements implementing one (or more) feature. Each component can be seen as a building block that plays a crucial role within the model composition process.

3.3 FMIt Multilayered Architecture Layers

The logical, multilayered architecture enables us to support a well-modularized design, thereby putting the heterogeneous, crosscutting concerns, previously

described in Figure 2, in shape. Figure 4 illustrates the component diagram, which focuses on presenting the group of elements responsible for performing each activity, specifically, which is, independent modules that play a determining factor in the integration of feature models.

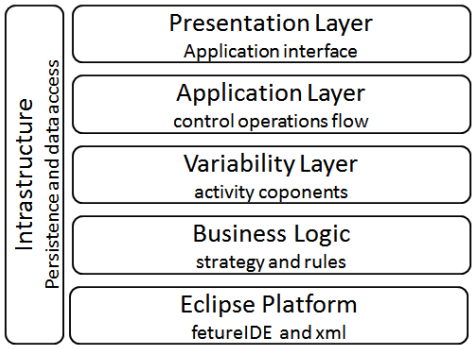

Fig. 4. The FMIt architecture layers.

The architecture is composed by five layers: (1) *Presentation layer* represents the topmost tier of the application gathering the input data required performing the functionalities and putting out the results to the integration; (2) *Application layer* encompasses FMIt's engine and its operators. It is responsible for organization, along with its operators, the integration process as a whole. The organization, plays a pivotal role by providing the main entry point, coordinating incoming integration requests, transforming the requests into commands for the operators, and rendering views; (3) *Variability layer* implements the variation points. For this, aspectual components weave the behaviors from design elements (from the business logic layer) to the operators (in the application layer). Aspectual components augment the operators with additional or alternative behaviors, i.e. strategies and their rules; (4) *Business Logic layer* defines a family of algorithms that implement the FMIt characteristics. These algorithms analyze the input models, seek to find the commonalities and differences between the input models, integrate the commonalities, and then evaluate the output models, and (5) *Infrastructure layer* accommodates the concerns related to handling exception, data access, persistence and logging, which are key crosscutting functionalities to put the integration process in practice

4. CASE STUDY

We evaluate this work by implementing a tool for integration of feature models based on the FMIt architecture. The tool, so-called FMIt, is an Eclipse Plug-in that allows a seamless integration with Eclipse Platform. In addition, it makes use of a range of Eclipse modeling technologies, including EMF, FeatureIDE, to implement all required activities described in the process of feature model integration. FMit ties together these technologies in such a way that makes it easy-to-use, even for users with little or no Java or XML coding experience. For example, FetureIDE reads and filters information from the tags of files written in XML and transforms it to an abstract data model in which input model elements can be manipulated as objects.

The term integration of feature models can be defined as a set of activities that must be performed with two (or more) input resource models, i.e., Feature Model A (FM_A) and Feature Model B (FM_B) in order to unify these models for the generation of a new model, that is, the desired feature model. The main challenge is to solve the conflicts that have arisen in the composition of these models. In figure 5, we present two feature models, which will be integrated.

4.1 Feature Model Integration Tool – FMIt Development

The implemented environment is an Eclipse platform plugin [25]. This implies in the use of the features offered by the platform, and, on the other hand, allowing users to work with FetuareIDE while using the Eclipse platform. Because many feature models are deployed in Java, such as FeatureIDE, Betty, Familiar, using the Eclipse platform also facilitates a possible code generation within the same development environment. The FeatureIDE [26] is an Eclipse-based framework, whose key role is to cover the entire development process and the incorporation of tools for the implementation of SPL in an integrated development framework.

In this scenario, the entire development process is defined: analysis, comparison, integration and evolution, being transformed until reaching the final goal, the FMIt tool. The following is briefly described the steps taken during the execution process, since they integrate the models:

1. Tooling Definition. First is specified FeatureIDE framework, which is an integral part of the ECLIPSE plug-in, for modeling as well as FMIt tool support.
2. Definition of the graphic model. This phase focuses on the definition of the elements and their relationships, FODA [3]. That is, the creation or import of the model, it is the graphic edition of the diagrams for composition.
3. Tooling Generation. This activity will be used to construct the new model. Displaying the results of the FMIt tool, inherent to the integration of two feature models, according to the applied strategies, returns as model output. The automatic or semiautomatic approach

is established by the threshold according to the business rules. Figure 5 shows an overview of the FMIt Tool together with some of the framework components highlighted with letters. The tool presents an initial view to the integration of specific models within the framework, and is discussed briefly below:

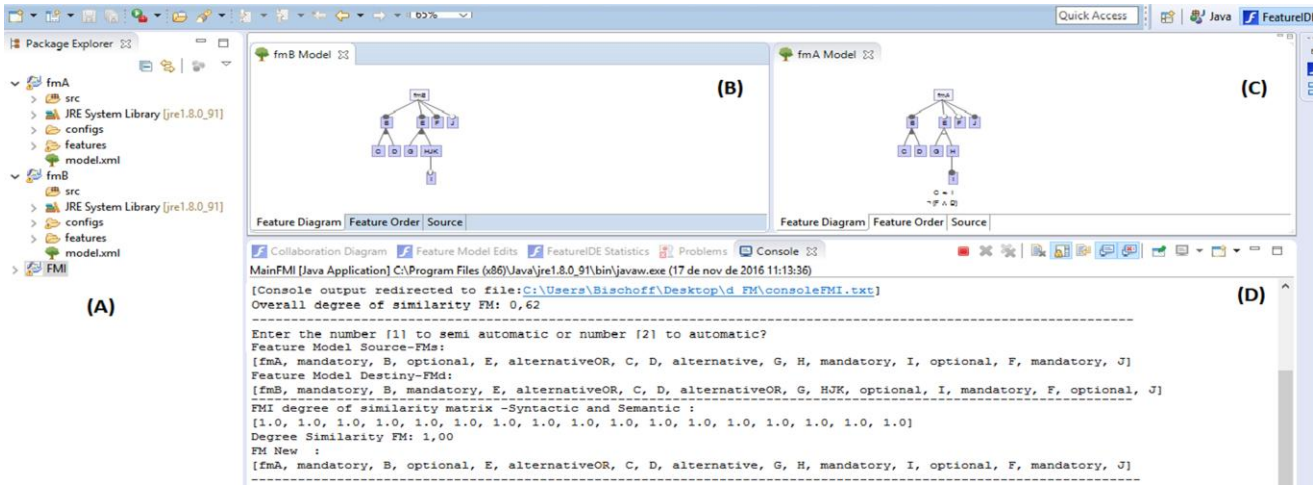

Fig. 5. FMIt tool

Package Explorer (A). For each new modeling project that is created, there is a need to create and import files that are used during the modeling process. The package explorer has as central functionality to allow the organization of the projects of the feature models, as well as to accommodate the integration tool.

Modeling View (B-C). The models that are created must necessarily be visualized with the objective of meeting two basic requirements when using models: comprehensibility and communication. Faced with this need, modeling view allows developers to view and edit models interactively. The feature models are accommodated in conjunction with the FMIt tool, to assist in exploring the inconsistencies when they exist, in this example we have two models of features, FMA and FMB to be integrated.

Console View (D). Once the models have been created or imported, an overview of the distribution of the present features is displayed and can be executed through the console, consisting of the usability of the tool.

5. CONCLUSIONS AND FUTURE WORK

This paper introduced a flexible, component-based architecture for supporting the development of model integration techniques, and an intelligible model integration workflow for aiding designers to comprehend the crucial integration activities and their relationships more properly. We also reported the FMIt tool, integration tools defined based on the FMIt-Arch. The preliminary results have indicated that the proposed architecture is able to support the development of integration tools for Feature Models and assist the development teams in their decision-making during the integration process. Studies are still required, other than case study presented, to check their usefulness and applicability in the academic and industrial process with the purpose of investigating its efficiency, effectiveness, and its effectiveness, seeking to hone the proposed technique.

The future investigations should seek to answer some questions such as: (1) do designers invest significantly more effort to develop a new integration technique than derive one from FMIt tool? (2) How effective is FMIt to combine realistic, semantically richer design models? (3) How do development teams observe the benefits of the integration process? Lastly, this work represents a first step in a more ambitious agenda on better supporting the elaboration of model composition techniques.

6. ACKNOWLEDGMENTS

This work was funded by CNPq Universal Project 14/2013 (grant number 480468/2013-3), Brazil.

REFERENCES

[1] I. Sommerville, Software Engineering. Addison Wesley, 9 edition, 2011.
[2] L. Bass, P. Clements, and R. Kazman. Software Architecture in Practice. Addison-Wesley, 1998.
[3] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson. "Feature-oriented domain analysis

(FODA) feasibility study." No. CMU/SEI-90-TR-21. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 1990.

[4] [4] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh. "FORM: A feature-oriented reuse method with domain-specific reference architectures." Annals of Software Engineering Vol. 5, No.1, 1998, pp. 143-168.

[5] [5] K. Czarnecki, S. Helsen, and U. W. Eisenecker. "Formalizing Cardinality-Based Feature Models and their specialization." Software process: Improvement and practice, Vol. 10, No.1, 2005, pp. 7-29.

[6] [6] J. Favaro, and S. Mazzini. "Extending FeatuRSEB with concepts from systems engineering." International Conference on Software Reuse. Springer Berlin Heidelberg, 2009.

[7] [7] M. Eriksson, J. Börstler, and K. Borg. "The PLUSS approach–domain modeling with features, use cases and use case realizations." International Conference on Software Product Lines. Springer Berlin Heidelberg, 2005, pp. 33-44.

[8] [8] D. Batory. "Feature Models, Grammars, and Propositional Formulas." In Proc. Int'l Software Product Line Conference, vol. 3714 of LNCS, Springer, 2005, pp. 7–20.

[9] [9] D. Benavides, S. Segura, and A. Ruiz-Cortés. "Automated analysis of feature models 20 years later: A literature review." Information Systems, Vol. 35, No. 6, 2010, pp. 615-636.

[10] [10] K. Czarnecki and U. W. Eisenecker. "Generative Programming: Methods, Tools, and Applications." ACM Press/Addison Wesley, 2000.

[11] [11] S. Apel, and C. Kästner. "An Overview of Feature-Oriented Software Development." Journal of Object Technology, Vol. 8, No. 5, 2009, pp. 49-84.

[12] [12] M. Acher, P. Collet, P. Lahire and R. B. France. "Comparing approaches to implement feature model composition." European Conference on Modelling Foundations and Applications. Springer Berlin Heidelberg, 2010, pp 3-19.

[13] [13] M. Acher, P. Collet, P. Lhire and R. B. France. "Composing feature models." International Conference on Software Language Engineering. Springer Berlin Heidelberg, 2009, pp. 62-81.

[14] [14] T. Berger, D. Lettner, J. Rubin, P. Grunbacher, A. Silva, M. Becker, M. Chechik and K. Czarnecki. "What is a feature?: a qualitative study of features in industrial software product lines." Proceedings of the 19th International Conference on Software Product Line. ACM, 2015, pp. 16-25

[15] [15] A. Classen, P. Heymans, and P. Schobbens. "What's in a feature: A requirements engineering perspective." International Conference on Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2008, pp. 16-30.

[16] [16] D. Beuche, and M. Dalgarno. "Software product line engineering with feature models." Overload Journal, vol. 78, 2007, pp. 5-8.

[17] [17] T. Thomas, D. Batory, and C. Kastner. "Reasoning about edits to feature models." 2009 IEEE 31st International Conference on Software Engineering. IEEE, 2009, pp 254-264.

[18] [18] S. Apel, J. M. Atlee, L. Baresi and P. Zave. "Feature interactions: the next generation". Dagstuhl Reports, Vol. 4, No. 7, 2014.

[19] [19] T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Bcker, K. Czarnecki and A. Wasowski. "A survey of variability modeling in industrial practice." Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems. ACM, 2013, pp.7.

[20] [20] S., Sergio, D. Benavides, A. Ruiz-Cortés. "Automated merging of feature models using graph transformations." Generative and Transformational Techniques in Software Engineering II. Springer Berlin Heidelberg, 2008. pp.489-505.

[21] [21] K. Farias, A. Garcia, J. Whittle, C. v. F. G. Chavez and C. Lucena. "Evaluating the effort of composing design models: a controlled experiment." Software & Systems Modeling, Vol. 14, No. 4, 2015, pp. 1349-1365.

[22] [22] K. Farias and T. C. Oliveira. "A guidance for model composition." International Conference on Software Engineering Advances - ICSEA. IEEE, 2007. pp.27-27.

[23] [23] S. Krieter, R. Schröter, T. Thüm, W. Fenske and G. Saake. "Comparing algorithms for efficient feature-model slicing." Proceedings of the 20th International Systems and Software Product Line Conference. ACM, 2016, pp.60-64.

[24] [24] K. Oliveira, K. Breitman, and T. C. Oliveira. "A Flexible Strategy-Based Model Comparison Approach: Bridging the Syntactic and Semantic Gap." J. UCS, Vol. 15, No. 11, 2009, pp. 2225-2253.

[25] [25] Eclipse Platform, www.eclipse.org, Accessed June 22, 2016.

[26] [26] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake and T. Leich. "FeatureIDE: An extensible framework for feature-oriented software development." Science of Computer Programming, Vol. 79, 2014, pp. 70-85.

[27] [27] K. Farias. "Empirical evaluation of effort on composing design models." Ph.D. thesis, Department of Informatics, PUC-Rio, Rio de Janeiro, Brazil, 2012.

[28] [28] K. Farias, L. Gonçales, M. Scholl, T. C. Oliveira and M. Veronez, "Toward an Architecture for Model Composition Techniques", The 27th International Conference on Software Engineering and Knowledge Engineering, Pittsburgh, PA, USA, 2015, pp. 656-659.

[29] [29] K. Farias, A. Garcia, J. Whittle, C. Lucena. "Analyzing the Effort of Composing Design Models of Large-Scale Software in Industrial Case Studies." In: 16th International Conference on Model-Driven Engineering Languages and Systems (MODELS'13), pp. 639-655, Miami, USA, September 2013.

[30] [30] K. Farias, A. Garcia, C. Lucena. "Evaluating the Impact of Aspects on Inconsistency Detection Effort: A Controlled Experiment." In: 5th International Conference on Model-Driven Engineering Languages and Systems (MODELS'12), Vol. 7590, pp. 219-234, Innsbruck, Austria, 2012.

[31] [31] K. Farias, "Empirical Evaluation of Effort on Composing Design Models." In: 32nd ACM/IEEE

International Conference on Software Engineering, Doctoral Symposium, Vol. 2, pp. 405-408, Cape Town, South Africa, 2010.

[32] [32] E. Guimarães, A. Garcia, K. Farias. “Analyzing the Effects of Aspect Properties on Model Composition Effort: A Replicated Study.” In: 6th Workshop on Aspect-Oriented Modeling at MODELS'10, Oslo, Norway, 2010.