Three strategies to Specify Multi-Instantiation in Product Lines

Raouia Triki, Camille Salinesi, Raúl Mazo
Centre de Recherche en Informatique (CRI)
University of Paris1-Panthéon Sorbonne
Paris, France
{raouia.triki; camille.salinesi}@univ-paris1.fr, raulmazo@gmail.com

***Abstract*—Product line engineering uses product line models to define the valid combinations of elements in a product and to configure them. Several modeling languages have been proposed to represent product line models. These languages have limits and they do not always fit the requirements of the context. For instance, in the industrial context, product lines contain often multi-instantiated features which are more difficult to model. This paper reports lessons learned during a project with an electronic supplier company called Rexel. Our objective in this study was to find out the appropriate modeling languages to model multi-instantiation, as required the Rexel's Electric Board that contains many multi-instantiated features. This paper presents (i) how three categories of modeling languages have been used in industry to model the multi-instantiation concept of an electric board; (ii) the limits and difficulties encountered with each category of modeling languages; and (iii) modeling strategies to handle multi-instantiation with each kind of language.**

Keywords—product lines; variability; modeling languages; multi-instantiation

I. Introduction

Product Line Engineering (PLE) is an emerging paradigm that enables developing products by reuse of artifacts from a product line. A Product Line (PL) is "a set of systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way"[1]. New products are generated with a configuration process that tries to match users' requirements to domain requirements. The variability of domain requirements is usually represented by means of a Product Line Model (PLM). Variability is "the ability of a system or artifact to be configured customized, extended, or changed for use in a specific context" [26]. Thus, a PLM represents, in an intensive manner, the collection of products that belongs to the product line. This representation comports variability constrains over the different artifacts of the domain that is being represented. Optionality, exclusion and multi-instantiation are some examples of these variability constraints in PLMs.

This paper focuses in how to represent and how to deal with multi-instantiation (i.e., the possibility to use multiple times an artifact in a particular product [6] [7]) in PLE. Multi-instantiation is not a new concept in PLE and it has been used several times in literature. For instance, there are modeling languages such as Cardinality-based Feature Models (CFM) [8], Textual Variability Language (TVL) [4] and a flavor of UML class diagrams used to represent PLs [5] that include the concept of multi-instantiation. These languages can be grouped in three categories: (1) category of modeling languages specifying constraints on sets of non-predefined instances, (2) category of modeling languages including built-in concepts to specify multi-instantiation, and (3) category of modeling languages without cardinality. The first category corresponds to the modeling languages that specify constraints on sets of non-predefined instances. Cardinality-based Feature Models (CFM) [8] is an example of language belonging to this category. CFM is an extension of the FODA language [14] and allows specifying individual cardinalities for each feature and group cardinalities for bundles of features. Textual Variability Language (TVL) [4] is a text-based feature modeling language and is another example of language that belongs to the first category. The second category of modeling languages includes built-in concepts to specify multi-instantiation The UML-Class flavor proposed by Clauss [5] is an example of language belonging to the second category. This category of diagrams is used to describe the structure of the system in terms of classes and their relationships. The third approach is a category of modeling languages without cardinality. The Feature-Oriented Domain Analysis (FODA) method [15] that represents the features of a particular domain and the relationships among them; and the Orthogonal Variability Model (OVM) [20] that provides a cross-sectional view of the variability across all software development artifacts, are examples of languages that belong to the third category.

These three modeling categories support the concept of multi-instantiation in different manners. That is why we are interested in the "best manner" to model the multi-instantiation in PLE. In order to identify that "best manner" to model multi-instantiation we compared the aforementioned categories in the context of a research action process. As a result, we identified the limitations and strengths of each of these categories. We analyzed these results and we identified the most suitable category of modeling languages to represent the concept of multi-instantiation and its associated constraints in the context of real product line engineering projects relying on an industry case: the product line developed by an electronic supplier company called Rexel[2]. Rexel supplies tailor-made electrical equipment and services to all professionals involved in the construction, maintenance, renovation, and development of all kinds of buildings and infrastructure.

978-1-4673-6630-4/15/$31.00 ©2015 European Union

[1] http://www.sei.cmu.edu/. SEI web page

[2] http://www.Rexel.com

In particular, this paper presents (i) how three categories of modeling languages have been used in industry to model the multi-instantiation concept of an electric board; (ii) the limits and difficulties encountered with each category of modeling languages; and (iii) modeling strategies to handle multi-instantiation with each kind of language.

The remainder of the paper is organized as follows. Section 2 presents the Rexel case. Section 3 presents the methodology applied for PL modeling. Then, Section 4 presents the Rexel PL modeling activity with the first category of modeling languages; Section 5 presents the Rexel PL modeling activity with the second category; and Section 6 presents the Rexel PL modeling activity with the third category. Examples tackled in sections 4, 5 and 6 present different parts of the same case study described in Section 2. Section 7 provides an analysis of the results. To finish, Section 8 presents related works and Section 9 presents the conclusions of the experiment.

II. The Electric Board Case Study

A. Context

Rexel is an electronic supplier company with over 2500 product references at every sales outlet, and manages their products sales by using product catalogs that present products extensively one by one. Products are presented in the catalog without mentioning in what type of site they should be used, or if their use corresponds to the French norm for electricity. Besides, dependencies and incompatibilities, among different products presented in the catalog are not explicitly defined and they are therefore difficult to maintain. Thus, Rexel decided to represent their products in an intensive way by means of product line models.

B. Electric Board Description

Electric boards are usually used for a particular habitat; for instance, an apartment, hospital or enterprise. This example was chosen for its simplicity with respect to other electric equipments and because it is a real and complete industrial product line that contains multi-instantiated elements. Rexel's electric board (see Figure 1) consists in a collection of electric equipment pieces that distributes, takes control and protects the individual circuits that feed each room. It also guarantees security of people and the entire electrical installation according to the NFC15-100 norm. This norm provides the regulations of design, construction and maintenance of electrical installations in France.

An accommodation must have room(s), kitchen(s), living room(s), WC(s) and an electrical box. An electrical box has optional components such as a socket piece, a door, a vertical comb busbar, an horizontal comb busbar which allows horizontal feed of rows, circuits or groups of circuits, electrical pieces of equipment of lighting control, electrical pieces of equipment of heating control and electrical pieces of equipment for programming of circuits. The lighting control includes a contactor or a remote control switch and a clock timer. The heating control includes a timer switch and a pilot wire administrator. The programming of circuits includes a remote dimmer switch and a switch clock. The horizontal comb busbar supplies electrical equipments of protection such as Surge Protection Device (SPD) to protect electrical devices from voltage spikes and differential switches to protect people from the risks of electric shock. There is one differential switch by horizontal comb busbar. The horizontal comb busbar also supplies a collection of electrical pieces of equipment such as breakers, fuses and Ground Fault Circuit Interrupters (GFCI) to protect circuits from short-circuit. Only GFCI protects both circuits and people. Each horizontal comb busbar always includes at least one circuit. There are several types of circuits, for instance, lighting circuit, PC16A circuit and dedicated circuits. Dedicated circuits are used for washing machines, dishwashers, dryers, ovens, freezers, mechanically controlled ventilation circuits, cooking circuits, water heater circuits and shutters circuits. Thus, there is one differential switch or one GFCI for each horizontal comb busbar in order to protect the corresponding circuits. The electric board must take into account constraints defined by the NF C15-100, which establishes rules for the different electric pieces of equipment.

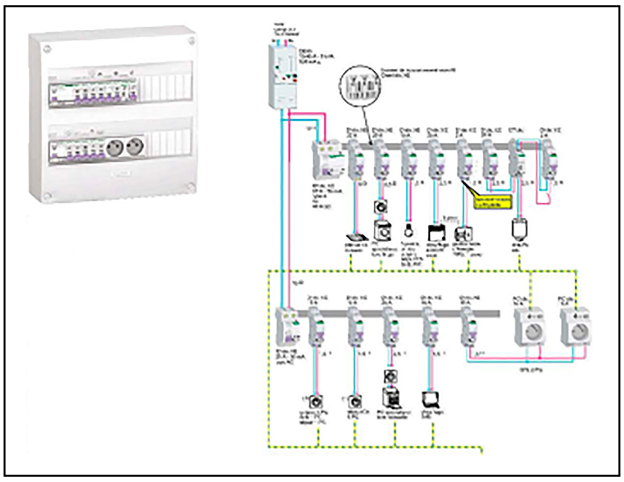

Fig. 1. Example of a Rexel's Electric Board

III. Methodology

The work reported in this paper was achieved through an "Action Research" methodology [19]. Action Research aims to improve practice by solving real problems and is conducted in order to investigate contemporary phenomena in their natural context [16]. Susman [28] has developed a detailed action research model with the different phases to be carried out in each cycle of the action research process. The Action research model consists in a certain number of cycles and five phases for each cycle (cf. Figure 2). As we were in the situation that Rexel really needed help to handle the non-trivial PL specification issue of multi-instantiation, we found the action research method suitable for this project.

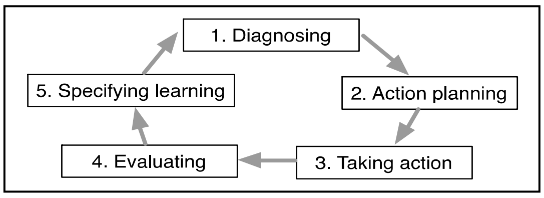

Fig. 2. Research Action Model [28]

The five phases of the action research cycle can be described as follows:

1) Diagnosing: defining the problem of how modeling the multi-instantiation concept in a product line.

2) Action planning: three categories of languages are considered to model the multi-instantiation.

3) Taking action: the first category that consists in specifying collection of constraints for non-predefined instances is used for modeling the multi-instantiation constraints.

4) Evaluating: an evaluation of the first category is carried out by observing whether the multi-instantiation is well modeled or not.

5) Specifying learning: interpretation of the general findings by specifying the strengths and limitations of the first category of languages.

Three cycles were performed: one for each category of language that was addressed in the project. At the end of each cycle, the problem was reassessed, and another cycle started with lessons learned in mind.

IV. First Cycle

The first category of modeling languages specifies constraints on sets of non-predefined instances. For example, TVL models the multi-instantiation by adding an explicit cardinality to the concerned feature [4]. TVL has an explicit enumeration type and also allows expressing constraints over the number of multi-instantiated features [4]. For instance, a "differential switch" has a type and a caliber, where the type can be "AC" or "Asi" and the caliber can be "25 amps", "40 amps" or "63 amps". This can be represented using different attributes types: enumeration for the type of the attribute that specifies differential switch, and integer for the caliber of differential switch.

```
enum diffSwitches in {AC, Asi};

Accomodation {
  int accomodArea;
  accomodArea <= 35 -> count( diffSwitch.filter(
   caliberDiffSwitch == 25 && typeDiffSwitch == AC))>= 1;

  accomodArea > 35 && accomodArea <= 100 -> count(
   diffSwitch.filter (
   caliberDiffSwitch == 40 && typeDiffSwitch == AC))>= 2;

  accomodArea > 100 -> count( diffSwitch.filter(
   caliberDiffSwitch == 40 && typeDiffSwitch == AC))>= 3;

  DiffSwitch {
    int caliberDiffSwitch in {25, 40, 63};
    diffSwitches typeDiffSwitch;
}},
```

Fig. 3. Excerpt of constraints specified with TVL

To define constraints like "For an accommodation area not exceeding 35m2, at least one differential switch with rated current of 25amps and with type of AC must be expected" can be expressed with TVL by using the functions count() and filter() applied to attributes. In fact, for each accommodation area, it is possible to define the number of differential switches that must be installed and their caliber and types. For example, Figure 3 shows constraints applied for accommodation area between 35m2 and 100m2; and constraints applied for accommodation area greater than 100m2. It is also possible to specify that each circuit must have at least one protection with a Breaker, GFCI or Fuse, using the or-decomposition constraint called "group someOf" and the xor-decomposition constraint called "group oneOf", as shown in Figure 4.

```
enum breakers in {AC, A, ASi};
enum gfcis in {AC, Asi};
enum fuses in {A, Asi};

Circuit [1..*] {
  ProtectionCircuit;
  group someOf {
    ProtectionCircuit [1..1] {
      group oneof {
        Breaker {
         int caliberBreaker in {2,6,10,16,20,25,32};
         breakers typeBreaker;
        },

        Gfci {
          int caliberGfci in {10, 16, 20, 25, 32};
          gfcis typeGfci;
        },

        Fuse {
          int caliberFuse in {10, 16, 20, 25};
          fuses typeFuse;
        }
      }
    }
  }},
```

Fig. 4. Excerpt of the Electric Board modeled with TVL

Likewise as shown in Figure 5, the cardinality-based feature model language allows to represent the fact that an horizontal comb busbar has at least one "*Protected circuit*" which in turn has at least one "*Circuit*" of type "*Lighting circuit*", "*PC 16A Circuit*", "*Dedicated Circuit*", or "*Other circuit*".

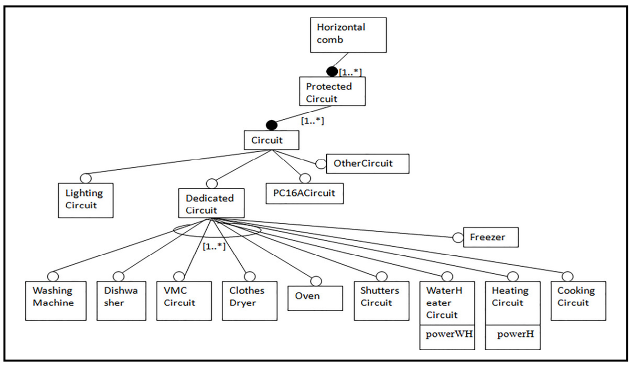

Fig. 5. Excerpt of an Electric Board modeled with CFM

Constraints applied over sets like « There must be at least three dedicated circuits for powering devices such as: washing machine, dishwasher, dryer, oven, and freezer » can be expressed with a CFM. However due to its tree structure, the cardinality-based FM language cannot express constraints such as "When the heating circuits and electric water heaters, whose total power exceeds 8kVA, are placed downstream of the same differential switch, an AC type differential switch and 40amps caliber must be replaced by an AC type differential switch and 63amps caliber", because it is not possible, with CFM, to calculate the sum of the powers of "heating circuit" and "water heater circuit". Moreover, far from the multi-instantiation, it is notable that CFMs do not make distinction between "concrete" and "abstract" features. For instance, "Protected Circuit" in Figure 5 is an abstract feature and "Lighting Circuit" is a concrete feature. Apart from what we observed in this cycle about feature multi-instantiation, we also observed that, once the electric board modeled as a feature model, it was difficult to know (and remember several days after) if certain features were represented as characteristics of the product line or just as decisions to take in the configuration process.

V. SECOND CYCLE

The second category consists in the modeling languages that include built-in concepts to specify multi-instantiation. This is, for instance, the case of the UML dialect proposed by Clauss [5] to specify with a concept of cardinality how many times a class can be instantiated. In this notation, classes are modeled using stereotypes to specify whether a class is a "variation", a "variant" or "optional". For instance, a "*circuit*" is a "variation" (cf. Figure 6) of: "*lighting circuit*", "*PC 16A circuit*", "*dedicated circuit*" or "*other circuit*", which are represented as "*variants*" by means of the inheritance concept of the UML class diagrams.

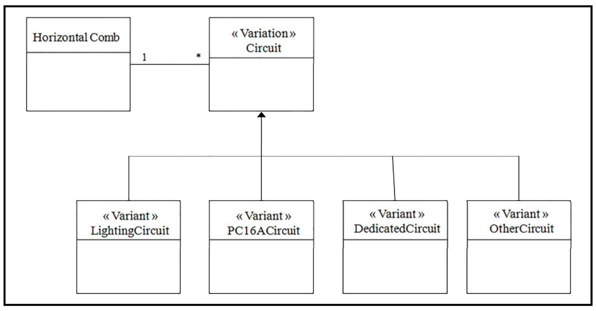

Fig. 6. Excerpt of Electric Board modeled with UML class diagrams

The limitation of this kind of notation is that all variations cannot be specified. For instance, it is not possible to model the different variations of the "dedicated circuit" because a variation cannot have in turn variations. Thus, constraints on non-modeled variants cannot be expressed.

VI. THIRD CYCLE

The modeling languages studied here do not offer an explicit concept to represent multi-instantiation. For instance, the FODA and OVM languages do not include the multi-instantiation concept. There are two possible strategies to handle the multi-instantiation phenomenon when using this category of language: (i) to explicitly specify all the possible instances in the product line model; and (ii) to use attributes to enumerate the number of times a feature should be instantiated.

A. Strategy (1): Explicit Specification of all Instances

This strategy supports multi-instantiation by explicitly specifying all the possible feature instances in the product line model; for example, each instance of "*breaker*" can be represented as a feature in the FODA model specifying its type and its caliber (cf. Figure 7) i.e. a *breaker* feature with "*A*" type and "*32*" caliber, a *breaker* feature with "*A*" type and "*16*" caliber, a *breaker* feature with "*AC*" type and "*16*" caliber, etc.

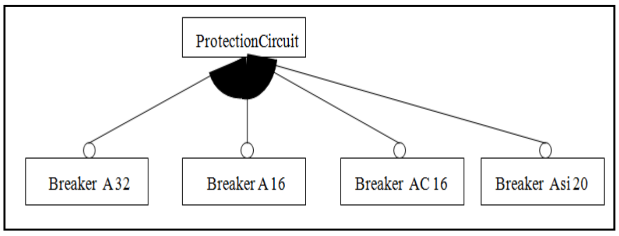

Fig. 7. Excerpt of Electric Board modeled with FODA language

In addition to the potentially very large number of features instances to represent, it is very hard to specify the constraints between these features. For example, the constraint "*VMC circuits must be protected with a maximum rated current of 16amps circuit breaker"* can be confusing because it is not clear for which instance the constraint should be applied. Thus, questions like "should the constraint be applied to the instance *Breaker A 16* or to the instance *Breaker AC 16*?" often arise when this strategy is used.

B. Strategy (2): Attributes to Enumerate Feature Instantiations

Another way to circumvent the lack of cardinalities is to record the number of instances of features. The idea is to represent the number of times that a feature can be instantiated by means of an attribute of feature or variant. For example, an electric board configuration can contain many breakers, to protect the different existing circuits, with different types and different calibers. Then, an attribute, called "*number*" for instance, can be added to the feature, variant or variation point to represent the number of instances of that element into a particular product. Figure 8 shows an extract of the *electric board* modeled with the OVM language in which the variation point called "*breaker*" can be instantiated many times: the attribute "*NbrBreaker*" saves the number of breakers instantiated in one configuration. For instance, in one configuration, there may be (i) two *A* type and *16amps* caliber breakers; and (ii) one *AC* type and *32amps* caliber breaker. Thus, the attribute "NbrA" saves the number of occurrences according to the variant "A" that is 2. The same for the attribute "NbrAc" that is equal to 1 occurrence, the attribute "Nbr16" that is equal to 2 occurrences, and the attribute "Nbr32" that is equal to 1 occurrence. Thus, the attribute "NbrBreaker" is equal to 3 occurrences that is the sum of all breakers contained in this configuration.

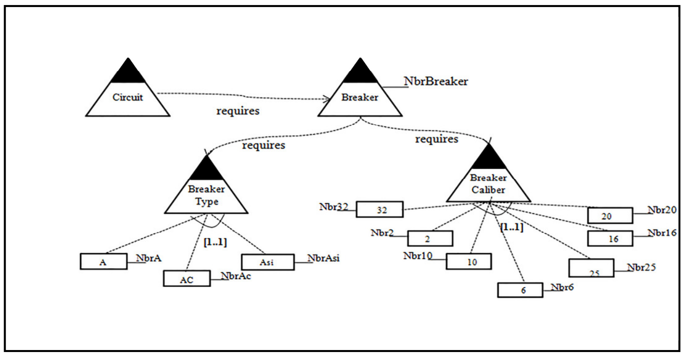

Fig. 8. Excerpt of Electric Board modeled with OVM language

Even if it is possible to represent the number of instances that a domain element can have with attributes, there are still limitations. Indeed, with this strategy, it becomes impossible to consider different instances individually. For instance, it is not possible to have both breaker calibers "*16*" and "*32*". Furthermore, some constrains like "*In the case of T1 type of housing, and if the equipment is not provided at least three specialized circuits are planned including a 32A circuit and two 16A circuits"* cannot be expressed.

VII. Analysis

This paper presents three categories of modeling languages that can be used to represent multi-instantiation in the context of product lines and shows how to use them with real case. At the end of our experiment we remarked that the first category of languages has the syntax richness to represent the electric board multi-instantiated elements and some constraints over them. We also noticed that the second category of languages has still limitations to specify some variations and constraints. Regarding the third category, we remarked that the use of explicit instances make it difficult to specify constraints on instances because it is not possible to distinguish which instance is concerned by the constraint. However, the use of attributes to count the number of instances is somewhat better than the previous strategy despite its weakness to express all constraints. In fact, with such strategy, it is not possible to specify in one configuration different instances with different values. Although the third category includes the more popular languages in PLE, it is also the least appropriate to represent multi-instantiation because (i) the syntax of the languages of this category does not define feature cardinalities, and (ii) these languages do not allow expressing constraints on feature instances. However, to choose one category of modeling languages depends on the context and the objectives that the engineers intend to achieve.

VIII. Related works

The literature proposes several comparative analyses of product line modeling languages. For instance, Djebbi & Salinesi [10] presents a comparative framework for evaluating FM languages intended to be used in real scenarios. Sinnema & Deelestra [24] proposes a classification framework of six variability modeling techniques: CBFM [10], COVAMOF [25], VSL [2], conIPF [13], Pure::Variants [3] and Koalish [4] based on a defined set of criteria. In the same context of comparative analysis between modeling languages, Heymans et al. [12] present a method for evaluating the quality of the semantics of feature notations by means of the SEQUAL framework [17]. However, none of these studies addresses the multi-instantiation problem or seek how to model this concept. Czarnecki et al. [7] carried out a comparison between feature modeling (FM) and decision modeling (DM) by identifying commonalities and differences between them and using ten dimensions such as (i) data types, and (ii) dependencies and constraints. Thus, they state like this paper do it that there are some FM (e.g, CFM) and DM (e.g., V-Manage [29]) languages that support the multi-instantiation and that some FM languages support composite types like the TVL that has an explicit enumeration type. Cordy et al. [6] propose the TVL as a solution for the multi-instantiated features without doing a comparative study between modeling languages as has been done in this paper. In particular, they define the cardinality of a feature to make possible the use of a feature multiple times in one configuration and they define constraints that can be applied on instances of a feature such as "*forall*" and "*exists*". Recently, Mazo [18] reports an empirical study in which he analyses the advantages and limitations of the feature modeling notation when it is used to represent industrial product lines with multi-instantiation and complex constraints.

IX. Conclusion

This paper presents how three categories of modeling languages can be used for modeling the multi-instantiation in product lines. Each category has strengths and weaknesses (cf. Table 1). Some languages are more appropriated for representing multi-instantiation than others. Thus, choosing an approach ultimately depends on the context and work targets.

As perspectives, it would be interesting to carry out a study that seeks ways to use advantages of TVL, which well model the multi-instantiation, to enhance FODA, which is the most widespread product line modeling language.

TABLE I. SUMMARY OF MULTI-INSTANTIATION MODELING CATEGORIES

Approaches	Advantages	Drawbacks	Languages

Approach 1		-Present multi-instantiated features plainly using feature cardinality -Use an explicit enumeration type for some languages like the TVL -Constraints on multi-instantiated features are easily expressed	There are some languages in this category that have some formal defects like the cardinality-based feature model language.	-TVL -Cardinality based feature model -Constraint programming over finite domain
Approach 2		Present multi-instantiation clearly using cardinality and stereotypes to instantiate classes	-Do not specify group features, only with OCL constraints. -Some variations and constraints cannot be specified	-UML -SysML -Ontologies
Approach 3	Explicitly specifying all possible instances	-All instances are explicitly modeled	-Too many instances presented in the model -Constraints are very complex to specify: expressing constraints on instances can be confusing	-FODA -OVM -DOPLER
	Enumerating the number of feature instantiations	-Attributes are simple to use	-It is impossible to have different instances -There are constraints that cannot be expressed: constraints on different instances	

REFERENCES

[1] T. Asikainen, T. Soininen, T. Männisto, "A Koala-based approach for modelling and deploying configurable softwareproduct families", 5th Workshop on Product Family Engineering, Springer Verlag Lecture Notes on Computer Science, vol. 3014, pp. 225-249, May 2004.

[2] M. Becker, "Towards a general model of variability in product families", Proceedings of the 1st Workshop on Software Variability Management, Groningen, Netherlands, February 2003.

[3] D. Beuche, H. Papajewski, W. Schröder-Preikschat, "Variability management with feature models", Science of Computer Programming, vol. 53 no°3, pp. 333-352, 2004.

[4] A. Classen, Q. Boucher, P. Heymans, "A text-based Approach to Feature Modelling: Syntax and semantics of TVL", Sci.Comput.Program,vol. 76, pp. 1130-1143, 2011.

[5] M. Clauss, "A proposal for uniform abstract modelling of feature interactions in UML", In Proceedings of the European Conference on Object-Oriented Programming, Workshop Feature Interaction in Composed System, 2001.

[6] M. Cordy, P.Y. Schobbens, P. Heymans, A. Legay, "Beyond Boolean Product-Line Model Checking: Dealing with Feature Attributesand Multi-Features", ICSE, pp. 472-481, 2013.

[7] K. Czarnecki, P. Grunbacher, R. Rabiser, K. Schmid , A. Wasowski, "Cool Features and Tough Decisions: A comparison of Variability Modeling Approaches", Vamos, 2012.

[8] K. Czarnecki , S.Helsen , U.W. Eisenecker, "Formalizing cardinality-based feature models and their specialization", Software Process Improvement and Practice, vol. 10, pp. 7–29, 2005.

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

[10] O. Djebbi, C. Salinesi, "Criteria for Comparing Requirements Variability Modeling Notations for Product Lines", CERE workshop at RE Conference, USA, pp. 20 – 35, 2006.

[11] M. Griss, J. Favaro, M. Allesandro, "Integrating Feature Modeling with RSEB", Proceedings of the Fifth International Conference on Software Reuse, Vancouver, BC, Canada, 1998.

[12] P. Heymans, P.Y. Schobbens, J.C. Trigaux, Y. Bontemps, R. Matulevicius, and A. Classen, "Evaluating formal properties of feature diagram languages", Software IET, vol. 2, pp. 281–302, 2008.

[13] L. Hotz, T. Krebs, K. Wolter, J. Nijhuis, S. Deelstra, M. Sinnema, J. MacGregor, ConWguration in Industrial Product Families –The ConIPF Methodology, IOS Press, ISBN 1-58603-641-6, July 2006.

[14] K. Kang, K. Lee, J. Lee, "FOPLE – Feature Oriented Product Line Software Engineering: Principles and Guidelines", Pohang University of science and technology, 2002.

[15] K. Kang, S. Cohen, J. Hess , W. Novak, S. Peterson, "Feature–Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, 1990.

[16] E. Koshy, V. Koshy, H. Waterman, "Action research in healthcare", Thousand Oaks, CA: Sage, 2011.

[17] J. Krogstie, "Using a semiotic framework to evaluate UML for the development of models of high quality", Unified modeling Language: Systems analysis, design and development issues, IDEA Group Publishing, pp.89-106, 2001.

[18] R. Mazo. Avantages et limites des modèles de caractéristiques dans la modélisation des exigences de variabilité. Journal "Génie Logiciel", No. 111, Paris-France, pp. 42-48, Dec. 2014.

[19] R. O'Brien, "An Overview of the Methodological Approach of Action Research", Roberto Richardson (Ed.) Theory and Practice of Action Research, Brazil, Universidade Federal da Paraíba, 2001.

[20] K. Pohl, G. Bockle, F. J. Van der Linden, "Software Product Line Engineering: Fundations, Principles and Techniques", Springer–Verlag, Berlin, DE, 2005.

[21] I. Rodrigues, N. Matos, S. Abreu, R. Deneckere, D. Diaz, "Towards constraint-informed information systems", RCIS, 2013.

[22] P. Runeson, M. Höst, "Guidelines for conducting and reporting case study research in software engineering, Empirical Software Engineering", vol. 14, pp. 131-164, 2009.

[23] P. Schobbens, P. Heymans, J. C. Trigaux, "Feature diagrams: A survey and a formal semantics", *Requirements Engineering, 14th IEEE international conference*, pp. 139-148, September, 2006.

[24] M. Sinnema, S. Deelstra, "Classifying variability modeling techniques", Information and Software Technology, vol. 49, pp. 717-739, 2007.

[25] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, "COVAMOF: A framework for modeling variability in software product families", Proceedings of the third Software Product Line Conference, Springer Verlag Lecture Notes on Computer Science, vol. 3154, pp. 197-213, August 2004.

[26] M. Sinnema, S. Deelstra, J. Nijhuis, J. Bosch, "Managing Variability in Software Product Families", Proceedings of the 2nd Groningen Workshop on Software Variability Management , 2004.

[27] D. Streitferdt, "Family-Oriented Requirements Engineering", PhD Thesis, Technical University Ilmenau, 2004.

[28] G. I. Susman, "Action Research: A Sociotechnical Systems Perspective", Ed. G. Morgan. London: Sage Publications, pp. 95-113, 1983.

[29] Europeen Software Institute Spain and IKV++ Technologies AG Germany, MASTER: Model-driven Architecture inSTrumentation, Enhancement and Refinement, 2002.