International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-6, August 2013

Managing Variability in Model Transformations for Model-Driven Product Lines

Naoufel Kraiem, Imen Boudich, Zuhoor Al Khanjari, Yassine Jamoussi

Abstract -The model-driven engineering is a theme in full expansion in both the academic and industrial world. It is a generative form of engineering in which all or part of an application is generated from templates. In this article, we studied the contribution of Model Driven Engineering (MDE) in the field of management of variability in Software Product Lines (SLPs). Indeed, the goal of software product lines is to minimize the cost of developing software in a particular application domain. This minimization is due to the design of reusable elements and not to the development of each program separately. We consider an approach to model-driven engineering and engineering fields (Product Lines) as two generative approaches that aim to automate the software development. Our goal is to create an approach for handling product lines using MDE.

Keywords: UML, Model Driven Engineering, Software Product Line, Variability.

I. INTRODUCTION

When dealing with the growing complexity of huge size software, software engineering aims to make software development an industrial process. Its main objective is to improve the reliability and quality of the software, when we increase productivity and efficiency of its production process. So, the question is no longer to develop a single software, but rather to develop a line of software that takes into account the factors of change and to minimize the cost and time of implementation. Generative approaches have emerged to meet these needs. They provide the tools for the automation of software development

Among these approaches, there is the development approach of software product lines. Software Product Lines (SPL)[1], or software product families[2][3], are emerging as a paradigm shift towards modeling and developing software system families rather than individual systems. SPL engineering embraces the ideas of mass customization and software reuse[4].

An important aspect of Product Line Engineering is to manage variability. The latter is a key activity in the development of product lines. Indeed, it is used to group the characteristics that differentiate the products of the same family. By explicitly modeling and managing variability, software productline engineering provides a systematic approach for creating adiversity of similar products at low cost in short time and with high quality [5].

The problem with this approach lies in the design of an architecture permitting the definition of several products. Different approaches show that the software product lines (SPL) can be implemented using the model-driven engineering (MDE) and the concept of successive refinement models. The MDE start from the idea that software systems can be described by one or several models, each with a view of the system under consideration. This increases the level of abstraction and plays the role of the basis for communication between the various actors which appropriate these different views of the system. The development is automated by this series of refinements, called transformation models. SPLs can build a set of member products, which are subject to variability i.e the member products will have a varying set of features integrated. Since model transformations can be used to instantiate and integrate components of a system, they should also integrate the feature[6].

Manuscript published on 30 August 2013.
* Correspondence Author (s)
Naoufel Kraiem*, Institute of Computer Science, Nanar, Tunisia
Imen Boudich, Institute of Computer Science, Nanar, Tunisia
Zuhoor Al Khanjari, Institute of Computer Science, Nanar, Tunisia
Yassine Jamoussi, Institute of Computer Science, Nanar, Tunisia

II. THE THEME OF THE PROBLEM

The "lines of software products" approach is a transposition of the production lines in the world of software[7] .The principle is no longer to develop each software separately, but by designing from reusable components, called "artifacts". An artifact may be a requirement, a model, architecture, a software component, or just one document.

One difficulty among others with this approach lies in the design of an architecture permitting the definition of several products. As well, members of a product line are characterized by their commonalities, but also differences (called "points of variation").

The management of variability is one of the key concepts of product lines. Variability can be identified by analyzing the requirements (functional and quality requirements) for products to be developed with the line of future products. Whenever there is a conflict or exclusive common needs across a range of product requirements, the product line will be flexible enough to implement these products.

SPL engineering is a process focusing on capturing the commonalities (assumptions true for each family member) and variability (assumptions about how individual family members differ) between several software products. Models have been used for long as descriptive artifacts, and proved themselves very helpful for formalizing, sharing, and communicating ideas. Modeling variability in SPL has thus already proven itself very useful, as highlighted by the popularity of feature modeling languages and their supporting tools [4].

Despite the existence of several approaches for modeling this variability. There is a lack in the use of model-driven engineering for handling product lines. Indeed, there is a lack of a mature approach for handling product lines in applying approaches and transformations of model driven engineering. This paper suggests an approach for modeling and implementing product lines based on MDE, incorporating feature diagrams, class diagrams and coherence constraints. Section 2 presents a state of the art product lines,

MDE, the relations between the two approaches is the study of existing modeling variability product lines approaches. Section 3 describes the suggested approach for modeling product lines by the proposal of a new meta-model feature diagram. Section 4 suggests a new support plug-in eclipce LDP ensuring modeling product line till its implementation. Section 5 concludes this work and gives some perspectives.

A. Software Product Line

A software product line is a set of products sharing a common architecture and a set of reusable components [8]. The software product lines adopt a top-down approach in the development of software systems. In this approach, we proceed by identifying common requirements to all products (commonality) and what differentiates (variability) requirements. The problem with this approach lies in the design of architecture for the definition of several products. Commonalities and variables characterize the members of a product line. The management of variability in the product line is a key activity in the development of product lines.

A .1.1 Domain Engineering and Application

The engineering software product lines is defined in the literature by distinguished [9] two levels of engineering: Domain Engineering and Application Engineering.

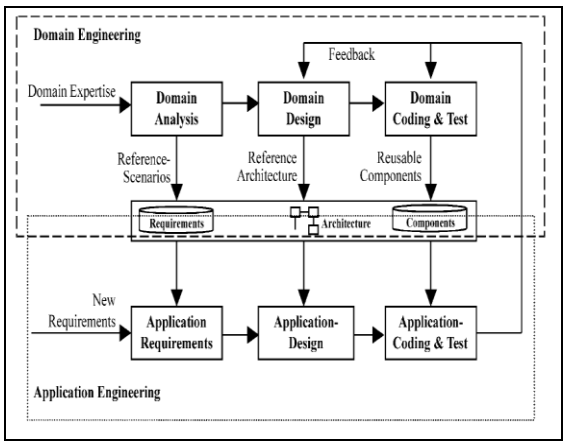

Figure 1: Engineering product lines

a. Domain Engineering

Domain engineering or (The engineering for Reuse) corresponds to the study of product line domain, identifying commonalities and variabilities between products, the establishment of generic software architecture and the implementation of this architecture. Indeed, the Domain engineering is the development and construction of reusable components known as the artifact name (specification document, model, code, etc.) That will be reused for the construction of reusable components products. For this reason, the domain engineering is considered a development for reuse. The Domain engineering has three activities[9]: analysis, design and Domain implementation.

*The domain analysis target is to study the domain of product line (health, finance, communication ...) and to identify commensalities and variability between products. There are Several methods for domain analyzing .

*The domain design is to establish a generic software architecture of the product line. [10] defining a software architecture of product line as a standard architecture that includes a set of components, connectors and constraints.

*The implementation of the domain consists of the implementation of the generic architecture defined in the domain design as components that will be reused in application engineering for the construction of each product

b. Engineering Applications

The domain engineering or engineering for reuse (Reuse by The engineering) allows you to search the optimal use to develop a new product from a product line by reducing the cost and development time and improve the quality. At this level, the results of the domain engineering are used for the derivation of a particular product. This branch corresponds to decision-making vis-à-vis the variation point.

B. Software Product Line and scoring variability

Variability is the ability of a system to be changed, customized and configured for a specific context [9]. A more goal-oriented definition of variability is also given by Bachmann and Clements [11].The Key software reuse systems, is a high degree of variability that facilitates the use of a system in a variety of contexts. It has become an important factor in the development of current systems and this is due to the large amount of options/alternatives offered by these systems. It is already known and evident in the development of product lines and the man-machine interfaces. Indeed, taking into account variation factors (technical, commercial or cultural) product lines allows to minimize costs and time embodiment.

The management of variability can also be described through a process oriented point of view [12]:

(i) identification of variability determines where variabilityis needed in the product line (list the featuresthat may vary between products),

(ii) constraining variability provides just enough flexibility for current and future system needs,

(iii) implementing variability selects a suitable variabilityrealization technique based on the previously determinedconstraints,

(iv) managing variability requires constant feature maintenanceand repopulation of variant features.

An important activity in the management of variability is to identify points of variation. A variation point can be seen as a decision point with several choices as known variants [10]. Each variant is a way of achieving variability. Indeed, it is a way of delaying design decisions. It is necessary to delay all decisions specific to a particular product. Failure to identify points of variation in the creation of the product line, may mean that the product line requires considerable maintenance activity to incorporate the necessary variability.

C. Driven engineering models

These Changes continue In today's software engineering where evolution over time has become standards, so there is a need to automatically produce the software and being adapting to the temporal and spatial changes to take into account the variability of these software.

The model-driven engineering (MDE) [13] aims to provide a framework for software development, which is seen as a series of refinements between models of the same system at different levels of abstraction. The MD Estarts with the idea that software systems can be described by one or several models; each one is a view of the studied system.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

This increases the level of abstraction and is the basis for communication between the various actors that appropriate these different views of the system. The development is automated by this series of refinements, called models transformation.

The MDE is interested in models that can be processed and used automatically [14].For example, to compose models representing different aspects of a system or to perform transformations between specification of different levels of abstraction. This automation necessarily involves models presentation. To be handled by a machine, a model must be written in a precise and clear language; which is the modeling language. It uses diagram approaches with symbols associated with names that represent the concepts of relations between them and various other annotations to represent the constraints.

D. The relationship between the two approaches

We can consider approaches of model-driven engineering and domain engineering (product lines) as two generative approaches that aim to automate the software development. For this, they can be seen as approaches to the automatic passage of a problem to a solution. We presented two generative approaches whose goal is to simplify software development for automation. They share a common idea: the use of knowledge about the problem that the software must solve. The domain engineering (product lines) is trying to capitalize the specificity of a field called business domain, and the expertise of users to produce all tools to automate development .

However, these two approaches are not completely disjoint [15]. We consider the MDE as a good instrument for the realization of the domain engineering. Indeed, the principles of MDE may be a good way for the realization of the models used in domain engineering.

Following this study, we can conclude that the common goal of these approaches is the use of knowledge related to the problem that the software must solve. In addition, we noted the importance of using MDE in order to make the product lines .

E. Modeling a Product Line:

The main objective of the variability is the adaptation of the system to the needs of users. In software systems, the variability corresponds to the different platforms on which these systems can run; the modes of interaction between the graphic interface and the user, the different features available. Variability representations must be capable of representing variability and commonalities and should be able to describe the effects/consequences of variability [16].Our study is based on the principal that handled variability in the context of existing product lines. The modeling notations variability can be graphics, text or text and graphics at a time.

E.1 Feature Diagram

Feature models (FMs) are the de facto standard for modeling variability of softwareproduct lines. The research effort is still intensive and aims at increasing the adoption of FMs in practice [17].

Feature models describe the features in a system, and allow to select a subset of features according to the selection/conjuration rules applied to the features. Typically, in the simpler case, a feature model is a tree where each sub tree is a feature [6].

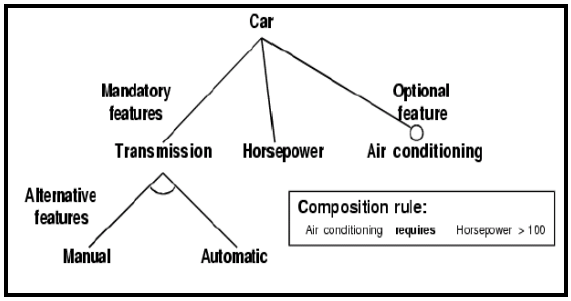

Figure 2: FODA notation

Fig. 2 shows an example of the diagram characteristics FODA notation which is considered one of the top modeling notations variability. The root of the tree represents the concept that is described. The nodes are the different characteristics of this concept.

The FODA notation represents three kinds of features:

*Mandatory features are features that are included by default in the description of a system (transmission and power for a car).

*Optional features are features that may not be included in the description of a system and are characterized by a graph in a circle (the air conditioning is optional in cars).

*Alternative characteristics are features that one alternative may be included in the description of a system and are connected by an arc indicating one feature can be selected (the transmission in a car can be manual or automatic).

We note that FODA is a clear and simple notation to master, despite that it has certain shortcomings when showing patterns of variation as the non representation of features for the choice of zero or numerous characteristics (many alternatives).

E.2 Extensions of UML diagrams:

To express variability, several studies suggest the use of use cases with some extensions.

Class diagram and sequence diagram: notation Ziadi[18]

The author of this notation proposed a modeling approach of product lines based on an extension of the use cases, class diagrams and future of UML sequence diagrams. These extensions allow to take into account the variability in a UML model. In addition to these extensions, it became necessary to define a set of architectural constraints expressed in OCL, describing consistency rules and dependencies between the elements of a product line.

From this model, many interesting problems linked to the world of product lines appear. This notation does not include consistency constraints that allow the facilitation of choice during the bypass. The derivation of products is indeed more than just a choice of options in an undetermined order.

E.3 *Examples of existing meta-models:*

In this section, we will introduce some notation of Tessier et al. [19] concerning the meta-model variability in product lines.

The authors of this notation offer a meta-model to model variability. This meta-model includes several key concepts that are explained below. A "variable element" is an element that has been specified as a variable directly by the user. In contrast, a "spread variable element" is an element that acquires the quality of being variable through its relationship with other model elements. The concept of (variationgroup) is a set of variable elements and defines a type of grouping between them.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

We took notes about this concept; see the different types of grouping identified by the authors: VariationGroupKind. To illustrate the type Alternative: a single element belonging to the group must be present in the model at any given time and appear in any product model. The problem with this meta-model is that it does not include the concept of optionality and the concept of relationship. The latter is implicitly modeled in the "variation group" notation. In order to express a simple dependency relationship between two model elements we must necessarily create a new variation group: the graphical notation could become very quickly incomprehensible, depending on whether many elements are in groups with different VariationGroupKind.

F. MDA approach and variability management:

Deelstra et al. [20] suggests the use of model-driven architecture (MDA) to manage the variability in software product lines. In this proposal, MDA is used as an approach to derive products into a particular type of software product lines (lines of configurable products).

Conceptually, a software system that is modeled according to the MDA approach specifies an application template for a family of products that implement the same functionality on different platforms. The choice for alternative platforms is a variation point in this product line. This variation point is separated from the application model so that it is no longer visible in the specification.

The advantage of MDA is that the variation point management of the platform is managed automatically by the processing step and is not a concern for the product developer. The underlying platform, however, is not the only point of variation which must be managed in a product line. Different members of a product line differ also conceptually, ie their functional characteristics, as well as in terms of infrastructure, i.e the shared software assets are used to implement alternative concepts.

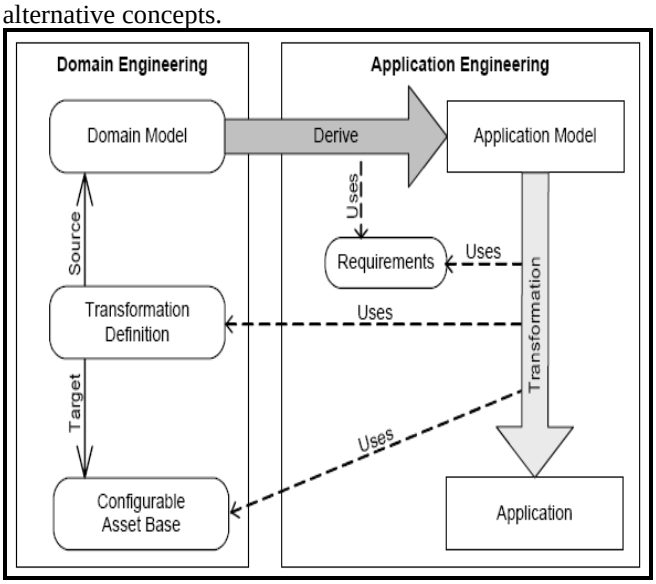

Figure 3: Engineering product lines in the world of MDA

Fig.3 shows the domain concepts and specifies the functionality of the product line regardless of the implementation details of the basic components. The definition of transformation defines how domain concepts are mapped with basic components. Based on product requirements, an application model is obtained by selecting other concepts included in the domain model. The application model is the result of using the definition of the transformation, the basic components and requirements of the product.

We note that despite the diversity of these approaches, there are always limitations and drawbacks. Indeed we found that these models are not artifacts of auxiliary documentation, but they can be a source of artifacts and be used for the automation of the course of needs analysis until code generation.

III. THE APPROACH

After defining the concept of variability in the previous session, we need to find a way to express it clearly, that is to say, to find a modeling language that allows us to include concepts linked to lines of software products, and represent our system.

Our draft approach to develop software product lines is based on the management of variability through the use of FODA diagram which reduces the discrepancy between the users of the solution and developers. We opt for the management of functional and technological variability. Regarding the management of the technological aspect, we adopt a model-driven approach. This approach allows you to manage software solutions on different platforms.

Variability in the management of software product lines has two fundamental challenges: the expression of common and variable characteristics and the development of applications using these features correctly. In our approach,we present a software product line based on models (MD-SPL). We separate the concepts linked to SPL to different areas and we build key assets like features models, meta-model s and different types of rules for transforming domain source models to a different (variable) models in a target domain.To achieve our goals, we will present the approach: We proposed a systematic approach for automating the development of product lines.

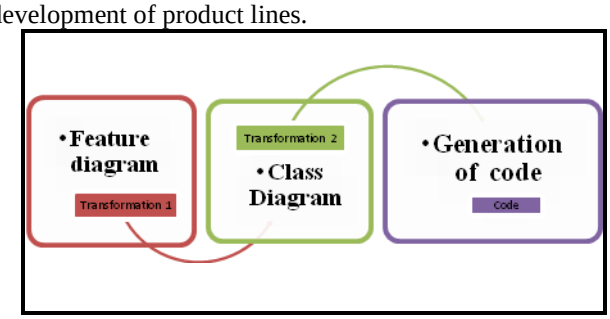

Figure 4: The proposed approach

Fig.4 shows this approach. Firstly, we propose a simple and effective meta-model product lines. In this section, we did transforamtion1. This transformation is between feature diagram and class diagram. Feature modeling is an important approach to dealing with variability at an abstract level in a hierarchical manner extensively used in software product lines. For its use in conjunction with other UML models and MDA approach, it is important to correctly integrate feature modeling into UML [21].

In the process of product development in line, after the goals and model characteristics are established, several UML models must be built (with the help of a catalog used to derive the characteristics into UML models).

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

A review of literature has shown that it is naive to make a simple transformation from UML features models diagrams. Therefore, we must go deep into the heart of the UML meta-model and derive concepts and features from more abstract elements without unwanted semantics in order to achieve a correct integration of feature modeling into it [21]. Therefore, we adopted a new meta-model that allows us to make an effective link between the two models.

Second, we created a tool to support the import of our meta-model and the transformation of class diagrams in code: This is the second transformation. In order to realize this transformation we chose Eclipse as a platform base with the relatively large number of available tools for this platform . We were particularly interested in the functional components provided by the EMF project, which carries an implementation of the basic principles of MDE approaches. We choose the tools for they can help us in the specification and implementation of different generators or compilers used in engineering applications.

A. Meta-model

A meta-model of software product lines, which also includes the notion of variability, has been proposed in this article. It allows us to reach the UML class diagram in order to represent features diagrams, such as FODA notation. After that, we present the main principles also the relationships identified in this meta-model concepts. An overview of the meta-model is available in Figure 5.

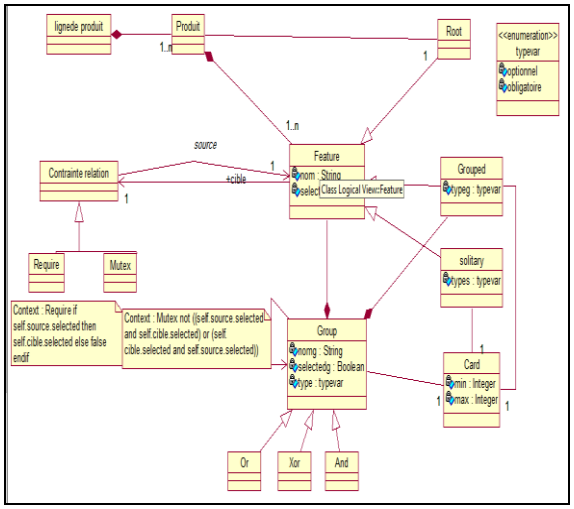

Figure 5: Meta-model proposed LdP

The starting point is the concept of "Root". Each "product line" contains products. Each product has a root. The latter is a class that comes from of the "feature" class. In fact, the feature is an artifact, a module or part of the family. It can be: common to all family members (mandatory), present only in some of them (optional). The feature itself can be "solitary" or belongs to a group "Grouped" or "root". Product line also includes multiple points of variation and these are presented by " group " .A group can be of different types : And all the features of this group must be present in the product line, however , the user has the choice between the features and finally Xor , which implies that there is a mutual exclusion between features, that is to say, if there is one that is selected, the other mu st not be simultaneously selected.

The notion of obligation and optionality is modeled by the "vartype." The class 'card' presents the cardinality which may have features and groups. Finally, contrainte_relation allows us to present the meta-model constraints that are relations of mutual exclusion and there are dependency between features.

Other complex constraints, used to validate the consistency of the product line are presented below using the OCL (Object Constraint Language).

Some examples of constraints proposed for the meta-model are:

•In the product line all mandatory features are selected

Context : solitary
ifself.type = Typevar::obligatoire then
self.selected
endif

•In a relationship of exclusion, if the source is selected the destination should not be and vice versa.

Context : Exclude
not ((self.source.selected and self.cible.selected or (self. cible.selected and self.source.selected))

• If a feature needs another and if the source of this relationship is selected, the destination must also be selected.

Context : Require
ifself.source.selected then
self.cible.selected
else
false
endif

IV. TOOLS

The characteristic modeling is an essential approach used in the development of common points and variability model of product lines.

Models have been used for long as descriptive artifacts and proved themselves very helpful for formalizing, sharing and communicating ideas.Modeling variability in SPL is thus already very useful by itself as highlighted by the popularity of feature modeling languages and their supporting tools [4] .Several tools exist for modeling feature:

*Pure::variant [22]:This is the modeling and configuration support of a characteristic of commercial tool.

*Xfeature [23]: This is a shocking newness introduction with the notion of cardinality, modeling based on XML.

*FeaturePlugin[24]: This tool supports cardinality-based modeling features, specialization of feature diagrams and layout-based features.

*RequiLine[25]: This is an engineering tool that was created to support the effective management of products. It allows you to shape the product using the characteristics and conditions to derive the specific product configurations model.

Next, we present a tool for functional modeling plug-in for Eclipse. The tool supports cardinality-based modeling features, constraints on the characteristics and the main ability of our tool include a test for our proposed meta-model and obtaining the implementation of LDP using model transformations.

The purpose of the plug-in is to provide an ease (we have only the feature diagram to draw) to design product line using our meta-model thereafter the automation of our approach. To realize such a goal, we define a systematic process to perform the domain engineering, and we apply the approach of MDE. We will design all artifacts involved as a model, making rigorous and unambiguous artifacts ready to handle by this tool.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

Indeed automation application engineering is carried out by means of model transformations that are automatically drift models defined in the domain engineering.

A. Implementation

The implementation of this plug-in under Eclipse is based on the frameworks EMF (Eclipse Modeling Framework), GEF (Graphical Editing Framework) and GMF (Graphical Modeling Framework) and make available all the elements to build modelers.

We chose these tools as they can help us in the specification and implementation of different generators or compilers used in engineering applications.

B. Case study

We also presented a possible model of the case study "a catalog for an eCommerce application" on which plug-in LDP was tested.

Electronic commerce or online sales, means the exchange of goods and services between two entities on the Internet with the aim of: looking for a product or service catalog browsing, simple or grouped demand, payment; delivery request, etc. This case consists of several modules; we are interested in management catalog module.

Management module product catalog:

For each product catalog it is necessary to provide a description (Productinformation). This description can optionally be supplemented by (1-5) images in two or three dimensions and accurately (0..4) three-dimensional images. Products catalog can be a categorized as (Categories):

(i) MultipleClassification represents the possibility for a product catalog belong to several categories.

(ii) MultipleLevel supports nested categories and description to add a text description to a category.

(iii) Finally Thumbnails indicates that the product can be represented as thumbnails and in this case implies that the support for two-dimensional images was chosen.

B.1 Engineering Domain

The domain engineering consists of developing and building reusable artifacts that will be reused for building applications, so we are dealing with the development for reuse. The three main activities are analysis, design and implementation of the domain.

B.1.1 Domain Analysis

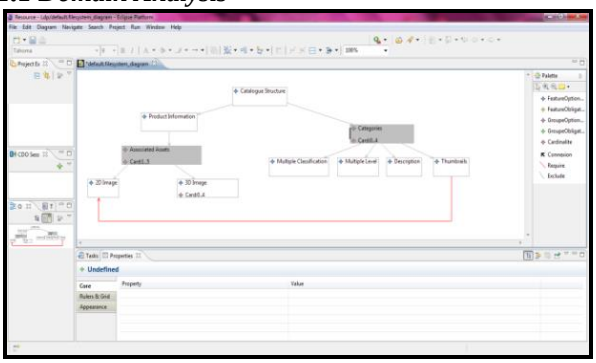

Figure 6: Diagram feature

In this section, we presented our graphical editor that allows LDP determines the features included in the field, presented in the form of mandatory or optional feature, optional or mandatory groups and the link between its concept regardless of the single link or require or mutex.

The result of this analysis is often called the domain model and for this reason also known as domain modeling. Formalisms and notations used for modeling the field vary according to the different practical approaches proposed. We modeled the formalism most widely used in the field of product lines: the pattern features, which will be presented in Fig. 6.

B.1.2 Design and implementation of field

The purpose of the design area is to establish a generic software architecture of the product line. A late goal to expect we chose the class diagram for the architecture of product line (a catalog for an e-commerce application). Indeed, the variability that we have identified during domain analysis must be explicitly specified in the architecture (see Fig. 7).

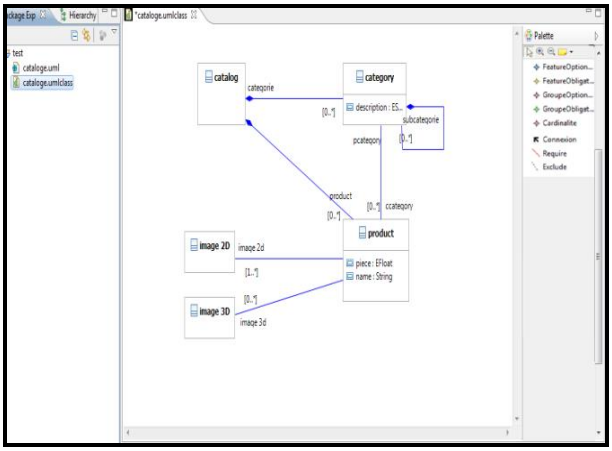

Figure 7: Class diagram

The implementation of the domain consists of implementing the generic architecture defined in the domain design as components that will be reused in application engineering for the construction of each product.

V. CONCLUSION

In this article, we presented the engineering product lines and its benefits in terms of cost and quality of development as well as model-driven engineering. In addition, we identified the reasons for working with the MDE paradigm and product lines approach. This allowed us to clarify the framework and feel the importance of treating this research problem. Subsequently, we reviewed the various existing modeling for Software Product Line solutions. Through the analysis of the proposed solutions, we were able to identify the key points of each approach and related issues. In the case of meta-model variability, a new alternative has been proposed to synthesize existing research and to treat their weaknesses.

However, we have deduced that current approaches lack flexibility in taking into account specific user requirements. This study included a solution to this problem, a newly proposed meta-model by using MDE devises such as transformations and a support tool(plug-in Eclipse LDP). This devise would ensure flexibility and efficiency when implementing the product line.

The prospects of this work consist of a more detailed study of specific and generic constraints for modeling product lines. We suggest at a later stage the refinement of our editor, first by adding controls notifications and error messages to ensure the validation of the developed models. Secondly by the automatic generation of class digraph From feature diagram using models transformations between our meta-model and the meta-model of class diagram.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)

In fact we will use "model-to-model transformation" and the QVT language to describe the similarities between the components of our meta-model and the components of meta-model class diagram.
Similarly, we expect the acquisition of more experiences with the application of our editor and lead the generation code based on another meta-model.

REFERENCES

1. Northrop, L. M. A framework for software product line practice. London : in Proceedings of the Workshop on Object-OrientedTechnology, pp. 365–376, Springer, UK, 1999.
2. Linden, F. van der. Software product families in Europe., : the esaps & cafe projects,IEEE Software, vol. 19, no. 4," , pp. pp. 41– 49, 2002.
3. Poh, G. Halmans and K .Communicating the variability of a software product family to customers.. Software and System Modeling, pp. pp. 15–36, 2003.
4. Jézéquel, Jean-Marc .Model-Driven Engineering for Software Product Lin. 2012, International Scholarly Research Network, p. 24 pages.
5. Klaus Pohl Andreas Metzger.Variability Management in Software Product Line. Engineering,
6. Sijtema, Marten. Managing variability in model transformations for model-driven product lines : extending the ATL model transformation language with variability management capabilities. 2010.
7. Clements P., Northrop L .Software Product Lines. 2001, Practices and Patterns.
8. Svahnberg M., Bosch J .Issues Conserning Variability in Software Product Lines. In Development and Evolution of Software Architectures for Product Families, Proceedings of International Workshop IW-SAPF- 3, pp. Vol 1429 of Lecture Notes in Computer Science, .
9. Czarnecki., W.U. Eisenecker and K. Generative Programming: Methods, Tools, and Applications. 2000., Addison-Wesley,.
10. Bass.L, P. Clements, and R. Kazman .Software Architecture in Practices. .Addison-Wesley, p. 1998.
11. Clements, F. Bachmann and P .Variability in software product lines.. 2005, Tech. Rep. cmu/sei-tr-012, Software Engineering Institute, Pittsburgh, Pa, USA, .
12. M. Svahnberg, J. van Gurp, and J. Bosch .A taxonomy of variability realization techniques: research articles. 2005, SoftwarePractice and Experience, pp. pp. 705–754.
13. D. C. Schmidt, r .Guest editor's introduction : Model-driven engineering, IEEE Compute. . 2006, p. 39.
14. FAVRE.J-M. oundations of Model (Driven) (Reverse) Engineering: Models.. Proceedings of the International Seminar on Language Engineering for Model-Driven Software Development.
15. Groher, [M. Völter and I.Product line implementation using aspect-oriented and model-drivensoftware development. 2007. , 11th International Software Product Line Conference (Kyoto, Japan).
16. Oster, Sebastian .Feature Model-based Software Product Line Testing.. 2012, TU Darmstadt, Darmstadt [Ph.D. Thesis].
17. Mathieu ACHER, Raphaël MICHEL, Patrick HEYMANS, Philippe COLLET, Philippe LAHIRE. Languages and Tools for Managing Feature Models.. 2012, in Proceedings of the 3rd International Workshop on Product LinE Approaches in Software Engineering co-located with ICSE'12.
18. Ziadi.T. Manipulation de Lignes de Produits en UML. s.l. : Rennes: Université de Rennes, 2004.
19. Tessier, Patrick, Servat, David and Gérard, Sébastien Variability Management on Behavioral Models.2008.
20. S.Deelstra, M.Sinnema, J.vanGurp and J.Bosch. Model Driven Architecture as Approach to Manage Variability in Software Product Families. 2003, Proceedings of the Workshop on Model Driven Architecture: Foundations and Applications.
21. Snirc, Valentino Vranic and Jan. Integrating Feature Modeling into UML. 2006, Proceedings of the NODe/GSEM, . Institute of Informatics and Software Engineering.Faculty of Informatics and Information Technology .Slovak University of Technology .
22. PV. Pure systems. 2011.
23. Pasetti, Rohlik and A. XFeature Modeling Tool. 2011, Automatic Control Laboratory, ETH Z• urich, Accessed.
24. K.Czarnecki., and M.Antkiewicz .FeaturePlugin: Feature modeling plug-in for Eclipse. 2004, In OOPSLA'04 Eclipse Technology eXchange (ETX).
25. RWTH Aachen. Lichter., T.von der Maßen and H.RequiLine. 2005.

Published By:
Blue Eyes Intelligence Engineering
and Sciences Publication (BEIESP)