Model Driven Software Product Line Process for Service/Component-Based Applications

Sami Ouali[1*], Naoufel Kraïem[2], Zuhoor Al-Khanjari[2], Youcef Baghdadi[2]

[1] RIADI Lab, ENSI, Compus of Manouba, Tunisia.
[2] Department of CS, Sultan Qaboos University, Oman.

* Corresponding author. Tel.:0021623695033; email: samiouali@gmail.com
Manuscript submitted April 28, 2014; accepted May 20, 2015.
doi: 10.17706/jsw.10.7.881-892

Abstract: The software reuse becomes the key for companies to improve development costs, time-to-market, and software quality. The Software Product Line Engineering (SPLE) and the Model Driven Engineering (MDE) are two new forms of software reuse. Software product lines are recognized as a successful approach to reuse in many domains (cars, printers, phones…) and especially in software development. Software Product Line Engineering proposes a set of methods, techniques and a common set of software artifacts for the production of customized software products as atomic or composite services to be reused in SOA-based applications. The main purpose of MDE is to enhance productivity by maximizing compatibility between software systems and simplifying the process of design via reuse of standardized models and design patterns. This paper presents a process for the construction of software product lines using model driven techniques. This process is based on a combination of SPLE and the MDE throw the use of visual techniques for modeling product lines and model driven techniques. We have validated the approach with a common example of software product line to evaluate the approach (airline travel agency reservation system). We propose a tool that supports some software product line visual techniques and model driven techniques.

Key words: Model driven, process, SOA, software product line, tool, variability.

1. Introduction

The study of software requirements is an essential step in the life cycle of any software. It consists on the study of functionalities and qualities which should be taken into account during design and be present or characterize the obtained software. Software development tends to focus on the construction of individual products. The evolution of software development, the concurrence, and the growth of product number's have motivated the emergence of software product line concept. Software Product Lines (SPL) are recognized as a successful approach to reuse in software development [1], [2]. SPLE allows companies to realize significant improvements in time-to-market, cost, productivity, and system quality. The purpose of this approach exploits the commonalities between software products while preserving the ability to vary the functionality between these products. That is, the difference between products is their variability, a key success factor in product lines and reuse [3]. Variability is the ability of an approach to produce a set of artifacts that differ from each other in a preplanned fashion [4]. In this definition, variations in a product line context must be anticipated and preserved.

SPLE distinguishes two layered levels of engineering [4]: Domain Engineering (DE) and Application

Engineering (AE). DE deals with the identification of commonalities and variabilities among products, the establishment of generic software architecture, and the construction of reusable components known as assets that will be reused for developing the products. AE is used to develop a new product from a product line. At AE level, the results from DE level are used to derive a specific product.

This paper presents a meta-model and a prototyping tool for requirement and feature modeling. The tool implements various models to assist stakeholders in the process of product configuration for SPLs. This paper describes an Eclipse plug-in for map, feature, and class modeling to model SPL, which assists SPL designer in the construction of SPL and the derivation of particular product. Providing tool as an Eclipse plug-in would facilitate the integration of these kinds of modeling with a development environment. The tool, itself, is built by using two Eclipse plugins: Eclipse Modeling Framework (EMF), and Graphical Modeling Framework (GMF). These plugins provide many possibilities through extensions points, which results in reducing the development effort.

The remainder of this paper is organized as follows. Section 2 summarizes a meta-model for intention and feature modeling in the context of SPLs. The proposed Model Driven for SPL process is described in Section 3. The supporting tool is presented in Section 4. Section 5 presents some related work. Finally, a conclusion section highlights our contribution and presents some research perspectives.

2. A Meta-Model for SPL

This section describes a meta-model that synthesizes the interesting points such as SPL, intention, features, etc. We choose to transform this meta-model into a UML profile to (i) facilitate the integration with UML models, and (ii) use it in our MD approach.

We try throw this meta-model to facilitate the relation between requirements model (MAP), features model and classes diagram. This meta-model is used in our process and in the generation of our tool support.

A MAP [5] involves two or more sections. Each section is composed of two intentions and one strategy (these concepts are presented in Section 3.1). A relationship between sections can be a cluster, multi-path or multi segment.

To further use the meta-model in our process, we make a modification, as shown in Fig. 1, by adding two types of sections: 'OptionalSection' and 'MandatorySection'. These new sections are added for the mapping between map model and features model.

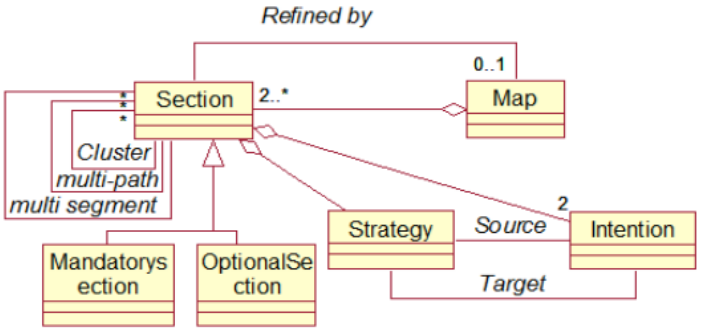


Fig. 1. Modification applied on map meta-model.

A PL contains features, as shown in Fig. 2. A product belongs to one PL. It is also composed of features that check some constraints (an exclusion and required relation) through the conflict and require relationships. Fig. 2 presents the different concepts of feature Model. The possible relationships between features are presented by FeatureGroup. There are three relationships between features: 'And', 'Xor', and 'Or'.

The relationships between a parent feature and its child features are categorized as: 'Mandatory' or 'Optional'.

Fig. 3 presents the relation between the features elaborated in the analysis domain and the components constructed in implementation domain; the goal of implementation domain is to develop the components participating in the architecture of the SPL). A feature is associated to a component. There are two kinds of components: composite or leaf, i.e. simple component. A leaf is a non decomposable component. It has one or more possible implementation. A composite component is decomposed into subcomponents.

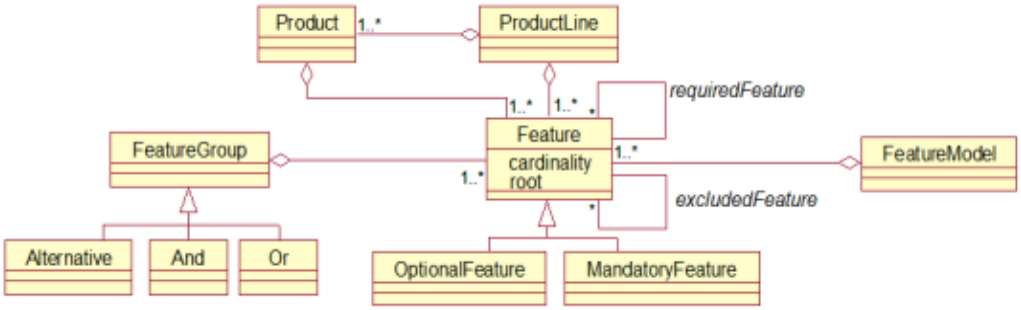


Fig. 2. Relationship between product line and features.

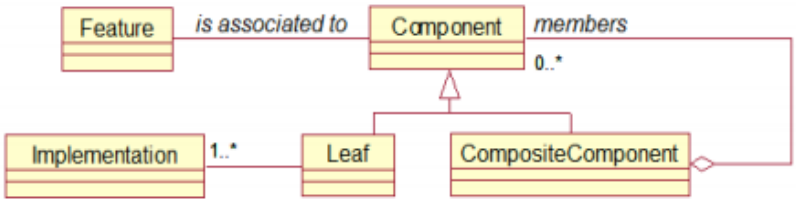


Fig. 3. Relationship between feature and component.

Another alternative to the implementation of assets with components is the use of services as shown in Fig. 4. There are two kinds of services: atomic service and composite service. An atomic service cannot further be decomposed. A composite service is composed of other services. A feature can be associated to a service.

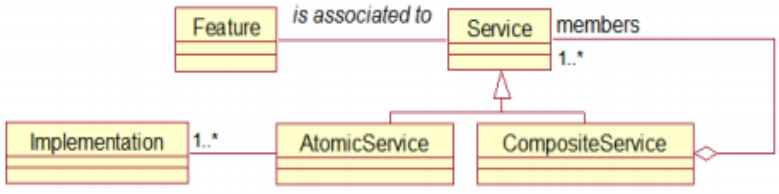


Fig. 4. Relationship between feature and service.

3. Model-Driven SPL Process

This section presents the notations used in the model-driven SPL process. These are map model and features model. It also presents some rules and patterns for mapping between map, features, and class model. Both DE and AE are covered in the proposed as shown in Fig. 5.

The DE process deals with the creation of core assets. These core assets are PL requirements, analysis models, PL architecture, reusable components and services. This process concerns with the elicitation of intentions and strategies using the map for the design of user's requirements. In the modeling of the requirements, SPL features intentionality is represented as maps. Features and their composition are

determined according to the intentional SPL meta-model proposed in [23]. Then, features models derive class diagrams in the design view that used further used to obtain a component architecture or service architecture throw through components or services repository. The component architecture view deals with the creation of the interface type and a component. The interfaces contain the signature of operations. The generated component implements the required and provided interfaces. Service architecture view is also realized by generation of services from design view.

The AE process deals with product derivation from the composition of existing artifacts created in the DE. That is, from existing components or services present in the components repository or services repository. It exploits the variability of the PL, which allows the satisfaction of users needs.

Our Model Driven Software Product Line Process tries to cover domain engineering and application engineering which are the main steps of Software Product Line lifecycle. In our process we try not only to coverage these steps but also to consider a new level which is the intentional one. We identify this layer to represent stakeholders' intentions. This layer is used in both of domain engineering and application engineering. We try throwing this layer to model the stakeholders' intentions for the software product line and for a particular product to construct in application engineering. The purpose is to propose an intentional process for software product line construction using the Model Driven approach for the passage between the proposed models. We present our process concerning domain engineering, application engineering, and the new intentional level. The domain engineering and the application engineering are the main steps of SPL lifecycle. They are related together.

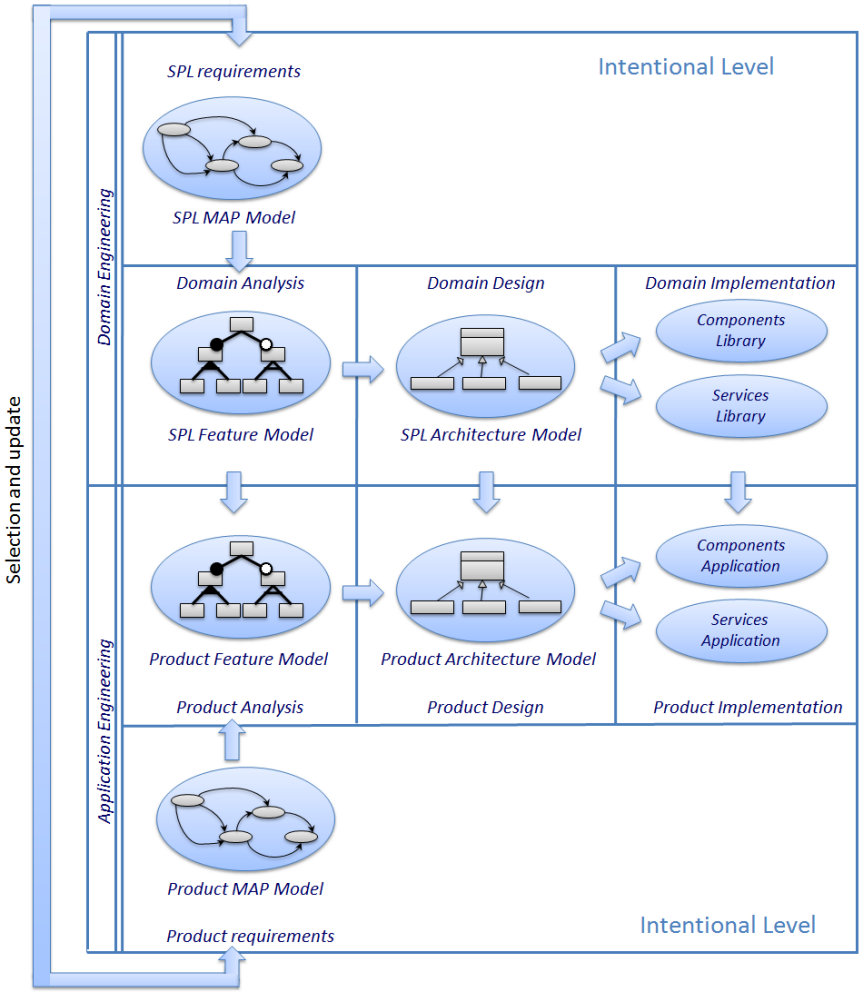


Fig. 5. Approach for SPL construction.

3.1. MAP Model

A MAP [5] is a process model expressed in a goal driven perspective that can provides a representation of a multi-facetted purpose based on a non-deterministic intentions and strategies. The directed nature of the graph shows which goals can follow which one. MAP is considered as intention-oriented process modeling that follows the human intention of achieving a goal [12]. A map is a directed graph from 'Start' to 'Stop', as the strategy shows the flow from the source to the target intention. It is composed of several sections. Intentions are represented as nodes of the graph. Strategies are the relationships between intentions.

The key concepts of the map are:

- **Intention:** it is a goal [6] that expresses what is targeted, i.e. a state that is expected to be reached or maintained.
- **Strategy:** it is a way or a means to achieve an intention.
- **Section:** it is a combination of two intentions and a source and target linked together by a strategy.

The inter-relationships between these concepts are:

- **Thread relationship**: a target intention to be achieved from a source intention, in many different ways. Each of these ways can be refined as a section in the map.
- **Path relationship**: establishes a precedence/ succession relationship between sections. For a section to succeed another, its source intention must be the target intention of the preceding one.
- **Bundle relationship**: several mutually exclusive sections having the same pair (Intention Source, Intention Target).
- **Refinement relationship**: a section of a map can be refined as another map through the refinement relationship.

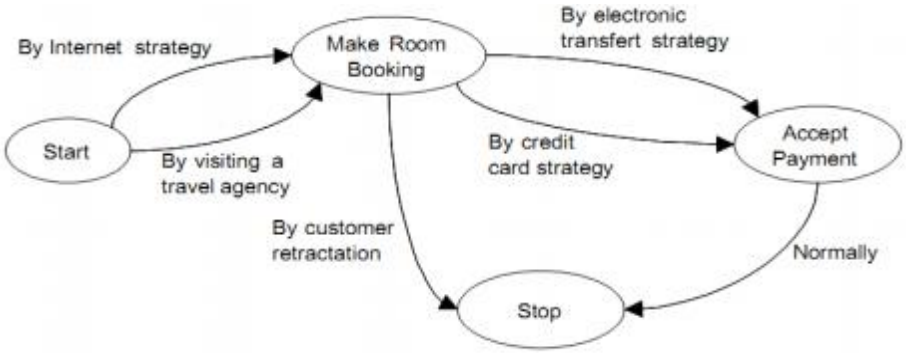


Fig. 6. Sample map for 'Make Confirmed Booking' (source: [24]).

For instance, 'Make Room', 'Flight', or 'Car Booking' is an intention to make a reservation for rooms, flights, or cars. The accomplishment of this intention leaves the system in the 'Booking made' state. Fig. 6 shows the map for making a 'Confirmed booking'. Two paths are present for the achievement of making a 'Confirmed booking'. This map involves a multi thread relationship which represents two different ways of achieving 'Accept Payment': 'credit card' and 'electronic transfer').

3.2. Feature Model

Feature modeling is a domain analysis technique, part of the Feature Oriented Domain Analysis (FODA) method [7], for developing software for reuse. This method has been applied in many domains such as telecom systems [8], [9], network protocols [10], and embedded systems [11]. Using feature modeling can help in generating domain design in SPL. Feature Models allow a representation of all possible products in a SPL, in terms of features. It represents the description of the mandatory features that are present in all the products of the PL along with variant features or optional features that do not appear in all the products.

In a feature model, we can find hierarchical relationships between features. The relationships between a parent feature and its sub features are categorized as mandatory features that are required in all the products, or optional features. A child feature can only appear in the products where its parent feature appears. FODA provides the following relationships between features:

- **Mandatory** is a relationship between a parent and its child feature. It indicates that when the parent is part of a product, the child also must be present.
- **Optional** is a relationship between a parent and its child feature. It indicates that if the parent is part of a product, the child may be part of it.
- **Alternative (Xor-relationship)** is a relationship between a parent and a set of its child features. It indicates that only one child feature can be selected. If the parent is part of a product, then one only child features must be in the product.
- **And-relationship** is a relationship between a parent and a set of its child features. It indicates that if the parent is part of a product, all its child features must be in the product.
- **Or-relationship** is a relationship between a parent and a set of its child features. It indicates that if the parent is part of a product, then one or more of the child features may in the product.

A features model defines constraints which are compatibility rules. These rules refer to some restrictions in features combinations:

- **Requires**: a feature X requires Y means that if X is included in a product, then feature Y must be included, but not vice versa.
- **Excludes**: a feature X excludes Y means that if X is included in a product, then feature Y should not be included and vice versa.

3.3. Patterns to Automate Transformations

Our interest in PL development process focuses on DE and AE, where the process to use goals and features models to establish architecture model. In this architectural model, we deal only with structural view, i.e. the class diagram of from UML model. We define some rules to try to automatically derive UML class diagram from requirement model.

We distinguish two different kinds of transformations: (i) SPL requirements are mapped into feature models, and (ii) the structural information of the feature models is mapped into class diagrams. These transformations are implemented using QVT.

The first transformation is the mapping between requirement model and analysis model, i.e. from MAP model to feature model, by using some transformations rules:

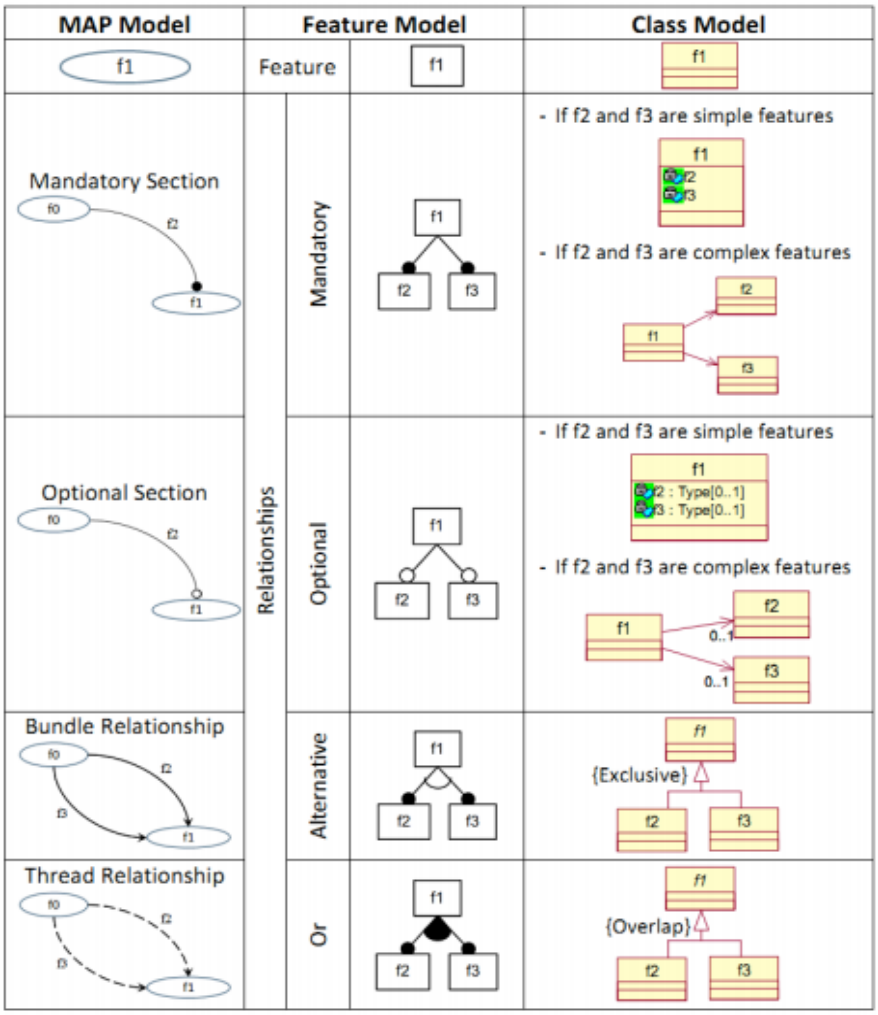
- A thread relationship in a map model representing a AND/OR relation between two strategies is equivalent to an Or-features-group.
- A bundle relationship representing a XOR relation between two strategies is equivalent to an Alternative-features-group.
- For mandatory and optional features, we have defined new concepts in MAP meta-model: mandatory and optional section. The same graphical representation in feature model is used for representing mandatory and optional sections.

Feature model are mapped into classes and attributes as shown in Table 1:

- A feature in the model tree can be simple or complex feature.
- The leaves of the tree can be simple features, in this cases a simple feature is mapped into an attribute of a parent class.
- A non-atomic feature is mapped into a class representing this feature.
- The mandatory features imply a unidirectional association with the right cardinality.
- The optional features are mapped as associations with 0..1 as cardinality

For alternative and OR groups of features, generalization concept is applied, where an alternative group of features is represented by a generalization of classes with exclusive as constraint between the subclasses, whereas Or group of features is represented by a generalization of classes with overlap as constraint between the subclasses.

Table 1. Rules for Automate Transformations



4. Supporting Tool

Based on the meta-model presented in Section 2, we developed an interactive, visual, tool for requirement modeling, feature capture, and architecture modeling. The tool will be extended to Feature Configuration. Our plugin implements map modeling, cardinality-based feature modeling through transformations and architectures of SPL construction. This tool is based on eclipse plug-ins. The tool uses the multiple extension possibilities of offered by eclipse platform through two very famous plug-ins: Eclipse Modeling Framework (EMF) [25], and Graphical Modeling Framework (GMF) [26]. These plugins provide many possibilities through extensions points, which can reduce the development effort. These plugins helps in constructing our modeldriven approach. The proposed tool is built by re-using these two Eclipse plugins. Providing a supporting tool for map, feature, and class modeling as an Eclipse plug-in is of paramount importance for with the integration of these kinds of modeling as part of a development environment. Object Constraint Language is used to describe constraints, rules and specifications. OCL statements can be specified as part of EMF Model or invoked directly from programming language (java).

4.1. Tool

Our tool allows the design of Software Product Line throwing the different phases of Domain Engineering and Application Engineering. We propose a set of graphical palette for the modeling of MAP model, Feature model, class model and Component model. Two of these graphical palettes are presented in Fig. 7 and Fig. 8. We develop not only palettes but also OCL transformations between the different models. In this way models are obtained from a source model using the automate rules of transformations.

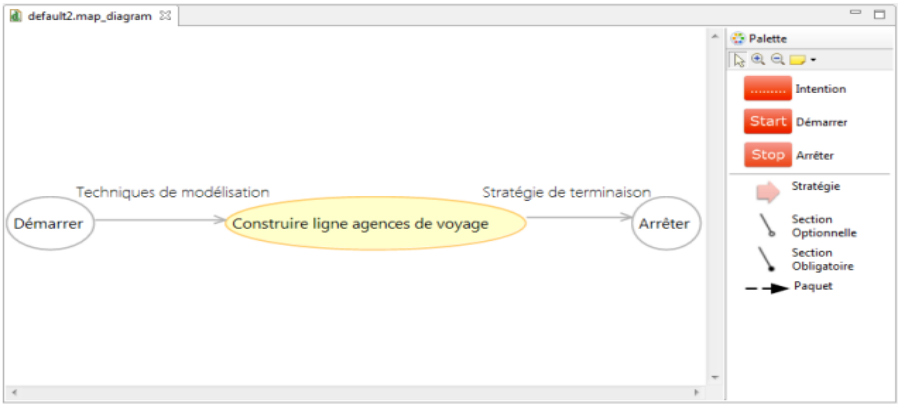


Fig. 7. Graphical palette for MAP modeling.

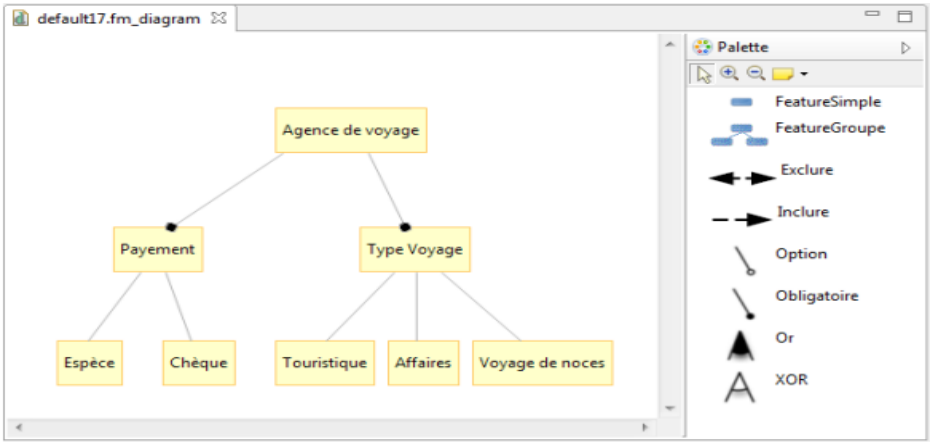


Fig. 8. Graphical palette for feature modeling.

In Fig. 9, we propose two views for features tree modeling where the first one present the structure of software product line and the second one present the configuration of a product line. The configuration concerns the choice of the features existing in the product issued from the software product line. Many ameliorations need to be done to add the generation of the products.

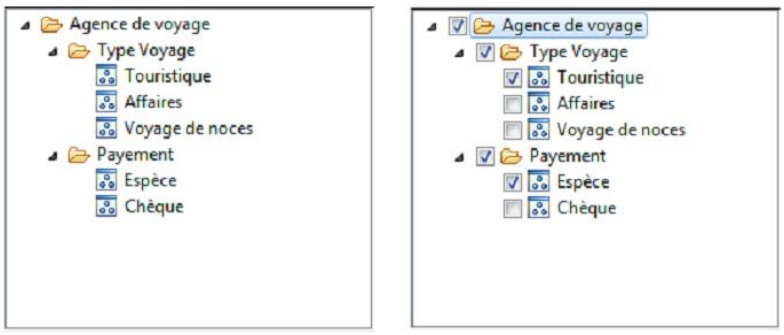


Fig. 9. Views for features tree modeling and configuration.

4.2. Case Study

We use the airline travel agency reservation system to illustrate and validate our approach. This common example of SPL shows we apply our approach and some rules and patterns to automate transformations from Maps to feature model. Fig. 10 shows the entire map with the purpose to 'Make Confirmed Flight Booking'. To reach the 'Accept Payment' intention that allows a customer to make the payment, three ways are possible: by 'Credit Card', 'Money Order', or 'Cash' strategy. Two ways are possible to reach the 'Stop' intention either because a customer retracts from making the booking (By customer retraction Strategy), or after payment (Normally).

Fig. 10. The map of 'make confirmed flight booking'.

5. Related Work

Several authors have proposed approaches relating to feature models and product architectures. The mapping between requirements and design is a complex task because of the flexibility and adaptability of the SPL, the different technology possibilities, etc. Van Lamsweerde [13] derives software architectures from the formal specifications of a system goal model by using heuristics like design elements (classes, states, etc.) Sochos *et al*. propose a mapping between requirements and architecture by modifying the feature models [20]. Bragança *et al*. [21] propose to obtain features from use case models. Laguna *et al*. [22] proposes some patterns to obtain design view (class diagram) and use case models from features models.

Many proposals express variability with UML models (Structural, functional or dynamic models). Some authors have proposed to change use case diagrams or class diagrams. Von der Maßen *et al.* [16] propose the extension of UML Meta model by the adding of new relationships "option" and "alternative". Clauß proposes the use of stereotypes to express the variability in architecture models [19].

Many propositions are concerned with the construction of editor for features modeling. The first one was AmiEddi [14], which supports feature modeling notation [4]. This notation did not support feature and group cardinalities. Another tool was proposed CaptainFeature [15], which implements a cardinality-based notation. Some other works try to integrate feature-based configuration. An initial prototype was proposed ConfigEditor [11]. ReqiLine [16] is a tool integrating feature modeling and requirements engineering. This tool allows the definition of various relationships between features but does not support cardinalities and the checking of models for consistency.

We also found some commercial tools (Pure: Variants [17] or GEARS [18], [27]). Pure: Variants support feature modeling and configuration by using a tree-view render without feature cardinalities. It allows modeling of constraints between features and uses Prolog-based constraint solver for the configuration. GEAR allows the modeling and configuration of software variants. It models variability as sets of parameters (Boolean for optional, enumeration for alternatives, set for subsets). These parameters are not arranged into tree or some hierarchies like in features model. The constraints can be defined and the

configuration will under account these constraints.

6. Conclusion

Software product line construction and derivation of particular product become a topic with a growing interest. We try throwing this paper to focus on the mapping of goal-oriented requirements to software architectures. We have aimed at fulfilling stakeholder's requirement of the SPL and the particular requirements. We have defined a Meta model to represent the different concepts related to the construction of SPL. Also, we have defined a process to construct SPL, derive particular product though configuration and model derivation.

We have emphasized the construction of components for Components-based applications and services for SOA-based applications. The process generates design views by using class diagram derived from a requirement model (i.e. which is Map model and features model). The process is supported by rules and mapping patterns. It offers also the building of a set of components and services which can be used in application engineering.

We propose a tool support using eclipse plugins (EMF and GMF) to facilitate the construction of SPL. We use this tool to illustrate and validate our process. A travel agency reservation system is taken as case study.

Our future works includes (i) the formal validation of the proposed rules and mapping patterns, (ii) the generation of assets of the SPL and (iii) the amelioration of our tool. The assets will be product line requirements, analysis models, product line architecture, reusable components, and services.

References

[1] Bachmann, F., & Clements, P. (2005). Variability in software product lines. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

[2] Bosch, J. (2000). *Design and Use of Software Architectures, Adopting and Evolving a Product-Line Approach*, Addison Wesley.

[3] Grup, J. V. (2000). The key to software reuse. Thesis, University of Groningem, Sweden.

[4] Czarnecki, K., & Eisenecker, W. (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

[5] Rolland, C., Prakash, N., & Benjamen, A. (1999). A multi-model view of process modeling. *Requirements Engineering Journal, 4(4),* 169-187.

[6] Salinesi, C., Mazo, R., Djebbi, O., Diaz, D., & Lora, M. A. (2011). Constraints: the core of product line engineering. *Proceedings of the 5th IEEE International Conference on Research Challenges in Information Science*. France.

[7] Saraswat, V. (1992). The category of constraint systems is cartesian-closed. *Logic in Computer Science*, IEEE Press.

[8] Mazo, R., Salinesi, C., Diaz, D., & Lora, M. A. (2011). Transforming attribute and clone-enabled feature models into constraint programs over finite domains. *Proceedings of the 6th International Conference on Evaluation of Novel Approaches to Software Engineering* (ENASE). Beijing, China.

[9] Lee, K., Kang, K. C., & Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. *Proceedings of the Seventh Reuse Conference* (pp. 62–77).

[10] Barbeau, M., & Bordeleau, F. (2002). A protocol stack development tool using generative programming. *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering* (pp. 93–109).

[11] Czarnecki, K., Bednasch, T., Unger, P., & Eisenecker, U. W. (2002). Generative programming for

embedded software: An industrial experience report. *Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering* (pp. 156– 172).

[12] Soffer, P., & Rolland, C. (2005). Combining intention-oriented and state-based process modeling. *Proceedings of the International Conference on ER'05* (pp. 47-62).

[13] Lamsweerde, V. A. (2003). From system goals to software architecture. *Formal Methods for software Architectures*.

[14] Selbig, M. (2000). A feature diagram editor: analysis, design, and implementation of its core functionality. Kaiserslautern, Germany.

[15] Bednasch, T., Endler, C., & Lang, M. Captain feature. Retrieved 2002-2004, from https://sourceforge.net/projects/captainfeature/.

[16] Von der Massen, T., & Lichter, H. (2003). RequiLine: A requirements engineering tool for software product lines. *Software Product-Family Engineering* (pp. 168-180).

[17] Pure-Systems GmbH. (2003). Variant management with pure: Consul. Technical White Paper. Retrieved from http://web.pure-systems.com.

[18] Krueger, C. W. Software mass customization. White Paper. Retrieved from http://www.biglever.com/papers/BigLeverMassCustomizat ion.pdf.

[19] Clauß, M. (2001). Generic modeling using Uml extensions for variability. *Proceedings of Workshop on Domain Specific Visual Languages at OOPSLA*.

[20] Sochos, P., Philippow, I., & Riebish, M. (2004). Feature-oriented development of software product lines: mapping feature models to the architecture. *Lecture Notes in Computer Science, 3263*, 138-152.

[21] Bragança, A., & Machado, R. J. (2007). Automating mappings between use case diagrams and feature models for software product lines. *Proceedings of SPLC 2007* (pp 3- 12).

[22] Laguna, M. A., & González-Baixauli, B. (2008). Product line requirements: Multi-paradigm variability models. *Proceedings of the 11th Workshop on Requirements Engineering WER 2008*.

[23] Ouali, S., Kraiem, N., & Ben, G. H. (2012). From intentions to software design using an intentional software product line meta-model. *Proceeding of the 8th International Conference on Innovations in Information Technology*.

[24] Rolland, C., & Salinesi, C. (2005). Modeling goals and reasoning with them. *Engineering and Managing Software Requirements* (pp. 189–217).

[25] Eclipse modeling framework. Retrieved from http://www.eclipse.org/modeling/emf/

[26] Graphical modeling framework. Retrieved from http://www.eclipse.org/modeling/gmp/

[27] Driss, M., Moha, N., Jamoussi, Y., Jézéquel, J. M., & Ghézala, H. H. B. (2010). A requirements-centric approach to web services modeling, discovery, and selection. *Proceedings of ICSOC'10 the 8th International Conference on Service Oriented Computing*, San Francisco, California, USA.

Sami Ouali is a PhD candidate in software engineering and computer science at National School of Computer Sciences University. His research focuses on the proposal of a construction method for a flexible software product line. He is also an assistant in the Higher Institute of Computer Science and Management of Kairouan.

Naoufel Kraiem is a HDR from University of Paris1, Sorbonne, France. He is currently an associate professor of computer science at SQU. His research interests include IT adoption and usage information modelling, software engineering, software product lines and CASE tools. His research work has been supported by funding of the CNRS, INRIA, MRT (Ministry of Research and Technology and Industry) and by the Commission of the European Communities under the ESPRIT Programmes (BUSINESS CLASS). He has

had several articles published in many journals such as Management Science Information Systems Research Communications of the ACM and IEEE Transactions. He has been invited to present his research in many countries in North America, Europe, Africa and in the Middle East. Prof. Naoufel Kraiem has been a member of over 20 program committees.

Zuhoor Al-Khanjari is currently an associate professor of software engineering in the Department of Computer at the SQU. His research also involves employing the software engineering strategies and methodologies to enhance the development of the e-learning content management systems. This research in e-learning is more beneficial to education community (SQU and other academic institutions across the country) particularly in encouraging the use of the efficient e-learning strategies in future education. Currently, his research concentrates on developing a structured methodology for developing e-learning software components (e.g. Learning Objects) and e-Gov.

Youcef Baghdadi holds a PhD degree in computer science from University of Toulouse I, Toulouse, France. He is now an associate professor at Sultan Qaboos University. He published papers in journals such as Theoretical and Applied Electronic Commerce Research, Enterprise Transformation, and Service Oriented Computing and Applications. His research interests include web services and service oriented architecture, enterprise interactions, electronic commerce, and social commerce.