Review of Software Product Line Models Used to Model Cloud applications

Anir Benlachgar and Fatima-Zahra Belouadha
Computer Science Department
Université Mohammed V – Agdal, Ecole Mohammadia d'Ingénieurs
Rabat, Morocco
Anir.benlachgar@gmail.com, belouadha@emi.ac.ma

***Abstract*— The Cloud computing presents advantages of reutilization, multi usage and resources sharing, and especially cost minimization. In particular, the software distribution model SaaS (Software as a Service) enables to produce multiuser services. This raises the concern of the adaptation of this type of applications to answer in a personalized way the different needs of users.. To achieve this goal, the proposed researches in this topic recommend or use techniques adopted from the Software Product Line engineering domain to model the variability (variable and customizable aspects) of the SaaS applications. This paper presents a review of the main models used in this context. The objective of this work is to assess the relevance of the studied models and their capacity to cover different aspects linked to the variability of the SaaS applications.**

***Keywords*— cloud computing; SaaS; service variability; variability models; software product line**

I. INTRODUCTION

Any SaaS (Software as a Service) application is supposed to be adaptable in order to allow customization with regards to functional and non-functional (mainly the performance and the quality of service (QoS)) needs of multiple users. The issue of adapting and personalizing SaaS applications constitutes a recent domain of research. Most of the related work emphasized on the modeling of this type of applications as a first milestone in the process of creating customizable SaaS solutions. It focuses essentially on modeling the variable and customizable aspects of the application that must, according to their nature, be communicated either to the clients or the developers of the application. As an example, the user may require a specific security level from different security levels proposed by the provider. Moreover, developers must know, for instance, the different security algorithms and techniques (encryption algorithm, digital signature, etc.) which are provided to secure different security levels for the application.

In this context, this article presents a review of the proposed models for describing the variability of the SaaS applications to assess their relevance. It is structured in eight sections. Section 2 presents some related works. Sections 3, 4 and 5 describe respectively the following models: Feature Model (FM) and Extended Feature Model (EFM), Decision Model (DM), and Orthogonal Variability Model (OVM). Section 6 presents a comparison of these four models. Finally, section 7 reports some conclusions of this work.

II. RELATEDWORKS

Few researches have been interested in modelling adaptable aspects of SaaS applications. Most of them adopted models used in the Software Product Line (SPL) to model variability. Mietzner et al. [1] use the OVM model to represent the external variability (communicated to client) and the internal one (visible for developers). This model provides a view of the SaaS application representing its different variations, while being optimized in terms of size and complexity. Ruehl et Andelfinger [2] propose to use a generic model which provides flexibility at two levels: the flexibility of use (configuration of the application according to the user needs), and the flexibility of change (e.g. the reconfiguration of the application). The authors recommend the approaches of SPL engineering to describe variants of applications thus to generate, at runtime, personalized applications. However, no concrete model for their approach has been presented. Schroeter et al. [3] adapt the FM model to dynamically configure multiuser cloud applications. They use the EFM model to represent features and QoS variability. To configure custom applications and also allow their dynamic reconfiguration in a dynamic staged configuration process, they use, in conjunction with the EFM model, a view model that allows users to perform configuration operations on the EFM, and a configuration process model that defines the steps for generating models corresponding to possible configurations. Schroeter et al. [4] propose an architecture which addresses the adaptation of SaaS applications in two steps: at the time of conception by modeling information reflecting the variable aspects, and during execution to optimize the execution environment. They focus on this last aspect by extending a Multi-Quality Auto tuning Architecture (MQuAT) [5] to evaluate the QoS. However, they don't specify a concrete variability model to use in the conception step. In the following, we describe the main SPL models recommended or used in the literature to customize SaaS applications.

III. FEATURE AND EXTENDED FEATURE MODELS

The FM model was proposed as part of the FODA (Feature Oriented Domain Analysis) method [6]. It provides a hierarchical graphic notation as a tree describing the characteristics of a SPL. Hereafter, we refer to a feature appearing in the top hierarchical level by master feature, and any feature associated with it and which is in a low level by

978-1-4799-0792-2/13/$31.00 ©2013 IEEE

slave feature. The different notations used by the FM model to describe the variability in SPL are as follows:

- *Feature*: refers to a common characteristic or a variable aspect of the SPL, such as functionality or a non-functional property (e.g. Level of security).
- *Mandatory*: corresponds to a link between master and slave features showing that the last one is mandatory.
- *Optional*: used to link a master and slave features in order to indicate that the last one is optional.
- *OR*: connects a master feature to a set of slave features, mentioning that one or more slave features may be selected to form features of a given product.
- *Alternative*: connects a master feature to a set of slave features to say that they are mutually exclusive (only one among these slave features can form a characteristic of a given product).

The EFM model [7, 8] is an extension of the FM model that uses the concept of *Attributed Feature* (described by a set of quantitative or qualitative attributes). It also introduces *Require* and *Exclude* links to respectively express the dependency and the mutual exclusivity between two features, and associates a cardinality with *OR* and *Alternative* links to indicate the number of features that are or could be grouped together to form features of a given product.

To illustrate how the EFM and implicitly FM models, as well as each of the studied models, describe the variability of software, we consider an illustrative example of a SaaS application intended for organizing a trip. The application has a high level security and allows by default to book flight tickets and make electronic payments. It provides also the option to book transfer from the hotel to the airport and also to select a room at the hotel. In this context the parameters Card number, Date, Ticket flight type and the Price type are mandatory parameters. In addition, the application requires that the Card number should correspond to a Visa or MasterCard and the ticket type to be Economic or Business. It provides the price as VAT (Value Added Tax) or VAT Excluded formats. Optional parameters are the type of transfer, the hotel class, and the type of the room. By default the type of transfer is Car. However, it is possible to book a Van in option. In addition the hotel class is either 3*, 4* or 5*. The type of room is either Single or Double. This parameter is required in case the hotel class is provided. Fig. 1 shows the EFM model of Trip service.

IV. Decision Model

The DM model was introduced in the synthesis method [9]. It is defined as a set of decisions sufficient to distinguish between members of a family of application engineering products. Several approaches based on the DM model were proposed. Schmid and John [10] provide a common basis based on this model, using different notations: tabular, textual, etc. According to the tabular notation, the DM considers seven elements to describe the variability of a product line:

- *Decision*: interprets a variable feature of a product.
- *Decision description*: is a textual description.
- *Type*: indicates the type of decision (Boolean or Enumeration). A boolean decision may or not be a feature of a product. Besides, any decision of type Enumeration is a feature described by a list of variants.
- *Interval*: corresponds to values indicating whether a boolean decision is a feature of a product or not, or mentioning variants of a decision of type enumeration.
- *Cardinality*: Indicates the number of values that together can make a decision from a set of values of an interval.
- *Constraint:* is a condition that must be satisfied to consider a decision as a characteristic of the SPL.
- *Visibility/relevance*: is a condition that must be checked to consider the decision as a product feature.

Fig. 2 illustrates the DM model for the Trip service example in tabular form.

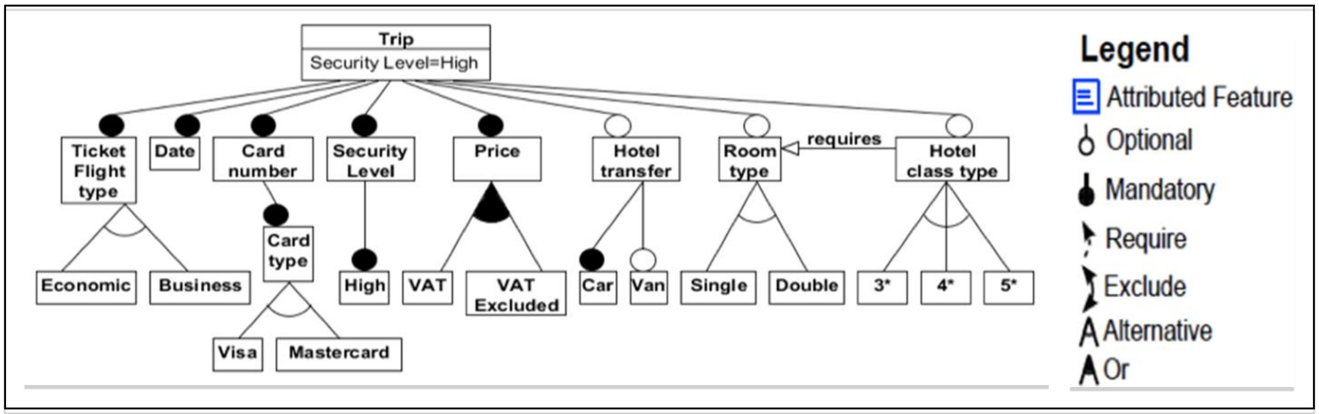
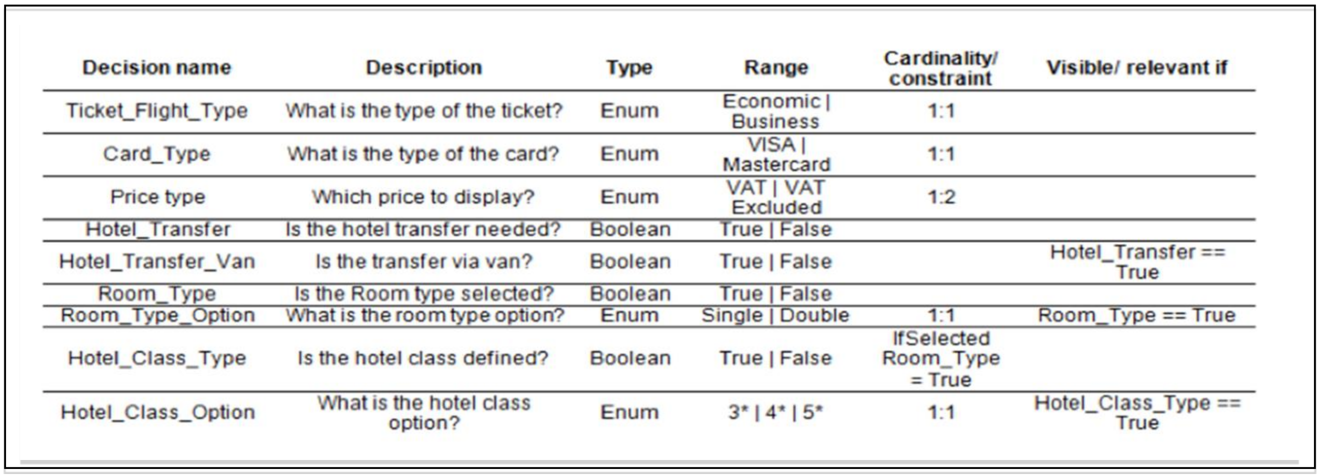


Fig. 1. EFM Model for Trip Service Example



Decision name	Description	Type	Range	Cardinality/ constraint	Visible/ relevant if
Ticket_Flight_Type	What is the type of the ticket?	Enum	Economic | Business	1:1	
Card_Type	What is the type of the card?	Enum	VISA | Mastercard	1:1	
Price type	Which price to display?	Enum	VAT | VAT Excluded	1:2	
Hotel_Transfer	Is the hotel transfer needed?	Boolean	True | False		
Hotel_Transfer_Van	Is the transfer via van?	Boolean	True | False		Hotel_Transfer == True
Room_Type	Is the Room type selected?	Boolean	True | False		
Room_Type_Option	What is the room type option?	Enum	Single | Double	1:1	Room_Type == True
Hotel_Class_Type	Is the hotel class defined?	Boolean	True | False	IfSelected Room_Type = True	
Hotel_Class_Option	What is the hotel class option?	Enum	3* | 4* | 5*	1:1	Hotel_Class_Type == True

Fig. 2. DM Model for Trip Service Example – Tabular notation

V. Orthogonal Variability Model

The OVM model is a graphic notation that only focuses on the variable aspects of SPL [11]. As shown in Fig. 3, it uses two main concepts: the Variation Point (VP) describing a variable aspect and Variant (V), and a set of links:

- *Mandatory VP* (MVP): a feature of each SPL product.
- *Optional VP* (OVP): an optional feature that not necessarily all SPL products should have.
- *Variant*: describes a possible instance of a VP.
- *Mandatory*: connects a VP to one or more V as default instances characterizing the products sharing this VP.
- *Optional*: connects a VP to one or more optional V that are not common features of products sharing this VP.
- *Alternative choice*: groups (totally or partially) a set of alternative optional variants.
- *Cardinality*: determines the number of alternative variants that can be grouped in the same time.
- *Requires*: expresses the dependency between VPs/Vs.
- *Excludes*: expresses the fact that two features (VPs or Vs) are mutually exclusive.

Fig. 4 shows the OVM model of the Trip service example.

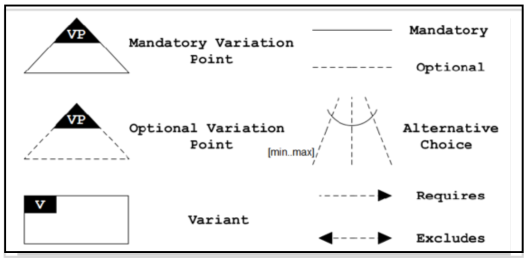


Fig. 3. OVM Model Elements

VI. Comparison of Variability Models

By analyzing the SPL models, we can identify elements necessary to describe the variability of software and that can be used to compare them as shown in Table 1. This table shows that the FM and EFM models present an ambiguity when describing variable properties and variants that are described respectively by the same element *Feature* and *Attributed Feature*. The DM model meanwhile, presents a description complexity when describing an optional variable property. Two decisions must be considered: a boolean decision stating that the property is optional and an enumerated one specifying the values of the variant (e.g. the room type in Trip service). This complexity is also observed when describing DM variants.

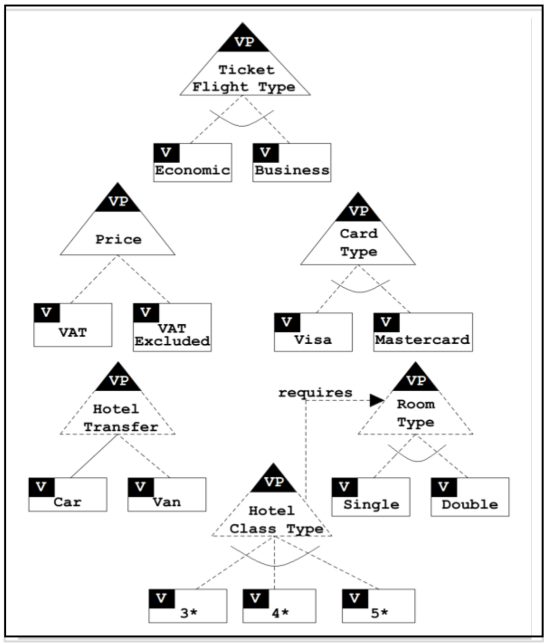


Fig. 4. OVM Model for Trip Service Example

TABLE I. COMPARISON OF MODELS DESCRIBING THE SPL

	FM	DM	OVM	EFM
Notation	Graphic	Tabular	Graphic	Graphic
Static properties	Feature	-	-	Attributed Feature
Mandatory variable properties	Feature	Enumerated decision	Mandatory Variation Point	Attributed Feature
Optional variable properties	Feature	Boolean decision + Enumerated decision	Optional Variation Point	Attributed Feature
Mandatory variant	Feature + Mandatory/OR/Alternative links	-	Variant + Mandatory/OR/Alternative links	Attributed Feature + Mandatory/OR/Alternative links
Optional variant	Feature + optional link	Enumerated range member or Boolean decision	Variant + Optional link	Attributed Feature + optional link
Aggregation	OR + Alternative links	Cardinality	Optional + Alternative links + Cardinality	OR + Alternative links + Cardinality
Dependencies relationships	Requires + Excludes links	Constraints	Requires + Excludes links	Requires + Excludes links
Conditions	-	Constraints	-	-
Non-functional properties	Feature	Decision	Variation Point	Feature attribute

Furthermore, at the level of aggregation, the OVM and EFM models tend to be more appropriate. They use OR and Alternative links associated with cardinality to express the possibility of grouping together many objects. On the other hand, the DM model associates cardinalities with decisions without specifying the regrouping type (e.g. the parameter Price can be provided as VAT and/or VAT Excluded). Besides, the FM, OVM and EFM models provide two explicit links (Requires and Excludes) to express the dependency relationships. This type of relationships can be expressed using constraints in the DM model but it would be difficult to automatically distinguish it from other types of conditions. The non-functional properties are described in the same way as any other property in the FM, DM and OVM models, and as feature attributes in the EFM model. This approach, even if it simplifies the description of this type of properties, it does not allow to represent an attribute (e.g. response time) that is less (or higher) than a given value.

Finally, we note that the OVM model remains the most appropriate compared to the studied models, even if it doesn't foresee to describe features by conditions and focuses on variability aspects and contains no elements to describe static properties (common features). We believe that it is more appropriate to model any static property according to standard models recommended for modeling the application (e.g. Web services standards).

VII. SYNTHESIS AND CONCLUSION

The literature revue shows that the SPL variability models are advocated for modeling SaaS applications in order to adapt them to users' needs. Besides, the results of comparison of these models lets us conclude that the OVM model is the most appropriate one because of its simplicity, its precision and its conciseness. However, we think that this model, along with its competitor the EFM do not cover, in addition to the conditions on the objects and the non-functional properties, the semantic aspect that allows describing with more precision the SaaS applications. Especially, when these applications are Web services, describing their variable aspects' semantics is important for their automatic discovery and composition. In this scope, the extension of the OVM model is a perspective of this work.

REFERENCES

[1] R. Mietzner, A. Metzger, F. Leymann and K. Pohl, "Variability Modeling to Support Customization and Deployment of Multi-Tenant-Aware Software as a Service Applications", PESOS'09, May 18-19, 2009, Vancouver, Canada.

[2] ST. Ruehl and U. Andelfinger. "Applying Software Product Lines to create Customizable Software-as-a-Service Applications", SPLC'11, August 21-26, 2011, Munich, Germany.

[3] J. Schroeter, P. Mucha, M. Muth, K. Jugel and M. Lochau, "Dynamic Configuration Management of Cloud-based Applications". SPLC'12, Vol. II, September 02 – 07, 2012, Salvador, Brazil.

[4] J. Schroeter, S. Cech, C. Wilke, U. Assmann and S. Goetz, "Towards Modeling a Variable Architecture for Multi-Tenant SaaS-Applications", VaMoS'12, January 25-27, 2012, Leipzig, Germany.

[5] S. Götz, C. Wilke, S. Cech and U. Assmann, "Sustainable Green Computing: Practices, Methodologies and Technologies", chapter Architecture and Mechanisms for Energy Auto Tuning. IGI Global. 2011.

[6] K. Kang, S. Cohen, J. Hess, W. Nowak and S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study", Technical report, CMU/SEI-90TR-21, 1990.

[7] D. Benavides, P. T. Martin-Arroyo and A. R. Cortés, "Automated reasoning on feature models", In Proceedings of the 17th International Conference on Advanced Information Systems Engineering (CAiSE '05), Springer, 2005, pp. 491-503.

[8] L. T. Passos, T. Berger, M. Novakovic, K. Czarnecki, Y. Xiong and A. Wasowski. "A study of non-boolean constraints in variability models of an embedded operating system". In Proceedings of the 15th International Software Product Line Conference, Vol. 2, pp. 1–2:8, 2011.

[9] Software Productivity Consortium Services Corporation, "Reuse-Driven Software Processes Guidebook", Version 02.00.03, Technical Report SPC-92019-CMC, 1993.

[10] K. Schmid and I. John, "A Customizable Approach to Full-Life Cycle Variability Management", Science of Computer Programming, Vol. 53, Issue 3, pp. 259–284, 2004.

[11] K. Pohl, G. Böckle and F. Van der Linden, "Software Product Line Engineering: Foundations, Principles, and Techniques", Springer, 2005.