*Available online www.jsaer.com*

***Journal of Scientific and Engineering Research*, 2017, 4(7):309-316**


**Review Article**

**ISSN: 2394-2630**
**CODEN(USA): JSERBR**

# Feature-oriented engineering of web applications

**Sidi Mohamed O. Moulaye Abdellahi[1], Mbaye SENE[2], Mohamed T. Kimour[3]**

[1]Department of Quantitatives Methods, Faculty Economics and Law, University of Nouakchott, Mauritania.
[2]Department of Computer Science and Mathematics, University of Cheikh Anta Diop-Dakar University, Senegal
[3]Department of Computer Science, University of Badji Mokhtar-Annaba University, Algeria

**Abstract** Web applications constitute crucial elements of the global information infrastructure. While the rise of their number and complexity is ever increasing; the cost and time effective development of web applications is one of the most challenging areas of software engineering. In this paper, we present a product-line approach to web application engineering. Software Product Line (SPL) is an emerging paradigm whose aim is to reduce both costs and time-to-market in the development of system families by the exploitation of commonalities among family members. Since web applications often share similar functionalities, instead of dealing with single products, firstly designing system families is an effective way leading to valuable benefits on development time and quality factors.

**Keywords** Web Applications, Software Product Lines, Feature diagram, Navigation model.

## Introduction

The requirements for web applications involve a great diversity of different functionalities and characteristics, but also a large number of complexities involved. The engineering of web applications implies the use of many different technologies in order to satisfy all user requirements [1-2]. On the other hand, the economic factors and the growing complexity of such applications require higher return on development time and cost [3]. This is why there is a need for adequate and efficient engineering methods for developing robust web applications.

Software product lines (SPL) are software families that share common functionality, where each member has variable functionality [4]. SPL engineering is one of the most promising approaches to reduce the development costs, as well as to increase the quality of families of similar software products [3-4].

As an established field with considerable methodological support, the main goal of SPL engineering is the agile and speedy development of member systems by taking advantage of reusable assets from all phases of the development lifecycle [3].

An SPL engineering requires to describe the products using variability models, such as Feature Models (FM) that contains only features and relationships between them. A feature is considered as a characteristic of the system that is observable by the end users [3]. Feature models describe which features are present in all the products, called core features, and which not, called variable features.

On the other hand, in web applications, we often find increasingly similar behaviors, shared by various products, used in similar environments to fulfill similar tasks [4-7].

Sharing a common infrastructure, which builds the core of every member, and reusing assets which can be delivered to deploy recurrent services is always more becoming profitable to effectively pursue planned software reuse and to lead to a considerable decrease in effort needed for the development of a single web application. This is why firstly designing product families instead of starting with a single product is an effective means upon software reuse [8].

SPL defines two major engineering processes; domain engineering, and product engineering. The former defines the feature model and its mapping to the identified reference architecture. The latter defines the process of using the reusable assets to derive the product. In doing so, the value of a SPL is finally realized in reuse-based application development.

In this paper, we present a product-line web engineering method, offering seamless design process of product families to take advantage from the emerging SPL paradigm, whose aim is to reduce both costs and time-to-market in the system families development, by the exploiting commonalities among family members.

We define a development process, based on three phases. We develop feature models and use cases with variability during the requirements modeling phase. Entity and navigation models are built during the analysis modeling, and the architectural model is identified at the design modeling phase. Our approach shows how to model variability in software product lines in the context of web domain by using defined multiple-view variability modeling, based on UML.

Moreover, in software engineering, product-line is a new paradigm, where systems share common characteristics "features". It is a set of products that share a common set of requirements, but each product has its own unique set of features. Product-line engineering mainly simplifies the design and maintenance of program families and addresses the needs of highly customizable applications in a cost-effective manner [8]. The motivation for it is to obtain desired benefits in terms of productivity, costs; effort, and time to market [9-11].

The structure of the paper is as follows. In section 2 concepts and foundation of SPL is introduced. Section 3 presents our approach using a case study. Section 4 compares our approach with some existing solutions that share the same objectives. Finally, section devoted to the conclusion and future work is closing the paper.

**Software Product Lines**

Software product line (SPL) is a software engineering paradigm for software development. SPL is important in promoting software reuse, leading to higher productivity and quality [4, 12] by realizing large scale software reuse, SPL approach for software development can generate important quantitative and qualitative improvements in productivity, time to market and customer satisfaction [4]. Their growing success comes from their ability to offer companies ways to exploit their software products commonalities to achieve economies of production [13].

A software product within a product line often has specific functionalities that are not common to all other members within the product line. Those specific functionalities are termed "variant features" in a product line. SPL paradigm involves the modeling of variant features [12].

A feature is a system property that is relevant to some stakeholder and is used to capture commonalities or discriminate among products in a SPL. Features are organized into a tree representation, called features diagram, with a specific notation for each variability category (mandatory, alternative and optional). A feature model refers to a features diagram accompanied by additional information such as dependencies among features. It represents the variability within a system family in an abstract and explicit way.

SPL engineering encompasses the creation and management of families of products for a particular domain, where each product in the family is derived from a shared set of core assets, following a set of prescribed rules [12]. By using a software product line, developers are able to focus on product specific issues rather than on questions that are common to all products [13].

The key concept to the development of system families is variability, intended as the ability to derive various products from the product family [12]. Variability is the ability to change or customize software systems [13]. It refers to the variable (optional, variation points and variants) features of an SPL [10].

Features describe an outstanding, distinguishable, user visible aspect, quality or characteristic of a software system or system [3, 14]. Features and their relationships are structured in a feature model. It is produced by domain analysis, through the analysis of commonalities and differences among a family of products.

Variability is not only used to define the SPL and the product derivation, but also to handle software evolution by anticipating some types of variability and construct a system in such a way that it is prepared for inserting predetermined changes [14]. Leveraging commonalities between members of a product family as well as across

similar product families emerged as an effective way of pursuing software reuse [4].

In general, features are represented diagrammatically as a tree diagram. Nodes in feature diagram show how some features are composed of lower level features, which may be mandatory (that must always be present), optional, or alternative selected at construction time or dynamically loaded at run time, at variation points. Various relationships exist among these features, such as generalization, aggregation, utilization, and mutual dependency.

SPL engineering is separated in two parts: Domain Engineering and Application Engineering. They constitute two relatively independent development cycles, i.e. development for reuse, meant as development of the product line itself, and development with reuse, also called product instantiation.

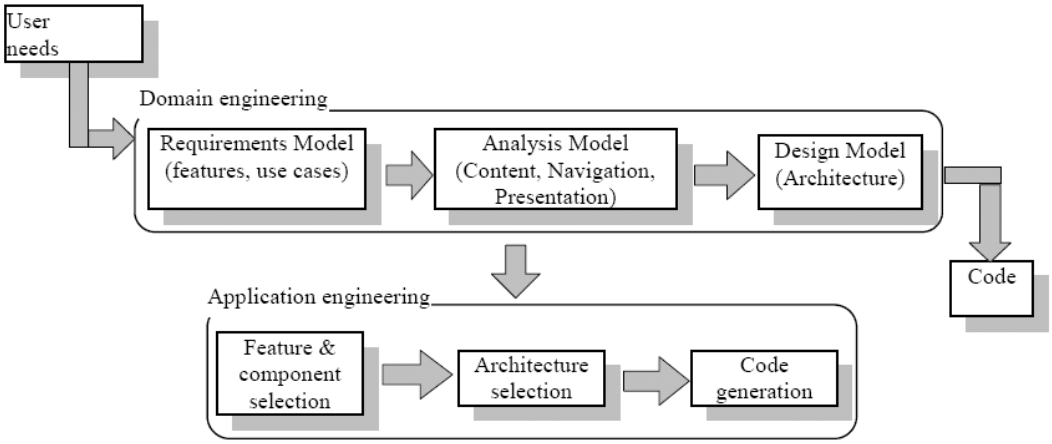


*Figure 1: SPL web engineering process*

Domain engineering involves, amongst others, identifying expressed as reusable and configurable requirements, analysis commonalities and differences between products' domain and design models, and so on. In general, any reusable work artefacts are work products of the domain engineering process, product is referred to as a reusable asset [13].

In contrast to domain engineering which covers a whole product line, application engineering focuses on deriving a certain product, using a subset of the shared software artifacts.

This is achieved by reusing domain artifacts exploiting the product line variability [4]. In this stage, each variation point, as defined in the previous stage, is bound to a specific variant, selected from the set of variants associated with it. A Product Configuration gives a list of selected and deselected features according to the variability model of the product line. This product configuration together with the feature mapping is used to derive the product implementation by reusing existing domain artifacts [14].

**SPL Web Engineering Approach**

Web applications can be considered as software products derived from a common infrastructure and assets which capture specific abstraction in the domain, e.g. user registration, shopping cart, and checkout in an online retailing system [1, 7, 16].

Naturally, our SPL web application engineering process (Fig. 1.) is conducted through two passes: domain engineering and application engineering. In the former, the analysis of commonalities and differences among a family of products, domain analysis produces a Feature model.

The purpose of domain engineering is to develop domain models that may be used in developing products for a given domain. As shown in Fig. 1, we define three phases in domain engineering: Requirements Modeling, Analysis modeling, and design modeling. Feature modeling is an activity that starts at the requirements modeling phase and may be refined along the remaining phases.

**A. Requirements model**

In our SPL Web engineering method, the requirements to be collected are not for one application but for an entire family of applications (domain engineering). We start by the requirements modeling where the feature

model and use cases are constructed (from the end user's point of view).

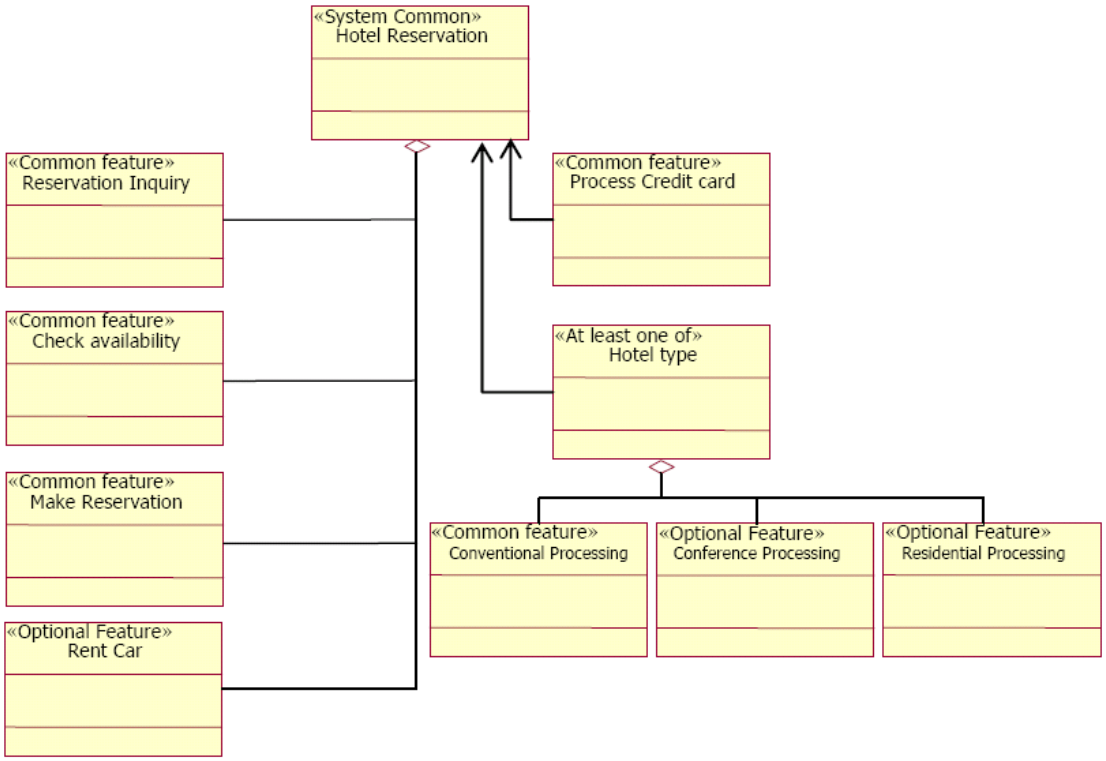


*Figure 2: An example of feature model for a Hotel Reservation System*

Feature modeling is the unifying view for modeling variability in software product lines [14]. Features can be incorporated into UML using the metamodeling concept [13][14], in which features are modeled as meta-classes and given stereotypes to differentiate between <<common feature>>, <<optional feature>>, <<default feature>> and <<alternative feature>>.

An example of a feature model is given for a hotel reservation system in Fig.2, which depicts System common feature, common features, optional features, as well as default features. Common features are: "Reservation inquiry", "Check availability", "make reservation", and "Confirm reservation".

The "at least one" feature is the "Hotel type". The default feature is the "conventional processing". The optional features are "conference processing" and "residential processing".

The functional requirements of a system are defined in terms of use cases and actors [14]. For a single system, all use cases are required. In a software product line, only some of the use cases, which are referred to as common use cases, are required by all members of the family [13, 14]. Other use cases are optional, in that they are required by some but not all members of the family. Some use cases may be alternative, that is different versions of the use case are required by different members of the family. We use the UML use case model, with stereotypes <<Common UC>> for common use case, <<optional UC>> for optional use case, or <<alternative UC>> for alternative use case. In the use case model for the SPL of Hotel reservation (Fig. 3), the "rent a car" as well as the "conference room reservation" use cases are optional. It is worth noting that variability can also be inserted into a use case through variation points, which specify locations in the use case where variability can be introduced [13-15].

The goal of use case analysis is to get a good understanding of functional requirements [15], whereas the goal of feature analysis is to differentiate between commonality and variability. Use cases and features complement each other. Optional and alternative use cases are mapped to optional and alternative features respectively, while use cases variation points are also mapped to features [13].

However, with use case modeling, it is more difficult to capture dependencies in use case variability, whereas a feature dependency can be explicitly captured in a feature model [14]. The same variability can be repeated in multiple use cases but is captured more effectively as one feature in a feature model.

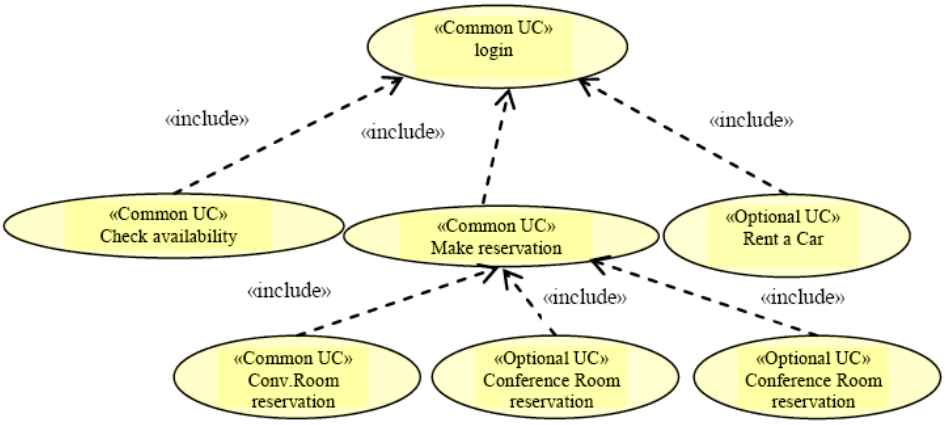


*Figure 3: Use Cases with variability*

**B. Analysis Model**

The analysis modeling aims to represent the functionality by using an object-oriented representation. In this phase the designer represents the main structures of the system and how they collaborate to achieve the system functionality. In UWE [6, 17], web application functionality is represented by three models: content, navigation, and presentation models. UWE provides the developer with guidelines to derive navigation model and presentation model from content model. UWE is an object-oriented approach based on the standard UML. UWE specifies a complete design life cycle for web-applications. It stipulates a number of models and model transformations according to OMG's Model Driven Architecture [6].

In our approach, we have follow UWE in the analysis model by defining the content model and the navigation model while adding variabilties to cope with the SPL approach.

**1) Content model with variability**

The content model (or entity model) aims to build a domain model trying to take into account as little as possible the navigation paths and presentation aspects. For a common Web application the use-cases and the feature model could be the base of the entity model identification in the domain.

We could use an incremental approach to identify classes. First, we identify the "active" entities in the system. Actors identified in the use-case will be considered as prime candidates for being listed as potential classes. Next, we identify the business domain ("passive") entities in the system.

It is worth noting that in data-oriented cases the base of the entity model should be the managed data, not the typical user activities and related use cases [18].

The entity model will be a UML class diagram which will determine the structure of the system, the classes and their relations (like association, aggregation, …).

In modeling software product lines, each class can be categorized according to its reuse characteristic using the stereotypes «Common entity» for a common entity, «Optional entity» for optional entity, in the class diagram representing the content model or the feature model. In UML 2.5, a modeling element can be described with more than one stereotype. For every feature in the product line, certain classes realize the functionality specified by the feature.

**2) Navigation model with variability**

The next stage in the development process is the navigation model design. The navigation model specifies which objects can be visited in a Web application [16][18]. Moreover, it defines the availability of the objects. In the navigation model's building process the developer takes crucial design decisions, such as which view of he conceptual model is needed and what navigation path are required to ensure the application's functionality [18]. The decisions are based on the conceptual model, use-case model and navigation requirements that the application must satisfy.

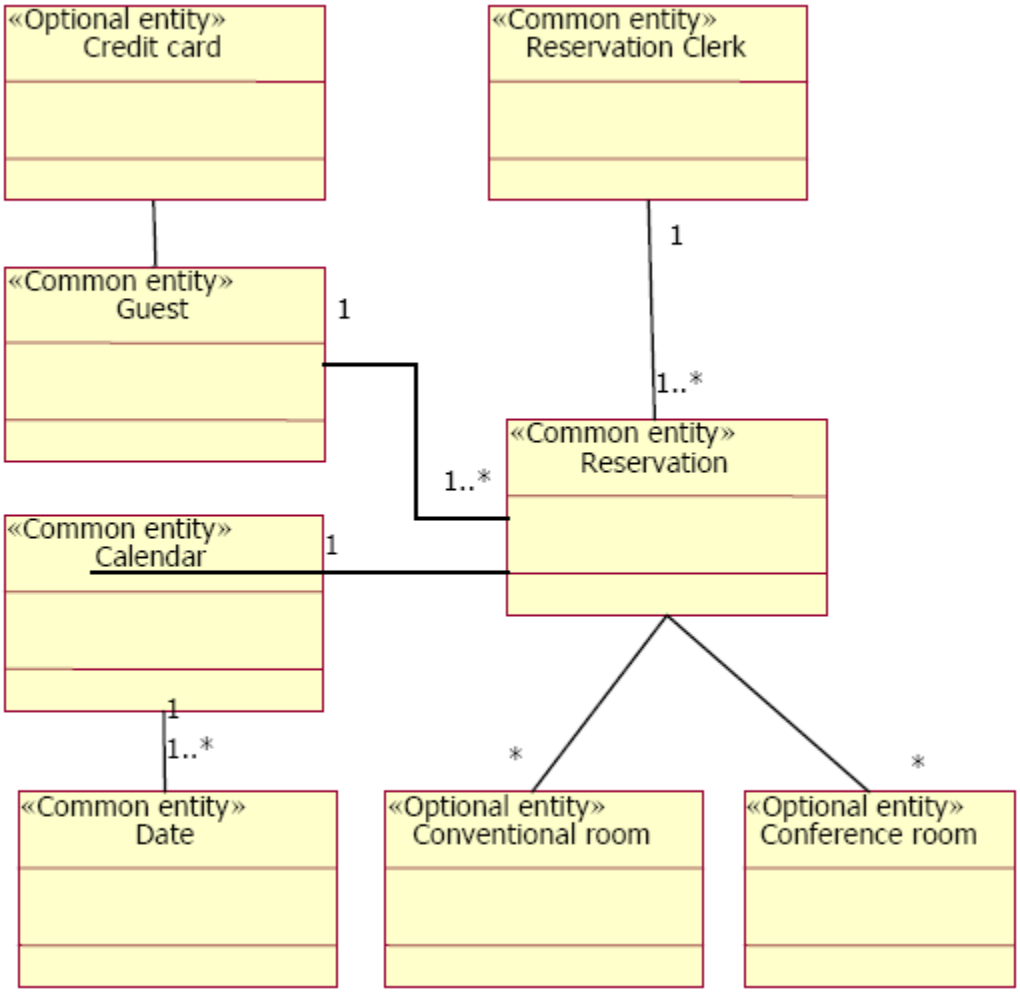


*Figure 4: Class entity model with variability for Hotel System*

The navigation modeling step uses the content modeling one as its input, which could be extended with additional associations [18]. To avoid a length greater than one in navigation path, such associations are generally added to represent a direct navigation. However, there could be some entity classes that are not targeted in the use-case model. In the navigation model, we do not represent them. Inside the web application, the navigation occurs along the associations.

We use them to describe the relation between navigation classes. In the user interfaces, such associations determine the hyperlinks. Although we use the navigation model as defined in UWE, we do not introduce new graphical elements in the representing diagrams; instead we add stereotypes in order to describe their functionality in the navigation. The approach could be seen in Fig.5, corresponding for the content model in Fig. 4 and the feature model in Fig. 2.

**C. Design model**

In the design modeling phase, we identify feasible architectures where the reusable software components and their configurations are built. With software product line strategy, software architecture presents a valuable framework for composing and adapting domain components. Generally, software architecture is built through the activities of partitioning high-level software specifications and allocating the divided functionalities into various architectural components, based on non-functional factors such as performance, security, availability, etc. [13-15]. Using the software architectures, domain components are composed and adapted for an organization's product lines. Finally, code may be automatically generated based on the selected software architecture.

**Related Works**

Today, the cost effective development of Web applications is one of the most challenging areas of software engineering. However, in the literature, there are very few works that deal with software product line paradigm in web applications [7, 19]. In [7], the authors proposed a framework called OOHDM–Java2a as a specialized architecture to develop Web applications. In such framework, one does not define variability at various development phases. The system family defined by OOHDM–Java2 is represented by the whole class of a web application; the commonalities are defined in a general manner. In [19], the authors show how model-driven product line engineering can be used to provide domain knowledge encapsulation, and how to automatic derive the product model from the domain one. These two proposed approaches suffer from the lack of straightforward separation of concerns in abstraction layers as done in our approach and do not show how to determine variabilities and to model them.

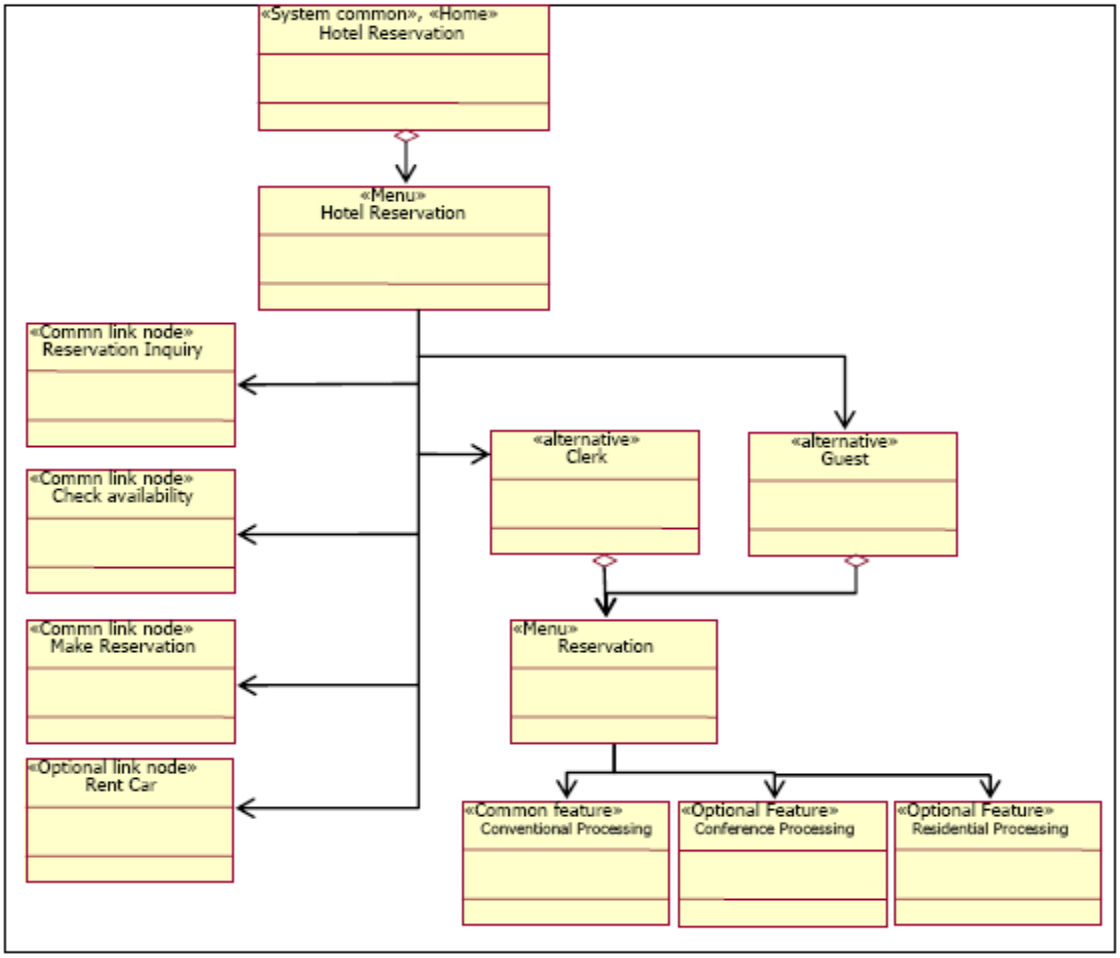


*Figure 5: Navigation model with variability*

**Conclusions and Future Work**

This work presents a contribution to the use of SPL in web engineering, by proposing an approach for modeling variability in software product lines in the context of web domain. We have developed a variability modeling in UML based multiple-view models of the SPL. Throughout three phases, these models define feature models and use cases with variability during requirements modeling, entity model and navigation model during analysis modeling, and architecture model during design modeling. We illustrate and demonstrate our method through a case study on the hotel management domain, clarifying and explaining the activities of the approach. As a future work, we are planning to apply our product line approach on web development to some other domain examples, while investigating the use of the existing SPL tools by our approach.

**References**


[1]. Roger, S. P. and David, L. (2009). Web Engineering: A Practitioner's Approach New York: McGraw-Hill. 2009.

[2]. Conallen, J. (2003). Building Web Applications with UML 2nd Edition, Addison Wesley, 2003.

[3]. Czarnecki, K., Antkiewicz, M. and Peter Kim, C. H. (2006). Multi-level customization in application engineering. Commun. ACM, 49(12):60-65, 2006.

[4]. Pohl, K. , Böckle, G. et Linden, F. V. D. (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Springer, New York, NY, 2005.

[5]. Ginegi, A., Murgesan, S. (2000). The Essence of Web Engineering, IEEE Multimedia, Vol. 8, no. 3, 2003.

[6]. Hennicker, R. and Koch N. (200). A UML-based Methodology for Hypermedia Design., UML´2000, LNCS 1939, Springer Verlag, 2000. 410-424.

[7]. Balzerani, L., Ruscio D. D., De Angelis, A. (2006). Supporting web applications development with a product line architecture, Journal of web engineering, vol. 5, no. 1 (2006) 025–042.

[8]. Riebisch, M., Streitferdt, D. and Pashov, I. (2004). Modeling Variability for Object-Oriented Product Lines. In: F. Buschmann, A.P. Buchmann, M. Cilia (Eds.): Object-Oriented Technology. ECOOP 2003 Workshop Reader. Springer, Lecture Notes in Computer Science, Vol. 3013, 2004, pp. 165 - 178.

[9]. Hallsteinsen, S., Hinchey, M., Park, S. and Schmid, K. (2008). Dynamic software product lines, Computer, vol. 41, no. 4, pp. 93–95, 2008.

[10]. Clements, P. and Northrop L. (2002). Software Product Lines: Practices and Patterns. Addison-Wesley, 2002.

[11]. Pohl K., Bockle, G. and Linden, F. J. V. (2005). Software Product Line Engineering: Foundations, Principles and Techniques. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[12]. Krueger C. W. (2006). New methods in software product line practice, Commun. ACM, vol. 49, no. 12, pp. 37–40, 2006.

[13]. Gomaa, H. (2005). Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures, Addison-Wesley, 2005.

[14]. Gomaa, H. and Shin, M.E. (2008). Multiple-View Modeling and Meta-Modeling of Software Product Lines, Journal of IET Software, Vol. 2, Issue 2, pp. 94-122, April 2008.

[15]. Jacobson, I., Booch, G., Rumbaugh, J. (1999). The Unified Software Development Process (Addison-Wesley, Reading, MA 1999) ISBN 0-201-57169-2.

[16]. Lou, B. (2014). Study on View-Oriented Navigation Modeling of Web Applications, TELKOMNIKA Indonesian Journal of Electrical Engineering, Vol.12, No.3, March 2014, pp. 2356- 236514.

[17]. Koch, N., Knapp, A.,Zhang, G. and Baumeister, H. (2008). UML-based web Engineering, chapter 7, pages 157-191. Human-Computer Interaction Series. Springer, 2008.

[18]. Adamkó, A. (2006). UML-Based Modeling of Data-oriented WEB Applications, Journal of Universal Computer Science, vol. 12, no. 9 (2006), 1104-1117, September 2006.

[19]. Martinez, J., Lopez, C., Ulacia, E. and Hierro, M. D. (2009): "Towards a Model-Driven Product Line for Web Systems", Proceedings of the Fifth International Workshop on Model-Driven Web Engineering (MDWE 2009), San Sebastián, Spain, June 22, 2009.