A First Approach to Build Product Lines of Multi-organizational Web Based Systems (MOWS)*

David Benavides[1], Antonio Ruiz-Cortés[1],
Miguel A. Serrano[2], and Carlos Montes de Oca Vázquez[2]

[1] Departamento de Lenguajes y Sistemas Informáticos,
E.T.S.Ingenieros Informáticos,
Universidad de Sevilla,
Avda. ReinaMercedes s/n, 41012 Sevilla, España
benavides@us.es, aruiz@lsi.us.es
[2] Centro de Investigación en Matemáticas
AP 402, Guanajuato, Gto., CP 36000, México
{masv,moca}@cimat.mx

Abstract. From the recent past and current state of the Internet, it is possible to forecast a wide growing of Multi Organizational Web–based Systems (MOWS). Therefore, the reduction of both costs and time–to–market is desirable. On the other hand, the success of building software in Product Lines (PL) is being demonstrated in different contexts reducing both time–to–market and costs. However, research on PL topics has not been oriented to include web–based assets. In this article, we propose a first approach to use PL methodologies to build MOWS. We identify quality aspects as a key point when building Product Lines of MOWS and we give a way to specify quality aspects in PL.

1 Introduction

The term "Product Line" (PL) was first introduced in 1976 by David Parnas [13] and it has been widely studied in Universities and in the industry since then [12,16]. The PL concept means that similar products share a common set of components and functionality (called core assets) and still, each individual product have some functionality specific to it (variable part). For the software industry, it is a good approach since similar parts (e.g. those belonging to the core assets) are developed only once and reused since then. The goal for organizations that follow a PL development approach is to be able to develop their products, by developing the variable parts only, and not developing the whole product.

MOWS have been described in [14] as a particular case of federated systems [1] or work–flows with multiple organizations [7]. The main difference between federated systems and MOWS is that in federated systems providers are often reduced in number, known beforehand and do not change along the system life–cycle. In MOWS, providers are numerous, they are not necessarily known a priori and they can change along the system life–cycle.

* This work was partially funded by the Spanish Ministry of Science and Technology under grant TIC2003-02737-C02-01 (WEBMADE)

Until now, PL engineering has not taken into account web–based assets. At the same time, a process to build MOWS reducing both time–to–market and costs is not well defined. In this paper, we propose a first approach that uses PL engineering when building MOWS. The products in the PL are composed by web–based assets offered by providers on the Internet. In this article, we focus on giving a step ahead to use PL methodologies to build MOWS pointing out the importance of quality aspects. In addition, we propose an approach to specify quality aspects in product lines.

The paper is organized as follows. In section 2, we give an introduction to MOWS. In Section 3, we briefly explain the PL concept and we mention some of the benefits of this approach. In Section 4, we present our approach to build PL of MOWS. Finally, in section 5, we present conclusions and related work.

2 MOWS in a Nutshell

The incredible success of the Internet has paved the way for an industry devoted to developing and running web services, which some authors tend to describe as the core of the next–generation Internet. Web services bring programmers a new way to develop advanced applications that are able to integrate a group of services available on the Internet into a single solution. These systems have been defined as Multi–Organizational Web–based System (MOWS) [5,14].

The reason why many experts forecast such a change from site-centric applications to web-service-based applications is the recent bankrupt of dotcoms. They have failed, not because of a lack of customers, but because of a lack of plans to make profit on them. In the near future, profit will be as important as ever, but the model will change from one based on competition to one based on collaboration. The success of the overall collaboration will lead the way to the success of each of the individual units.

We provide an example to illustrate MOWS. Consider that someone is interested in setting up a web portal specialized in providing movies. The portal offers a potentially infinite catalog of films and the same functionality as a domestic video player. The portal has three kinds of web services: a service for streaming videos over the Internet, a service for managing catalogs and keeping them up-to-date, and a service for managing virtual shops. Consequently, the portal becomes a service that integrates other more basic services that are provided by other organizations.

3 Product Lines in a NutShell

PL engineering has been widely used in several industries such as hardware or telecommunications. For instance, there are PL of processors that share some commonalities but still each processor has some specific characteristics. However, the PL approach has not been widely adopted in the software industry [4].

The PL concept means that in an industry there are a set of products sharing a common part, called core assets, and each of those products adds a set of specific characteristics, called variable parts. PL is a promising approach to achieve one of the main goals of software engineering which is software reuse.

It has been said that PL engineering is a good approach for product–oriented organizations [2], which have a well known market domain. PL engineering might not be suitable for organizations that are project-oriented because each project might have a different domain.

A PL includes an application domain, where PL engineering is applied, and assets that cover the derivation of products of such domain. An asset is a partial solution such as a component, a design document, or knowledge that engineers use to build or modify software products in PL [18]. The term 'domain' may have different meanings for the customers and the software engineers. We refer to domain in the same way that Bosch does [2]. He makes difference between *application domain* and *software domain*.

- *Application domain* is the definition from the customer's point of view where they refer to the product as a whole as being part of a particular domain. For instance, a word processor can be considered as part of the domain of building office suits software.
- *Software domain* is the definition from the software engineers' point of view where they refer to the product that may belong to different domains. For instance, the word processor may contain functions from different domains such as graphical user interfaces, text parsing, spell checking, etc.

3.1 Activities

There are three main activities in product line engineering: core assets development (also found in the literature as domain engineering), product development from core assets (also found in the literature as application engineering), and management tasks (the interaction between the first two activities). Core asset development is the part of the process where the analysis, design, and management of the assets of the application domain is done. Product development deals with the derivation or construction of new products from the different assets in the application domain. If there is a part of a product that is needed and it does not exist in the application domain, it may be developed, bought, rented or even shared from other domain. An optimal state for organizations would be to put most of the effort on core asset development activities and to put minimal effort on product development. Unfortunately, this situation does not happen often.

3.2 Example: A PL of Web Videos

We use the example of a web video provider mentioned in section 2 to illustrate a PL. The set of core assets are those that give the functionality of streaming videos on the Internet, manage catalogs and keep them up-to-date, and manage virtual shops. There could be a product line in this application domain. One product could offer access trough mobile phones, another could offer access trough digital TVs or other similar devices with different format. Both products share the same core and each of them has a variable part. The organization offering this service would be clearly product-oriented since it is specialized in a specific application domain. Once the set of core assets are developed, the main activities will be addressed to integrate a short amount of assets covering the variable part of each different product.

3.3 Benefits

PL approach has several benefits:

- Cost reduction because of assets reuse.
- A reduction of time–to–market because once the core assets are developed it takes less time to produce a new product.
- Many activities of the product life-cycle are shared among the products in the PL.
- There are common requirements, architecture, models, planning and so on.
- It is predictable that a quality improvement will appear due to the reduction of rework.

The main weakness we have identified in this approach is that if an organization wants to follow a PL approach, a large amount of work has to be carried out beforehand (e.g., defining the set of core assets, processes, architecture, common requirements).

4 Our Approach

Up to now, there have not been efforts to create a methodology to develop PL of MOWS. We propose a first approach to build MOWS in PL that follows three steps:

1. Include web–based assets in PL.
2. Study the impact of including web-based assets in the PL methodology.
3. Identify and specify quality aspects in the PL.

Our approach brings together two separates worlds of software engineering: MOWS and PL. We use PL to improve the techniques to produce MOWS and to profit from benefits of the PL approach such as cost and time-to-market reduction and quality improvement.

Our approach offers two important benefits:

- Reuse of components when developing MOWS.
- Reduction of work in core asset development because some of the core assets could be rented or outsourced.

Following with the example of a web video provider, it is possible to identify an application domain in this context and some parts that appear in all the products of this PL. Three kinds of web services, at least, will shape the set of core assets: a

service for streaming video on the Internet, a service for managing catalogs and keeping them up–to–date and a service for selling videos. From this base, it is possible to define different products in the PL. Movies could be visualized through mobile phones or through digital TVs. They are two different products addressed to different customers. Moreover, they share a set of common core assets and they have variable parts (e.g., a web video server or a mobile video server).

Features diagrams are used in the literature and industry to specify PL [6,8,11,9,17]. This type of diagram specifies a PL differentiating the common and variable parts.

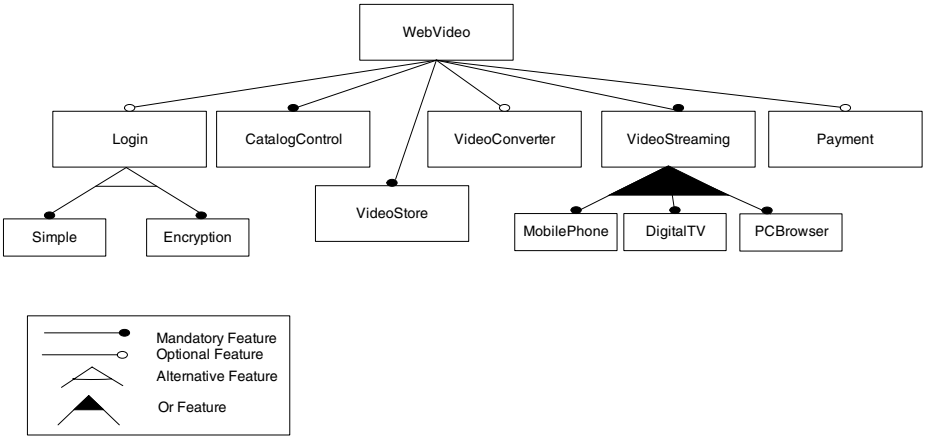

Fig. 1. Feature Model of a PL of Web Videos

In figure 1, we use the notation proposed in [6, pag. 86] to clarify our example. In this notation, there are four types of relations between features:

– Or-features: an or-feature of a set of or-features may be present in a feature model instance if its parent feature is included. Then, one or more features of the set may be present. It has the same meaning of an OR logical relation.

- **Mandatory features:** a mandatory feature is present in a feature model instance when its parent feature is present.
- **Optional Feature:** an optional feature may or may not be present in a feature model instance when its parent feature appears.
- **Alternative Features:** an alternative feature of a set of alternative features may be present in a feature model instance if its parent feature is included. Then, exactly one feature of the set is present. It has the same meaning of a XOR logical relation.
- **Or–features:** an or–feature of a set of or–features may be present in a feature model instance if its parent feature is included. Then, one or more features of the set may be present. (It has the same meaning of an OR logical relation).

In this example of a Product Line of Multi–Organizational Web–Based System (MOWS-PL), it is possible to define different products. In figure 1, there are five parts where different decisions define the set of possible products:

- First decision refers to whether to include a login service (optional feature) or not.
- Second decision implies to have a product in the video server with an image converter service or without it (optional feature).
- Third decision: if the login service has been including in the product, it will be a simple login service or an encrypted login service. In this case it is possible to decide only one option (alternative features).
- Forth decision represents the possibility of having visualization through a mobile phone, or a web browser or a PC Internet browser. However, it is possible to decide on having one or more services at the same time (or-features).

– Last decision represents the possibility of including a payment functionality (notice the possible dependency between this feature and the login feature. If a virtual payment is needed, an encrypted login service might be needed. This complex problem about dependencies is out of the scope of this paper).

With the present technologies and web services provided over the Internet, every feature of the feature diagram can be implemented by a web service. Several services interacting at the same time conform a MOWS, therefore it is possible to define a MOWS Product Line (MOWS–PL). As discussed in section 2, a particularity of MOWS is that providers are not necessary known beforehand and may change along the system life cycle. Thus, the key point that differentiate product lines using MOWS from those using local components, is that quality aspects of the features become more important. The functionality represented by a feature may be provided by more than one service, the decision of which service will provide the specific functionality is going to be based on quality aspects.

Up to now, we have not found any proposal in which quality aspects have been taken into account when specifying PL. With the inclusion of web–assets, the quality aspects become fundamental. Therefore, quality aspects need a specific way to describe them.

We propose to augment the feature diagram of figure 1 with quality features. In figure 2 only a fragment (Video Streaming) of the diagram is presented because of space constraints.

To be understandable, a feature diagram that includes quality information, must be accompanied with a catalogue of attributes describing the quality features. In figure 2, the catalogue is defined using the QRL (*Quality Requirement Language* notation [14]. Furthermore, we use discontinuous lines in quality features to distinguish between functional and quality features.

A quality feature is directly related to a quality attribute. Nevertheless, the values of these attributes should not appear in the feature model because different instances of the PL can have different values for each quality feature.

5 Conclusions and Related Work

There is a significant amount of research done about PL issues. The European research project, called CAFE [16] [16] whose main research area is on PL, has the collaboration of some of the most important organizations with representation in Europe. However, they have not addressed the introduction of web-based assets in PL.

There are some works in the literature suggesting the need of dealing with quality features. Kang *et. al* [9, pag. 38] suggested the need of taking into account quality features since 1990, when they described a classification of features. Later in 1998 Kang *et. al* [10] made an explicit reference to what they called 'non-functional' features (i.e., quality features or extra–functional features). However the authors proposed to include in the same model both functional and non-functional features without making any distinction in the model between them. Later in 2001 Kang *et. al* [3], proposed some guidelines for feature modeling. The authors made a distinction between functional and quality features and pointed out the need of an specific method to include quality features. However, they did not provide an specific way to do it.

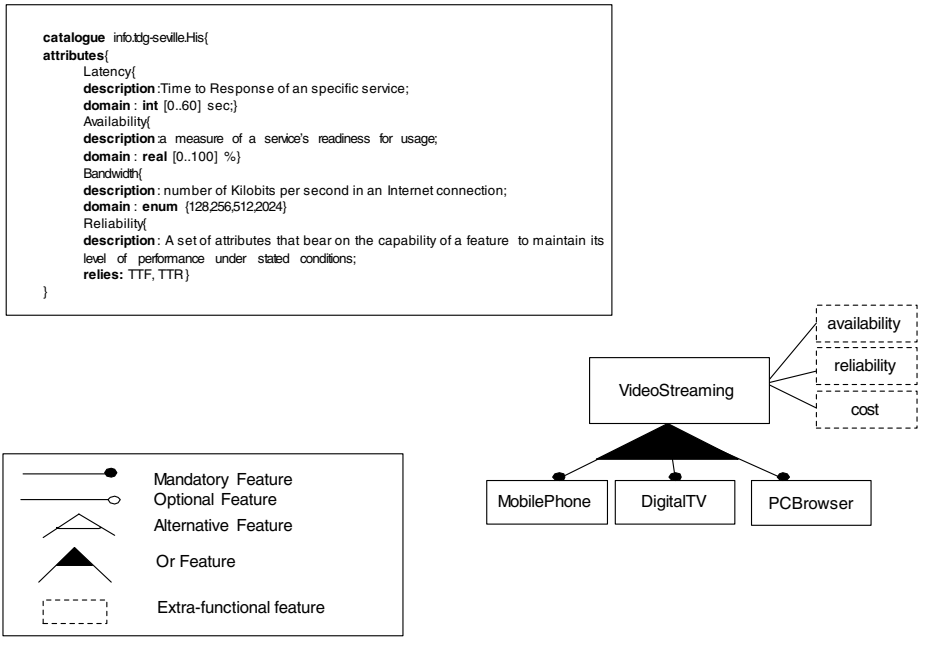

Fig. 2. Fragment of a feature model including quality features

In [15], the authors made a distinction between functional and what they called "parameters". The authors marginally introduced a way to deal with this kind of features. There is a drawback in this proposal: the authors proposed to include the values of the parameters in the feature model. With this proposal, the possible instances of the PL are limited to the values specified in the feature model. In this paper we introduced a way to specify quality features separating the definition of this kind of features at feature level and the range of values at instance level.

Any of the previous works was not focused on the inclusion of web-based assets. They only marginally describe the need of coping with quality features without giving an specific way to do so. In this paper, we identified that quality might become a major research topic when building Product Lines of MOWS (MOWS–PL). We have given a brief description of our approach to deal with quality features in MOWS-PL. We are currently working on implementing automated support tools for the derivations of products in MOWS–PL. This automates support will take into account quality issues.

References

1. P. Bhoj, S. Singhal, and S. Chutani. SLA management in federated environments. *Computer Networks*, 35:5–24, 2001.
2. J. Bosch. *Design and Use of Software Architectures*. Addison-Wesley, 1^{th} edition, 2000.

3. G. Chastek, P. Donohoe, K.C. Kang, and S. Thiel. Product Line Analysis: A Practical Introduction. Technical Report CMU/SEI-2001-TR-001, Software Engineering Institute, Carnegie Mellon University, June 2001.
4. P.C. Clements and L. Northrop. *Software Product Lines: Practices and Patterns*. SEI Series in Software Engineering. Addison–Wesley, August 2001.
5. R. Corchuelo, A. Ruiz-Cortés, J. Mühlbacher, and J.D. García-Consuegra. Object–Oriented Business Solutions. In *Chapter 18 of ECOOP'2001 Workshop Reader, LNCS nº 2323*, pages 184–200. Springer–Verlag, 2002.
6. K. Czarnecki and U.W. Eisenecker. *Generative Programming: Methods, Techniques, and Applications*. Addison–Wesley, may 2000. ISBN 0–201–30977–7.
7. P. Grefen, K. Aberer, Y. Hoffner, and H. Ludwig. CrossFlow: Cross–organizational workflow management in dynamic virtual enterprises. *International Journal of Computer Systems Science & Engineering*, 15(5):277–290, 2000.
8. M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, pages 76–85, Canada, 1998.
9. K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson. Feature–Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.
10. K.C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. FORM: A feature–oriented reuse method with domain–specific reference architectures. *Annals of Software Engineering*, 5:143–168, 1998.
11. K.C. Kang, J. Lee, and P. Donohoe. Feature–Oriented Product Line Engineering. *IEEE Software*, 19(4):58–65, July/August 2002.
12. L. Northrop. SEI's Software Product Line Tenets. *IEEE Software*, 19(4):32–40, July/August 2002.
13. D.L. Parnas. On the design and development of program families. *IEEE Transactions on Software Engineering*, SE-2(1):1–9, March 1976.
14. A. Ruiz-Cortés, R. Rorchuelo, A. Duran, and M. Toro. Automated Support for Quality Requirements in Web-Services-Based Systems. In *Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001)*, pages 184–200. IEEE-CS Press, 2001.
15. D. Streitferdt, M. Riebisch, and I. Philippow. Details of formalized relations in feature models using ocl. In *Proceedings of 10th IEEE International Conference on Engineering of Computer–Based Systems (ECBS 2003), Huntsville, USA. IEEE Computer Society*, pages 45–54, 2003.
16. F. van der Linden. Software product families in Europe: The Esaps & Café Projects. *IEEE Software*, 19(4):41–49, 2002.
17. J. van Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01), IEEE Computer Society*, pages 45–54, 2001.
18. J. Withey. Investment Analysis of Software Assets for Product Lines. Technical Report CMU/SEI-96-TR-010, Software Engineering Institute, Carnegie Mellon University, November 1996.