Product Line Development of Web Systems with Conventional Tools*

Miguel A. Laguna, Bruno González-Baixauli, and Carmen Hernández

Department of Computer Science, University of Valladolid,
Campus M. Delibes, 47011 Valladolid, Spain
{mlaguna,bbaixauli,chernan}@infor.uva.es

Abstract. Development of software product lines is a challenge for small organizations. Although the use of feature models is necessary to manage variability, we propose to use conventional tools for the rest of development activities. Traceability between the features and the UML architectural models is achieved by means of the package merge mechanism of UML 2. A similar strategy is applied at the implementation level, using packages of partial classes. The combination of these techniques and conventional IDE tools make the developments of product lines easier as it removes the need for specialized tools and personnel. This article reports a successful experience with these techniques in the domain of web applications.

Keywords: Software Product Lines, Feature Model, Variability, Traceability.

1 Introduction

Software product lines (SPL) are a proven reuse approach in industrial environments, based on the idea that each product of the SPL can be built from a common set of assets [3]. However, this approach is complex and requires a great effort by the companies that take it on [5]. The research we carry out aims to simplify the change from a conventional development process into one that benefits from the product line advantages in small and medium enterprises (SME).

As specific SPL development techniques, we must pay special attention to the variability and traceability aspects at each abstraction level. We need models that represent the product line and a mechanism to obtain the configuration of features that represent the best combination of variants for a specific application. There is wide agreement about using feature diagrams in some of their multiple versions like FODA [10] or FORM [9] to fulfill those requirements. A feature diagram is a tree, where the root node is the *concept*. The edges are used to decompose this concept into more detailed features by several types of decompositions.

Additionally, we must connect the optional features with the related variation points of the architectural models that implement the product line through traceability links. This explicit connection allows the automatic instantiation of the domain framework in each specific application, selecting or not the optional parts with respect to the particular functional and non-functional user requirements. However, this

* This work has been founded by the Junta de Castilla y León (VA-018A07 project).

M. Gaedke, M. Grossniklaus, and O. Díaz (Eds.): ICWE 2009, LNCS 5648, pp. 205 – 212, 2009.

traceability is not easily managed for several reasons [13]. An optional feature can be related to several elements in a UML model and vice versa. On the other hand, the same basic modeling mechanisms of variability (the specialization in class diagrams or the <<extend>> relationship in use cases) are used to express two variability levels: the design of the product line architecture and the design of a specific application that also has variations (for example two valid and alternative payment forms within a sales system). The solution to this problem has been achieved by modifying or adapting the UML structural and behavioral models, moving from the standard. The works of Gomaa [8] and Clauß [4] are examples of this approach, using stereotypes. Another solution proposed by Czarnecki in [6], consists of annotating the UML models with *presence conditions*, so that each optional feature is reflected in one or several parts of a diagram. All these solutions involve UML modifications. One of our initial restrictions was to maintain unchanged the UML meta-model, in order to use conventional CASE tools to model the product line. Other obligations were to locate at one point on the model all the variations associated to each optional feature or to separate the SPL variability from the variability of the specific applications.

In a previous work [11], we proposed the UML 2 package merge mechanism to represent the SPL architectural variations. This mechanism permits a clear traceability between feature and UML models to be established, associating a package to each optional feature, so that all the necessary changes in the model remain located. Therefore, the architectural model (including structural –class diagrams-, behavioral -use cases-, and dynamic –interaction diagram- models) is built starting from a *Base* package that gathers the common SPL aspects. Then, each variability point detected in the feature model originates a package, connected through a <<merge>> relationship with its parent package. These packages will be combined or not, when each product is derived, according to the selected feature configuration.

To support the approach, we have developed a Feature Modeling Tool (FMT[1]) and integrated it into Visual Studio, using the Microsoft DSL tools. The aim was to introduce product line as one of the project types provided by the development platform. Figure 1 shows the feature explorer view and the configuration tool (Figure 4 in Section 3 depicts a general view). The interface and underlining meta-model of FMT is similar to the *fmp* plug-in [1] and compatible with it, allowing the direct import of *fmp* models. The advantages of FMT are direct integration into the Visual Studio IDE and the possibility of visual representation and manipulation of features and *mutex/require* constraints. As additional benefits, the package structure of the product line and the configuration files can be directly generated.

To test the proposal in realistic situations, this article reports the practical experience with these techniques in the development of an e-commerce product line, as a representative example of web systems. A distinctive characteristic is the use of conventional CASE and IDE tools. In particular, we have used .NET and MS Visual Studio as development platform, with the FMT tool incorporated into the platform. Personnel involved vary from granted postgraduate students to undergraduates working on their term projects, but they are not specialists in SPL development. Sections 2 and 3 of this article are devoted to the description of the case study analysis and design and Section 4 concludes the article and outlines future work.

[1] http://www.giro.infor.uva.es/FeatureTool.html

2 Case Study: e-Commerce

The starting point has been the feature model proposed by Lau in [12], where a complete domain analysis using feature models is accomplished. However, the SPL is not implemented, providing us with a starting point to contrast the technique, since the packages that we must implement are imposed by an external independent study. The aim was not to implement hundreds of packages, but to reach a result with enough variability to show that it is possible to develop a functional product line in the web systems domain using a conventional development platform. Figure 1 (left) shows the initially selected features. Some of them are mandatory, as *Catalog, Product Information* and *Categories*. Others can be incorporated or not to the final product as *Shipping* and *Billing Address* or *Multilevel Categories*.

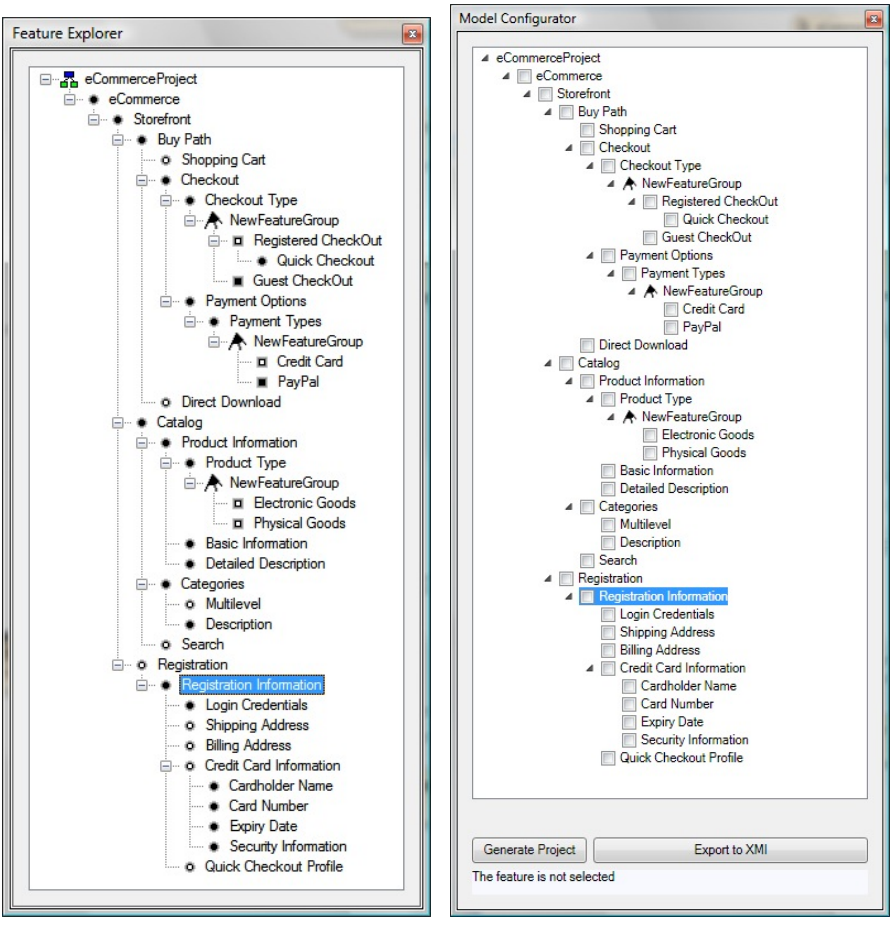

Fig. 1. FMT details of the e-Commerce product line: features tree-view and configuration tool

At this moment, the common part of the product line and a dozen packages have been developed. Therefore, we can already generate hundreds of e-commerce systems, from a minimal combination (that is, the simplest purchase process) to a typical portal with registered users, shopping cart, credit card secure payment, catalogs with multiple categories, search criteria, etc. The developed packages form a basic product family, but it continues growing with the successive packages in development.

3 Product Line Design and Implementation

The design has respected the basic ideas of the architecture proposed by Lau in [12], but organizing them in packages. The basic model of the product line is shown in Figure 2. This structure is automatically generated (in XMI format) from the Feature Modeling Tool using the transformation defined in [11] and now incorporated into the tool. The number of included features (near 40) is noticeably greater than the number of generated packages, as the design elements linked with mandatory features are included in existing packages. Most packages are optional and mutually independent. However, more complex situations have also appear: for instance, the *Electronic* and *Physical Product* packages correspond to an OR (1..2) structure in the feature model. This implies that at least one product type must be chosen. Another special situation is that the *Physical Product* package always requires the *Shipping Address* package to enable the effective shipment of the physical items. These restrictions are contemplated during the feature configuration process and automatically reflected in the valid packages combinations.

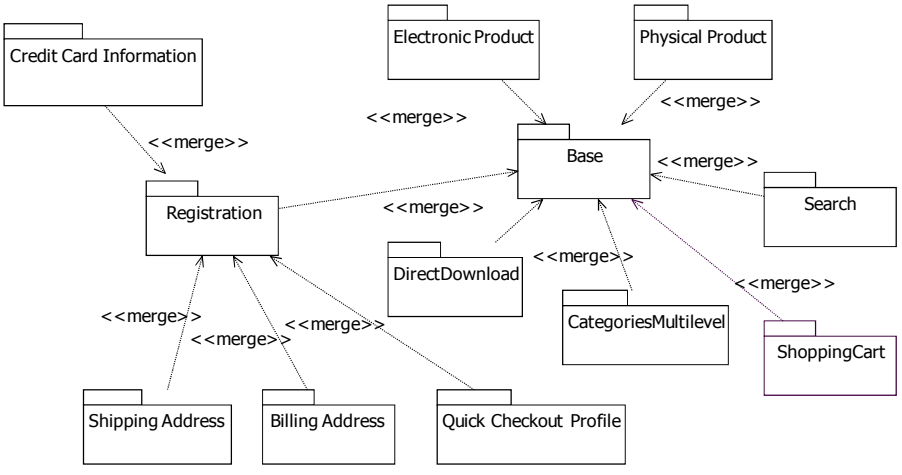

Fig. 2. Packages of the electronic commerce product line

We have achieved the complete development of the Base and eleven optional packages. Some of them are the following:

- *CategoryMultilevel* Package (supports multi-level categories in the catalog)
- *Search* Package (implements multiple criteria searches)

- *DirectDownload* Package (an electronic product can be directly downloaded)
- *ShoppingCart* Package (visualizes the detail of the products added to the cart at any time)
- *ElectronicProduct* and *PhysicalProduct* Packages (at least one must be included to enable the purchase process)

Other packages enable registration facilities, including a simplified payment process, when the user facilitates its payment preferences.

To trace the relationships between the different levels, this package organization is used throughout the development cycle of the product line: features, requirements (use cases and domain models), design (classes and interaction diagrams), and finally implementation. To extend traceability to the implementation, we use the concept of partial class of some languages. For example, C# permits to organize a project in packages with partial classes and, later, when two or more packages that contain classes with the same name are selected, the compilation process combines them in a unique class. The approach reproduces the same strategy used in requirements and design levels at the implementation level. Consequently, once the packages have been implemented, to derive a concrete application, we must uniquely indicate the selected packages corresponding to the feature configuration to the compiler. Thus, the goal of one-to-one traceability from features to code is achieved. The actual implementation of the e-commerce PL has been done using the .NET/ ASP as platform, C# as language, and Microsoft Visual Studio (including FMT) as IDE tool. In addition to domain classes, the implementation details must also solve two main problems: the user interface and data persistence. In the case of persistence, a pragmatic solution has been adopted, using a database that contains all the possible tables and columns. We continue working on the right solution: the *ad hoc* generation of the database schema as a part of the automated product configuration process. Consequently, the methods that handle access to the database use partial data structures, so that some columns of the database tables remain unused.

As for the user interface problem, we have used a combination of templates, cascade style definition files, and dynamic containers. In ASP.NET, it is possible to create master pages that (combined with .css files) serve as templates to the web system. Since we are not developping a closed application, each concrete product in the SPL will possibly have a different main page from the view point of final users. The variability mechanism of the template is achieved by using dynamic containers (*ContentPlaceHolder*) that will be filled in a dynamic way as specified in the code of each concrete application. As shown in Figure 3, each template is built from several pieces. The most interesting parts are on the left, where the different menus are, and the central part, corresponding to the specific content of the page. In both cases, a dynamic editable container has been used. To fill the containers, the methods of the associated C# classes (code behind in .NET terminology) add the needed dynamic controls, as well as their particular behavior. At the time of the configuration of each final product of the SPL, the compiler must recognize the necessary packages and the corresponding dynamic controls. To achieve this, a set of XML based configuration files are used to indicate paths, packages, etc. In addition to the default configuration files provided by the platform, other specific files have been added to each package. Thus, each page builds itself in a systematic way, using a name convention and the information about the necessary controls.

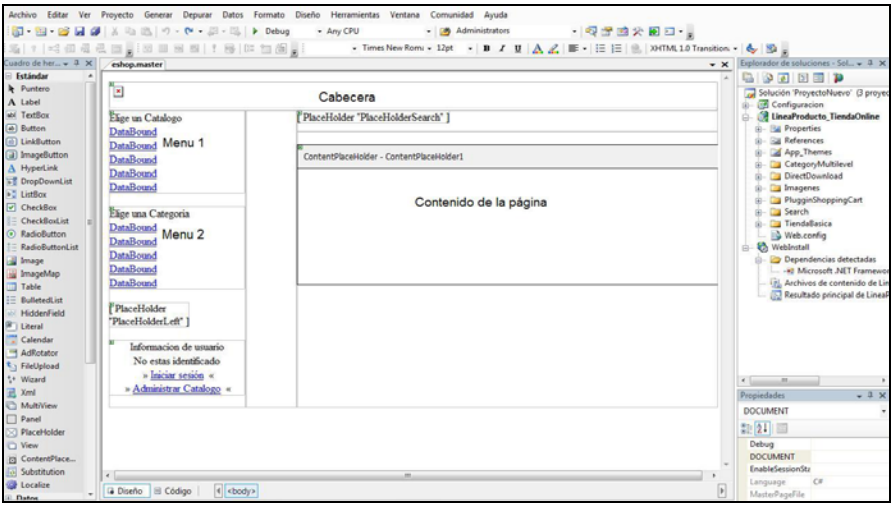

Fig. 3. Design view of a template with two dynamic containers

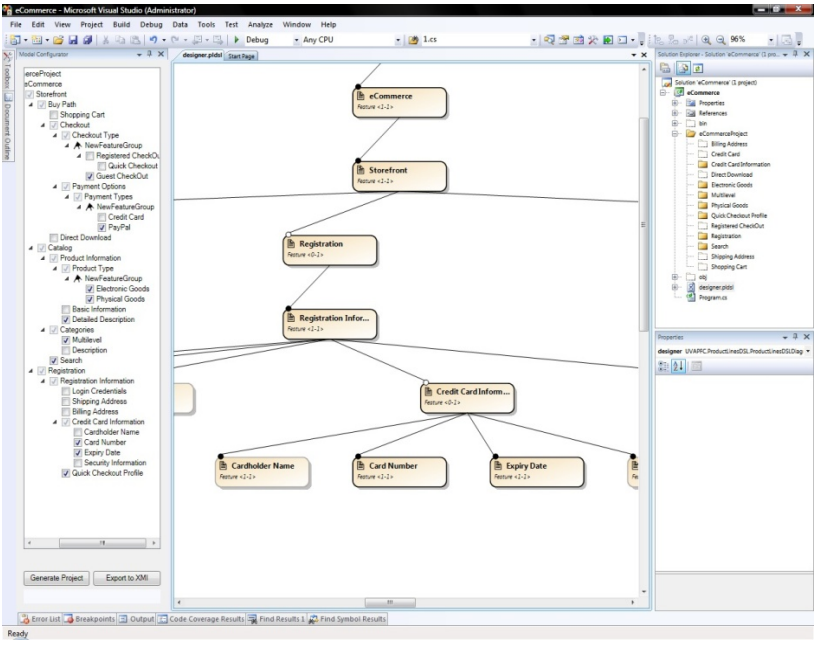

Fig. 4. Configuration process of the e-commerce product line with the detail of the selected (yellow) packages in the *solutions* explorer (right)

Figure 4 shows the way the packages are organized and configured. The available packages and their relationships are managed inside the IDE platform. In fact, the selection process is done using the FMT *configuration tool* (left part of Figure 4) and

this selection is automatically reflected in the Visual Studio *solution explorer* (packages of the right part of Figure 4). Once selected and validated, the project configuration is set and the compiler generates the final specific product that can be deployed and installed in minutes in the production server.

The results include the generation of several hundreds of variants, simply configuring and recompiling the SPL project into a concrete product. All the products include the basic purchase process, but specific products can include: Registered users, electronic or physical products, search facilities, credit or *PayPal* secure payment methods, etc. The details of how to install and to configure a product can be consulted in [7]. In Figure 5, two examples of final products with different degrees of complexity can be appreciated. To summarize, a realistic e-commerce product line has been developed, using a seamless approach, based on UML package merging and partial classes. At the same time, the necessary implementation techniques to handle variability at code level have been established. In general, the experience with graduate students has been satisfactory as they have reached the objectives with a reasonable effort (three to four months, four students working part time).

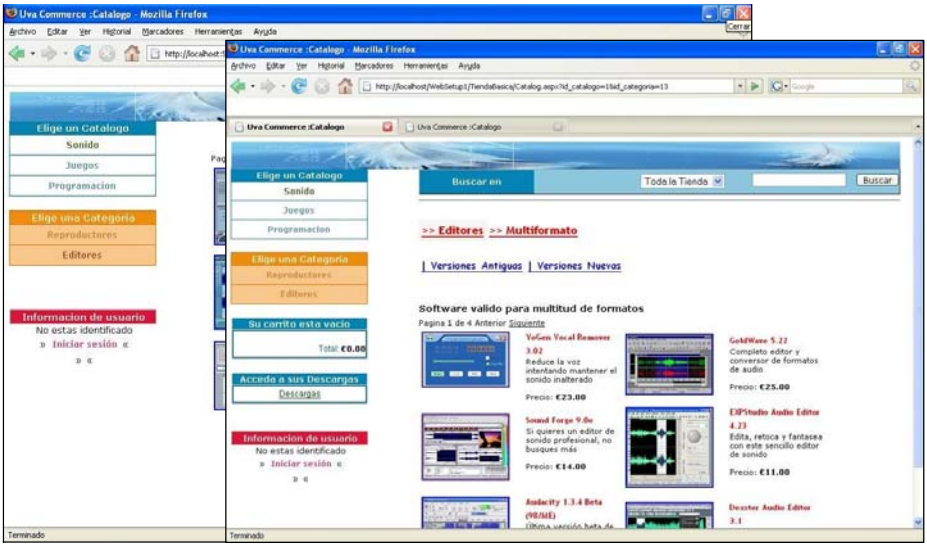

Fig. 5. Two variants of the electronic commerce product line as seen in a web browser

4 Conclusions

In this work the feasibility of a product line development approach in the web applications domain has been shown. The use of package merge and partial class mechanisms enables the automated generation of each product from the features configuration. Furthermore, the use of conventional CASE and IDE tools can simplify the adoption of this paradigm, avoiding the necessity of specific tools and techniques as in previous alternatives. The approach has been successfully applied to the design and implementation of an e-Commerce product line, based on a previous feature

analysis published in the literature. As a part of this work, a Feature Modeling Tool have been developed and incorporated into the Visual Studio IDE. This direct integration allows generating automatically (through XMI standard files) the UML package model structure and configuring the final products from the Feature Modeling Tool. Therefore, the configuration process is more transparent and straightforward for the application engineers.

Some commercial tools, such as Big-Lever Gears (www.biglever.com) or pure::variants (www.pure-systems.com) offer similar functionalities. Batory et al. have developed AHEAD, a set of java based tools that implements the *Feature Oriented Programming* paradigm [2]. Though these solutions are valid, the learning of new modeling or implementation techniques and the need of specialized CASE and IDE tools represent barriers for the adoption of the product line approach in many organizations; we therefore believe that the solution presented here improves the abovementioned proposals.

Current work includes the development of other product lines (in particular in the domain of non-lucrative associations), and the enrichment of the e-commerce case study. In this case, the objective is to evaluate the scalability of the proposal as the optional features increase.

References

1. Antkiewicz, M., Czarnecki, K.: Feature modeling plugin for Eclipse. In: OOPSLA 2004 Eclipse technology exchange workshop (2004)
2. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling Step-Wise Refinement. IEEE TSE (June 2004)
3. Bosch, J.: Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach. Addison-Wesley, Reading (2000)
4. Clauß, M.: Generic modeling using Uml extensions for variability. In: Workshop on Domain Specific Visual Languages at OOPSLA (2001)
5. Clements, P.C., Northrop, L.: Software Product Lines: Practices and Patterns. SEI Series in Software Engineering. Addison-Wesley, Reading (2001)
6. Czarnecki, K., Antkiewicz, M.: Mapping Features to models: a template approach based on superimposed variants. In: Glück, R., Lowry, M. (eds.) GPCE 2005. LNCS, vol. 3676, pp. 422–437. Springer, Heidelberg (2005)
7. García Gil, C., Izquierdo, Á., Juan, C.: Desarrollo de una Línea de Producto Software de comercio electrónico. PFC (2008), `http://giro.infor.uva.es`
8. Gomaa, H.: Object Oriented Analysis and Modeling for Families of Systems with UML. In: ICSR6, pp. 89–99 (2000)
9. Kang, K., Kim, S., Lee, J., Kim, K.: FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Eng. 5, 143–168 (1998)
10. Kang, K.C., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report, CMU/SEI-90-TR-21 (1990)
11. Laguna, M.A., González-Baixauli, B., Marqués, J.M.: Seamless Development of Software Product Lines: Feature Models to UML Traceability. In: GPCE 2007 (2007)
12. Lau, S.: Domain Analysis of E-Commerce Systems Using Feature-Based Model Templates., MSc Thesis, ECE Department, University of Waterloo, Canada (2006)
13. Sochos, P., Philippow, I., Riebish, M.: Feature-oriented development of software product lines: mapping feature models to the architecture. In: Weske, M., Liggesmeyer, P. (eds.) NODe 2004. LNCS, vol. 3263, pp. 138–152. Springer, Heidelberg (2004)