FACULTY OF SCIENCE
Department of Computer Science
Web & Information Systems Engineering
MASTER AFTER MASTER OF COMPUTER SCIENCE

An Approach for Experimenting with Website Designs

A thesis presented in fulfillment of the thesis requirement for
Master after Master of Computer Science degree

by

Sanaa Alsarraj

Student ID: 88449

Promoter: Prof. Dr. Olga De Troyer

August 2009

Abstract

Nowadays, Internet marketing is more widely spread and more competitive. This brought new challenges to developers in creating good web interfaces that would increase the sales or improve the marketing. Many experiments show that different designs can influence the customer's behavior. Therefore an environment for testing and studying customer interactions with different designs would be highly welcome. In this thesis we present an approach for facilitating the achievement of such web experiments. The approach allows us to produce different variants of webpage's interface. The variants share some common features but differ in other parts. In addition to that, the approach allows us to modify it to be adapted with some cultures of different clients. All this is done by reusing the core assets of the primary design and reusing the original code. The approach is based on two main technologies: software product line engineering and the aspect-oriented programming. The combination and integration of those two technologies allow us to generate a family of web applications while reducing time and efforts and achieving good performance. This is obtained by following several stages, including feature modeling to specify the common and the variable features between the different web interfaces, and capturing the variable features by aspect codes that modify the original implementation.

Acknowledgements

This research project would not have been possible without the support of many people.

I wish to express my deepest gratitude to my supervisor, Prof. Dr. Olga De Troyer who was abundantly helpful and offered invaluable assistance, support and guidance.

My large gratitude are also due to the members of the research group WISE.

I wish to express my love and gratitude to my husband; for his understanding and support through the duration of my studies.

I would like to express my special thanks to my parents for their encouragement and support they gave me during my study.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

The most influence of the ICT evolution is the Internet and the World Wide Web. In the economical field the Internet has opened new opportunities for business and commerce. Thousand of people are connected to the Internet from all around the world, mostly using search engines and surfing shopping websites. Internet can facilitate the communication and exchanging information among individuals and companies and has become an economic tool for both peoples of developed and developing countries.

In practice marketing over the Internet brought many challenges to websites developers, as the web page interface is the transmission medium and the message between sender (producer/company) and the receiver (customer/web user). Our main objective in this research work is to produce an approach for experimenting with website design to get the user interface that maximizes sales.

This section is divided as follow: first we give an overview of Internet marketing and its challenges, second we present culture customization concepts for the web and reviewing the culture values framework, third we talk about websites development and testing and finally in subsections four and five we address our research problem and our approach to perform a subjective steps of action to the solution of the research problem.

1.1 Internet marketing

Internet marketing can be defined as "*the use of the Internet and related digital technologies to achieve marketing objective"* [1].
The term e-commerce is used in the same context including e-commerce transactions. Internet marketing is usually associated with buying and selling over the Internet, actually there are many success stories about companies on the Internet one of them is the amazon.com for online shopping (website depicted on figure 1).

Indeed Internet represents many benefits for the marketing field [1], both for the customer and for the organization. For the customer it gives wider choices of products, services and prices from different suppliers. For the organization it gives the opportunity to expand into new markets, compete on larger business and develop new product and services, in addition to the improvement of the communication channel between the customer and the organization and providing control by better marketing research through tracking the customer behavior.

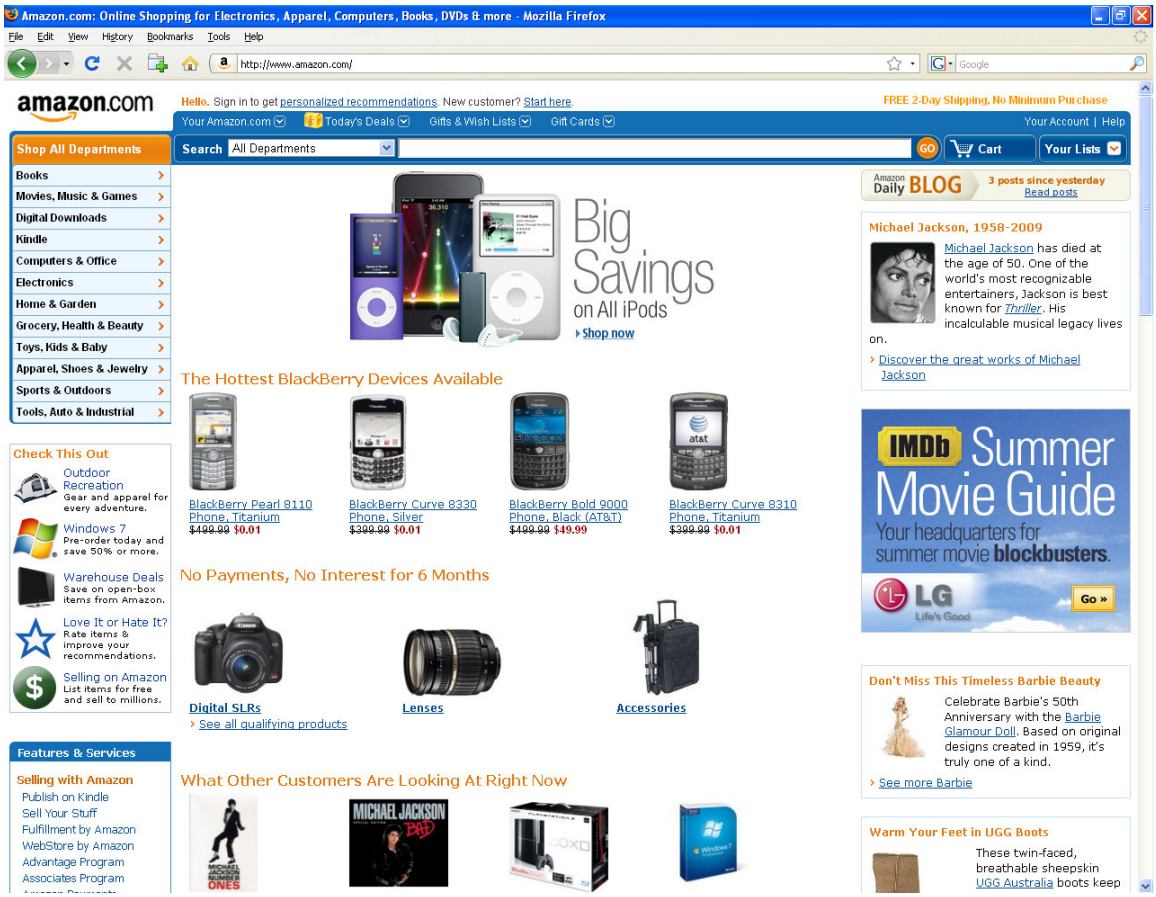


Figure1. amazon.com

Many people think that Internet marketing is just only access to a Website, but it is more than that. The Internet brings many and big challenge for the organizations in order to have effective Internet presence and the important thing is how this effective Internet presence is achieved. One of the key elements here is effective website design. Website design is considered as one of the key success factors because it can impact the buying process for a new purchaser. For example, one of the most powerful features of the websites is their facility to the purchase process, or in other words how to influence the customer to decide to purchase and how to not lose him if he decide.
The term conversion rate describes the percentage of success of the purchase process. The conversion rate is defined as the "percentage *of visitors who take a desired action"* [2] or usually the percentage of site visits that result in purchase, a simplified calculation of the conversion rate might be:
Number of orders / Number of visitors x 100 = Conversion Rate Pct.

Achieving a high conversion rate depends on many factors, one of them is the website interface design: how much is it attractive, well presented and usable.

Small changes in the web interface can affect the conversion rate and increase or decrease the percentages of sales. For example one small change could account for a 1,000 percent increase in volume [3]. These changes are considered as the variables visible on the web interface. There are thousands of such variables on a site, some of these variables have a major impact on increasing the conversion rate, but this is also depend on the target customers.

1.2 Culture customization on the Web:

The Internet and the World Wide Web provide global commerce without any national boundaries. However an important aspect that must be taken into account when designing e-commerce site is the culture of the target customers because it may widely affect their behavior.
Many researches shown that the culturally adapted web content enhance the usability, accessibility and website interactivity. Many companies realize this point and build their sites sensitively to the demographic profile of their customers.

1.2.1 Website globalization

Entering the global market has brought significant challenges to companies in order to expand their market and to keep the largest amount of customers. Customers satisfaction is the key factor here. Customers prefer to view content relevant to their market**.** From this point of view the idea of Website Globalization came.

Website Globalization enhances and facilitates visitor experiences on the site and motivates them to purchase or perform any other desired activity.

Website globalization is achieved by providing access to the site's content and functionality to visitors in different target markets and in their own languages.

According to the explicitly exhibited global or local features on the website, website globalization can be classified into five categories [4]:

1- Standardized websites: have the same content for all customer segments (domestic and international); there is no tend for translation or localization.
2- Semi-localized websites: have the same content for all customers, but provide contact information about foreign subsidiaries to address the needs of their international customers.
3- Localized websites: offer country-specific web pages with translation, wherever relevant.
4- Highly localized websites: offer country-specific URLs with translations, and include relatively high levels of localization by providing country-

specific information such as: time, data, zip code, number formats and so on.

5- Culturally customized websites: provide a complete immersion of the culture of the target market.

1.2.2 Cultural values framework

The authors in [4] introduced a framework for the cultural values (dimensions) that addresses the behavioral component of culture. Each country is a blend of all of the relevant cultural values: as such, for true cultural customization, all relevant values must be included. So if one of these cultural values is emphasized in a website, this make the site more closely customized to that cultural value. However, it is not the mere presence of these values that matters, but the degree to which they are emphasized in a website.
Here we present the five cultural values and describe how to emphasized them in the design.

1- individualism-collectivism

This cultural dimension focuses on an individual's relationship. Individualist societies have some characteristics like: no much ties between individuals, confirm on personal freedom, population are independent and individual decision-making is encouraged.
If the country of interest is toward individualistic, the following can be incorporated into a website: good privacy statement, independence theme, images with a single person, product uniqueness, and personalization features.

While in collectivist societies individuals are connected with strong relations, they believe in the importance of the goals of the group as a whole, there is emphasis on group decision-making, and group obligations, and have extended family structures. If the country of interest is toward collectivist a website can be adapted to conform to this value by incorporating the following: community relationships, clubs/chat rooms, newsletter, family themes, symbols and pictures of national identity, loyalty programs, and links to local websites.

2- Uncertainty Avoidance

This dimension focuses on how cultures adapt to changes, cope with uncertainty, ambiguity and have predictability and willingness for risk.
People from cultures high on uncertainty avoidance tend to have low tolerance for uncertainty and avoid ambiguous situations, have more formal rules, prefer details, put emphasis on security and risk avoidance, and view conflict and competition as threatening. While people from cultures low on uncertainty avoidance accept generalization, prefer fewer rules, approve of risk, tend to be modern and accept competition and conflict. To customize websites on this value, the following features can be incorporated: To emphasize high on uncertainty avoidance: a strong customer service section including customer service options, contact information (including phone numbers and email), link to

frequently asked questions (FAQ) page, and providing simple navigation manner. While to emphasis low on uncertainty avoidance, the above website features are not important and complex navigation with a multitude of link choices is acceptable.

3- Power distance

This dimension is related to how the society accepts the social hierarchy, social inequalities, and authority.
People in societies high on power distance dimension emphasize on the hierarchical social structure and status, refer to the power and authority. While people in societies low on power distance accept less social hierarchy, and tend to have egalitarian and equal rights for all.
To customize websites on this value, the following can be incorporated: To emphasize high power distance: company hierarchy information, quality assurance and awards, vision statement, pride of ownership, appeal, and proper titles. While for those countries that are low on power distance, the above website characteristics are not important.

4- Masculinity- femininity

The masculinity- femininity dimension describes the gender roles in different cultures, Masculine cultures emphasis on values like assertiveness, ambition, success, and performance. In such culture, there are clear gender roles, masochism is acceptable, people are direct and decisive. While in a country with feminine cultures people emphasis on values like: beauty, modesty, harmony and nature, and, they are less inclined toward fantasy and imagery.
To customize websites on this value, the following can be incorporated: To emphasize masculinity: quizzes and games, realism theme, product effectiveness, and clear gender roles, depiction of women in traditional roles, such as models, wives, and mothers; depiction of men as macho, strong, and in positions of power. While to emphasize femininity, websites must incorporate aesthetics and harmony and a soft-sell approach to marketing their products.

5- High-low context

People act and communicate within a context, this context gives the meaning for the words and symbols and other elements we use, some symbols or signs have different meaning in different cultures.
The high-low context diminution help us to understand how people in different cultures communicate in their daily live, with the correct level of context that must be followed. In high context cultures most information is embedded in the context, they use symbols and indirect verbal expressions. On the other hand in low context cultures, messages are straightforward and detailed, people use precise word to convey meanings and they rely on the spoken context.

If the country of interest is toward high-context, a website can be adapted to conform to this value by incorporating the following: politeness and indirectness and aesthetics. While If the country of interest is toward low-context, a website can be adapted to conform to this value by incorporating the following: a hard-sell approach, emphasize on clarity and directness.

1.3 Websites development and test

Websites development

Websites development passes many phases, figure 2 summarizes them [1]. The main phases are:

1. Domain name registration, more usually referred to as web addresses or URLs.
2. Selecting the right partner to host content: this will usually be an Internet service provider (ISP) for small and medium companies, but for larger companies the web server may be inside the company itself.
3. Analysis for website development: this includes the analysis and design activities. Analysis includes understanding the requirements of the audience of the site and the requirement of the business.
4. Test review and revise content: this is very critical phase and the results of the tests should be analyzed carefully.
5. Publish the website.
6. Promote the website.

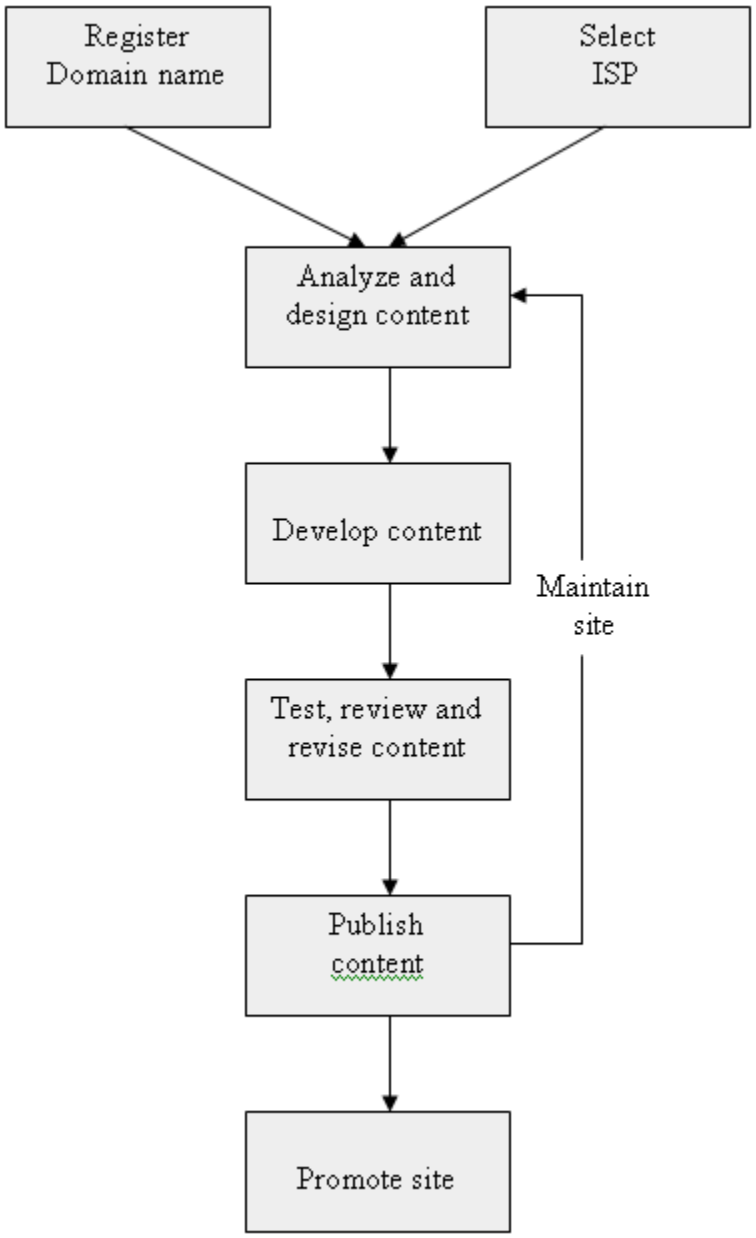


Figure 2. Summary of process of website development

Experiments on websites

It may be hard for a designer to decide which suitable button, background color or any other interaction component to choose for their webpages. They also need to continuously improve and evolve their sites by making frequently small adjustments. But how does he knows which change will have the highest impact on the customer experience or in other word which choice/change will impact the conversion rate? For this reason, several techniques have been developed to measure the reaction of the user and the benefits of design modification. These techniques are known as A/B test [6][28].

A/B, A/B/C, …,A/../N testing is a way of finding out which changes help your users' performance. It provides a controlled method of measuring the effectiveness of the alternatives of the interface design. The idea behind this test is to create one or more versions of the web interface, each one have different or modified features (changes), and then presenting it to randomly selected subset of users (for example the original version of the interface is presented to 50% of the users and the modified version is presented to the other 50%) in order to analyze their reactions. The user responses will determine the performance of the different versions or variants.

Usually the test compares one or more variants of a single site element or factor. For more factors / variations, we need to deploy multivariate testing which can be viewed as a combination of many A/B tests.

1.4 Problem statement

As experiences have shown that small changes to the interface of e-commerce websites (including culture customization) may cause big differences in the amount of purchases, the modifications on web interfaces, as well as the interface itself, should be carefully planned and evaluated.

Researchers have developed techniques to measure the user reactions to the modifications like the previous discussed A/B test, which is built on developing more than one web interface to be used in an experiment. However, the problem is how to development quickly different versions of a specific website interface and how to choose the components to vary. This is also the problem of website customization where there is a necessity for small modifications in some part of the interface in order to adapt the website to a specific culture. We don’t want to create a complete new website to produce the different variants of the interface; it would be less costly and more efficient if we can use the original one to obtain a new version that has some additional or modified features.

1.5 Approach

The underlying idea behind our approach for website experimenting is to create one or more different versions of a web interface by incorporating new or modified features with reuse of the original interface. For this purpose we rely on two different techniques:

1- **Software product line engineering** which enables us to describe the commonality and the variability features of a system (in our case a web interface) by using feature modeling.

2- **Aspect oriented programming**, which enables us to express the cross cutting concerns (in our case, the actual implementation of the modified features) and allows to automatically weave them into the original code of the system.

Therefore, the first step in our approach is to define a feature model. This feature model will consist of some common and variable features in the web interface. Once we have build the feature model we can configure it to produce different variants, one of these variants will be considered as the original web interface for the application.

Then we associate each variable feature with an aspect. Defining these aspects will allow us to modify or replace a specific module in our program in order to produce the website with the desired feature. This is achieved by means of join points. Once a join point has been matched, the program can run the code corresponding to the new advice (before, after, around). The weaving process of the original code and the aspect will happen at compile time.

This approach has many advantages:

- It reduces the costs and the efforts by better code reusability;
- The original source code does need to be modified, and the designer does not have to develop new variant implementations;
- The code for the aspects can add or modify some component in our interface to produce new features.

As a proof-of-concept implementation, we have used Java Servlets to build the web application. This technology is easy to learn and widely used for building dynamic web applications. We have selected AspectJ as aspect oriented programming language, which is an extension to java and work with all java programs.

In addition, the proof-of-concept implementation should meet the modularity principle in programming [24], which means breaking down the design of a program (here our web application) into individual components (modules). Each

module can perform a function or an implementation of a specific component in our web interface and then returns control back to the main program. This principle is a requirement in our approach for allowing more control over our interface components when modified or replaced by the aspects (in practice aspects will modify or replace these modules). Figure 3 shows the idea of modularity principle in programming.

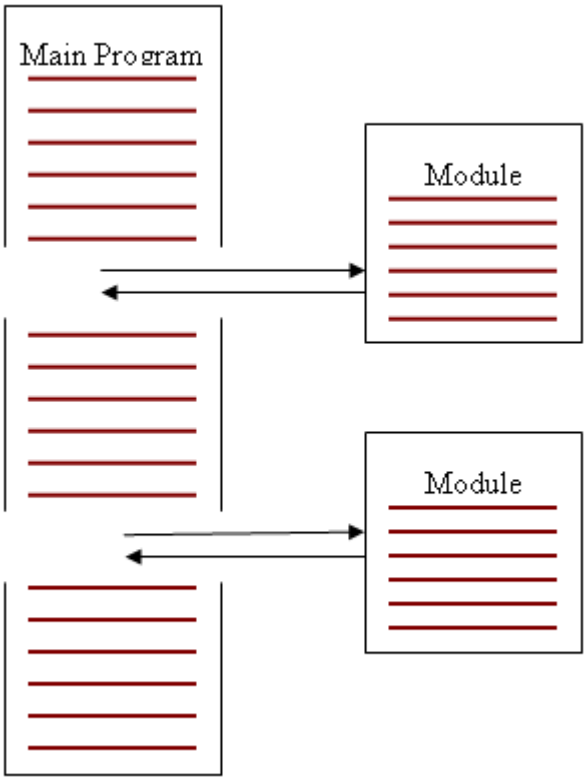


Figure 3. Principle of modularity in programming

Chapter 2: Background

In this chapter, we describe the main foundations on which our approach relies. This chapter is divided as follow: section 1 gives an introduction about feature modeling and the associated notations, section 2 gives backgrounds about object-oriented programming and introduces AspectJ as aspect oriented programming tool, and finally in section 3 we present Java Servlet as a programming environment for building dynamic web applications.

2.1 Feature modeling

2.1.1 Product line software engineering

It has become necessary in the field of software development implement or update software systems using existing software assets.
Product line software engineering (PLSE) is *"an emerging software engineering paradigm, which guides organizations toward the development of products from core assets rather than the development of products one by one from scratch*" [7]. This is achieved by the modeling of commonalities and variabilities between product variants. Exploiting the commonalities allows reusing the core assets, which implement most product functionality, where the variabilities are the new features. The commonalties are some requirements or features of a specific product in the product line that are the same in all product variants. So the process of developing and implementing these variants is by reusing the shared features. The main benefit of the technique is reducing the cost and the time of developing multiple, similar software products. It provides an effective way to benefit from code reuse in software application development. Also another benefit is the improvement of product quality of all product variants by improvement the quality of the core assets.
Figure 4, presented in [8], shows the cost for developing N kinds of systems as single systems compared to the product line software engineering where the second one provides lower development cost.

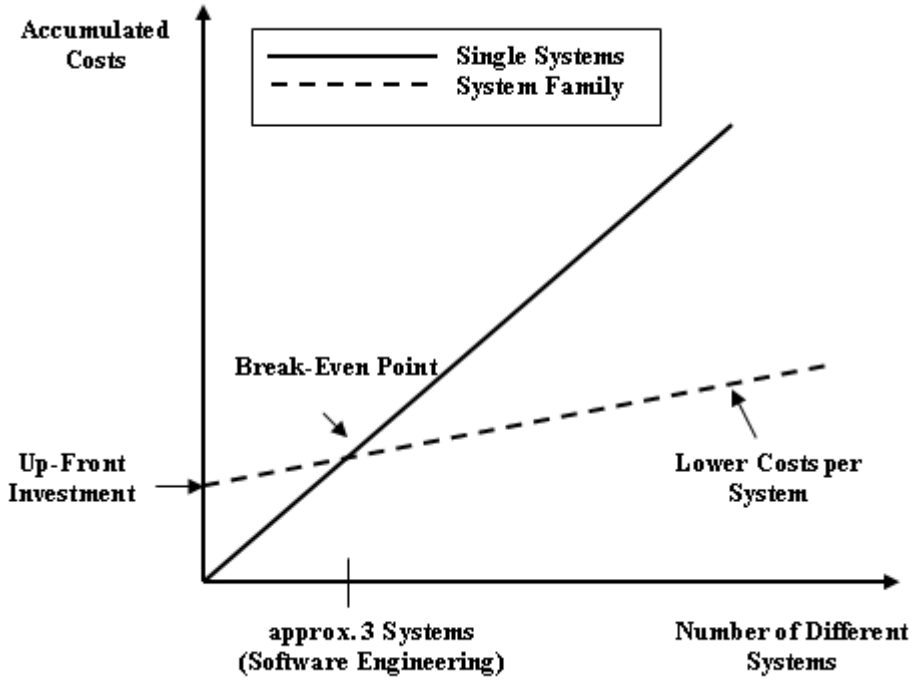


Figure 4. The cost for developing N kinds of systems as single systems compared to the product line software engineering

The so-called paradigm *domain engineering (DE)* is used to capture the common and variable properties of an application from a "domain" perspective,

Domain engineering is defined as "the *process of producing, maintaining and cataloging reusable assets which includes domain analysis and the subsequent construction of components, methods, and tools that address the problems of system/subsystem development through the application of the domain analysis"* [9].

The domain analysis has several phases, domain modeling being one of them. In the domain modeling, we address the features of software in the domain. Figure 5 depicts the three phases of the method and lists the products of each [9] .

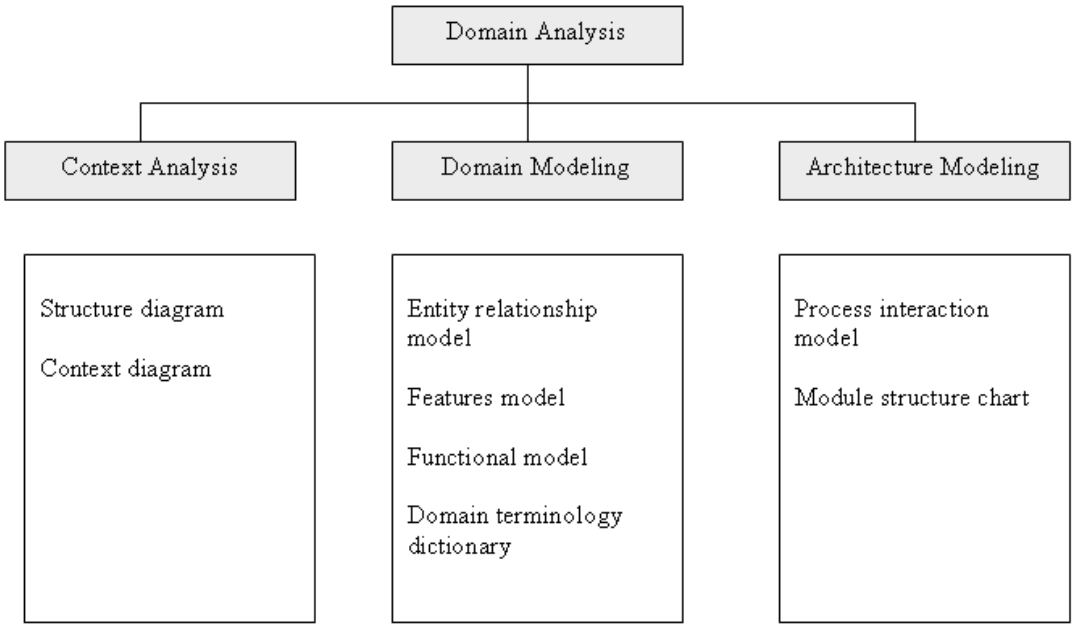


Figure 5. Phases and Products of Domain Analysis

2.1.2 Feature Modeling

2.1.2.1 Overview

The basic entity in a feature model is *the feature*. A feature is considered as a key distinctive characteristic of a product. Different domain analysis methods use the term "feature", for example FODA [9] defines a feature as « *a prominent and user-visible aspect, quality, or characteristic of a software system or systems.* »

In general, three categories of features [11] are distinguished in an application:

Functional features**:** services that are provided by the applications.
Interface features: features that are related to what and how information is presented to the users.
Parameter features: Properties that represent environments in which applications are used.

Feature modeling is defined as "*the activity of modeling the common and the variable properties of concepts and their interdependencies and organizing them into a coherent model referred to as a feature model* " [12].

A feature model represents the common and the variable features of a concept or product and the relationships between the variable features by a hierarchical structure of the features. The root of the hierarchy always represents a concept feature.

It is important to note that the feature modeling must focus on identifying external visible characteristics of products in terms of commonality and variability, rather than describing all details of products such as other modeling techniques (e.g., functional modeling, object-oriented modeling, etc.) do [7].
The main benefits of the feature models are using them for the development of an application of software systems, by describing the possibilities of the system characteristics and properties.

A feature model consists of:

- Feature diagram: a graphical AND/OR hierarchy of features, that captures structural or conceptual relationships among features.

- Features definitions: describe all the features such that each feature has a well-defined meaning, or *semantics*. It contains a short description about the feature's semantic. This is helpful when the feature is implemented by other models.

- Composition rules for features**:** describe which combinations of features are valid or invalid.

2.1.2.2 Notations

The variability in feature model or the parental relationships between features is/are expressed through different notations like notations for:

Mandatory features: theses are common features among different products.

Optional Features**:** these are the different features among products.

Or features group: this is group of features indicates that at least one of the sub-features must be selected.

Alternate features group (xor): this is a group of features that indicates that no more than one feature can be selected for a product.

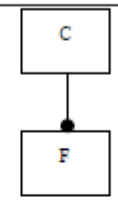
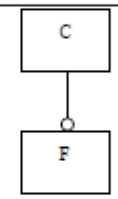
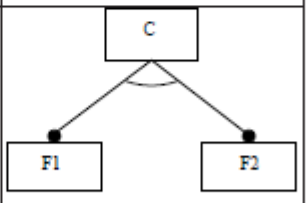
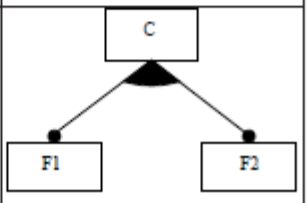
In addition to the parental relationships between features, there are also cross-tree constraints. The most common are:

- A requires B: this means that the selection of feature A in a product implies the selection of feature B.

- A excludes B: this means you cannot select the feature A and the feature B at the same time.

Table 1 provides an overview of some commonly features relationships types [13].

Table 1. Features types

Type	Notation
Mandatory	
Optional	
Alternative	
Or	

Cardinality-based feature notation

Some authors propose to use the form [n,m] with n being the lower bound of features and m the upper bound of features that can be selected in order to express how many features there should be. For example the cardinality [2…5] means at least 2 features and at most 5 can be selected for a product. In this case, mandatory and optional features are special cases of features with cardinalities [1…1] and [0…1] respectively [14].

Also there is a group cardinality to express both alternative features group and or feature group. In order to know how many features could be chosen in a feature group, this is expressed as <n-m>.

2.1.2.3 Approaches of feature modeling

In the literature, there exist a number of approaches for feature modeling:

FODA [9]: Feature Oriented Domain Analysis is a method for analyzing and representing commonality and variability of applications in a domain. FODA models are often expressed as diagrams consisting of a set of graphical symbols forming a configuration tree. There are four types of features in feature model: Mandatory, Optional, and Alternative features, in addition to the Or groups of features and cardinality aspects. Besides that there are two types of cross cut relations between features: require and excludes relations which are expressed in textual form.

FORM [15]: Feature Oriented Reuse Model is as an extension of FODA. It is a method for analyzing the commonality among applications in a domain during application development. This commonality includes services, operating environments, domain technologies, and implementation techniques. A reusable feature model is used to represent the commonality as an AND/OR graph, where AND nodes indicate mandatory features and OR nodes indicate alternative features selectable for different applications.

FeatuRSEB [16]: The FeatuRSEB method is a combination of FODA and the Reuse-Driven Software Engineering Business (RSEB) method [10][35]. RSEB is a use-case driven systematic reuse process, where variability is captured by structuring use cases and object models with variation points and variants. FeatuRSEB feature diagrams are directed acyclic graphs (DAGs), with the decomposition operator ‘or’, ‘xor’ and ‘and’. Variation points are features whose sons are decomposed through ‘or’ or ‘xor’ while those sons are called variants. In addition, the constraints ‘requires’ and ‘mutex’ are represented by dashed arrows.

2.1.2.4 Verification

To capture the valid combinations of feature in a feature diagram, the most common approach is to use mathematical logic. The feature model can be easily translated into a logical expression. By substituting all the selected features in the expression by true, and by false if unselected, we can generate the set of possible variants and test the validity of each variant within the product line.

The following table, table 2, describes all the feature model relations and the equivalent formal definitions [27]:

Table 2. Feature model relations and the equivalent formal definitions

Feature Diagram Primitive	Semantics
r is the root feature	r
f_1 optional sub-feature of f	$f_1 \Rightarrow f$
f_1 mandatory sub-feature of f	$f_1 \Leftrightarrow f$
$f_1, \ldots, f_n$ alternative sub-features of f	$(f_1 \vee \ldots \vee f_n \Leftrightarrow f) \wedge \bigwedge_{i<j} \neg(f_i \wedge f_j)$
$f_1, \ldots, f_n$ or sub-features of f	$f_1 \vee \ldots \vee f_n \Leftrightarrow f$
f_1 excludes f_2	$\neg(f_1 \wedge f_2)$
f_1 requires f_2	$f_1 \Rightarrow f_2$

As an example we use an online shopping system. The feature model for this system is shown in figure 6. The root feature represents the concept of the E-shop. The system implements a catalogue, payment, and has the modules security and optionally a search feature. Providing a catalogue is mandatory. The payment process is an “or group” of two features (bank transfer or credit card). At least one of them must be selected in the implementation, while the security polices has the alternative between high or standard implementation and only one of them must be selected. In addition to that, there is a require (implies) cross tree relationship between the credit card feature and high security policy feature which means if we select the payment process to be credit card, this requires to select the high security policy. It must be noted that each approach of feature modeling has its own symbols and notation in the feature model diagram.

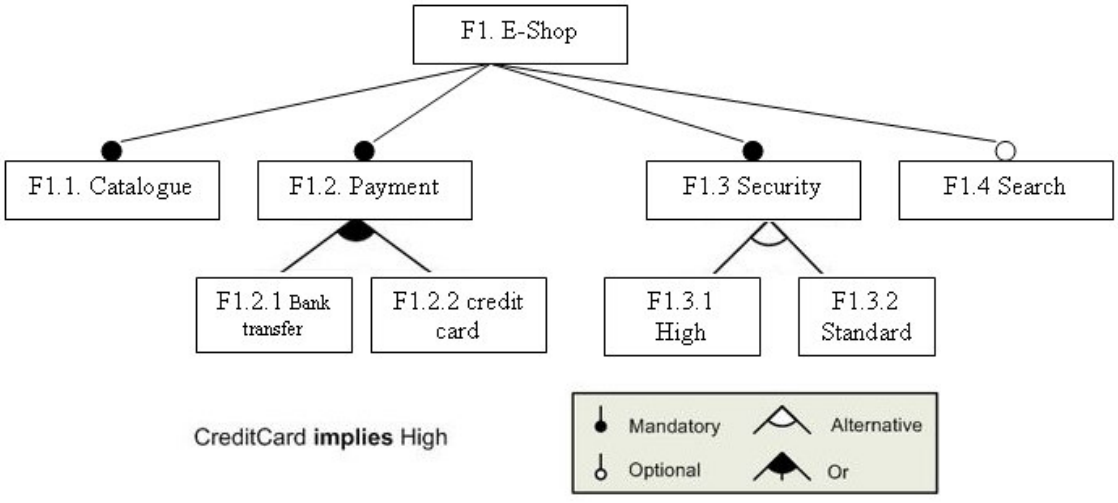


Figure 6. A feature diagram representing a configurable e-shop system

2.1.2.5 Feature modeling tools

There are number of tools supporting the creation of feature models:

1- Feature Model DSL [36]: makes it possible for a user to design feature models using Visual Studio 2008. See figure 7 for a screenshot.

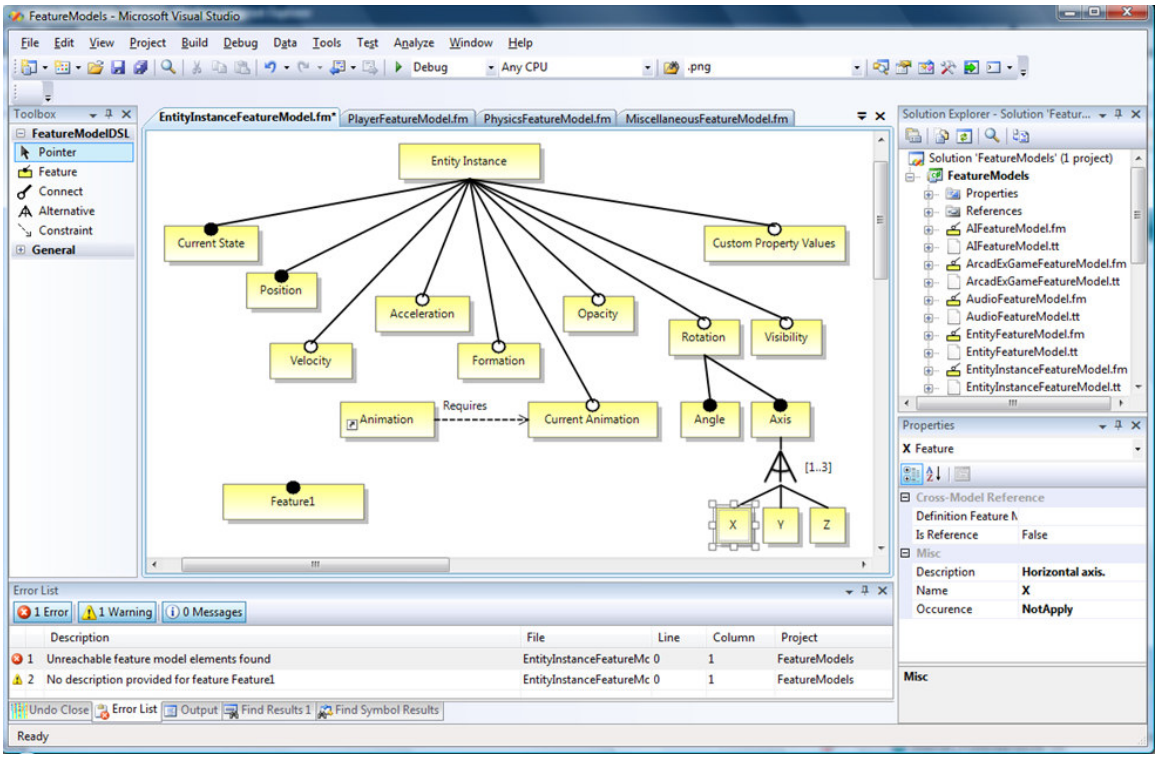


Figure 7. Feature Model DSL

2. XFeature [37]: is a feature modeling tool which supports the modeling of product families and allows instantiating them to create configurations (applications). The tool is provided as a plug-in for the Eclipse platform. See figure 8 for a screenshot.

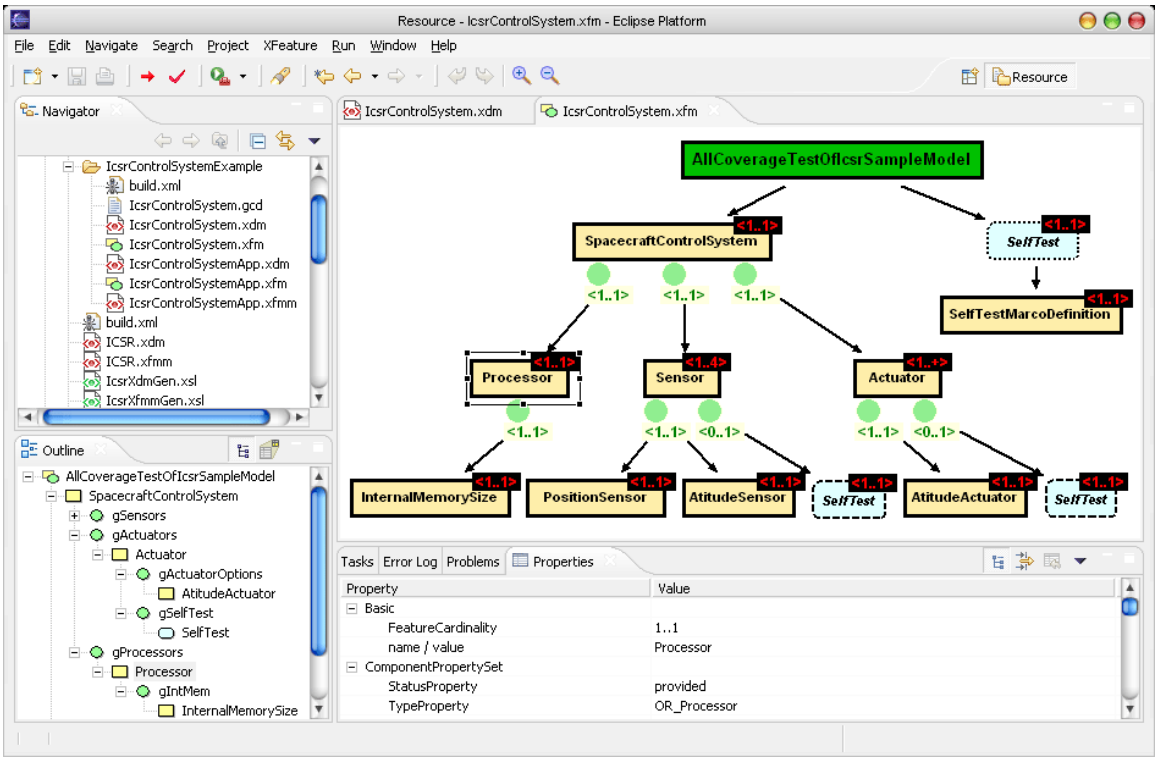


Figure 8. XFeature Tool

3. Feature modeling tool:

This is a tool to design feature modeling and also allows managing the variability of product lines [38]. It is using Visual Studio. See figure 9 for screenshot.

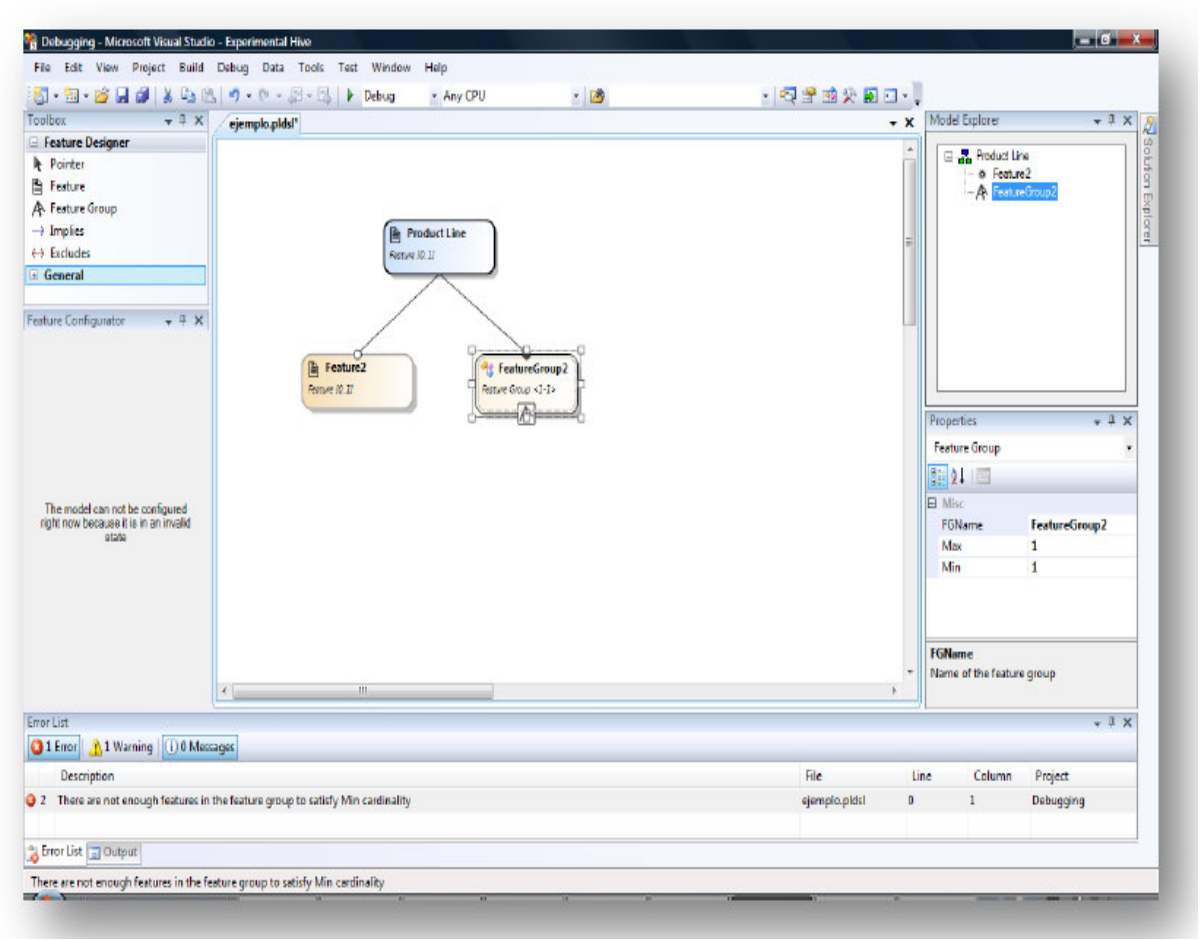


Figure 9. Feature Modeling Tool

2.2 Object Oriented Programming (AOP)

Object-oriented (OO) programming is considered as the main programming paradigm. In OO application, the software system consists of a number of objects (defined using classes), which cooperate to achieve the objective of the system. However, OO has its limitations. For example, if there are crosscutting concerns that affect sections of many classes in the whole application, adding the code that handles these concerns will result in very complex tangled code that is repeated in many places, making the program difficult to reason about and difficult to change.

Aspect-Oriented Programming (AOP) provides a solution to this problem The idea is by providing a mechanism (the aspect), for expressing the cross cutting concern and automatically weave them into the code of the system resulting into an easier, more understandable program, which is easier to develop and maintain.

An aspect encapsulates behaviors that affect multiple classes. Using aspects keeps the structure of the system's architecture the same, where a single aspect can contribute to the implementation of a number of methods, modules, or objects. With AOP, we start by implementing the project using an OO language (for example, Java), and then we deal separately with crosscutting concerns in the code by implementing aspects. Finally, both the code and aspects are combined into a final executable form using an aspect weaver.

The weaving process is achieved by linking aspects with other application types or objects to create an advised object [17]. This can be done at compile time (using the AspectJ compiler, for example), at load time, or at runtime. Note that the original code doesn't need to know about any functionality the aspect implements; it needs only to be recompiled without the aspect to regain its original functionality. Figure 10 shows how the weaving process works compared to a normal compilation process.

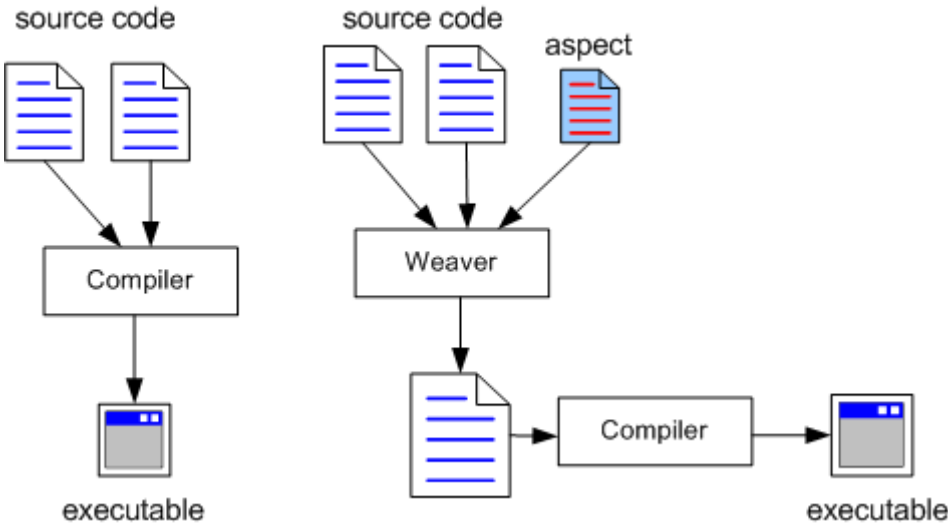


Figure 10. Aspect Weaver

2.2.1 Aspect Oriented Programming for Java: AspectJ

AspectJ is an aspect-oriented extension to Java. It provides flexibility to java programmers as it has the following compatible properties [20]:

- All legal Java programs are considered as legal AspectJ programs.
- All legal AspectJ programs run on standard Java virtual machines.
- The existing tools for java can be extended to work with AspectJ.

Component of AspectJ

1- Join Points

Join points are certain well-defined points (places) in the execution of the program, for example points at which an object receives a method call and points at which a field of an object is referenced. Table 3 summarizes the different kinds of join points of AspectJ [18].

Actually any aspect-oriented language has a join point model in order to coordinate between the aspect and the non-aspect code. The AspectJ model is considered as a graph:

- The nodes represent the join points.
- The edges are control flow relations between the nodes.
- Each join point is reached twice: before the action described begins executing and when it returns.

Figure 11 shows the idea of the AspectJ join point model:

- Large circles represent objects.
- Square boxes represent methods.
- Small numbered circles represent join points.

The first three lines of code build the objects below, executing the last line starts a computation that proceeds through the labeled join points.

```
Point pt1 = new Point(0, 0);
Point pt2 = new Point(4, 4);
Line  ln1 = new Line(pt1, pt2);

ln1.incrXY(3, 6);
```

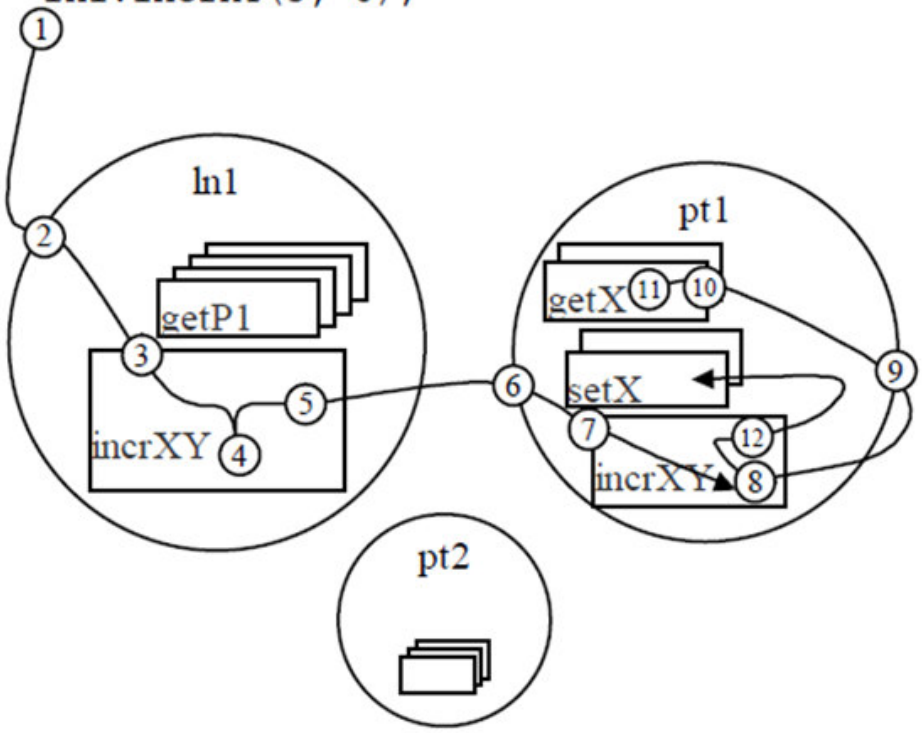


Figure 11. Illustration of AspectJ model for representing join points

The list below describes some of the labeled join points of figure 8:

1. A method call join point corresponding to the incrXY method being called on the object ln1.
2. A method call reception join point at which ln1 receives the incrXY call.
3. A method execution join point at which the particular incrXY method defined in the class Line begins executing.
4. A field get join point where the _p1 field of ln1 is read.
5. A method call join point at which the incrXY method is called on the object pt1.
8. A method call join point at which the getX method is called on the object pt1.
11. A field get join point where the _x field of point pt1 is read.

Control returns back through join points 11, 10, 9 and 8.

12. A method call join point at which the setX method is called on p1. And so on, until control finally returns back through 3, 2 and 1.

Table 3. Kind of join points of AspectJ

Kind of join point	**Points in the program execution at which…**
method call constructor call*	A method (or a constructor of a class) is called. Call join points are in the calling object, or in no object if the call is from a static method.
method call reception constructor call reception	An object receives a method or constructor call. Reception join points are before method or constructor dispatch, i.e. they happen inside the called object, at a point in the control flow after control has been transferred to the called object, but before any particular method/constructor has been called.
method execution* constructor execution	An individual method or constructor is invoked
field get	A field of an object, class or interface is read
field set	A field of an object or class is set.
exception handler execution	An exception handler is invoked
class initialization	Static initializers for a class, if any, are run.
object initialization	When the dynamic initializers for a class, if any, are run during object creation.

2- Pointcut

Pointcuts are the AspectJ mechanisms for declaring set of join points. AspectJ includes two kinds of designator:

1- Primitive pointcut designators
such as
calls(signature)
receptions(signature)
executions(signature)

Example:
The pointcut designator

```
receptions(void Point.setX(int))
```

This matches all method call reception join points at which the Java signature of the method call is “void Point.setX(int)” that happens every time a Point instance receives a call to change its x coordinate.

Pointcuts also can be combined using and, or and not operators (‘&&’, ‘||’ and ‘!’). For example the following code for the compound pointcut designator matches to all receptions of calls to a Point instance to change its x or y coordinate.

```
receptions(void Point.setX(int)) ||
receptions(void Point.setY(int))
```

2- User-defined pointcut designators

User-defined pointcut designators are defined with the pointcut declaration.
Example:

```
pointcut moves():
receptions(void FigureElement.incrXY(int, int)) ||
receptions(void Line.setP1(Point)) ||
receptions(void Line.setP2(Point)) ||
receptions(void Point.setX(int)) ||
receptions(void Point.setY(int));
```

This defines a new pointcut designator called moves(), that identifies a set of primitive pointcut designators, which is matched whenever a figure element receives a call of a method that can move it (changing the x or y by any way). The user-defined pointcut designators can be used wherever a pointcut designator can appear.

The following table (table 4), presented in [34], shows examples of pointcuts that capture the methods and constructors execution in a program.

Table 4. Execution of methods and constructors pointcuts

Pointcut	**Description**
execution(public void MyClass.myMethod(String))	Execution of myMethod() in MyClass taking a String argument, returning void, and with public access
execution(void MyClass.myMethod(..))	Execution of myMethod() in MyClass taking any arguments, with void return type, and any access modifiers
execution(* MyClass.myMethod(..))	Execution of myMethod() in MyClass taking any arguments returning any type
execution(* MyClass.myMethod*(..))	Execution of any method with name starting in "myMethod" in MyClass
execution(* MyClass.myMethod*(String,..))	Execution of any method with name starting in "myMethod" in MyClass and the first argument is of String type
execution(* *.myMethod(..))	Execution of myMethod() in any class in default package
execution(MyClass.new())	Execution of any MyClass' constructor taking no arguments
execution(MyClass.new(..))	Execution of any MyClass' constructor with any arguments
execution(MyClass+.new(..))	Execution of any MyClass or its subclass's constructor. (Subclass indicated by use of '+' wildcard)
execution(public * com.mycompany..*.*(..))	All public methods in all classes in any package with com.mycompany the root package

3-Advice

An advice is the code that is executed at each match of the join point in a pointcut. It is the actual implementation at join points (what to do at the join points), Each piece of advice is associated with a pointcut (named or anonymous) and specifies behavior that it wants to execute before, after, or around, the join points that the pointcut matches. An advice declaration may contain parameters whose values can be referenced in the body of the advice. The three kinds of advice are:

- before advice runs at the moment the join point is reached, for example before the method runs.

- around advice runs when the join point is reached and has control over whether method itself runs at all, in other words it can decide whether the original join point should be executed or not (the advice code runs instead of the original code of the method in this case).
- after advice runs at the moment the control returns through the join point, for example just after the method is ended.

The following code illustrates an after advice that runs some code after returning from the matched joinpoints of the pointcut accountUpdate.

```
pointcut accountUpdate(Account acc, Money amount) :
  execution(void Account.*(Money)) &&
  args(amount) &&
  this(acc);

after(Account acc, Money amount) returning :
  accountUpdate(acc,amount) {
  acc.getStatement().recordTransaction(
  thisJoinPointStaticPart, amount);
}
```

4- Inter-type declarations

When an aspect takes complete responsibility on behalf of other types (for example other classes), this is called inter-type declaration. An inter-type declaration inside an aspect looks just like the definition of a normal method or field in the aspect, except that the method or field name is preceded by a type name. Here's a simple example [29]:

```
public aspect AccountSession{
  private SessionContext Account.context;
  public SessionContext Account.getSessionContext() {
    return context;
  }
  public void Account.setSessionContext(SessionContext ctx)
   {
    context = ctx;
  }
```

The AccountSession will manage a SessionContext context on behalf of the Account class, and that it provides getSessionContext() and setSessionContext() methods on behalf of the Account class.

5-Aspect

Aspect is a new class-like language element that allows developers to encapsulate across cutting concerns. Aspects are defined by aspect declarations, which have a form similar to that of class declarations. Aspect declarations may include pointcut declarations, advice declarations, as well as all other kinds of declarations permitted in class declarations.
Figure 12 shows the relationship between join points, aspects, pointcuts, advice and our application classes [19].

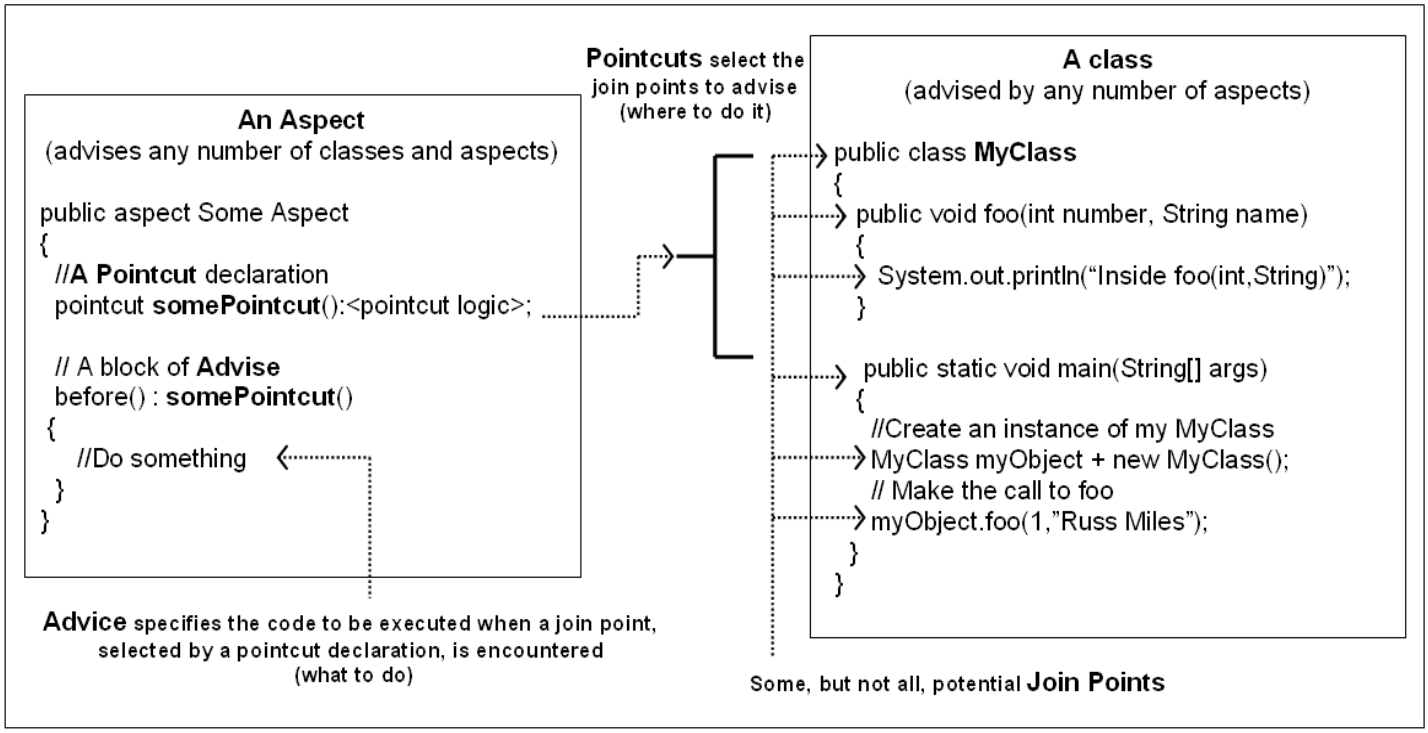


Figure 12. The relationship between aspects, pointcuts, and advice

To put it all together let's consider the following simple example.

```
public class HelloWorld {

public static void main(String[] args) {
        printHello ();
}

public void printHello () {
        System.out.println ("Hello world");
 }

 }
```

Now, we have our existing Java code in HelloWorld.java. Let's assume that we would like to print a message after executing the printHello method

The following code is the AspectJ implementation.

```
public aspect HelloFromAspectJ {
    pointcut afterPrint() : execution(public void
Helloworld.printHello());

    after(): afterPrint() {
       System.out.println("Hello from AspectJ");
     }
}
```

First we define an aspect in the same way we define a Java class. Like any Java class, an aspect may have member variables and methods. Then we define a pointcut, named afterPrint that picks out the execution of the HelloWorld.printHello method. This pointcut is used in the definition of the advice that will be executed after the pointcut afterPrint. Finally we implement the advice.

2.3 Java servlets

Java servlets is a technology used to develop dynamic content for web-based applications. It receive all the benefits of the mature Java language, including portability, performance, and reusability, and the ability to access the entire family of Java APIs, including the JDBC API to access databases.

Servlets do not face any of the problems faced by the classical Common Gateway Interface CGI programming. CGI creates a process for every request which limits the number of requests the server can handle, while in Servlets, once a servlet is loaded, the server only makes simple method calls and does not need to create a new process for each request.

Java Servlets are used to generate HTML response dynamically based on the user request. The dynamic response generation is achieved by HTTP request-response process. When the servlet receives a request to be processed, the web server creates the objects of the classes HttpServletRequest as HTTP request and HttpServletResponse as HTTP response and passes these two objects to the Servlet instance in order to handle the incoming HTTP request information and execute the application logic, and then generates the response, once the response has been completed it is returned to the client by writing it to the HttpServletResponse.

Servlet Life Cycle

The servlet life cycle is one of the most exiting features of the servlets. The life cycle of a servlet is controlled by the web server or web container (one example of the container is Apache Tomcat which is an open source container) in which the servlet has been deployed. When a request is mapped to a servlet, the container performs the following steps [21].

1. Create and initialize the servlet: If an instance of the servlet does not exist, the web container loads the servlet class, creates an instance of the servlet class and then initializes the servlet instance by calling the init method.
2. Handle zero or more service calls from clients: The server invokes the service methods, passing a request and response objects.
3. Destroy the servlet and then garbage collects it: If the container needs to remove the servlet, it finalizes the servlet by calling the servlet's destroy method.

The servlet lifecycle is illustrated in the figure 13.

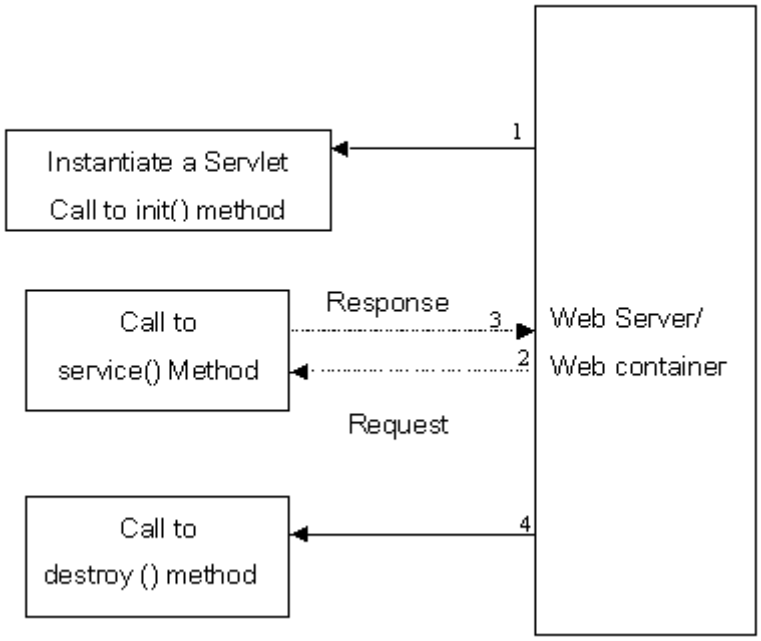


Figure 13. Servlet Lifecycle

Sending HTML Information:

The HTTP servlet can return three kinds of things to the client:

1- A single status code: to indicate success or failure.
2- Any number of http headers.
3- A response body.

The response body is the main content of the response. For an HTML page, the response body is the HTML itself, also it can be of any type and of any length. The servlet is provided with a set of classes that treat HTML as just another set of java objects, this approach can greatly simplify the task of generating HTML and make the servlet easier to write, easier to maintain, and more efficient.

The following code will show the more usual case where HTML is generated. The first step is done by setting the Content-Type response header, we need to set response headers *before* returning any of the content via the PrintWriter.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Myservlet extends HttpServlet {
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println("<HTML>\n" +
"<HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
"<BODY>\n" +
"<H1>Hello World</H1>\n" +
"</BODY></HTML>");
}
}
```

The following figure, figure 14, is the output

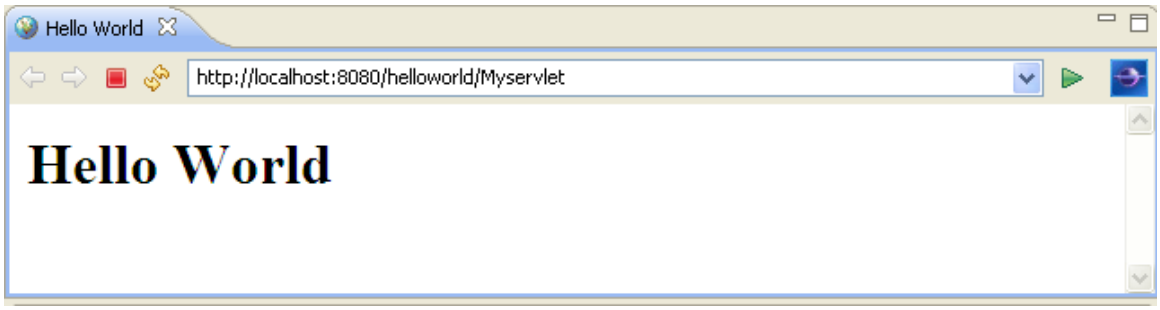


Figure 14. Servlet output: Hello World example

Deploying a Java servlet that uses AspectJ

As proposed, we will use java servlet to build our web application and AspectJ as aspect oriented programming language. Assume that we are using Eclipse AspectJ project and Apatche Tomcat as web server, then we need to do the following steps [19]:

1- Compile the java servlet and aspect as normally.
2- Create a new web application directory in Tomcat by creating a subdirectory in the webapps directory of the Tomcat called mywebapplication.
3- Inside mywebapplication create a subdirectory called WEB-INF that contains classes and lib directory.
4- Copy the class files generated from the compilation of java servlet and aspect to the above classes directory.
5- Copy the aspectjrt.jar from the aspectj installed directory to the above lib directory.
6- Then restart your web application.

The directory structure for the deployed web application should be as in figure 15.

Figure 15.The deployed web application and file structure

Chapter 3: An Approach for experimenting with website designs

3.1 Our approach

Our work proposes an approach for experimenting with website designs. As already explained in the introduction, it is based on the core asset development of software product lines (SPL) and aspect-oriented software development (AOSD). The approach can be used for identifying, and managing variability of web application for example in the case of website design testing but also for website customization.

The contribution of this approach is that the construction of two or more different versions of the website interface can be done by reusing the core assets of the original one.

As we stated before we relay onto two techniques: software product lines and aspect oriented programming:

- Software product lines allow us to capture the commonality and the variability between products family. Since different versions of a Web application will share similar behaviors, designing these applications by reusing the common features between them is more efficient as it reduce the time and effort and improves the consistency among all them. Feature modeling is used for specifying and modeling the variability among the different versions of our applications.
- Aspect oriented programming allow us to capture the variable features in the stage of variants implementation by defining aspects that modify the original code. In this case, the desired variants are achieved by reusing the original code. So, there is no need for new implementation; aspects can add or modify the variable features among the different versions of the web interface.

However, for this approach to be effective, as mentioned before, the implementation of the original interface should meet the modularity principle in programming. The benefit of this lays into facilitating the achievements of the modifications and changes on the originals components (modules) by aspects.

Our approach consists of a number of stages depicted in figure 16:

1- Specifying the feature model of the web interface. As already pointed out, several tools are available for feature modeling.
2- Defining the different variants of the interface. This includes feature model configuration by selecting the variable features.
3- Developing the original system code (if it is not yet existing): Here, we have used the Java servlet technology, one of the popular programming languages for Web application development. It is an easy to learn and has

excellent capabilities. However, the approach itself is not limited to this technology.

4- Writing aspect code for the variants and variable features: we have select AspectJ as aspect oriented programming tool to write the aspect code for our application. Each variable features that does not exist in the original interface has to be associated with an aspect that modify the original code in order to get the desired new feature.
5- Collecting all the aspects associated with the features that were selected for the variant; and compile the java servlet and aspect as normally.
6- The aspect tool then invokes the weaver to produce the actual variant code by weaving the original system code with the collected aspects.

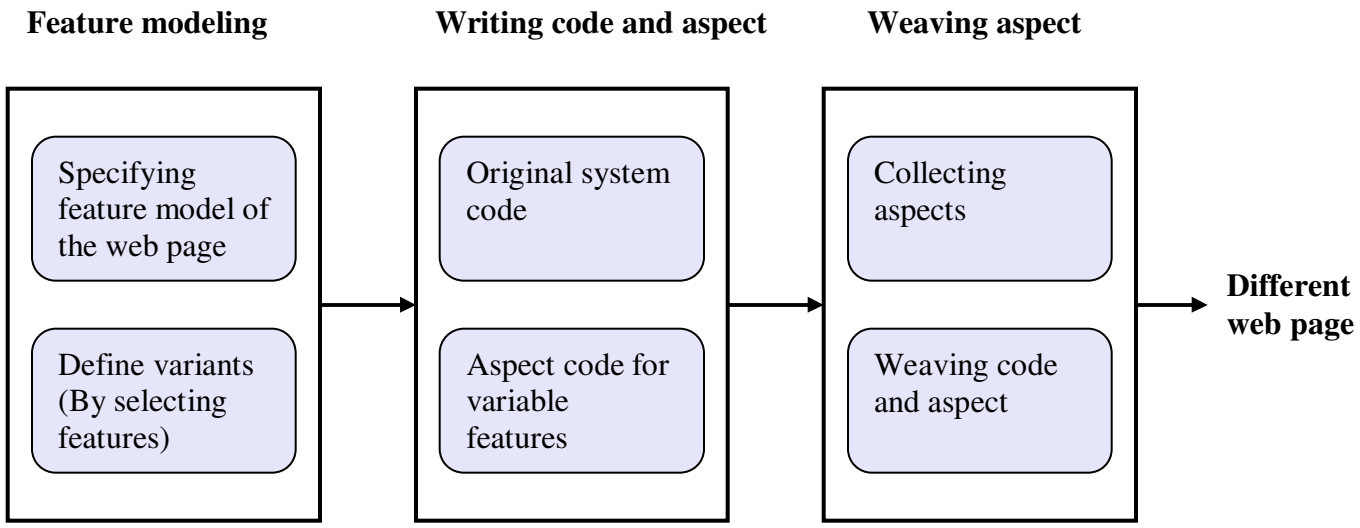


Figure 16. The Stages of our approach

3.2 Case studies

3.2.1 Changing the checkout screen of a shopping cart.

We introduce an example to illustrate our approach. It is a simplified checkout screen of a shopping cart for a CD shop. Assume we want to test how small changes could affect the purchase process. We aim to be able to produce different versions /variants of this interface. One variant is treated as the original (and has the core features) and we also want another variant that has small modifications of some component or factors.
Figure 17 shows the original interface, called variant A, and figure 18 shows the modified interface, called variant B. There are two main differences: variant B

includes checkout button twice placed at top and at bottom, and the “total” text presented in different color.

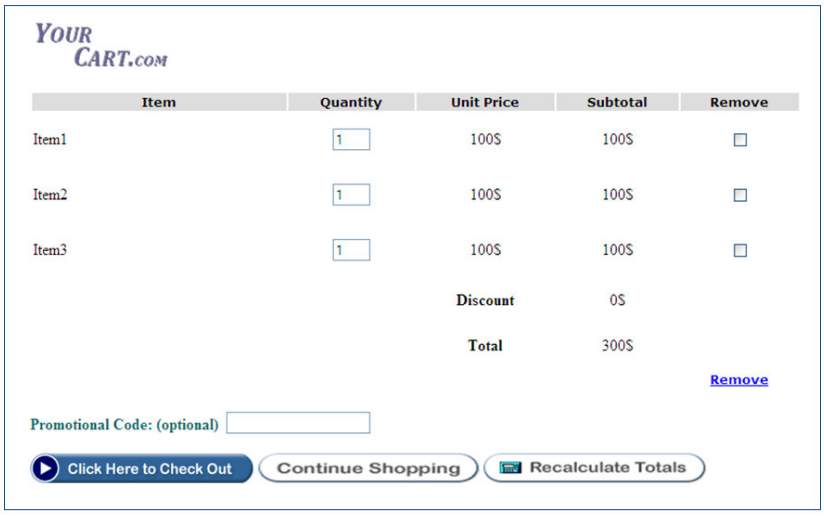


Figure 17. Checkout screen variant A

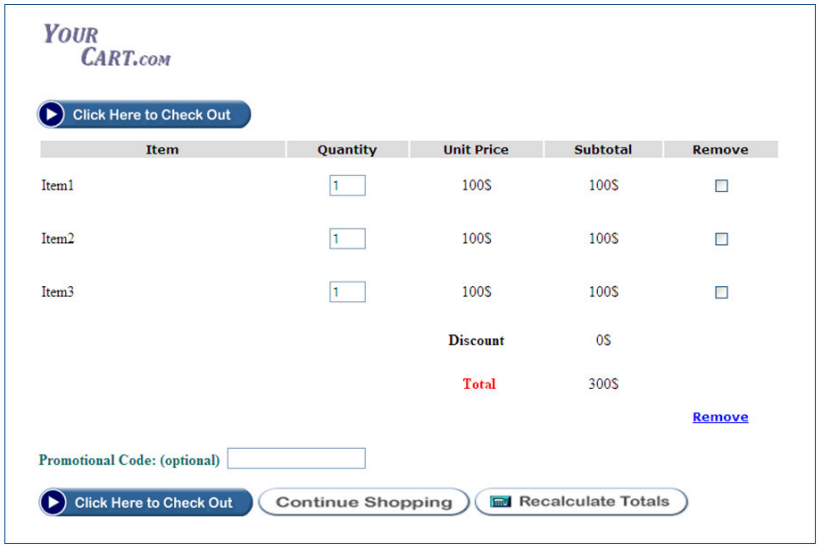


Figure 18 .Checkout screen variant B

Specifying the feature model:

The first step in our approach is to build our feature model of our screen.
We use the future modeling tool as tool to build and configure our feature model [38].

Our feature diagram is depicted in figure 19. The root feature represents the concept checkout screen. The screen has a checkout button, continue button, total display and optionally discount offer, and recalculate button. Providing a checkout button is mandatory, the position of the button on the screen is an “or group” of two features (“placed at top” or “placed at button”). At least one of them must be selected in the implementation while the total display has two alternative for being displayed: either using “same color” (as the rest of the table elements) or in “different color”, and only one of them must be selected. In addition to that, there is a require (implies) cross tree relationship between the “discount offer” feature and “recalculate button”; this means that selecting the discount offer in the interface (variant) also requires to select the “recalculate button” in the interface (variant).

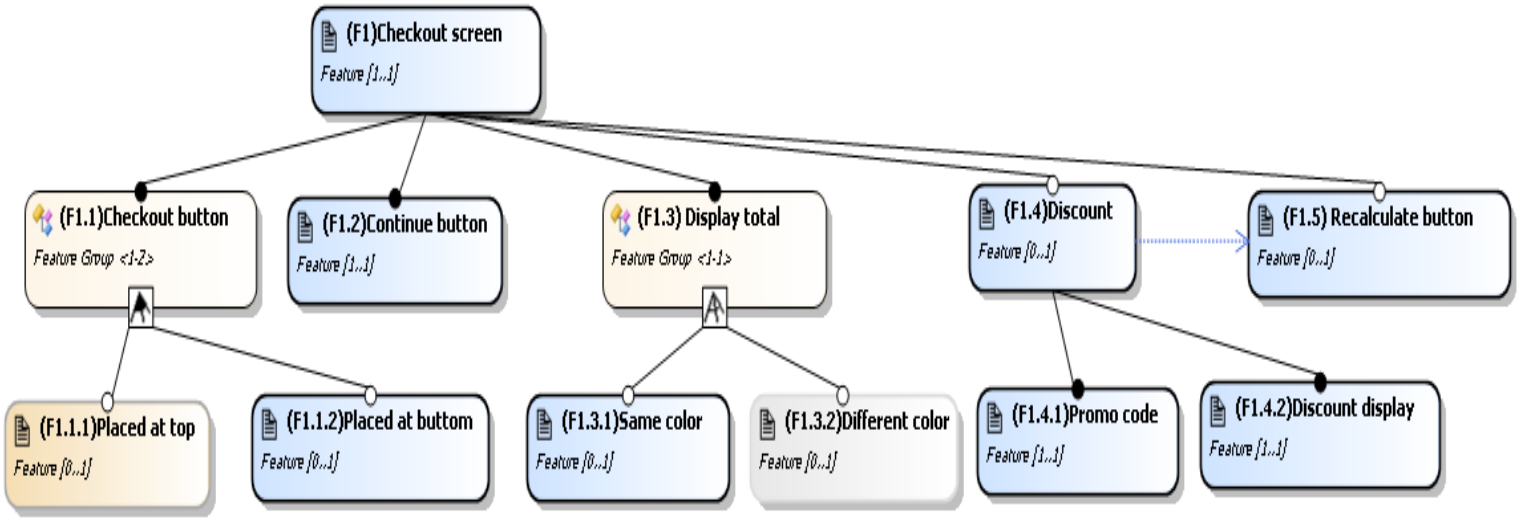


Figure 19. The feature model diagram of the checkout screen

Here in table 5 we summarize all the features of our feature diagram with a description of each feature and its existence on the original interface. The features that exist in the original interface will be treated as the core features and contains all the common features (all mandatory features) in addition to some variable features which represent the variability between the variants.

Table 5. Features of the checkout screen

Feature name	**type**	**description**	**Existence in the original interface**
F1	Root feature	Checkout screen	yes
F1.1	Mandatory	Checkout button	yes
F1.1.1	Or group	Checkout button placed at top	Modified

F1.1.2	Or group	Checkout button placed at bottom	yes
F1.2	mandatory	Continue button	yes
F1.3	mandatory	Display total	yes
F1.3.1	Alternative group	Display total in the same color	yes
F1.3.2	Alternative group	Display total in different color	Modified
F1.4	optional	Discount offer	yes
F1.4.1	mandatory	Text field for promotion code	yes
F1.4.2	optional	Display discount	yes
F1.5	optional	Recalculate button	yes

Selecting the variants:

After building the feature diagram we have to configure it to produce our variants. Different combinations of interface features will result in different interface variants.
For our example, we want to have two variants, variant A as the core variant (will be implemented as the original interface) and variant B as a modified interface.

Variant A (Original interface): has the following features:
- Checkout button placed at bottom.
- Continue button.
- Total displayed with the same color.
- Presenting discount offer.
- Presenting recalculate button.

Variant B (Modified interface): has the following feature:
- Checkout button placed at bottom and placed at top.
- Continue button.
-Total displayed with the different color.
- Presenting discount offer.
- Presenting recalculate button.

The tool we use in building our feature diagram allow us to configure our feature diagram manually to produce correct combination of features as a particular variants. Figure 20 shows the feature configuration screen.

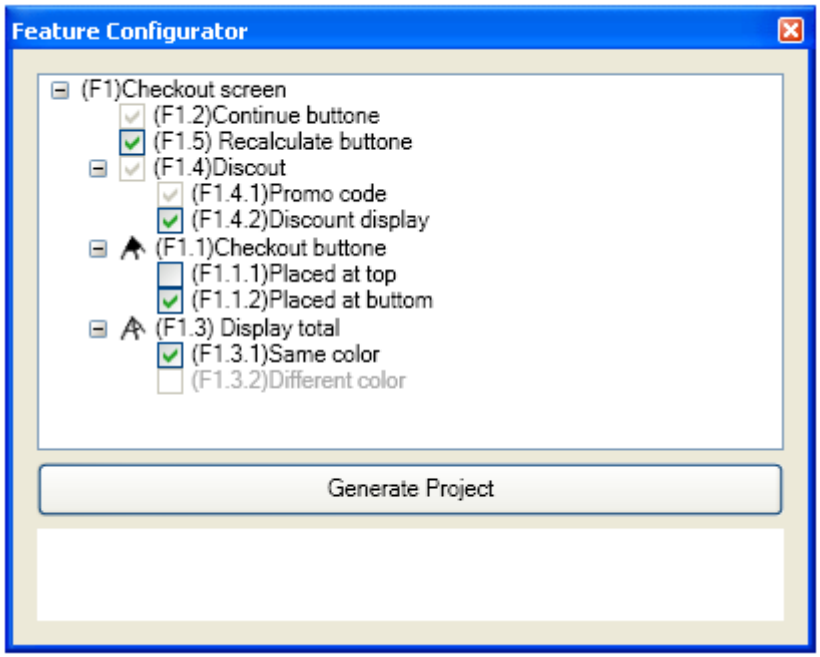


Figure 20. Feature model configuration for specific variants

Implementing the variant:

Variant A: Original interface

This variant is developed as any dynamic web application. As proposed before our implementation technology for it the Java servlet technology. We have defined a class ShoppingCart that allows for the addition and removal of different items. This class contains a number of methods that render the different elements in the web page interface, such as:

- addLogo();
- addItemList();
- addPromoCode();
- checkoutButton();
- continueButton();
- recalculateButton();
- addFooter();
- getTotal();
- getDsicount();
- printTotal();
- remove();

It is left to the programmer if he/she want to add additional methods, but as mentioned before, in our approach we followed the modularity principle in programming when implementing our interface. This means each element (part)

presented in the interface should have its own method that implements it separately.
Separating the implementation of the components makes it easier later to apply changes and modifications to individual components and decreases the chance that modifying one component will adversely impact the other.

Variant B: Modified interface

This variant has two different features selected:

- Placement of checkout button at top as well as bottom.
- Displaying total in different color.

The rest of the features is common with the original interface.

Now, realising these two variable features is done by defining two aspects for each of them. In this case the original source code does not need to be modified; all the specification of the modification is included in the defined aspects which modify the original system in a particular way.

- In order to implement the feature “checkout button at top” (F1.1.1 in the feature diagram), we define an aspect called checkoutOnTop, which has a pointcut that intercepts the execution of the method that prints the tables of the items (addItemList), and applies an before-type advice which will execute the method checkoutButton and print the button in the screen before execution of the method addItemList, and by this we realize our feature.

The following code is the AspectJ implementation:

```
public aspect checkoutOnTop {


public pointcut Topcheckout(HttpServletRequest request,
HttpServletResponse response) :
                                execution(private void
ShoppingCart.addItemList(HttpServletRequest, HttpServletResponse))
&& args(request, response);


before(HttpServletRequest request, HttpServletResponse response)
throws IOException : Topcheckout(request, response)
         {
            ShoppingCart.checkoutButton(request, response);
         }

}
```

Regarding the feature of displaying the text “total” in a different color (red color for example) we define an aspect called replaceTotal, which has a pointcut that

intercepts the execution of the PrintTotal method, and by applying an around-type advice, that replace the code of the method as we want, we apply a simple change which is changing the text color. The following code is the implementation:

```
public aspect replaceTotal {

public pointcut recolortotal(HttpServletRequest request,
HttpServletResponse response) :
            execution(private void
ShoppingCart.PrintTotal(HttpServletRequest,
HttpServletResponse)) &&
            args(request, response);

void around(HttpServletRequest request, HttpServletResponse
response) throws IOException : recolortotal(request, response)
      {

            PrintWriter out = response.getWriter();
            out.println("<tr><td width=255 height=44></td><td
height=44><td height=44><p align= center ><font
color=#FF0000><b>Total :</b></td>");

}
}
```

Here the implementation of the application by java servlet offer us great flexibility in adding changes to the interface elements by aspects declarations because most of these changes could be a matter of arranging the HTML code in a specific way to get the desired look, in addition we can also add additional functionality to some components and elements. And here the importance of following the modularity principle in programming appears, as each element could be treated separately without affecting the other elements.

Weaving process:

The waving process is done automatically at the compile time, so we collect all the related classes and aspects and compile them as usual (we use Eclipse as tool for building our application with Java servlet), and then follow the steps of deploying Java servlet that uses AspectJ to deploy our application.

3.2.2 Adapting a CD shop page to some cultural dimensions

In this case study we illustrate how that our approach can be used to customize websites to specific cultural dimensions, we aim to produce different versions of the website interface where each version differs in presenting some parts or

components according to culture of the target customers (for example adding additional components, using specific pictures or providing additional links).
As we have seen before, the cultural values framework presents the five cultural values and describes how to emphasize a specific one on the design of the website interface.

We apply our approach on a simple CD screen (figure 21) for an online CD shop.

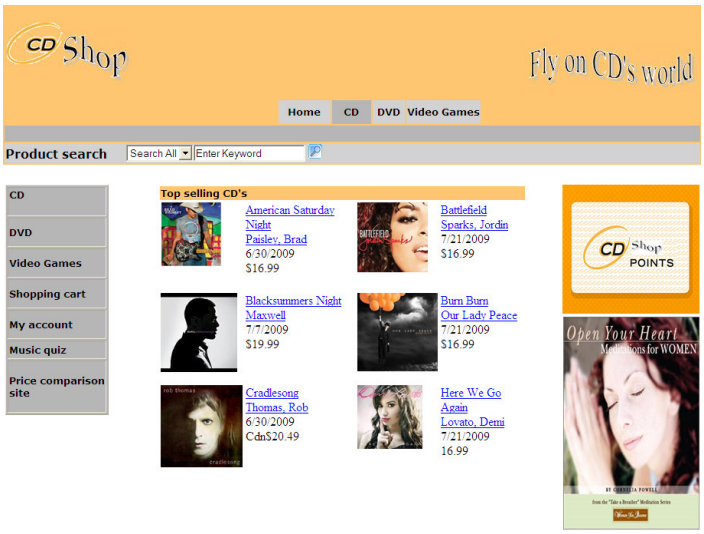


Figure 21. CD screen variant A

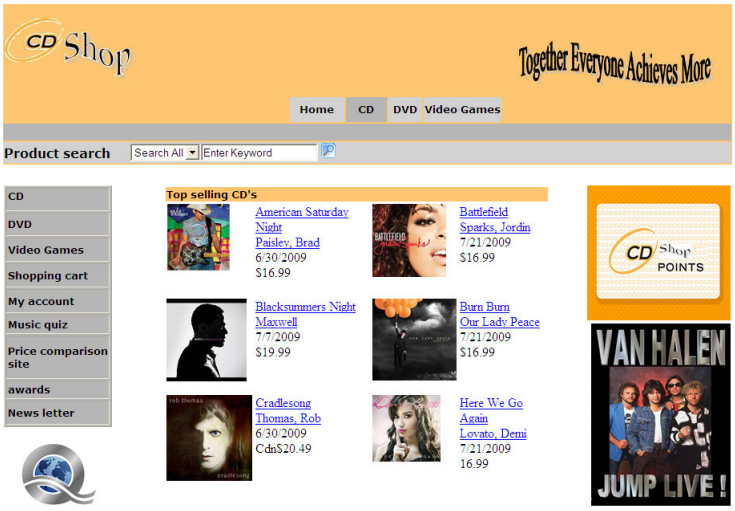


Figure 22. CD screen variant B

Before starting specifying the feature model, we want to illustrate how emphasizing some components in the design customize the website on specific cultural values. We follow the guidelines of the cultural values framework for the

adaptation of the website to conform to specific cultural dimension. Below we give some examples to be included in the interface for each dimension. Note that these are just some examples; the interface could include additional component to emphasis specific cultural value.

1- individualism-collectivism

Individualism requires:

- Picture of people (customers, models) as individuals.
- Slogan referring to freedom.

Collectivism requires:

- Link to subscribe to newsletter that is providing info on new releases of favorite artist.
- Picture of people (customers, models) as group.
- Slogan referring to team work.

2- Power distance

High power distance requires:

- Mentioning of awards won.
- Quality certification label.

Low power distance requires:

- No mentioning of the awards or the certifications.

3- Masculinity- femininity

Masculinity requires:

- Link to music/media quiz.
- Pictures of female posing with typical female CDs.

Femininity requires:

- Pictures of females and males

4- High-low context

High context requires:

- Slogan referring to the best choice.
- More and bold colors in the site.

Low context requires:

- Slogan referring to leader in quality.
- Link to price-comparison sites.

5- Uncertainty avoidance:

High on uncertainty avoidance requires:

- Link to frequently asked questions page.

Low on uncertainty avoidance requires:

- No such links.

Specifying the feature model:

When specifying the feature model of the webpage interface we include all the feature and components (buttons, menus, pictures) the page could include (mandatory, optionally, Or group, Alternative, …) in addition to specifying the cultural values as optional features.
Linking the cultural values with some features is done by means of a require (implies) cross tree relationship between the specified cultural value and the features related to it. This implies that selecting a specific cultural value requires to select the specific features. The relation between two opposite cultural values is expressed by an exclude relationship between the two features of these values. The feature model diagram is depicted in figure 23.

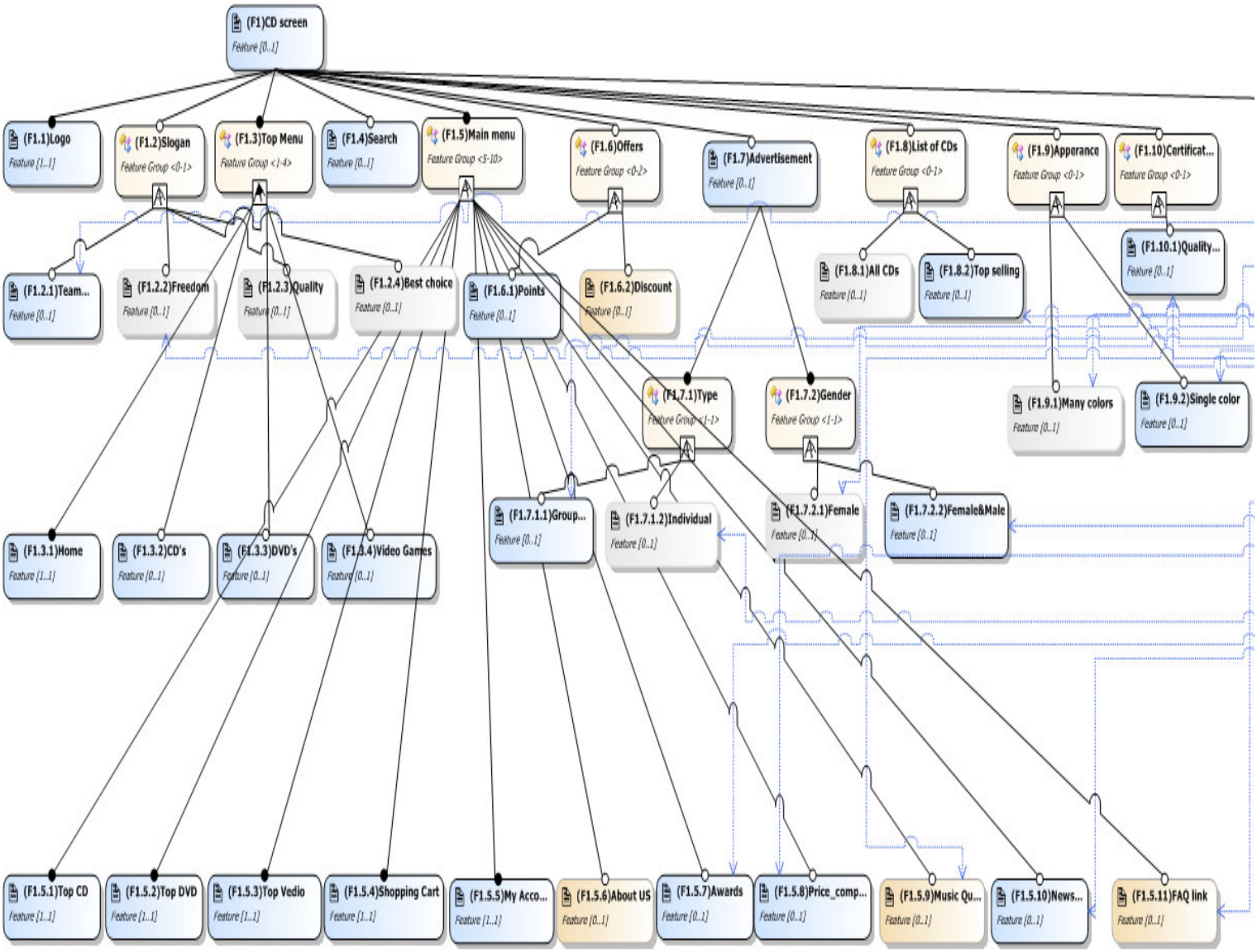


Figure 23. The feature model diagram of the CD screen (part 1)

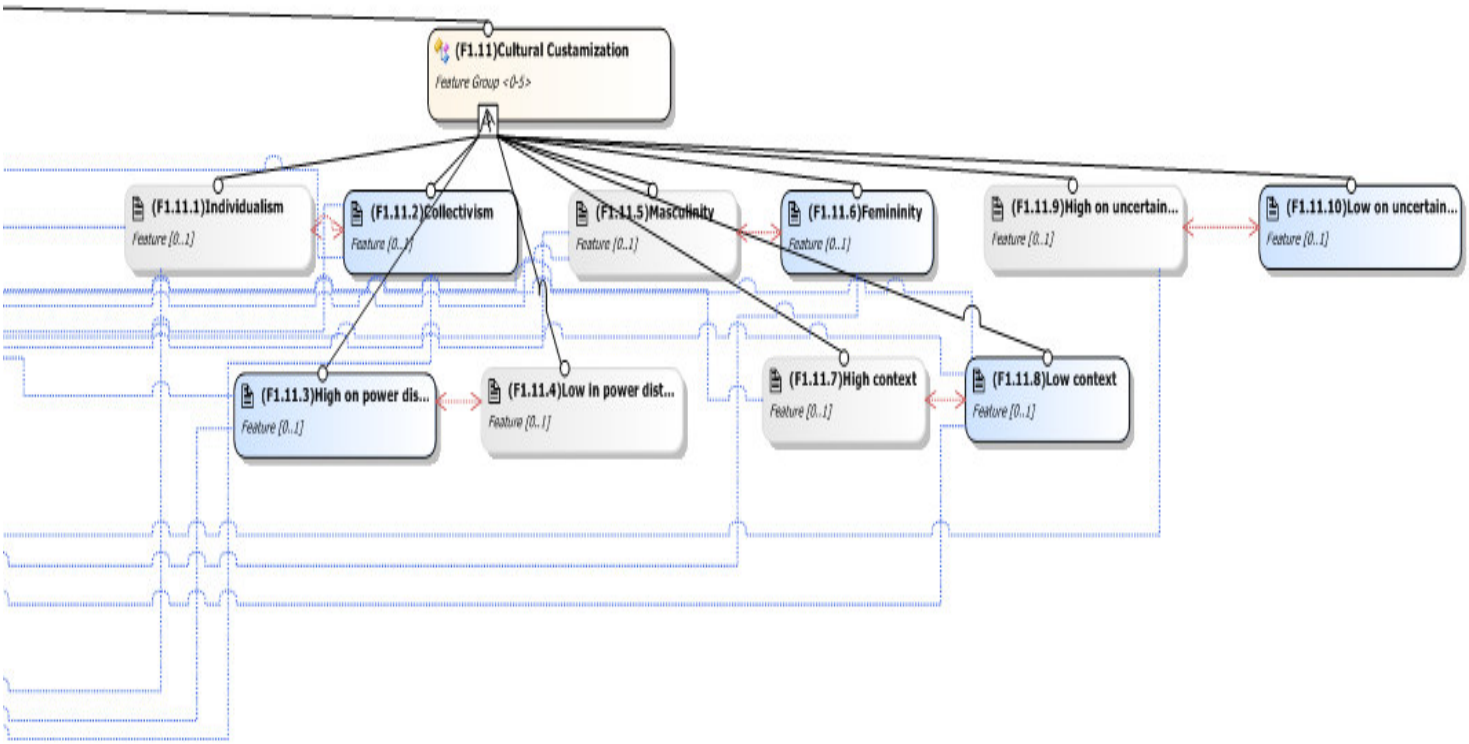


Figure 23. The feature model diagram of the CD screen (part 2)

Selecting the variants:

In fact, each country has its own mix of the culture values, thus all cultural values must be included and presented in website to be culturally customizable, However, it is not the mere existence of these features on the page that makes it customized, but the degree of emphasising it.
So to see which features should be included in the websites to be culturally customized, we start by selecting the desired cultural value and then see what features are required to be included in our webpage to be customized to that culture value. And then we can add other features if we want. In this stage, when configuring the feature model, it helps us to discover the conflicts between some features. Example conflicts are for instance, if we try to add a specific feature in the interface but we have selected a cultural value that exclude that feature, or if a feature relate to more than one cultural value but one of them should be selected. In these cases we have to solve the problem by discarding some features.

<u>Variant A: Original interface</u>
In our case study the original interface is customized to the following cultural values: Individualism, low on power distance, masculinity, low context, low on uncertainty avoidance.

So all the required feature related to these values are:

- Picture of people (customers, models) as individuals.
- Slogan referring to freedom.
- Link to music/media quiz.
- Pictures of female posing with typical female CDs.
- Slogan referring to leader in quality.
- Link to price-comparison sites.

Variant B: Modified interface

This interface is customized to adapt the following cultural values:
Collectivism, high on power distance, femininity, low context, low on uncertainty avoidance.
So there are some features that should be added or modified in the original interface according to the selected cultural values.
Actually a conflict between the two features (slogan referring to team work, Slogan referring to leader in quality) occurs because they are in an alternative group of features and only one can be selected. To solve this conflict we discard the slogan referring to leader in quality, and are satisfied with the slogan referring to the team work.
Table 5 describes all the features depicted in the web interface of the page and mentions which ones are included in original interface and which ones are modified in the other variant.

Table 6. Features of the CD screen

Feature name	**type**	**description**	**Existence in the original interface**
F1	Root feature	CD screen	
F1.1	Mandatory	Logo	Yes
F1.2	Optional	Slogan	Yes
F1.2.1	Alternative	Team work	Modified
F1.2.2	Alternative	Freedom	Yes
F1.2.3	Alternative	Quality	
F1.2.4	alternative	Best choice	
F1.3	Mandatory	Top menu	
F1.3.1	Or group	Home	Yes
F1.3.2	Or group	CD	Yes
F1.3.3	Or group	DVD	Yes
F1.3.4	Or group	Video games	Yes
F1.4	optional	Search	Yes
F1.5	mandatory	Main menu	Yes
F1.5.1	Or group	Top CD	Yes

F1.5.2	Or group	Top DVD	Yes
F1.5.3	Or group	Top video	Yes
F1.5.4	Or group	Shopping cart	Yes
F1.5.5	Or group	My account	Yes
F1.5.6	Or group	About us	
F1.5.7	Or group	Awards	Modified
F1.5.8	Or group	Price comparison site	Yes
F1.5.9	Or group	Music quiz	Yes
F1.5.10	Or group	News letter	Modified
F1.5.11	Or group	FAQ link	
F1.6	Optional	Offers	Yes
F1.6.1	Or group	Points	Yes
F1.6.2	Or group	Cards	
F1.7	Optional	Advertisement	Yes
F1.7.1	mandatory	Type	Yes
F1.7.1.1	Alternative	Group of people	Modified
F1.7.1.2	Alternative	individual	Yes
F1.7.2	Mandatory	Gender	Yes
F1.7.2.1	Alternative	Female	Yes
F1.7.2.2	Alternative	Female and male	Modified
F1.8	Mandatory	List of CDs	Yes
F1.8.1	Alternative	All CDs	
F1.8.2	Alternative	Top selling CD's	Yes
F1.9	Mandatory	Appearance	Yes
F1.9.1	Alternative	Many colors	
F1.9.2	Alternative	Single color	Yes
F1.10	Optional	Certification	Yes
F1.10.1	Or group	Quality certification	Modified
F1.11	Optional	Cultural values	Yes
F1.11.1	Or group	Individualism	Yes
F1.11.2	Or group	Collectivism	Modified
F1.11.3	Or group	High on power distance	Modified
F1.11.4	Or group	Low in power distance	yes
F1.11.5	Or group	Masculinity	Yes
F1.11.6	Or group	Femininity	Modified
F1.11.7	Or group	High context	
F1.11.8	Or group	Low context	Yes
F1.11.9	Or group	High on uncertainty avoidance	
F1.11.10	Or group	Low on uncertainty avoidance	Yes

Implementing the variant:

The original interface is implemented as usually. For the modified one we have to write aspects for all of the modified features listed in table 5.
For example the following code will replace the slogan of the company:

```
public aspect TeamWork_slogan {

public pointcut replace_slogan(HttpServletRequest request,
HttpServletResponse response) :
execution(private void CdScreen.Freedom_slogan(HttpServletRequest,
HttpServletResponse)) && args(request, response);

      void around(HttpServletRequest request, HttpServletResponse
response) throws IOException : replace_slogan (request, response)
      {
// alternative code here to replace the slogan

      }
}
```

Chapter 4: Related work:

There is several related work based on Software product lines, aspect-oriented programming and on a combination of them applied on some software development. For example in [30] they present Koriandol, a product line architecture designed to develop, deploy and maintain families of web applications which often share similar behaviors and components (commonalities) and a number of variant points (often referred to the delayed design decisions). Their definition of a web application involves the selection of components, which are assembled in a prescribed way, as depicted in the UML class diagram of figure 24. These components have some variable features. Variation in Koriandol is accomplished by means of selecting components with different features in order to put them to use in specific products, rather than writing code and keeping the variants distinguished. Producing any system within that scope becomes a matter of exercising the variation of the components and architecture that is, configured and then assembled and tested. The main benefit is maximizing code reusing and minimizing code writing.

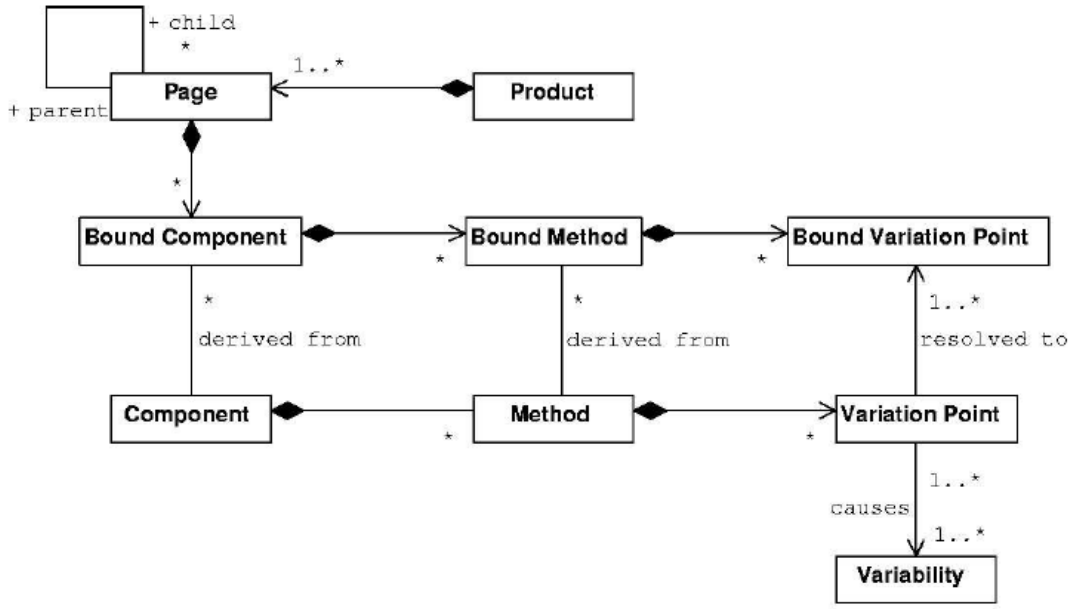


Figure 24. Associations among the assets in a product

In this work feature models are used to represent common and variable features of the components, figure 25 shows an example of the News component. After a feature has been identified and modeled, a reusable component which implements it is developed (in the previous example a mapping between the notation of feature diagrams and HTML widgets (e.g., check boxes, radio button and so on) has been defined), When completed, the component is registered into

the repository and becomes available to all applications. Figure 26 shows the architecture of the Koriandol system

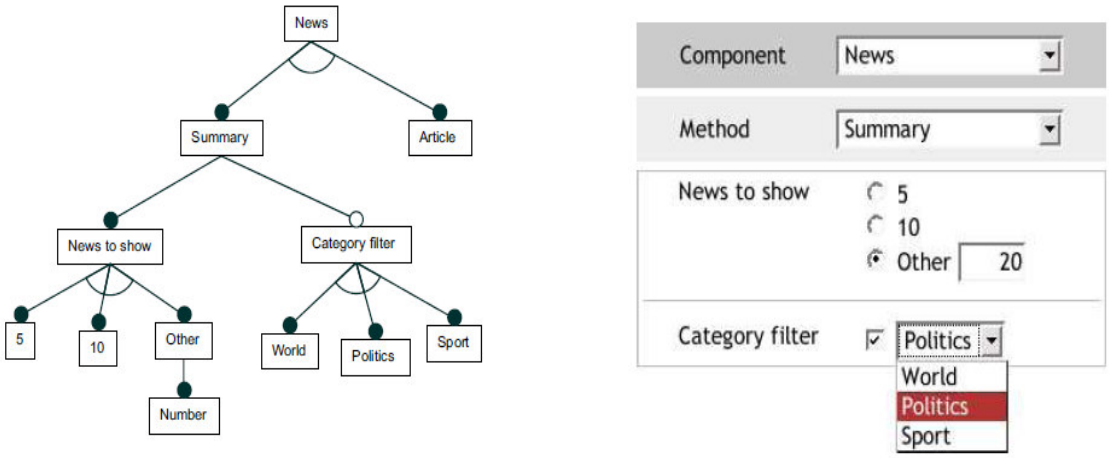


Figure 25. Feature diagram for a News component

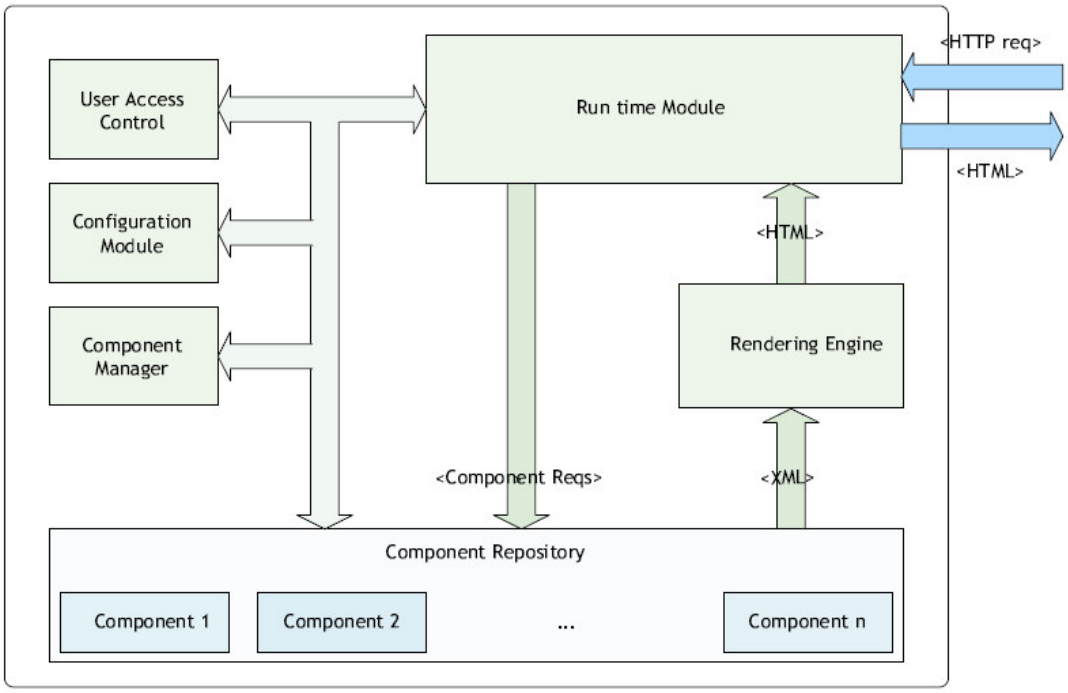


Figure 26. Koriandol system

When a client request (HTTP Request) is sent from the browser to the run time module, it interprets it according to the correspondent page specification and identifies which components have to be invoked to serve that particular request, then feature request action is sent to the component that was selected and configured during the variability determination for the instantiation of the

requested page. The method returns XML data which is passed to the rendering engine which generates HTML fragment by means of transformation stylesheets. Once all component requests are realized the obtained HTML segments (HTML contents) are put together and returned to the user (HTML Page).
This work is related to ours in the area of exploring the commonalities and the variabilties of the systems by using feature modeling, but they have their own implementation, “the Koriandol system”, which supports variability determination through built-in reflective mechanisms. While in our work achieving the variability is realized through defining aspects that cross cut some modules of the original system code and modify or replace them.

Using aspect oriented programming to achieve web changes is described in [31]. The authors proposed an approach to web application evolution in which changes are represented by aspect-oriented design patterns and program schemes. These changes could be kind of integration changes, grid display changes, input form changes…etc, change realization they have proposed actually is accompanied by its own specification. This means that the initial description of the changes to be applied is application specific, each application specific change can be seen as a specialization of some generally applicable change, this is depicted in figure 27 in which a general change with two specializations is presented. Thus, they determine the generally applicable change whose specialization is the application specific change and adapt its realization scheme.

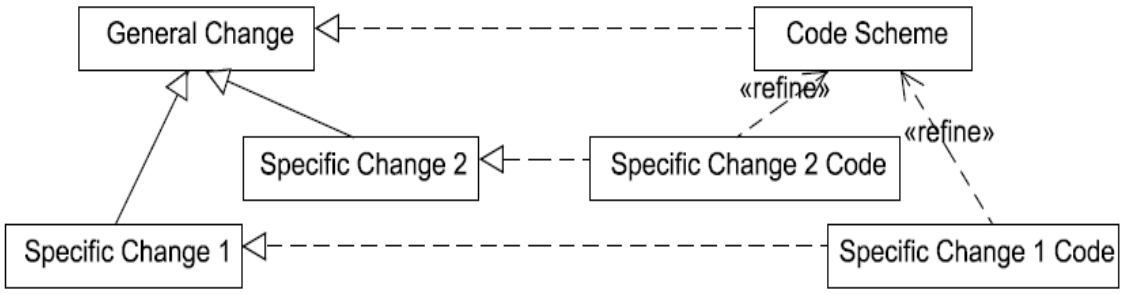


Figure 27. General and specific changes with realization.

The main difference between our work and their work is that we rely on feature modeling to describe the commonalties and the variabilites of the different versions of our web application interface and they specify changes in general as application specific, so a catalogue of the changes is developed, in which the generalization specialization relationships between change types is explicitly established.

The idea of combining software product lines and aspect-oriented software development techniques already exists in software engineering. The author in [32] presented detailed guidelines on how FOA (Feature-oriented analysis), and AOP (aspect-oriented programming) can be combined. He saw that one-to-one mapping between features and aspects is not a practical solution because if aspects implementing features are dependent on each other their variation may cause changes to other parts of the assets. To address the problems, he has proposed a product line engineering method that combines FOA and AOP to enhance reusability, adaptability, and configurability of product line assets. He proposed that commonalities must be decoupled from variabilities in product line asset development, That is, common features are used to design base modular components, and variable features are used to define aspectual components, which adapt or extend the base modular components. This does not mean that all common features have to be designed as base modular components. Common features can also be defined as aspectual components, if they have crosscutting concerns. Here crosscutting concerns can be classified into two categories: homogeneous and heterogeneous. Homogeneous crosscuts (e.g., logging) add the same code fragments at different join points of multiple components, they can be effectively defined as separate aspects using AOP, whereas heterogeneous crosscuts (e.g., services) add different code, and they can be incorporated into existing modular components or defined as separate aspectual components. This work provides us with guidelines when implementing the commonalities and the variabilities in our work especially when implementing the part of variabilities between the different versions of the web interface with aspects.

Also another contribution in managing variability of SPL of web applications using AOSD is presented in [33]. They propose an aspect oriented framework to manage variability in software product lines, figure 28 shows the two main activities of the proposed framework: 1- Domain Engineering activity in which the commonality and the variability of the product line are created. 2- Application Engineering activity in which the applications of the product line are built by reusing assets.

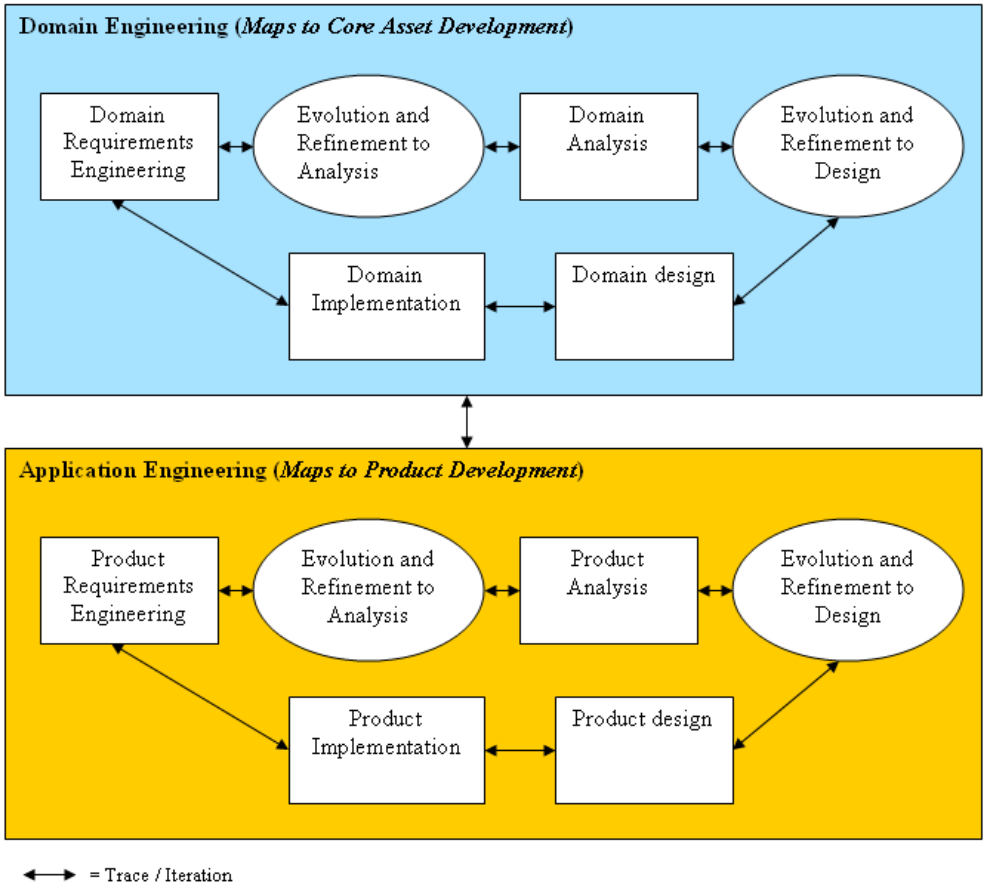


Figure 28. Aspect-Oriented Product Line Framework

In domain engineering activity they propose for capturing the commonality and variability “the use cases” where use cases are categorized as kernel, optional, or variant, and mapping rules are used to establish relationships between them. For this mapping they create table 6 which works as ”a production plan”, as it describes how the products can be created from the core assets, and shows which concerns are involved in the architecture. The next step is defining aspects from the established crosscutting concerns, creating a refined use case diagram with them (figure 29) and following proposed mapping rules. Several phases followed then as shown in figure 28. Finally in the application engineering activity, the developers follow the production plan, in order to recognize the variation points of specific product depending on the variabilities that were defined as aspects in every stage of the domain engineering activity.
We share the basic idea with this work by combining SPL with AOSD but we use feature models to specify the variabilities between the applications(web interfaces) while they build a table to describe the different variants and then translate it into a use case sequence diagram and defined aspects.

Table 7. Kernel concerns vs. crosscutting concerns with variants

Crosscutting Concerns		**Kernel Concerns**		
Optional and Non-Functional Requirements	**Variants**	*Kernel concern 1*	*Kernel concern 2*	*Kernel concern n*
Optional use case 1	Variant 1		X	
	Variant 2			X
Non-functional requirement 1		X	X	
Non-functional requirement n		X	X	X

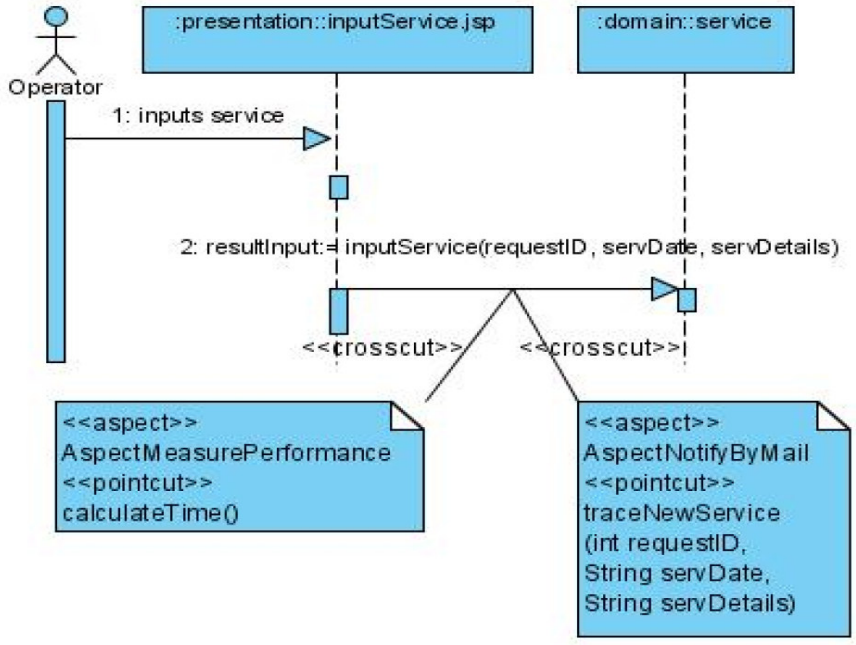


Figure 29. Fragment of the sequence diagram for the use case “Input Service”

Chapter 5: Conclusion

During the last years Internet marketing has experienced a remarkable growth, however it also brought many challenges to websites developers as website design is considered as one of the key success factor of the business because it can impact the buying process for new purchasers. Also experiences have shown that small changes to a website interface including adding small modification in order to adapt the website to a specific culture affect the percentage of site visits that result in purchase (conversion rate). Many techniques have been developed to measure the user's reactions to the modifications of the site interface like the known A/B test, in which the designers develop more than one version of a web interface and present it to randomly selected subset of users in order to analyze their reactions, and determine which version has the highest performance. However, the problem was how to develop quickly different versions/variants of a specific website interface, and how to choose components of the interface elements (features) to vary. Since these variants share common features, designing these variants by reusing the common features between them is more efficient as it reduces the time and effort and improves the consistency among all them.

In this thesis we present an approach for website experimenting that enable us to create one or more different versions of a web interface by incorporating new or modified features with reuse of the original code. In this case we don't need to create a complete new website to produce a different variant of the interface; we simply can use the original one to obtain a new version that has some additional or modified features. To achieve this, we mainly rely on two techniques:1- Software product line engineering and Aspect oriented programming. Software product lines allow us to capture the commonality and the variability between products family throw feature modeling, in our case feature modeling is used for specifying and modeling the variability features of the website interface and then by selecting different sets of features from the model we can produce different variants of the interface. One of these variants will be considered as the original web interface for the application and the others are modified ones that can be used in the tests. Aspect oriented programming allow us to capture the variable features of the variants by defining aspect codes for each of the variable features that does not exist in the original website. Then the weaving process of the aspect code and the original code will produce new modified code that creates a new web interface variant. So there is no need for a complete new implementation of an application or for modifying applications manually, which may lead to cumbersome maintenance problems. Aspects can add or modify the variable features among the different versions of the web interface. There is however one requirement for being able to apply this approach, the implementation of the original code should meet the modularity principle in programming. i.e. the program should be divided into modules to represent the interface components. This is needed to allow the aspects to effectively change the modular components (interface features).

Our approach includes the following stages: building the feature model, configuring it to produce different variants where one of these variants must be considered as the original web interface for the application and should be developed as any web application (we have proposed Java servlet as programming tool). Then achieving the other variants is by incorporating aspects in the original code. Each variable feature in these variants is associated with an aspect (we have used AspectJ as aspect oriented implementation language) that add or modify a specific feature in the original interface. Once all the aspects related to the variant interface have been defined, we collect them with the original code; the weaving process of the original code and the aspect will happen at compile time.

To illustrate our approach we have applied it on two case studies. In the first case study, we have seen how we can apply small changes in a checkout page of a shopping cart. We have defined the feature model of the page and then configured two variant. We have considered the first one as the original interface and for the other one we have defined aspect code for each of the variable features and thus the other variant can be produced easily.

In the second case study, we have shown how we can apply our approach in the case of producing culturally customized websites. These sites reflect the culture of the target market. By often small modifications in some part of the interface we can adapt the website to a specific culture. Our feature model in this case study include the five cultural values presented in [4], together with a set of required features for each cultural value to be added to the interface to customize it. Configuring the feature model by selecting different features allow us to produce a mix of different culture values and thus producing different variants of the web interface each of them can serve a particular target society. The implementation of these variants is as described before and is also reusing the original interface code and defines aspects for each of the modified features.

The main benefit of applying our approach is maximizing code reusing by reusing the code of the core assets of the original application, in this way we just need to write some aspect to obtain the desired new features.

Future work could include the development of a software environment to support our approach, as well as more experiments with different use cases to validate the approach.

References

[1] Dave Chaffey, Richard Mayer, Kevin Johnston, Fiona Ellis-Chadwick, Internet Marketing Strategy, Implementation and Practice, Pearsoneduc 2000.

[2] Randy Duermyer, How to Convert Your Web Site Traffic to Cash – Intro: <http://homebusiness.about.com/od/yourbusinesswebsite/a/conversion.htm>.

[3] Bryan Eisenberg, How to Increase Conversion Rate 1,000 Percent: < http://www.clickz.com/1756031>.

[4] Nitish Singh, Arun Pereira, The culturally customized Web site: Customizing Web Sites for the Global Marketplace, Butterworth-Heinemann, 2005.

[5] Bryan Eisenberg, How to Improve A/B Testing: <http://www.clickz.com/3500811>.

[6] Split A/B testing: <http://www.webcredible.co.uk/user-friendly-resources/web-usability/ab-testing.shtml>.

[7] Kwanwoo Lee, Kyo C. Kang and Jaejoon Lee, Concepts and Guidelines of Feature Modeling for Product Line Software Engineering , IEEE Software, 2002

[8] Pohl, Klaus, Böckle, Günter, Linden, Frank J. van der, Software Product Line Engineering Foundations, Principles and Techniques, Springer, 2005.

[9] Kang, K. C., S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute (Carnegie Mellon), 1999.

[10] Rubén Fernández, Miguel A. Laguna, Jesús Requejo, Nuria Serrano, Development of a Feature Modeling Tool using Microsoft DSL Tools, < http://www.giro.infor.uva.es/fmt.pdf>.

[11] Matthias Riebisch: Towards a More Precise Definition of Feature Models. Position Paper. In: M. Riebisch, J. O. Coplien, D, Streitferdt (Eds.): Modelling Variability for Object-Oriented Product Lines. BookOnDemand Publ. Co., Norderstedt, 2003.

[12] László Lengyel, Tihamér Levendovszky, Hassan Charaf, Constraint handling in Feature Models, 5th International Symposium of Hungarian Researchers on Computational Intelligence, 2004.

[13] Hai Wang,Yuan Fang Li, Jing Sun3 Hongyu Zhang and Jeff Pan, A Semantic Web Approach to Feature Modeling and Verification, Workshop on Semantic Web Enabled Software Engineering (SWESE'05), 2005.

[14] Krzysztof Czarnecki, Chang Hwan Peter Kim , Cardinality-Based Feature Modeling and Constraints: A Progress Report, ACM 1-59593-193-7/05/0010., 2005.

[15] Kyo C. Kang, Sajoong Kim , Jaejoon Lee , Kijoo Kim , Euiseob Shin and Moonhang Huh , A feature-oriented reuse method with domain specific reference architectures, Annuals of Software Engineering, 5, 1998.

[16] Griss, M.L., Favaro, J., d'Alessandro, M., Integrating feature modeling with the RSEB, Proceedings of the Fifth International Conference on Software Reuse, 1998.

[17] Yasser EL-Manzalawy , Aspect Oriented Programming:
< http://www.developer.com/design/article.php/3308941 >.

[18] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kerste2, Jeffrey Palm and William G. Griswold, An Overview of AspectJ, ACM, 2001.

[19] Russ Miles, AspectJ Cookbook, O'Reilly Media, 2004.

[20] AspectJ: Frequently Asked Questions:
<http://www.eclipse.org/aspectj/doc/released/faq.php>.

[21] Jason Hunter, William Crawford, Java Servlet Programming, O'Reilly, 1998.

[22] Stefan Kuhlins and Axel Korthaus, Java Servlets versus CGI Implications for Remote Data Analysis, 23rd Annual Conference of the Gesellschaft fur Klassifikation, 1999.

[23] Java servlet life cycle:
< http://www.javaonnet.com/2005_04_01_archive.html>.

[24] Modular programming: <http://www.answers.com/topic/modularity>.

[25] Improve modularity with aspect-oriented programming
<http://www.ibm.com/developerworks/java/library/j-aspectj/>.

[26] Modular programming:
<http://www.pcmag.com/encyclopedia_term/0,2542,t=modular+programming&i=47184,00.asp>.

[27] Feature model: <http://en.wikipedia.org/wiki/Feature_model>.

[28] Conversion Rate Optimization:
<http://www.maximize-conversion-rate.com/conversion_optimization/2009/01/-conversion-rate-optimization-101-.html>

[29] Adrian Colyer, Andy Clement, George Harley, Matthew Webster, Eclipse AspectJ: Aspect-Oriented Programming with AspectJ and the Eclipse - AspectJ Development Tool, Addison-Wesley Professional, 2004.

[30] Luca Balzerani, Davide Di Ruscio, Alfonso Pierantonio, Guglielmo De Angelis, Supporting Web Applications Development with a Product Line Architecture, Journal of Web Engineering, Vol. 5, No. 1, 2006.

[31] Michal Bebjak, Valentino Vrani´c, and Peter Dolog, Evolution of Web Applications with Aspect-Oriented Design Patterns, proceedings of 2nd International Workshop on Adaptation and Evolution in Web Systems Engineering, AEWSE 2007, in conjunction with 7th International Conference on Web Engineering, ICWE 2007.

[32] Kwanwoo Lee, Kyo C. Kang, Minseong Kim, Sooyong Park, Combining Feature-Oriented Analysis and Aspect-Oriented Programming for Product Line Asset Development, Software Product Line Conference10th International, 2006 .

[33] Germán Harvey Alférez Salinas, Poonphon Suesaowaluk, An Aspect-Oriented Product Line Framework to Support the Development of Software Product Lines of Web Applications, Proceedings of the 24th South East Asia Regional Computer Conference, 2007.

[34] Learn AspectJ to better understand aspect-oriented programming:
<http://www.javaworld.com/javaworld/jw-03-2002/jw-0301-aspect2.html?page=3>

[35] Yves Bontemps, Patrick Heymans, Pierre-Yves Schobbens, and Jean-Christophe Trigaux, Semantics of FODA Feature Diagrams, The 8th international Conference on Software Product Line Conference (SPLC'04), 2004.

[36] Feature Model DSL: <http://featuremodeldsl.codeplex.com/>.

[37] XFeature Tool : <http://www.pnp-software.com/XFeature/Home.html>.

[38] Feature modeling tool : <http://www.giro.infor.uva.es/FeatureTool.html>.