Evolution of Web Applications with Aspect-Oriented Design Patterns

Michal Bebjak[1], Valentino Vranić[1], and Peter Dolog[2]

[1] Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technology
Slovak University of Technology,
Ilkovičova 3, 84216 Bratislava 4, Slovakia
bebjak02@student.fiit.stuba.sk, vranic@fiit.stuba.sk

[2] Department of Computer Science
Aalborg University
Fredrik Bajers Vej 7, building E, DK-9220 Aalborg EAST, Denmark
dolog@cs.aau.dk

Abstract. It is more convenient to talk about changes in a domain-specific way than to formulate them at the programming construct level or—even worse—purely lexical level. Using aspect-oriented programming, changes can be modularized and made reapplicable. In this paper, selected change types in web applications are analyzed. They are expressed in terms of general change types which, in turn, are implemented using aspect-oriented programming. Some of general change types match aspect-oriented design patterns or their combinations.

1 Introduction

Changes are inseparable part of software evolution. Changes take place in the process of development as well as during software maintenance. Huge costs and low speed of implementation are characteristic to change implementation. Often, change implementation implies a redesign of the whole application. The necessity of improving the software adaptability is fairly evident.

Changes are usually specified as alterations of the base application behavior. Sometimes, we need to revert a change, which would be best done if it was expressed in a pluggable way. Another benefit of change pluggability is apparent if it has to be reapplied. However, it is impossible to have a change implemented to fit any context, but it would be sufficiently helpful if a change could be extracted and applied to another version of the same base application. Such a pluggability can be achieved by representing changes as aspects [5]. Some changes appear as real crosscutting concerns in the sense of affecting many places in the code, which is yet another reason for expressing them as aspects.

This would be especially useful in the customization of web applications. Typically, a general web application is adapted to a certain context by a series of changes. With arrival of a new version of the base application all these changes

have to be applied to it. In many occasions, the difference between the new and the old application does not affect the structure of changes.

A successful application of aspect-oriented programming requires a structured base application. Well structured web applications are usually based on the Model-View-Controller (MVC) pattern with three distinguishable layers: model layer, presentation layer, and persistence layer.

The rest of the paper is organized as follows. Section 2 establishes a scenario of changes in the process of adapting affiliate tracking software used throughout the paper. Section 3 proposes aspect-oriented program schemes and patterns that can be used to realize these changes. Section 4 identifies several interesting change types in this scenario applicable to the whole range of web applications. Section 5 envisions an aspect-oriented change realization framework and puts the identified change types into the context of it. Section 6 discusses related work. Section 7 presents conclusions and directions of further work.

2 Adapting Affiliate Tracking Software: A Change Scenario

To illustrate our approach, we will employ a scenario of a web application throughout the rest of the paper which undergoes a lively evolution: affiliate tracking software. Affiliate tracking software is used to support the so-called affiliate marketing [6], a method of advertising web businesses (merchants) at third party web sites. The owners of the advertising web sites are called affiliates. They are being rewarded for each visitor, subscriber, sale, and so on. Therefore, the main functions of such affiliate tracking software is to maintain affiliates, compensation schemes for affiliates, and integration of the advertising campaigns and associated scripts with the affiliates web sites.

In a simplified schema of affiliate marketing a customer visits an affiliate's page which refers him to the merchant page. When he buys something from the merchant, the provision is given to the affiliate who referred the sale. A general affiliate tracking software enables to manage affiliates, track sales referred by these affiliates, and compute provisions for referred sales. It is also able to send notifications about new sales, signed up affiliates, etc.

Suppose such a general affiliate tracking software is bought by a merchant who runs an online music shop. The general affiliate software has to be adapted through a series of changes. We assume the affiliate tracking software is prepared to the integration with the shopping cart. One of the changes of the affiliate tracking software is adding a backup SMTP server to ensure delivery of the news, new marketing methods, etc., to the users.

The merchant wants to integrate the affiliate tracking software with the third party newsletter which he uses. Every affiliate should be a member of the newsletter. When selling music, it is important for him to know a genre of the music which is promoted by his affiliates. We need to add the genre field to the generic affiliate signup form and his profile screen to acquire the information about the genre to be promoted at different affiliate web sites. To display it, we need to

modify the affiliate table of the merchant panel so it displays genre in a new column. The marketing is managed by several co-workers with different roles. Therefore, the database of the tracking software has to be updated with an administrator account with limited permissions. A limited administrator should not be able to decline or delete affiliates, nor modify campaigns and banners.

3 Aspect-Oriented Change Representation

In the AspectJ style of aspect-oriented programming, the crosscutting concerns are captured in units called aspects. Aspects may contain fields and methods much the same way the usual Java classes do, but what makes possible for them to affect other code are genuine aspect-oriented constructs, namely: *pointcuts*, which specify the places in the code to be affected, *advices*, which implement the additional behavior before, after, or instead of the captured *join point*[3], and *inter-type declarations*, which enable introduction of new members into existing types, as well as introduction of compile warnings and errors.

These constructs enable to affect a method with a code to be executed before, after, or instead of it, which may be successfully used to implement any kind of *Method Substitution* change (not presented here due to space limitations). Here we will present two other aspect-oriented program schemes that can be used to realize some common changes in web application. Such schemes may actually be recognized as aspect-oriented design patterns, but it is not the intent of this paper to explore this issue in detail.

3.1 Class Exchange

Sometimes, a class has to be exchanged with another one either in the whole application, or in a part of it. This may be achieved by employing the Cuckoo's Egg design pattern [8]. A general code scheme is as follows:

```
public aspect ExchangeClass {
   public pointcut exhangedClassConstructor(): call(ExchangedClass.new(..);
   Object around(): exhangedClassConstructor() { return getExchangingObject();}
   ExchangeObject getExchangingObject() {
      if (. . .)
         new ExchangingClass();
      else
         proceed();
   }
}
```

The exhangedClassConstructor() is a pointcut that captures the ExchangedClass constructor calls using the **call**() primitive pointcut. The around advice captures these calls and prevents the ExchangedClass instance from being created. Instead, it calls the getExchangingObject() method which implements the exchange logic. ExchangingClass has to be a subtype of ExchangedClass.

[3] Join points represent well-defined places in the program execution.

The example above sketches the case in which we need to allow the construction of the original class instance under some circumstances. A more complicated case would involve several exchanging classes each of which would be appropriate under different conditions. This conditional logic could be implemented in the getExchangingObject() method or—if location based—by appropriate pointcuts.

3.2 Perform an Action After an Event

We often need to perform some action after an event, such as sending a notification, unlocking product download for user after sale, displaying some user interface control, performing some business logic, etc. Since events are actually represented by method calls, the desired action can be implemented in an after advice:

```
public aspect AdditionalReturnValueProcessing {
   pointcut methodCallsPointcut(TargetClass t, int a): . . .;
   after(/* captured arguments */): methodCallsPointcut(/* captured arguments */) {
      performAction(/* captured arguments */);
   }
   private void performAction(/* arguments */) { /* action logic */ }
}
```

4 Changes in Web Applications

The changes which are required by our scenario include integration changes, grid display changes, input form changes, user rights management changes, user interface adaptation, and resource backup. These changes are applicable to the whole range of web applications. Here we will discuss three selected changes and their realization.

4.1 Integration Changes

Web applications often have to be integrated with other systems (usually other web applications). Integration with a newsletter in our scenario is a typical example of *one way integration*. When an affiliate signs up to the affiliate tracking software, we want to sign him up to a newsletter, too. When the affiliate account is deleted, he should be removed from the newsletter, too.

The essence of this integration type is one way notification: only the integrating application notifies the integrated application of relevant events. In our case, such events are the affiliate signup and affiliate account deletion. A user can be signed up or signed out from the newsletter by posting his e-mail and name to the one of the newsletter scripts. Such an integration corresponds to the Perform an Action After an Event change (see Sect. 3.2). In the after advice we will make a post to the newsletter sign up/sign out script and pass it the e-mail address and name of the newly signed up or deleted affiliate. We can seamlessly combine multiple one way integrations to integrate a system with several systems.

Introducing a *two way integration* can be seen as two one way integration changes: one applied to each system. A typical example of such a change is data synchronization (e.g., synchronization of user accounts) across multiple systems. When the user changes his profile in one of the systems, these changes should be visible in all of them. For example, we may want to have a forum for affiliates. To make it convenient to affiliates, user accounts of the forum and affiliate tracking system should be synchronized.

4.2 Introducing User Rights Management

Many web applications don't implement user rights management. If the web application is structured appropriately, it should be possible to specify user rights upon the individual objects and their methods, which is a precondition for applying aspect-oriented programming.

User rights management can be implemented as a Border Control design pattern [8]. According to our scenario, we have to create a restricted administrator account that will prevent the administrator from modifying campaigns and banners and decline/delete affiliates. All the methods for campaigns and banners are located in the campaigns and banners packages. The appropriate region specification will be as follows:

```
pointcut prohibitedRegion(): (within(application.Proxy) && call(void *.*(..)))
   || (within(application.campaigns.+) && call(void *.*(..)))
   || within(application.banners.+)
   || call(void Affiliate.decline(..)) || call(void Affiliate.delete(..));
}
```

Subsequently, we have to create an around advice which will check whether the user has rights to access the specified region. This can be implemented using the Method Substitution change applied to the pointcut specified above.

4.3 Introducing a Resource Backup

As specified in our scenario, we would like to have a backup SMTP server for sending notifications. Each time the affiliate tracking software needs to send a notification, it creates an instance of the SMTPServer class which handles the connection to the SMTP server and sends an e-mail. The change to be implemented will ensure employing the backup server if the connection to the primary server fails. This change can be implemented straightforwardly as a Class Exchange (see Sect. 3.1)

5 Aspect-Oriented Change Realization Framework

The previous two sections have demonstrated how aspect-oriented programming can be used in the evolution of web applications. Change realizations we have proposed actually cover a broad range of changes independent of the application

domain. Each change realization is accompanied by its own specification. On the other hand, the initial description of the changes to be applied in our scenario is application specific. With respect to its specification, each application specific change can be seen as a specialization of some generally applicable change. This is depicted in Fig. 1 in which a general change with two specializations is presented. However, the realization of such a change is application specific. Thus, we determine the generally applicable change whose specialization our application specific change is and adapt its realization scheme.

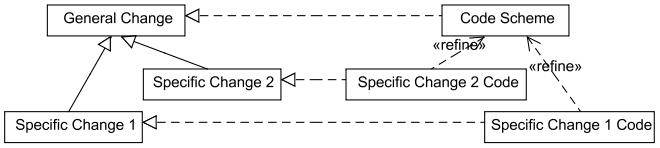

Fig. 1. General and specific changes with realization.

When planning changes, it is more convenient to think in a domain specific manner than to cope with programming language specific issues directly. In other words, it is much easier to select a change specified in an application specific manner than to decide for one of the generally applicable changes. For example, in our scenario, an introduction of a backup SMTP server was needed. This is easily identified as a resource backup, which subsequently brings us to the realization in the form of the Class Exchange.

6 Related Work

Various researchers have concentrated on the notion of evolution from automatic adaptation point of view. Evolutionary actions which are applied when particular events occur have been introduced [9]. The actions usually affect content presentation and navigation. Similarly, active rules have been proposed for adaptive web applications with the focus on evolution [4]. However, we see evolution as changes of the base application introduced in a specific context. We use aspect orientation to modularize the changes and reapply them when needed.

Our work is based on early work on aspect-oriented change management [5]. We argue that this approach is applicable in wider context if supported by a version model for aspect dependency management [10] and with appropriate aspect model that enables to control aspect recursion and stratification [1]. Aspect-oriented programming community explored several specific issues in software evolution such as database schema evolution with aspects [7] or aspect-oriented extensions of business processes and web services with crosscutting concerns of reliability, security, and transactions [3]. However, we are not aware of any work aiming specifically at capturing changes by aspects in web applications.

7 Conclusions and Further Work

We have proposed an approach to web application evolution in which changes are represented by aspect-oriented design patterns and program schemes. We identified several change types that occur in web applications as evolution or customization steps and discussed selected ones along with their realization. We also envisioned an aspect-oriented change realization framework.

To support the process of change selection, the catalogue of changes is needed in which the generalization-specialization relationships between change types would be explicitly established. We plan to search for further change types and their realizations. It is also necessary to explore change interactions and evaluate the approach practically.

Acknowledgements The work was supported by the Scientific Grant Agency of Slovak Republic (VEGA) grant No. VG 1/3102/06 and Science and Technology Assistance Agency of Slovak Republic contract No. APVT-20-007104.

References

[1] E. Bodden, F. Forster, and F. Steimann. Avoiding infinite recursion with stratified aspects. In Robert Hirschfeld et al., editors, *Proc. of NODe 2006*, LNI P-88, pages 49–64, Erfurt, Germany, September 2006. GI.

[2] S. Casteleyn et al. Considering additional adaptation concerns in the design of web applications. In *Proc. of 4th Int. Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems (AH2006)*, LNCS 4018, Dublin, Ireland, June 2006. Springer.

[3] A. Charfi et al. Reliable, secure, and transacted web service compositions with ao4bpel. In *4th IEEE European Conf. on Web Services (ECOWS 2006)*, pages 23–34, Zürich, Switzerland, December 2006. IEEE Computer Society.

[4] F. Daniel, M. Matera, and G. Pozzi. Combining conceptual modeling and active rules for the design of adaptive web applications. In *Workshop Proc. of 6th Int. Conf. on Web Engineering (ICWE 2006)*, New York, NY, USA, 2006. ACM Press.

[5] P. Dolog, V. Vranić, and M. Bieliková. Representing change by aspect. *ACM SIGPLAN Notices*, 36(12):77–83, December 2001.

[6] S. Goldschmidt, S. Junghagen, and U. Harris. *Strategic Affiliate Marketing*. Edward Elgar Publishing, 2003.

[7] R. Green and A. Rashid. An aspect-oriented framework for schema evolution in object-oriented databases. In *Proc. of the Workshop on Aspects, Components and Patterns for Infrastructure Software (in conjunction with AOSD 2002)*, Enschede, Netherlands, April 2002.

[8] R. Miles. *AspectJ Cookbook*. O'Reilly, 2004.

[9] F. Molina-Ortiz, N. Medina-Medina, and L. García-Cabrera. An author tool based on SEM-HP for the creation and evolution of adaptive hypermedia systems. In *Workshop Proc. of 6th Int. Conf. on Web Engineering (ICWE 2006)*, New York, NY, USA, 2006. ACM Press.

[10] E. Pulvermüller, A. Speck, and J. O. Coplien. A version model for aspect dependency management. In *Proc. of 3rd Int. Conf. on Generative and Component-Based Software Engineering (GCSE 2001)*, LNCS 2186, pages 70–79, Erfurt, Germany, September 2001. Springer.