CONCERNS VS COMPONENTS FOR WEB DEVELOPMENT

Antonia M. Reina, Jesús Torres, Miguel Toro, Juan A. Álvarez
Languages and Systems Department. University of Seville
Avda. Reina Mercedes, s/n. 41012 Seville

ABSTRACT

The Web has been growing and evolving in the last couple of decades at a very high speed, causing web applications to be more and more complex. Aspect oriented programming gives us a way of reducing the complexity of software development process, in the sense that it is easier to reason about concepts separately. This paper wants to explore the way in which aspects are being applied during the development of web applications and, at the same time, compare the benefits of using aspects with the benefits of other emerging strategies for developing web applications, such as Cocoon, a component-oriented web framework publishing.

KEYWORDS

Aspect-oriented programming, XML, Web development, advanced separation of concerns.

1. INTRODUCTION

In the last few years, the growing of World Wide Web has made many companies think of web applications as an important source of income. Thus, we have reached the point where to obtain sophisticated Web applications that meet the customer's demands is crucial for many companies. Aspect oriented programming (AOP, for short) can be seen as a way of reducing the complexity of applications, in the sense that it is easier to reason about isolated problems. Hence, it seems that if we apply this principle to web development, we end up reducing the complexity of web applications.

The main goals of this paper are to know the strengths and drawbacks of aspect oriented technology for web development and find out the concerns that have been addressed during the web development process. We also want to compare the development process with AOP and the development with another emerging technology. Thus, the first stage of this work has been the development of a web application, on the one hand, using aspect technology, and, on the other hand, using Cocoon2 [Mazzocchi, 2002], which it has been chosen because it stresses the separation between presentation and content. Cocoon2 is a web framework publishing that is part of the Apache project, and it is introduced in Section 2.

The aspectual version has been developed with AspectJ [The AspectJ Team, 2002], because it is the most popular and most extended aspect-oriented proposal. However, in the near future, we want to experiment with other aspect-oriented proposals. Section 3 gives an overview of aspect oriented technology. Section 4 is devoted to give a brief explanation of the application developed and to compare both technologies. Finally, in section 5, we will conclude the paper and point out our future lines of research.

2. A FRAMEWORK BASED ON COMPONENTS: COCOON

The rapid evolution of the Web has made many technologies bring up in order to face these changes. But, we were looking for a technology that was capable of separating presentation and content. By definition, technologies such as ASP, JSP or PHP mix logic and presentation. Although they give us some mechanisms to minimize this mixture, logic and presentation still jumble due to the use of HTML. On the other hand, Cocoon [The Apache Project, 2002] is an emerging technology based on XML and XSLT which stresses the separation of logic, content and presentation. Moreover, it is based on components, which also can be seen as a way of separating things, in this case according to functionality.

Cocoon is part of the Apache XML Project. It started in 1998 having in mind the idea of separating style from content, but Cocoon 1.x was designed when the XML technology was very young and there was very little experience. Therefore, in 1999, a new release of Cocoon was born in order to solve some problems of the earlier one. This new release is called Cocoon2 and it can be seen as a web publishing framework. The two most important requirements that Cocoon2 had to fulfill were:

- *Execution speed efficiency and memory efficiency*. This requirement implied the choice of SAX [The SAX Project, 2002] processing model instead DOM (Document Object Model) to interpret XML, because it is simpler than DOM and it is appropriate when many or very large files have to be processed.
- *Scalability*. As a result of this one, Cocoon2 has been thought as a reusable component model based on the use of XML [Bray et al., 2000] and XSL [Deach, 1999] technologies. These components communicate with each other using SAX events. That means that every Cocoon's component receives XML and sends XML. These XML documents are processed via SAX in order to get efficient executions.

Figure 1 shows a scheme which represents a Cocoon pipeline for serving up a web page with static and dynamic content. The pipeline will process the XML file's content as follows: The component called *generator,* will transform file.xml into a SAX stream, and will send it to the component *SQL transformer*. The SQL transformer will be in charge of processing the embedded SQL sentences and it will replace them in the SAX stream for its result. Afterwards, the *transformer* will apply the style defined in stylesheet.xsl. Finally, the *serializer* will finish the stream and it will return the content in HTTP format to the end user.

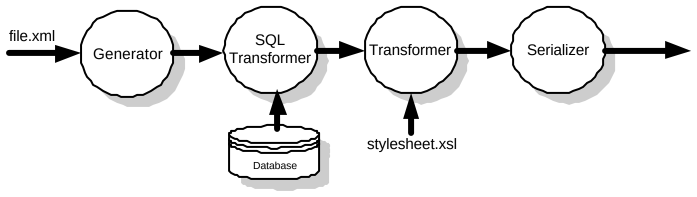


Figure 1. Cocoon pipeline for serving up a web page with static and dynamic content.

3. ASPECT ORIENTED PROGRAMMING

One of the most widespread emerging technologies in the last few years is known as aspect oriented programming or advanced separation of concerns. This approach is based on the 'divide and conquer' principle in the sense that its main goal is to obtain a better separation of concerns. This principle has been commonly applied in software engineering. The aspect-oriented programming (AOP) community [Elrad et al., 2001] states that it is possible to make programs better, if we specify separately the different concerns, properties or areas of interest of a system and the description of their relationships.

If we look at the tools developed to support the different proposals and the number of messages sent to mailing lists, AspectJ seems to be the most popular proposal. AspectJ [The AspectJ Team, 2002] is an aspect-oriented, general purpose extension to Java. It tries to capture the concerns that scatter all over the program code. These concepts are known as aspects, and to mix them with the code that implements the basic functionality they use a new construct, known as pointcut. Pointcuts are points where the code that implements the basic functionality can be augmented, either before, or after these specific points. This extra-code is wrapped into an aspect.

Concerning to web development, there have been some concerns that have been addressed and separated while deploying web applications [Kilesev, 2002], they are: security, design by contract, exception handling, logging, tracing, profiling, pooling or catching. We think that although a few concerns have been addressed to develop web applications, a more intense research work should be done to address concerns that are critical for web development, such as those related to performance, security or navigation [Reina & Torres, 2002], because they are likely to influence on the final success of the product.

4. ASPECTS VS COCOON

Having in mind the detection of the strengths and shortcomings of these to proposals, we have developed the same web application using both of them. On the one hand, we have implemented the application using the Cocoon framework, and, on the other hand, we have developed it using AspectJ and JSP technology (Java Server Pages) [Roth&Pelegrí-Llopart, 2003]. The choice of JSP has been imposed due to the use of AspecJ, which is based on Java technology. The main aim of the application is the management of a little news service where users post messages grouped by subject. Access to the server requires a user registration by means of a login name and a password. Once the user is registered, he should choose the subjects he is interested in. Thus, when the user wants to read the news, he only will browse the ones related to the subjects he is interested in.

Firstly, we are going to make a general comparison. After that, we are going to focus on two of the most interesting concerns addressed in AspectJ, and we will try to locate them in the Cocoon implementation.

If we look at the characteristics of the products we have obtained, we have to point out that while the AspectJ version has 40 files with source code, the Cocoon version only has 21. Attending the number of code lines written, we can realize that the Aspect J version has 2005 lines, and the Cocoon one has only 888. The size of AspectJ source code doubles the Cocoon one.

4.1 Authentication

Cocoon considers that a very important point for building web applications is authentication. Thus, it addresses this concern via a module. The *authentication handler* is an object that controls the access to the resources. Each resource can be related to one authentication handler, and one authentication handler manages the right accesses to all the resources that are related to it.

The authentication framework controls the authentication policy by means of actions: the *auth-protect* action, the *auth-login action*, the *auth-logout* action and the *auth-loggedIn* action. The *auth-login* and the *auth-logout* action control the authentication whereas the *auth-loggedIn* action controls the application flow.

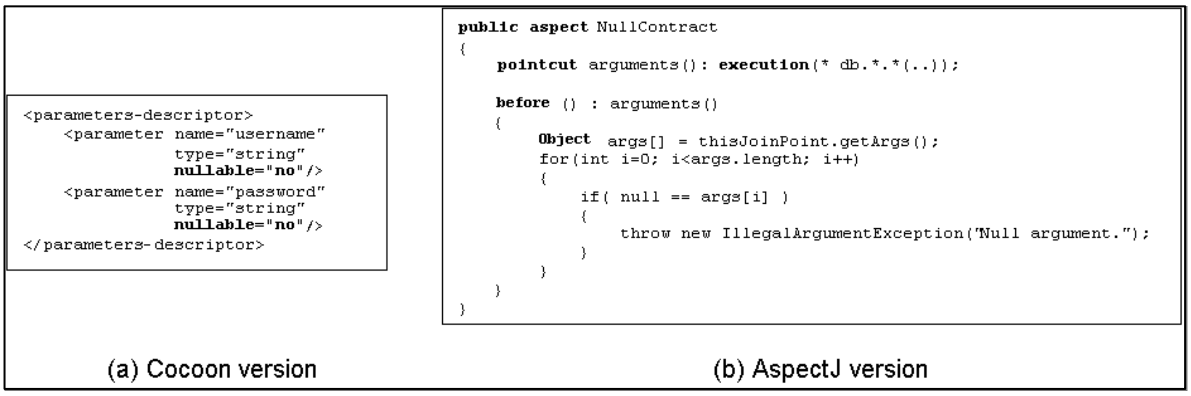
Thus, the Cocoon version allows us to define the authentication policy, but however, it is not localized on a concrete place. It is spread throughout a few XML files.

We can implement authentication in AspectJ by means of two pointcuts. On the one hand, although the AspectJ definition of authentication aspect is perfectly localized, it is less understandable than the Cocoon one, because Cocoon uses XML, what it means a higher abstraction level. One the other hand, AspectJ authentication is better adaptable to changes, especially if we want to add characteristics that haven't been taken into account in the Cocoon version.

4.2 Design by contract

This concern tries to validate some conditions that have to be fulfilled by some methods. In our case, we have tried to prove that the database accesses are made with not null parameters. Cocoon uses a *form-validator*, where describes the characteristics of the parameters of the form. Figure 2(a) shows how the property nullable expresses the parameter can't be null.

On the other hand, if we use AspectJ, we only have to point out those database method calls and check if all the arguments of those calls are null or not. Figure 2(b) shows the implementation of NullContract aspect. By means of the constructor *pointcut*, you can specify certain points in the code (in this case, those where a method related to a database operation is called), and then, using an *advice*, another constructor, you write the logic you want to execute, in this case, before the *pointcut*. Again, looking at the two implementations, we can be aware that AspectJ implementation introduces a better concern localization, but the Cocoon one is much clearer.



```
<parameters-descriptor>
    <parameter name="username"
               type="string"
               nullable="no"/>
    <parameter name="password"
               type="string"
               nullable="no"/>
</parameters-descriptor>
```

```
public aspect NullContract
{
    pointcut arguments(): execution(* db.*.*(..));

    before () : arguments()
    {
        Object args[] = thisJoinPoint.getArgs();
        for(int i=0; i<args.length; i++)
        {
            if( null == args[i] )
            {
                throw new IllegalArgumentException("Null argument.");
            }
        }
    }
}
```

Figure 2. Requiring the existence of not null parameters

5. CONCLUSION

We have introduced two emerging technologies, on the one hand, Cocoon, dedicated specifically to produce web applications, and, on the other hand, aspect-oriented programming, that is more general and a new way to face the application development. We think that the application of the advanced separation of concerns' principles can be very valuable for web development, because it can help us reduce the complexity of applications, and, at the same time, to improve reusability and maintenance. But, we have to compare aspect-oriented web development with other web development techniques. We can take advantages of this comparison in two different ways, firstly, to obtain the strengths and drawbacks of both technologies, and, secondly, because the non aspect-oriented technology can help us identify new concerns.

Moreover, we have developed a web application using both technologies. Thus, we have realized that many of the concerns separated using AspectJ have come into the focus of the Cocoon designers. If we compare both techniques, we can conclude that using AspectJ we can localize better the concerns, whereas using Cocoon, they are better understood, because they are expressed in a higher level language. We also need more lines of source code to describe a concept using AspectJ than using Cocoon. On the other hand, it is easier the addiction of new concerns to the AspectJ version. Thus, we think it is a good idea to mix both proposals and take advantage of both of them.

According to our future lines of research, we are working on the separation of other concerns such as navigation. In addiction, we also want to try other advanced separation of concerns strategies, such as HyperJ [The HyperJ Team, 2000].

REFERENCES

Deach, S, 1999. *Extensible Stylesheet Language (XSL) Specification.* W3C Recommendation.

Elrad, T. et al., 2001. Aspect-Oriented Programming. *Communications of the ACM*, Vol. 44 No. 10, pp. 29-31.

Kilesev, I., 2002. *Aspect-Oriented Programming with AspectJ.* Sams Publishing.

Mazzocchi, S, 2002. *Introducing Cocoon 2.* Electronic Publisher: Xml.com.

Reina, A. M., Torres, J, 2002. Separating the Navigational Aspect. *In proceedings of the workshop of Aspect-Oriented Programming for Distributed Computing Systems.* Vienna, Austria. IEEE Press.

Roth, M. and Pelegrí-Llopart, E., 2003. *Java Server Pages Specification. Version 2.0.* Sun Microsystems.

The Apache Project, 2002. *The Apache Cocoon Web Page.* http://xml.apache.org/cocoon/.

The AspectJ Team, 2003. *The AspectJTM Programming Guide.* http://www.eclipse.org/aspectj/

The HyperJ Team, 2000. *The HyperJ Web Page.* http://www.alphaworks.ibm.com/tech/hyperj

The SAX Project, 2002. *The Sax Project Web Page.* http://www.saxproject.org