

Aspect-Oriented Web Development vs. Non Aspect-Oriented Web Development

A. M. Reina¹, J. Torres¹, and M. Toro¹

Languages and Systems Department,
University of Seville,
Avda. Reina Mercedes s/n,
41012 Seville, Spain
{reinaqu, jtorres, mtoro}@lsi.us.es

Abstract. Aspect oriented programming is a way of reducing the complexity of software development, in the sense that it is easier to reason about isolated concepts. The complexity of web applications has been increasing in the last few years, mainly, because customers are requesting more and more. We think that advanced separation of concerns can be a good strategy to reduce the complexity of web applications. In order to pool the drawbacks and strengths of aspect oriented technology for web development, we have developed the same application, on the one hand, using AspectJ and Java Server Pages (JSP), and, on the other hand, using Cocoon, a web framework based on XML for publishing web pages. While developing the application, a few concerns brought up, but we are going to focus this paper on two of them, authentication and design by contract, because they help us give a clearer explanation of our results.

1 Introduction

Aspect oriented programming can help us reduce the complexity of applications, in the sense that it is easier to reason about isolated, separated concerns. On the other hand, the number of web applications has been remarkably increasing because the internet has become accessible to the mass audience, and, also, companies see the net as a gateway to new markets and making money. This situation has caused customers to ask for more and more requirements when they demand web products, and, as a consequence of this, the complexity of these applications has been growing gradually. Hence, we think that web development can be improved using aspect-oriented proposals.

Having in mind the previous idea, we think that a good way to know the strengths and drawbacks of aspect oriented web development is developing a web application, on the one hand, using aspect-oriented technology, and, on the other hand, using another web development framework. After that, we could compare the results of both processes. In addition to this, we want to survey the different concerns that should be addressed while developing software for web environment.

AspectJ [3] seems to be the most popular AOP proposal, and this choice forced us to develop the aspect-oriented version using JSP (Java Server Pages) [5]. On the other hand, one of the clearer requirements we asked for the non aspect-oriented technology was that we should be allowed to separate presentation and content.

This requirement forced us to use some XML-based [1] technology, and, finally, we decided to develop the non aspectual application with Cocoon[6], a web framework publishing that is part of the Apache project. Because Cocoon let us describe content using XML, and specify presentation with XSL (Extensible Stylesheet Language)[2].

This paper is structured as follows: Section 2 introduces the Cocoon's philosophy of work; section 3 points out the different concerns that have been addressed in order to develop the

web application, and those which should be addressed; section 4 explains our experience using both technologies, and analyses the pros and cons of both of them. Finally, in section 5, we conclude the paper and point out our future lines of research.

2 Developing web applications

Current technologies for web development, such as ASP, JSP or PHP mix logic, data and presentation, because source code is intermingled with HTML. A first separation can be obtained by means of style sheets, but, however, logic is still intermingled with data. Therefore, we decided to use Cocoon2[6], an emerging technology which stresses the separation of logic, content and presentation.

Cocoon is part of the Apache XML Project and can be seen as a web publishing framework. The Cocoon project started in 1998, but it was designed when XML technology was very young and there was very little experience. Therefore, in 1999, a new release of Cocoon arose, known as Cocoon2. This new release was designed for execution speed efficiency, and memory efficiency, and due to this requirement the choice of SAX [9] processing model was made.

SAX is an API that let a programmer interpret an XML file. SAX is an alternative to using DOM (Document Object Model) to interpret XML. It's simpler than DOM and it is appropriate when many or very large files have to be processed.

Another distinguished quality of Cocoon is scalability, thus, it has been thought as a reusable component model. These components communicate with each other using XML documents. 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. Cocoon model divides the development of web content in three levels:

XML creation The XML file is created by the content owners.

XML processing The requested XML file is processed and the logic contained in its logic-sheet is applied.

XSL rendering The XML document is rendered by applying an XSL stylesheet to it and formatting it.

One of the main aims of Cocoon is the separation of content, style, logic and management based on XML technology. The processing of content is achieved by assembling components in a pipeline. Hence, you can define XML documents and transformations to be applied on them in order to generate a presentation format (HTML, PDF, SVG, ...). Logic can be applied to the XML files, so that the XML can be dynamic.

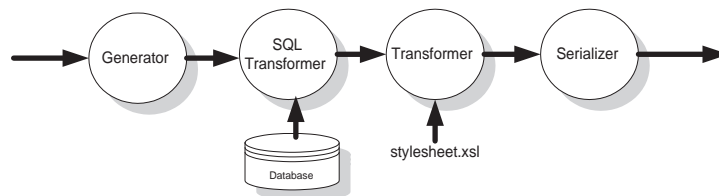


Fig. 1. Cocoon pipeline for serving up a web page with static and dynamic content.

Figure 1 depicts an example of a Cocoon pipeline suitable for serving up an HTML document obtained from an XML document with static content and dynamic content. This

pipeline has four components: a generator, an SQL transformer, a transformer and a serializer. The generator reads a file (file.xml) and turns it into an XML SAX stream. The second component, the SQL transformer, processes the SQL statements embedded in the original XML document and replaces with an XML result certain tree fragments. Afterwards, the XSL transformer accepts an XML stream and applies the style (stylesheet.xsl) to send out another XML stream. Finally, the serializer outputs the content in an HTML format.

3 Aspects and web development

Although aspect oriented programming is a very young area of research, from the very beginning there have been concerns, such as synchronization or distribution, that have had the attention of researchers due to their clear crosscutting nature. But there are other concerns that don't crosscut so clearly, and haven't had the focus of the aspect oriented community.

Following the proposal by Kilesev[3], we have addressed the following concerns:

Security. This concern is really authentication. Authentication is the process of determining whether someone is who is declared to be. This aspect tries to prevent that unauthenticated users have access to some web pages.

Design by Contract. A contract is something that should be guaranteed before calling a method on a class, but, also, the class should guarantee certain properties after the call. This is a way to check if certain conditions are fulfilled before executing a method. Some programming languages have implemented this concern using the notion of assertion. We have used this concern to be sure that methods related to the database are called without any null parameter.

Exception Handling. It is a simple way of applying an exception handling policy, in such a way that all exceptions should be handled by notifying the end user that something went wrong.

Logging. This concern encapsulates the logger behaviour. When certain points during the execution of a program are reached, a message is printed out.

Tracing. It is a debugging tool very similar to a logger, but it only tracks one type of event, a method execution.

Profiling. A debugging concern which measures the execution time consumed in some methods. This concern can be very helpful for detecting some bottlenecks.

Pooling. Pooling is a strategy to obtain faster database connections. When a database and all its associated files are closed, the connection and server resources are freed. If the same application needs the database services again, a new connection will have to be established and server resources will have to be asked for again, wasting resources, and, of course, slowing down the application. If we maintain a pool of connections and server resources, we will obtain faster database connections.

Caching. Caching is the retention of data, usually in the application, to minimize network traffic flow and/or disk accesses. If we catch database information on the application server, the database server can be relieved of its repetitive work.

Although all the concerns enumerated above have been addressed during the development of our web application, there are some of them that are crucial for the success of the final product. These key aspects are: pooling, caching and security.

On the one hand, pooling and caching are very important because they can have influence on response time, which is an important requirement, because a user can be bored waiting for a response. On the other hand, authentication is vitally important, because a web application can be a piece of a cake for an intruder, and needs to be protected. But also, we think

there are other key concepts, such as navigation[4], that should be addressed during web development. We also think that a more intense research work should be done to address key elements during web development; especially the ones related to performance, security, navigation and user interface.

4 Cocoon vs Aspects

Having in mind the comparison of both technologies, we have focused on the concerns we have addressed in AspectJ, and then we have tried to locate them on the Cocoon implementation. For space reasons, we aren't going to mention all the concerns enumerated in the previous section. Thus, we have chosen authentication and design by contract, because they are the most interesting ones to proceed to explain our results.

4.1 Authentication

Cocoon considers that a very important point for building a web application 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 access rights to all the resources that are related to it.

Figure 2 shows part of the authentication policy defined in our application. You can see how you need a username and a password to access to the main page. If the user isn't authenticated, then he will be redirected to the login page. This behaviour is defined by means of an action. This action is part of the `db-authenticator`, but also, although we have not depicted it in Figure 2, we need to give the XML file `auth.xml`, and another action, which is part of the `session-validator`, to describe the complete authentication policy.

```

<map:match pattern="do-login">
  <map:act type="form-validator">
    <map:parameter name="descriptor"
      value="context://mount/aspectNews/descriptors/params.xml"/>
    <map:parameter name="validate" value="username,password"/>
    <!-- If params are right, authenticate -->
    <map:act type="db-authenticator">
      <map:parameter name="descriptor"
        value="context://mount/aspectNews/descriptors/auth.xml"/>
      <!-- If authentication is right, go to the main page-->
      <map:redirect-to uri="index"/>
    </map:act>
    <!-- If user isn't authenticated....-->
    <map:redirect-to uri="login"/>
  </map:act>
  <!-- If params aren't right...-->
  <map:redirect-to uri="login"/>
</map:match>

```

Fig. 2. Part of the authentication policy

Figure 3 depicts an implementation of the authentication aspect in AspectJ. There are two pointcuts `startTag()` and `sessionParam()`. When the join points defined by the first pointcut are reached, an `around()` advice is executed, and, if the user has already been authenticated the operation will proceed. But if he isn't, then we have to check if the user had requested a page or he had just completed the login information in the login page. In

the first case, an exception will be thrown, and, in the second case, his credentials had to be checked.

The second pointcut is used for protecting any method that has anything to do with a session from an unauthorized user's access.

<pre> public aspect Authentication { declare parents: UserTag extends TagSupport; declare parents: StoriesTag extends BodyTagSupport; pointcut startTag(): call(int *.doStartTag()) && (target(TagSupport) target(BodyTagSupport)); pointcut sessionParam(HttpSession sess): call(* *.*(..,HttpSession, ..) && args(sess)); int around() throws JspException: startTag() { PageContext pc; Object o = thisJoinPoint.getTarget(); if (o instanceof TagSupport){ pc = ((TagSupport)o).getPageContext(); } else if(o instanceof BodyTagSupport){ pc = ((BodyTagSupport)o).getPageContext(); } else { throw new JspException("Unknown target class: "+ o.getClass().getName()); } } } </pre>	<pre> String user = (String)pc.getSession().getAttribute("user"); if (user == null) { ServletRequest req = pc.getRequest(); user = req.getParameter("user"); String pass = req.getParameter("pass"); if ((user == null) (pass == null)) { throw new NotAuthenticated(); } } try { if (!UsersDb.check(user, pass)) { pc.getServletConfig().getServletContext(). \\\ getRequestDispatcher("/login.jsp"). \\\ forward(req, pc.getResponse()); return javax.servlet.jsp.tagext.Tag.SKIP_BODY; } } catch(Exception e) { throw new JspException(e); } pc.getSession().setAttribute("user", user); } return proceed(); } before(HttpSession sess) throws JspException: sessionParam(sess) { if(sess.getAttribute("user") == null) { throw new NotAuthenticated(); } } } </pre>
--	--

Fig. 3. Authentication aspect

Although an authentication policy is defined in the Cocoon implementation, it is not localized in a concrete place. It is scattered throughout a few XML files. However, the AspectJ definition of the authentication aspect, is perfectly localized, but it seems much harder to understand than the definition of the Cocoon one, because with XML you can get a higher level of abstraction to express concerns.

4.2 Design by contract

The implementation of this concern using AspectJ has been reduced, and we only are going to worry about checking if the arguments of the methods related to the database are null or not. Figure 4 shows the implementation of the NullContract aspect. The aspect has a pointcut which picks an execution of all the database methods. If any null parameter is found, then an `IllegalArgumentException` is thrown.

On the other hand, Cocoon uses a form-validator, where the characteristics of the parameters of the form are described. Figure 5 shows a snippet of the form validator. The property nullable indicates that the parameter can not be null.

Again, although the Cocoon implementation is scattered (because we need to specify the nullable property for all the parameters that have to be checked), it is much clearer than the AspectJ one.

5 Conclusions and Further Work

We have developed the same web application using two emerging technologies, one based on components and XML, and the other one, using the aspect-oriented principles. If we

```

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.");
            }
        }
    }
}

```

Fig. 4. Design by contract in AspectJ

```

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

```

Fig. 5. Specifying the not null condition in Cocoon

compare both approaches, we will conclude that although the aspect oriented one has a better concern localization and encapsulation, the one based on XML is more expressive, because it can use a higher level of abstraction to express concerns. Therefore, it is a good idea to mixture both approaches, and this is one of our future lines of research.

In addition to this, most of the concerns addressed aren't key for web applications. We think that a more intense research work should be done to find and treat properly web concerns. Following this line, we are involved in the study of navigation.

If we focus on the concerns addressed in section 3, there are a few that are very important for the success of a web application. For example, security and concerns related to the response time are critical to the application. Thus, we think that a classification of concerns should be done, according to how important they are.

On the other hand, our first experience concerning to the separation of concerns has been with AspectJ, but we also want to try other proposals, such as HyperJ[8].

References

1. Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E.: Extensible Markup Language (XML) 1.0 (Second Edition). W3C Recommendation. October, 2000. <http://www.w3.org/TR/REC-xml>.
2. Deach, S.: Extensible Stylesheet Language (XSL) Specification. April, 1999. <http://www.w3.org/TR/WD-xsl>.
3. Kilesev, I.: Aspect-Oriented Programming with AspectJ. Sams Publishing, 2002.
4. Reina, A. M., Torres, J.: Separating the navigational aspect. In proceedings of the Workshop of Aspect-Oriented Programming for Distributed Computing Systems. Vienna, Austria. 2002.
5. Roth, M., Pelegr-Llopert, E.: Java Server Pages Specification. Version 2.0. January, 2003.
6. The Apache Project.: <http://xml.apache.org/cocoon/>.
7. The AspectJ Project: <http://www.eclipse.org/aspectj/>
8. The HyperJ home page:<http://www.alphaworks.ibm.com/tech/hyperj>
9. The SAX Project.: <http://www.saxproject.org>.