Processing XML using Java

Barry Cornelius
Computing Services, University of Oxford
Date: 1st May 2001
http://users.ox.ac.uk/~barry/papers/
mailto:barry.cornelius@oucs.ox.ac.uk

---

1 XML **1**
2 Some XML applications **3**
3 XSL Transformations **4**
4 Three approaches **4**
5 Using a standalone tool **5**
6 Using JavaServer pages **6**
7 Support in browsers **10**
8 Comparison of the three approaches **12**
9 Examples in the curriculum **12**

---

1 XML

1.1 What is XML?

Although originally HTML was a language in which much of the details of how a WWW page was displayed by a browser was left to the browser to decide, over the years, HTML has evolved so that the author has more and more control over the appearance of a page.

This means that a WWW page is an intermingling mix of text giving the data to be displayed together with other text describing how it is to be displayed.

```
<table border=1>
   <tr>
      <td><b>inkjet cartridges</b></td>
      <td><em>HP Deskjet print cartridge 51645A</em></td>
      <td bgcolor="yellow">19.50</td>
   </tr>
...
</table>
```

XML is another markup language. However, an XML document does not usually contain presentation details: instead, it is just used to describe data. And the tags of the language are chosen by the author of the document rather than being determined by the language. For this reason, it is called the *eXtensible Markup Language*:

```
<consumables>
   <product>
      <category>inkjet cartridges</category>
      <item>HP Deskjet print cartridge 51645A</item>
      <price>19.50</price>
   </product>
   ...
</consumables>
```

The WWW page:
http://www.dur.ac.uk/barry.cornelius/java/xml.processing/code/consumables.xml
contains a complete XML document. Elsewhere in this paper, a URL like this will be abbreviated to CODE/consumables.xml, where CODE is an abbreviation of:
http://www.dur.ac.uk/barry.cornelius/java/xml.processing/code

1.2 Some jargon

Here is some jargon:

```
<price>19.50</price>
```

is called an *element*. It is introduced by a *start tag*. In this example, the start tag is:

```
<price>
```

And it is terminated by an *end tag*:

```
</price>
```

A start tag may have one or more *attributes* as in:

```
<price units="pounds" country="UK">19.50</price>
```

An element may be *empty* as in:

```
<price amount="19.50"></price>
```

and this can be abbreviated to:

```
<price amount="19.50"/>
```

Another example is:

```
<unknown_price></unknown_price>
```

and this can be abbreviated to:

```
<unknown_price/>
```

1.3 The main goals of XML

As I see it, there are three main goals of XML:

1. To provide a language that just describes some data. If it is necessary to describe how this data is to be displayed (and in some situations this is not the case), then this is described in a separate document.
2. To allow the author to use their own tags. In this way, it is possible for an author to choose tags that are appropriate to the data.
3. To insist on the language being used in a correct manner rather than being tolerant, e.g.,
   - each start tag must have an end tag;
   - the values of attributes must be quoted.

1.4 Well-formed, DTDs and valid

Although WWW browsers are lenient about the quality of the HTML that they receive, the XML specification does not allow XML parsers to be lenient. Any document that is not *well-formed* is not allowed to be fixed: it must be reported as being in error.

An XML document may use a *document type definition* (*DTD*). This can either appear in the same file as the XML document or it can appear in a separate file. The DTD is a sequence of rules describing the markup language that is used by the XML document.

```
<!ELEMENT consumables (product*)>
<!ELEMENT product (category, item, price)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT item (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

An example of a DTD is in the file `CODE/consumables.dtd`.

An XML document that conforms to a DTD is said to be a *valid* document.

Even if an XML document uses a DTD, not all XML parsers actually check that the document conforms to the DTD.

1.5 XML schemas

More recently, *XML schemas* have been introduced. They can be used to define the structure of some XML (instead of using a DTD). Two of the main differences are:

1. An XML schema is written as an XML document (which means that there is less to learn and that it can be parsed by an XML parser).
2. When producing an XML schema, not only can you use predefined types, your schema can also introduce new types that help you to define the structure of the XML.

1.6 Resources about XML

1. Jon Bosak, ‘XML, Java and the future of the Web’,
   http://www.ibiblio.org/pub/sun-info/standards/xml/why/xmlapps.htm
2. Elliotte Rusty Harold, ‘XML Bible’,
   IDG Books, 1999, 0-7645-3236-7.
3. Charles Goldfarb and Paul Prescod, ‘The XML Handbook’,
   Prentice Hall, 2000, 0-13-014714-1.
4. Brian E. Travis, ‘XML and SOAP Programming for BizTalk Servers’,
   Microsoft Press, 2000, 0-7356-1126-2.
5. Some of the main WWW pages about XML are

   http://www.w3c.org/XML/,
   http://www.xml.com/, http://www.xml.org/, http://www.ibm.com/xml/,
   http://www.ibiblio.org/xml/, http://www.oasis-open.org/,
   http://www.oasis-open.org/cover/ and http://www.ucc.ie/xml/.

2 Some XML applications

As we have seen, XML allows you to produce a markup language to describe some data where you choose the tags to be used. Once you have chosen a set of tags, and perhaps have produced a DTD to describe the structure, you can then produce XML documents that use these tags. The act of producing a new markup language creates a new *XML application*.

Although we can all write our own XML applications, it becomes useful to establish agreed XML applications. Examples of XML applications that have been produced include:

1. CDF, Channel Definition Format,
   http://msdn.microsoft.com/workshop/delivery/cdf/reference/CDF.asp
2. CML, Chemical Markup Language,

   http://www.xml-cml.org/
3. ebXML, an XML for electronic business,

   http://www.ebxml.org/
4. GeoTech-XML, Geotechnical Engineering,

   http://geotech.civen.okstate.edu/Gml/
5. MathML, Mathematical Markup Language,

   http://www.w3.org/Math/
6. SMIL, Synchronized Multimedia Integration Language,

   http://www.w3.org/AudioVideo/
7. SVG, Scaleable Vector Graphics,

   http://www.w3.org/Graphics/SVG/
8. TexML,

   http://www.alphaworks.ibm.com/tech/texml
9. XGL,

   http://www.xglspec.org/
10. XHTML, a rewrite of HTML as strict XML,

    http://www.w3.org/MarkUp/

There are many more. One list of XML applications is given at:
http://www.xml.org/xmlorg_registry/.

3 XSL Transformations

3.1 XSL

Often XML documents are provided on the WWW so that they can be read by other people. Suppose some other person wants to read our data, but suppose that their XML reading program requires only part of the data, or it requires the XML to be written in a different way. What would be useful is a way of describing how one XML document can be transformed into another one.

This forms one of the roles of the *eXtensible Style Language* (*XSL*). So an XSL document can be used to describe how to transform an XML document into some other XML document. This is called an *XSL Transformation* (*XSLT*). As HTML is reasonably close to XML, an XSL document is often used to describe how to transform an XML document into HTML.

3.2 The `consumables` element

The XML document shown earlier contained data describing some products and how much they cost. The outermost element of this XML document is the `consumables` element. Suppose we now wish to produce HTML from this XML document.

Here is part of an XSL document that can do this:

```
<xsl:template match="consumables">
   <html>
   <body>
   <table>
      <xsl:apply-templates/>
   </table>
   </body>
   </html>
</xsl:template>
```

This is a rule that says: when you come across a `consumables` element, output:

```
<html><body><table>
```

then process the elements between the `consumables`'s start tag and its end tag; and then output:

```
</table></body></html>
```

You can see that the notation that is used by an XSL document is XML, and so XSL is yet another XML application.

3.3 The `product` element

The other rule that we need in our XSL document is one that copes with a `product` element. For this we can use:

```
<xsl:template match="product">
   <tr>
   <td><xsl:value-of select="category"/></td>
   <td><xsl:value-of select="item"/></td>
   <td><xsl:value-of select="price"/></td>
   </tr>
</xsl:template>
```

This extracts the characters embedded in the `category`, `item` and `price` elements of a `product` element and places them in the cells of the HTML table.

3.4 Other aspects of XSL

A complete XSL document is given at `CODE/consumables.xsl`.

There are many other aspects to the use of XSL to transform documents. These include the possibilities of:

1. re-ordering the processing of the elements of an XML document;
2. looping over elements;
3. deciding which elements to include;
4. sorting with respect to some element.

4 Three approaches

Three approaches for processing an XML document will now be considered.

4.1 Approach One: using a tool to process XML

We could write a program to process an XML document. The program could be used to produce HTML which we could then make available on the WWW. More details about this are given in Section 5.

The problem with using a tool is that we have to remember to run it everytime the XML document is updated. The remaining two approaches enable the user of a browser to work directly on the XML document.

4.2 Approach Two: processing XML with a JavaServer page

If a webserver supports the running of server-side programs, then a browser visiting a page could trigger a webserver to run a program that reads an XML document and produces HTML. Although this could be done using a CGI program or by running a PHP script, Section 6 looks at how a Java program could be run by a webserver.

4.3 Approach Three: get the browser to process the XML

Increasingly, browsers are providing support for the use of XML and XSL. So Section 7 looks at how they are supported by Microsoft's Internet Explorer and Mozilla.

5 Using a standalone tool

5.1 Creating a tool in the Java programming language

You could write your own program to process an XML document. There are two APIs that make these programs easy to write: they are the *SAX API* and the *DOM API*.

The SAX API allows you to walk through an XML document. It generates events for each part of a document. For example, it generates an event for each start tag, for each end tag, for the characters that appear between a start tag and an end tag, and so on. And in your Java program you can provide an object that handles all of these events.

The DOM API allows a Java program to build a data structure that represents an XML document. Your program can then wander around the data structure at will.

So, suppose we want to write a Java program to transform some XML into HTML. Although the DOM API could be used to do this, it makes more sense to use the SAX API as this task can be done by simply walking once through the XML document.

Although there are many software developers that provide SAX, the code that you write that uses the SAX API is almost the same no matter whose implementation you use. The only difference will be the code that you use to create an instance of the SAX parser. The right-hand side of the following statement supposes that we are using the Xerces parser from the Apache Software Foundation:

```
XMLReader tXMLReader = new org.apache.xerces.parsers.SAXParser();
```

We also need to create the handler that is to handle the events generated by the SAX parser:

```
Handler tHandler = new Handler();
```

This object then needs to be registered with the SAX parser:

```
tXMLReader.setContentHandler(tHandler);
```

Then the parser is let loose on the document:

```
FileReader tFileReader = new FileReader("consumables.xml");
InputSource tInputSource = new InputSource(tFileReader);
tXMLReader.parse(tInputSource);
```

It is the `Handler` class that is used to state what should happen when each event arises. It has to override the methods of the `DefaultHandler` class (which has methods that do nothing). For example, consider:

```
public class Handler extends DefaultHandler
{
   public void startElement(String pURI, String pLocalName,
                            String pQualifiedName, Attributes pAttributes)
   {
      System.out.println(pLocalName);
   }
}
```

This `Handler` class would cause the name of each element to be output when the start tag of the element is parsed.

The full code of a program that produces HTML from the `consumables.xml` file is given at CODE/ConvertConsumables.java and CODE/Handler.java.

5.2 Using a tool to create HTML from XML and XSL

Instead of sprinkling the HTML we want to generate all over a Java program, we could instead use the XSL stylesheet that was produced in Section 3.

Many software developers provide a tool to produce the document that results from applying the rules of an XSL stylesheet to an XML document.

For example, if you are using Xalan from the Apache Software Foundation, the following command line could be used to produce the file `consumables.html`:

```
java org.apache.xalan.xslt.Process -in consumables.xml -xsl consumables.xsl -out consumables.html
```

This command line assumes that the classpath has been set to include the files `xerces.jar` and `xalan.jar`.

5.3 Resources about SAX, DOM and XSLT

XML parsers (including the SAX and DOM APIs) and XSLT processors are available from:

1. http://xml.apache.org/xerces-j/ and http://xml.apache.org/xalan/
2. http://msdn.microsoft.com/xml/default.asp
3. http://users.iclway.co.uk/mhkay/saxon/
4. http://www.jclark.com/xml/xp/ and http://www.jclark.com/xml/xt.html

Some articles and books about SAX, DOM and XSLT include:

1. David Megginson, 'SAX2: Quick Start',
   http://www.megginson.com/SAX/Java/quick-start.html
2. Mazmul Idris, 'XML and Java Tutorial Part 1',
   http://www.developerlife.com/xmljavatutorial1/default.htm
3. Brett McLaughlin, 'Java and XML',
   O'Reilly, 2000, 0-596-00016-2.
4. Brett McLaughlin, 'All about JAXP: Sun's Java API for XML Parsing',
   http://www.ibm.com/software/developer/library/x-jaxp/index.html
5. Dave Pawson, 'XSL Frequently Asked Questions',
   http://www.dpawson.co.uk/xsl/xslfaq.html
6. Ken Holman, 'What is XSLT?',
   http://www.xml.com/pub/a/2000/08/holman/index.html
7. Paul Sandoz, 'FOP Slide Kit',
   http://www.sun.com/software/xml/developers/fop/
8. Doug Tidwell, 'Tutorial: Transforming XML documents [into HTML, SVG, and PDF]',
   http://www.ibm.com/xml/

6 Using JavaServer pages

6.1 Servlets

Instead of running a tool to transform an XML document into HTML, you could get a webserver to run a program that does the transformation. This can be done in many ways, including the use of a CGI script, a PHP script, a servlet using SAX, or by using a JavaServer page.

In Section 6.4, we will get the webserver to do the transformation by using a tag library in a JavaServer page, but first it is useful to have a detour introducing the ideas of servlets (Section 6.1), JavaServer pages (Section 6.2) and JavaBeans (Section 6.3).

A *servlet* is a WWW page that is written in Java. When a visitor's browser accesses the URL of a servlet, its bytecodes will be run. This will done either by the webserver itself or the webserver will get some other program to arrange for run the bytecodes. Rather than starting a new process to run the bytecodes, this program just runs them in a new thread.

The Java code of the servlet does some work and generates some HTML. The webserver gets the HTML from the servlet and passes that to the visitor's browser.

Suppose we want a WWW page that uses a WWW form to get some data from a visitor. Suppose we then want to run some Java source code that processes that data and generates some HTML which is to be fed back to the visitor.

Here are the HTML instructions of a WWW form:

```
<HTML>
   <BODY>
      <FORM METHOD="POST" ACTION=
         "http://altair.dur.ac.uk:8080/barry.cornelius/servlet/choice">
         Please enter your choice<BR>
         <INPUT TYPE="text" NAME="choice">
         <BR>
         <INPUT TYPE="submit" VALUE="Submit choice">
         <INPUT TYPE="reset">
      </FORM>
   </BODY>
</HTML>
```

and here is some Java source code:

```
import javax.servlet.http. HttpServlet;
import javax.servlet.http. HttpServletRequest;
import javax.servlet.http. HttpServletResponse;
import java.io.            IOException;
import java.io.            PrintWriter;
import javax.servlet.      ServletException;
public class choice extends HttpServlet
{
   public void doPost(final HttpServletRequest request,
                      final HttpServletResponse response)
         throws ServletException, IOException
   {
      final String tChoiceString = request.getParameter("choice");
      response.setContentType("text/html");
      final PrintWriter tResponsePrintWriter = response.getWriter();
      StringBuffer tStringBuffer = new StringBuffer();
      tStringBuffer.append("<html>\n" );
      tStringBuffer.append("<title>Reply</title>\n" );
      tStringBuffer.append("Your choice was: " + tChoiceString + "\n" );
      tStringBuffer.append("</html>\n" );
      tResponsePrintWriter.println(tStringBuffer);
      tResponsePrintWriter.close();
   }
}
```

The above texts are in the files `CODE/choice.html` and `CODE/choice.java`.

The above code illustrates one of the possibilities for the source code of a servlet: the class is derived from `javax.servlet.http.HttpServlet` and it overrides the `doPost` method of that class. When the `doPost` method is called, it is passed two arguments: the `request` argument enables access to the values that have been supplied on the WWW form, and the `response` argument provides a means of passing back HTML to the webserver.

In order for this to work, you need a webserver that understands servlets. One possiblity is to use tomcat, a program provided by the Apache Software Foundation, or to use Apache's webserver augmented by tomcat.

Although I have successfully configured Apache's webserver to use tomcat, I have chosen not to put this into active use as a badly written servlet could be cpu-bound and this would lead to poor performance of the computer running the webserver. Instead, at the University of Durham, we use tomcat on its own (i.e., without Apache's webserver) on a different computer, a multi-processor machine called `altair.dur.ac.uk`.

The above WWW form can be accessed by going to the URL:
`http://www.dur.ac.uk/barry.cornelius/tomcat/choice/choice.html`

We could now provide some Java source code that uses the SAX API to generate some HTML from an XML document. This code could be based on the code of `ConvertConsumables.java` and `Handler.java`, files that were mentioned in Section 5.1. In this way, we could provide a servlet that automatically generates HTML from an XML document.

6.2 JavaServer pages

The code of a servlet is a mix of Java source code doing some work and Java source code that generates HTML instructions. Because a lot of the code consists of Java statements generating HTML, the code is not easy to read.

A *JavaServer page* is a WWW document that is written in a mix of elements of a markup language (e.g., HTML) and elements that are dynamically generated by executing some Java code. The webserver will either itself transform, or instead arrange for another program to transform, the JavaServer page into Java source code (a *servlet*) which is then executed and its output is then passed to the webserver.

Suppose we want a WWW page that uses a WWW form to get a maximum price from a visitor. Suppose we then want to run some Java source code that queries a database generating some HTML giving all the products that are not greater than this price.

Here are the HTML instructions of a WWW form:

```
<HTML>
   <BODY>
```

```
      <FORM METHOD="POST" ACTION=
         "http://altair.dur.ac.uk:8080/barry.cornelius/prices/prices.jsp">
         What is your choice of price?<BR>
         <INPUT TYPE="text" NAME="testprice">
         <BR>
         <INPUT TYPE="submit" VALUE="Submit price">
         <INPUT TYPE="reset">
      </FORM>
   </BODY>
</HTML>
```

and here is the code of a JavaServer page:

```
<%@page language="java" import="java.sql.*" %>
<%
   String tMaxPriceString = "1000000000000.0";
   if (request.getParameter("testprice") != null)
   {
      tMaxPriceString = (String)request.getParameter("testprice");
   }
   final Driver tDriver =
         (Driver)Class.forName("org.gjt.mm.mysql.Driver").newInstance();
   final Connection tConnection = DriverManager.getConnection(
         "jdbc:mysql://www.dur.ac.uk/Pdcl0bjc_prices", "", "");
   final PreparedStatement tPreparedStatement =
         tConnection.prepareStatement(
               "SELECT * FROM consum WHERE price < "
               + tMaxPriceString + " ORDER by price");
   tPreparedStatement.setQueryTimeout(0);
   final ResultSet tResultSet = tPreparedStatement.executeQuery();
%>
<html>
<head>
<title>Access to the prices database</title>
</head>
<body bgcolor="#FFFFFF">
<table>
<%
   while (tResultSet.next())
   {
%>
<tr>
<td>
   <%= ((tResultSet.getObject("price")!=null)?tResultSet.getObject("price"):"") %>
</td>
<td>
   <%= ((tResultSet.getObject("goods")!=null)?tResultSet.getObject("goods"):"") %>
</tr>
<%
   }
%>
</table>
</body>
</html>
<%
   tResultSet.close();
   tConnection.close();
%>
```

The above texts are in the files `CODE/prices.html` and `CODE/prices.jsp`.

The above WWW form can be accessed by going to the URL:
`http://www.dur.ac.uk/barry.cornelius/tomcat/prices/prices.html`

A JSP is a page of HTML with Java source code inserted between

```
<%
```

and

```
%>
```

brackets. So the HTML is a lot easier to read if a JSP is used instead of a servlet.

6.3 Using a JavaBean from a JSP

One of key aspects of programming in an object-oriented programming language (such as Java) is that you create objects and apply methods to objects. An object whose class provides appropriate `get` and `set` methods is called a *JavaBean*. A JavaServer page can easily create a JavaBean and access its `get` and `set` by using the elements `jsp:useBean`, `jsp:setProperty` and `jsp:getProperty`.

Here is a JavaServer page that creates and then accesses a JavaBean called `tAdder`:

```
<html><body>
<p>start</p>

<jsp:useBean id="tAdder" scope="page" class="addition.Adder" />

<p>effectively this applies setFirst(27) to tAdder<p>
<jsp:setProperty name="tAdder" property="first" value="27" />

<p>effectively this applies setSecond(42) to tAdder<p>
<jsp:setProperty name="tAdder" property="second" value="42" />

<p> now apply getSum to tAdder </p>
<p>sum is <jsp:getProperty name="tAdder" property="sum" /> </p>

<p>effectively this applies setFirst(5) to tAdder<p>
<jsp:setProperty name="tAdder" property="first" value="5" />

<p> now apply getSum to tAdder again </p>
<p>sum is <jsp:getProperty name="tAdder" property="sum" /> </p>

<p>finish</p>
</body></html>
```

and here is the code of the `Adder` class:

```
package addition;
public class Adder
{
   private String iX = "0";
   private String iY = "0";
   public void setFirst(String pFirst)
   {
      iX = pFirst;
   }
   public void setSecond(String pSecond)
   {
      iY = pSecond;
   }
   public String getSecond()
   {
      return iY;
   }
   public String getSum()
   {
      int tResult = Integer.parseInt(iX) + Integer.parseInt(iY);
      return "" + tResult;
   }
}
```

The above texts are in the files `CODE/addition.jsp` and `CODE/Adder.java`.

The above JavaServer page can be executed by going to the URL:
`http://altair.dur.ac.uk:8080/barry.cornelius/addition/addition.jsp`

The use of JavaBeans helps to separate the HTML from the Java source code.

As well as the three tags used above, there are three other standard tags that are used in JSPs: `jsp:include`, `jsp:forward` and `jsp:plugin`.

6.4 Using a tag library in a JSP

Besides the six standard tags, in a JavaServer page we can also use tags from a *tag library*. Another project of the Apache Software Foundation has developed a number of tag libraries: this is the *jakarta-taglibs project*. Of interest to us is a tag library containing tags for processing XML input and applying XSL transformations.

A JavaServer page that uses the tags of the xsl tag library to read `consumables.xml`, apply the transformations described in `consumables.xsl` and generate the resulting HTML is:

```
<%@page language="java" %>
<%@taglib uri="http://jakarta.apache.org/taglibs/xsl-1.0"
          prefix="xsltlib" %>
<html>
<head>
<title>Consumables example</title>
</head>
<body>
<xsltlib:apply xml="/xml.processing/consumables.xml"
               xsl="/xml.processing/consumables.xsl" />
</body>
</html>
```

The above text is in the file `CODE/Consumables.jsp`.

The above JavaServer page can be executed by going to the URL:
`http://altair.dur.ac.uk:8080/barry.cornelius/xml.processing/jsp/Consumables.jsp`

In the above JSP, the tag prefix `xsltlib` is associated with the URI
`http://jakarta.apache.org/taglibs/xsl-1.0`
Three files are needed to get this to work with tomcat:

- It is necessary to associate `http://jakarta.apache.org/taglibs/xsl-1.0` with the location of a file defining the tag library. This can be done by putting the following into a file called `web.xml`:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
   "http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
   <taglib>
      <taglib-uri>http://jakarta.apache.org/taglibs/xsl-1.0</taglib-uri>
      <taglib-location>/WEB-INF/xsl.tld</taglib-location>
   </taglib>
</web-app>
```

- The file `xsl.tld` contains a definition of the tag library together with names of the Java classes (e.g., `org.apache.taglibs.xsl.ApplyTag`) that implement the tags. It contains lines like:

```
<tag>
   <name>apply</name>
   <tagclass>org.apache.taglibs.xsl.ApplyTag</tagclass>
   ...
   <attribute>
      <name>xml</name>
      <required>false</required>
      <rtexprvalue>true</rtexprvalue>
   </attribute>
   ...
</tag>
```

- The file `xsl.jar` contains the `.class` files of these classes.

6.5 Resources about JSPs

1. Sun's main page about JavaServer pages is at:
   http://java.sun.com/products/jsp/
2. Jakarta's main page about tomcat is at:
   http://jakarta.apache.org/tomcat/
3. 'JavaServer Pages Tag Libraries',
   http://java.sun.com/products/jsp/taglibraries.html
4. 'Developing XML Solutions with JavaServer Pages Technology',
   http://java.sun.com/products/jsp/html/JSPXML.html
5. Duane K. Fields and Mark A. Kolb, 'Web Development with JavaServer Pages',
   Manning Publications, 2000, 1-884777-99-6.

7 Support in browsers

As time moves on, WWW browsers play catch-up with new developments. We now look at the support for XML (and XSL) that is provided in recent versions of browsers: we look at version 5.5 of Internet Explorer (IE5.5) and Milestone 18 of Mozilla augmented by the code of Mozilla's XSLT project (M18XSLT). Milestone 18 was released in October 2000. Since the release of Mozilla 0.9.1 on 6th June 2001, the code of the XSLT project has formed part of Mozilla. The latest release of Mozilla can be downloaded from

`http://www.mozilla.org/releases/`.

7.1 Use of CSS1 with XML

Both IE5.5 and M18XSLT provide support for XML with *CSS1* (version 1 of *Cascading Style Sheet*). To use CSS, you need to augment an XML document with a `xml-stylesheet` processing instruction. This results in the text that is in the file

`CODE/css1.xml`:

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="css1.css"?>
<!DOCTYPE consumables SYSTEM "consumables.dtd">
<consumables>
...
</consumables>
```

The `type` attribute of this `xml-stylesheet` processing instruction says that this XML document is to be transformed using a CSS stylesheet and the `href` attribute gives the URL of the file containing the actual CSS instructions.

The file `css1.css` might contain:

```
consumables { display: block }
product     { display: block }
category    { font-size: x-large;
              color: black }
item        { font-size: large;
              color: red }
price       { background-color: yellow;
              color: blue }
```

7.2 Use of CSS2 with XML

CSS2 introduces some new aspects like the handling of tables. So instead we could use the file

`CODE/css2.xml`:

```
<?xml version="1.0" standalone="no"?>
<?xml-stylesheet type="text/css" href="css2.css"?>
<!DOCTYPE consumables SYSTEM "consumables.dtd">
<consumables>
...
</consumables>
```

where the file `css2.css` contains:

```
consumables { display: table }
product     { display: table-row }
category    { display: table-cell; font-size: x-large;
              color: black }
item        { display: table-cell; font-size: large;
              color: red; text-indent: 0.1in }
price       { display: table-cell; background-color: yellow;
              color: blue; text-align: right; text-indent: 0.1in }
```

Although these instructions are obeyed correctly by M18XSLT, they are not understood by IE5.5.

7.3 Use of XSL with XML

Both IE5.5 and M18XSLT can use XSL to process an XML document. You need to insert the following `xml-stylesheet` processing instruction in your XML document:

```
<?xml-stylesheet type="text/xsl" href="consumables.xsl"?>
```

Inconveniently, there are two important differences in the XSL that is understood by IE5.5 and M18XSLT:

1. IE5.5 requires a rule in the XSL stylesheet for processing the outermost level of an XML document. So IE5.5 needs the extra rule:

   ```
   <xsl:template match="/">
      <xsl:apply-templates/>
   </xsl:template>
   ```

2. IE5.5 requires the following element to identify the *namespace* being used:

   ```
   <xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
   ```

   whereas an M18XSLT XSL document requires:

   ```
   <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="html"/>
   ```

This is because IE5.5 has an old version of Microsoft's XML Parser (MSXML). The WWW page

`http://www.netcrucible.com/xslt/msxml-faq.htm`

gives details of how you can obtain Version 3.0 of MSXML (November 2000) and get IE5.5 to use it.

The files

`CODE/xsl.xml` and `CODE/consumables.xsl`

contain the texts that can be used by both M18XSLT and the modified version of IE5.5.

8 Comparison of the three approaches

Approach One uses a standalone tool to process an XML document. The resulting HTML document can then be put into an area that is visible from the WWW. The disadvantage of using Approach One is that you need to remember to run the tool each time the XML document is altered. However, this approach has the advantage that there are no prerequisites for either the webserver or the WWW browser.

Approach Two assumes that you can get your webserver to run a program to produce HTML from the XML document. Its advantage is that the HTML is always up-to-date as it is generated each time from the XML document. The first disadvantage of this approach is that you need a webserver that can run some appropriate program. Another disadvantage is that there may be some delay whilst the program is producing the HTML.

At the moment, Approach Three is the one that has the most problems. There are two main problems: firstly, if you want to provide the same XML and XSL documents for both Internet Explorer and Mozilla, an IE5.5 visitor will need to install a new MSXML and modify IE5.5 to use this. Secondly, the browsers that are used by most visitors are not modern browsers and so they are incapable of dealing with XML.

9 Examples in the curriculum

9.1 Computer Science projects during 1999-2000

Each year, second year students in the Department of Computer Science at the University of Durham engage in a Software Engineering Group project. During the various stages of the life cycle of the project, they produce many files that contain the documentation and code of the project. Unfortunately, these files are managed in an ad-hoc fashion.

During 1999-2000, two Computer Science students, Isabel Delacour and Michael Creasy, undertook final year projects that investigated how a single source XML document could be used to manage the documentation and code. They also looked at how this XML document could be viewed in different ways, or processed to extract a particular piece of source code or a particular test case. Although Isabel chose to utilise the XML support provided by a particular browser (Internet Explorer 5.0), Michael instead chose to process the XML document using servlets processed by an JServ servlet engine controlled by an Apache webserver.

Both Isabel and Michael were supervised by James McKinna, a Lecturer in the Department of Computer Science at the University of Durham.

9.2 Engineering projects during 2000-2001

During 2000-2001, two final year students in the School of Engineering are involved with developing parts of an XML application in the area of Geotechniques.

Alex Cubitt is investigating ways in which Geotechnical structures (such as dams, foundations, embankments and retaining walls) can be represented in XML. Besides designing a suitable markup for this area of Geotechniques, he will also be looking at displaying the data of such an XML document using Java's 2D/3D graphics.

Rob Shields is looking at representing information about boreholes. As well as designing suitable XML for this area, he will be looking at how to convert between an existing established format called AGS and his XML markup language (and back again). Like Alex, he will also be looking at ways of producing graphical output from the XML.

Both students are supervised by David Toll, a Senior Lecturer in the School of Engineering at the University of Durham. It is hoped that the work of these projects will contribute to the international Geotech-XML project.