Web-WAP with JSP and XML

Xiangdong Yang

A Project Report
Submitted to Committee Members:
Dr. W. Homer Carlisle (Chair)
Dr. David A. Umphress
Dr. Wen-Chen Hu

In Partial Fulfillment of the
Requirements for the
Degree of
Master in Software Engineering

Auburn, Alabama

May 2001
ABSTRACT

Web-WAP with JSP and XML

Xiangdong Yang

Master in Software Engineering, May 2001

Directed by Dr. W. Homer Carlisle

In order for web-based services to have the broadest use it has become increasingly important that they be accessible from the widest variety of clients. Recent years have witnessed a proliferation of different types of user-oriented clients for web-based applications: PDAs, WAP-enabled mobile phones, etc.

This project mainly focuses how to generate both web pages and WAP pages from the same content on a web site. JavaServer Pages (JSP) and XML are selected as implementation technologies for the Web-WAP applications. JSP and XML are natural partners for developing Web applications that use heterogeneous data sources and support multilingual clients.

There are two approaches utilized in this project to fulfill the purpose. In the multiple pipeline approach, Different JSP pages are used to generate client-specific outputs, either in html or wml, from the same content. Another approach combines the single and multiple pipeline methods in the same servlet container, JSP is integrated with XML technology to generate client-specific markup languages by applying transformations to incoming dynamic XML data.

The implementation environment for JSP is Tomcat, and XML is Cocoon. A n-tier client/server architecture is implemented involving JSP, XML, XSLT, XSP, ESQL, Java, JDBC, MySQL, HTML, WML languages.
# TABLE OF CONTENTS

LIST OF FIGURES .................................................................................. IV

1. **Introduction** .................................................................................. 1

2. **Overview of technologies** .............................................................. 2
   2.1. **JSP technology** ....................................................................... 2
       2.1.1. **Basics of JSP** ................................................................. 2
       2.1.2. **Advantages of JSP** ......................................................... 6
   2.2 **XML technology** ...................................................................... 8
       2.2.1. **Introduction to XML** ..................................................... 8
       2.2.2. **XML and JSP** .............................................................. 9
   2.3. **WAP and WML** ................................................................... 10

3. **Application environment, architecture and implementation** ........ 12
   3.1. **Application environments** ..................................................... 12
       3.1.1. **Tomcat as JSP application environment** ....................... 12
       3.1.2. **Cocoon as XML application environment** ................. 14
       3.1.3. **WAP development environment** ................................ 15
   3.2. **Architecture and implementation** ......................................... 15
       3.2.1. **Overall architecture** .................................................... 15
       3.2.2. **Multiple pipeline approach** ....................................... 16
       3.2.3. **A different combination approach** .............................. 24

4. **Discussion and conclusion** ......................................................... 48

Reference .............................................................................................. 50
LIST OF FIGURES

Figure 1 JSP process architecture .................................................. 7
Figure 2 WAP request-response exchange diagram .............................. .. 11
Figure 3 Tomcat structure ..................................................................13
Figure 4 Overall application logic ......................................................... 16
Figure 5 Application logic of the multiple pipeline approach ................. 17
Figure 6 the architecture for the coffee application .............................. 25
Figure 7 Application logic of the combination approach ...................... 26
1. Introduction

In order for web-based services to have the broadest use, it has become increasingly important that they can be accessible from the widest variety of clients.

Recent years have witnessed a proliferation of different types of user-oriented clients for web-based applications: PDAs, WAP-enabled mobile phones, and voice clients. Currently, the number of PDA and mobile phone users in the world is exploding. Worldwide, there are about 380 million mobile phone users, compared to 200 million PC. By 2003, mobile phone users are expected to reach at least one billion. The emergence of a new technology enables all these mobile phone users to access the Internet. This technology is WAP.

Web pages today are often written in html (hypertext markup language) that is rather too complex for mobile phones with their slower operating speeds. Mobile phones will need to access pages that are written in WML (wireless markup language). WML greatly simplifies download times and presentation. Thus, an existing web site that needs to go to wireless world can put the output in the wml format.

The main focus of this project is to how to generate both web pages and WAP pages from the same content on a web site. But, finding the correct technologies for this purpose is definitely not a trivial task. JavaServer pages (JSP) and XML are selected and discussed as a way to fulfill the purpose of this project.

There are some drawbacks in conventional web site design, with CGI for instance, there are problems in scalability, problems in integration of backend data and business logic, and problems in manageability and personalization[1]. JSP is the Java-based
technology for generating dynamic Web pages with cross-platform and cross-Web-server support. This makes creating dynamic HTML and XML Web pages even simpler, while offering all of the benefits of the Java platform.

XML is a markup language for documents containing structured information. When XML is combined with XSLT (eXtensible Stylesheet Language Transformation) technology it allows one to generate pure data content, and have that content uniformly styled either at predetermined times, or dynamically at runtime. Both of JSP and XML are open source software. JSP and XML are chosen for study and we will overview these technologies in the following chapter.

2. Overview of technologies

2.1 JSP technology

2.1.1 Basics of JSP

As mentioned, there are some drawbacks in conventional website development. An easier, platform- and server- independent approach is needed to build high-performance, dynamic web applications. The JavaServer Pages (JSP) technology is the answer from JavaSoft. JSP is a major extension of Java Servlet technology that makes building and maintaining dynamic pages much easier, and supports the distributed development model.

So what are JavaServer Pages? They combine markup language (HTML, WML or XML) with the advantages of Java to produce a dynamic web page. Each page is automatically compiled to a servlet by the JSP engine. The first time it is requested, and then executed. JSP provides a variety of ways to talk to Java classes, servlets, applets and
the web server. With it, you can split the functionality of your web applications into components with well-defined public interfaces glued together by a simple page.

A JSP page is executed by a JSP engine or container, which is installed on a web server, or on an application server. When a client asks for a JSP resource the engine wraps up that request and delivers it to the JSP process along with a response object. The JSP processes the request and modifies the response object to incorporate the communication with the client. The engine (container) then wraps up the response from the JSP page and delivers it to the client. The underlying layer for a JSP is the servlet implementation.

JSP technology is truly an extension of servlet technology. However, JSP pages make servlets much easier to write and maintain by changing the approach to creating servlets. Following is a very simple JSP called hello.jsp with a page directive.

```html
<! -- following are the directives -- >
<%@ page language = "java" import = "java.util.*" session = "true" info = "my first page" errorPage = "error.jsp" %>
<html><body>
<!-- following are the scripting elements -- >
<%
for (int i = 0; i <= 2; ++i) {
Hello World<br>
%
%
</body></html>
```

As shown from the hello.jsp example, the JSP syntax is fairly simple and straightforward, and can be classified into directives, scripting elements, and standard actions. JSP directives are messages for the JSP engine, which do not directly produce any visible output, but tell the engine what to do with the rest of the JSP page. JSP directives are always enclosed within the `<%@ ... %>` tag. Scripting elements are used to include scripting code (usually Java code) within the JSP, which allow you to
declare variables and methods, include arbitrary code, and evaluate an expression. The scripting element is enclosed between `<% %>` [1]. The output of this JSP is shown below:

![Image of JSP output]

The `hello.jsp` is simple, but, the java source file that translated by compilation from the hello.jsp code is not simple, which looks like this:

```java
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext..HiddenTag;
import java.io.PrintWriter;
import java.io.IOException;
import java.io.FileOutputStream;
import java.io.FileInputStream;
import java.util.Vector;
import org.apache.jasper.runtime.*;
import java.beans.*;
import org.apache.jasper.JasperException;
import java.util.*;

public class _0002fhello_0002ejshello.jsp extends HttpServlet {
    public String getServletInfo() {
        return "my first page";
    }

    static {
```
public _0002fhello_0002ejsphello_jsp_4() {
}

private static boolean _jspx_inited = false;

public final void _jspx_init() throws JasperException {
    
    public void _jspService(HttpServletRequest request,
    HttpServletResponse response)
    throws IOException, ServletException {
        JspFactory _jspxFactory = null;
        PageContext pageContext = null;
        HttpSession session = null;
        ServletContext application = null;
        ServletConfig config = null;
        JspWriter out = null;
        object page = this;
        String _jspx_page_context = null;
        try {
            if (_jspx_inited == false) {
                _jspx_init();
                _jspx_inited = true;
            }
            _jspxFactory = JspFactory.getDefaultFactory();
            response.setContentType("text/html;charset=8859_1");
            pageContext = _jspxFactory.getPageContext(this, request, response,
            "error.jsp", true, 8192, true);
            application = pageContext.getServletContext();
            config = pageContext.getServletConfig();
            session = pageContext.getSession();
            out = pageContext.getOut();
            
            out.write("\r\n<html><\r\n<body><\r\n";
            for ( int i = 0; i <=9; ++i) {
                out.write("\r\nHello World <br>\r\n");
            }
            out.write("\r\n</body>\r\n</html>");
        } catch (Exception ex) {
            if (out.getBufferSize() != 0)
                out.clearBuffer();
            pageContext.handleErrorPageException(ex);
        } finally {
            out.flush();
            _jspxFactory.releasePageContext(pageContext);
        }
    }
}
The translated java code shown above is generated by Tomcat, which is the combined JSP1.1 and Servlets 2.2 reference implementation being developed by the Apache organization. Judged from above-mentioned sample, JSP, and the JSP syntax is fairly simple and straightforward, and, JSP will simplify web development by leaving programming tasks which requiring significant developer expertise to servlet engine. Following are the advantages to using JSP.

2.1.2 Advantages of JSP

Write Once Run Anywhere™ (JavaSoft): JSP is set to be one of most important part of the Java 2 Platform Enterprise Edition, which is portable across Windows, Linux, UNIX and MacOS and compatible with a wide range of Web Servers (IIS, Apache, Netscape Enterprise Server) and Application Servers from Sun, IBM and others. JSP technology brings the "Write Once, Run Anywhere" paradigm to interactive Web pages. JSP pages can be moved easily across platforms, and across web servers, without changes.

"Dynamic content can be served in a variety of formats: There is nothing that mandates the static template data within a JSP page to be of a certain format. Consequently, JSP can service a diverse clientele ranging from conventional browsers using HTML/DHTML, to handheld wireless devices like mobile phones and PDAs using WML or other B2B (business-to-business) applications using XML." (quoting from [2])

Separating content generation from presentation: Using JSP technology, web page developers use HTML or XML tags to design and format the results page. They use JSP tags or scriptlets to generate the dynamic content on the page (the content that
changes according to the request). The logic that generates the content is encapsulated in tags and JavaBeans components and tied together in scriptlets, all of which are executed on the server side. Figure 1 [2] illustrates this approach. A page connects to a data source through a server-side object, transforms the information into data abstractions, and then renders the data using JSP elements. On the server, a JSP engine interprets JSP tags and scriptlets, generates the content required (for example, by accessing JavaBeans components, accessing a database with JDBC technology, or including files), and sends the results back in the form of an HTML (or XML) page to the browser. This helps authors protects the own proprietary code while ensuring complete portability for any HTML-based web browser.

Figure 1. JSP process architecture

"Separation of static from dynamic content: With servlets, the logic for generation of the dynamic content is an intrinsic part of the servlet itself, and is closely tied to the static presentation templates responsible for the user interface. Thus, even minor changes made to the user interface typically result in the recompilation of the
servlet. This tight coupling of presentation and content results in brittle, inflexible applications. However, with JSP, the logic to generate the dynamic content is kept separate from the static presentation templates by encapsulating it within external JavaBeans components. These are then created and used by the JSP page using special tags and scriptlets. When a page designer makes any changes to the presentation template, the JSP page is automatically recompiled and reloaded into the web server by the JSP engine.” (quoting from [2])

2.2 XML technology

2.2.1 Introduction to XML

XML is a markup language for documents containing structured information. A markup language is a mechanism to identify structures in a document [5]. The XML specification defines a standard way to add markup to documents. XML is different from HTML. In HTML, both the tag semantics and the tag set are fixed. The World Wide Web Consortium (W3C), in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount. For people who want to disseminate information widely, features supported by only the latest releases of Netscape and Internet Explorer are not useful.

XML specifies neither semantics nor a tag set. It provides a facility to define tags and the structural relationships between them. Since there’s no predefined tag set, there
can't be any preconceived semantics. The semantics of an XML document could be defined by the applications that process them.

XML alone is useless without some defined semantics: even if an application is able to parse a document, it must be able to understand what the markup means. This is why XML-only browsers are meaningless and are no more useful than text editors. This is one of the reasons why XSL (the eXtensible Stylesheet Language) was proposed and designed. XSL is divided into two parts: transformation (XSLT) and formatting objects (sometimes referred to as FO, XSL:FO, or simply XSL). Both are XML DTDs (Document Type Definition which describe the syntax of a language implemented in XML) that define a particular XML syntax. Every XSL or XSLT document is a well-formed XML document[4].

2.2.2 XML and JSP

By now nearly everyone in the software industry is aware of the advantages of XML and Java. Web developers have come to conclusion that XML and Java are the perfect pair because they complement each other. XML provides the platform-independent data — portable documents and data. Java contributes the platform-independent process with portable object oriented software solutions [3].

As part of the general-purpose Java software platform, JSP goes further to provide a number of capabilities that are ideally suited for working with XML. JSP pages can contain any type of text-based data, so it is straightforward to generate documents that contain XML markup. Also, JSP pages provide an abstraction mechanism to encapsulate functionality for ease of use within a JSP page.
JSP can consume XML data. It is easy to use multiple data sources, including XML sources, in a JSP page. XML can be converted to server-side objects and extract object properties using JSP. Currently you have to write code, using DOM or SAX and encapsulate it into a custom tag or a JavaBeans component to create these objects.

JSP can generate XML-based markup languages for various types of web-based clients. Using JSP to generate XML content is as easy as generating HTML content. A set of client-specific JSP pages is used to generate client-specific markup output. Each type of client requires a different JSP output page. Executing a JSP page to generate markup is much more efficient than performing a XSLT transformation to generate the same markup.

JavaServer Pages technology and XML are natural partners for developing Web applications that use heterogenous data sources and support multilingual clients. As part of the general-purpose Java software platform, JSP technology provides developers with the capability to adopt architecture for XML processing that best satisfies their particular application requirements.

2.3 WAP and WML

The number of mobile phone users in the world is exploding everyday. WAP stands for the Wireless Application Protocol. In fact, WAP is the technology that makes it possible to link wireless devices (such as mobile phones, PDA etc.) to the Internet by translating Internet information so it can be displayed on the display screen of a mobile telephone or other portable device. In order to translate back and forth between the Web content and its binary encoding for WAP clients, we need a layer of software that would
sit between the Web server and the wireless Web. That layer of software is called a WAP Gateway. The WAP request-response exchange diagram is shown in Figure 2 below:

![WAP request-response exchange diagram](image)

Figure 2. WAP request-response exchange diagram [1]

If you want to serve up simple WAP content, you can use an existing web server. A mobile will connect to your server through a WAP gateway located in their mobile operator's network.

The WAP Protocols cover both the application level (WML and WML Script; collectively known as WAE), and the underlying transport layers (WSP and WTP, WTLS and WDP). WML is an XML language, just like XHTML or MathML or SVG. This means that WML has a fixed set of tags and associated attributes that are defined in the WML Document Type Definition (DTD). The DTD specifies the rules of WML, you don’t have to know how to read the DTD in order to learn those rules, but you have to learn the rules in order to create WML documents.

In order to connect mobile phones to the internet, the mobile phone we are using as our client must be WAP-enabled, and the website one is viewing from the internet must be in WAP-format (this means that the web pages must be written in WML), in addition to the WAP gateway. Actually, there are quite a few conversion programs that translate an existing website into a WAP-format on the market today. But, many people feel that creating such conversion program are problematic, and that creating WAP pages
from web content directly for mobile devices is the way to go. Since that the content and
display of the content on mobile phone will be compromised from that a PC. Knowing
what and how to compromise this information may be something that is beyond the scope
of conversion software.

3. Application environment, architecture and
implementation

3.1 Application environments

3.1.1 Tomcat as JSP application environment.

Tomcat, the only official Reference Implementation for the Java Servlet 2.2 and
JavaServer Pages 1.1 Technologies, is selected here as JSP application environment.
Tomcat is the flagship product of Apache Software Foundation. Binary downloads of the
Tomcat server are available from the http://jakarta.apache.org/downloads/bin/index.html
page. Tomcat can be served as a stand-alone container, or as a server add-on. It is
recommended that real world sites use a web server, such as Apache, for serving the
static content of the site, and use Tomcat as a Servlet/JSP add-on. In order to have them
work together, basically, we need to, first, set up the Tomcat, modify web server’s
httpd.conf file, then install a web server adapter, at last, modify tomcat’s server.xml file.

The tomcat structure is shown in Figure 3. You can see from the figure:
• bin is used for containing startup/shutdown scripts

• conf is used for containing various configuration files including server.xml and web.xml.

• doc is used for containing miscellaneous documents regarding Tomcat

• lib contains various jar files that are used by Tomcat

• logs is where Tomcat places its log files

Figure 3. Tomcat structure

• src contains servlet APIs source files

• webapps contains web applications

Problems when setting up Tomcat at our Apache-powered web server (interact.cse.eng.auburn.edu) were in the way Tomcat invokes the java interpreter (kaffe). When our system administrator tried to run the Hello World example servlet, Apache invokes Tomcat to start the servlet (via mod_jk, using the AJP12 protocol which the Tomcat process is listening for on socket 8007), But Tomcat dies with the following error:

```
[root@interact bin]# HANDLER THREAD PROBLEM: java.net.SocketException: Invalid argument
java.net.SocketException: Invalid argument
```
An alternative implementation environment is a commercial web hosting site which support Tomcat. MediaHost (http://dev.startcom.org/index.html) is such a good choice, which is housed on a Intel i686 with a RedHat 7.0 Platform with JServ-1.1.2, JDK-1.2.2, JSDK2.0, JSSI-1.1.2, JSP (gnuJSP-1.0)/Tomcat 3.2, and COCOON-1.8.2. Cocoon will be used as XML implementation environment for this project, which will be discussed in detail.

3.1.2 Cocoon as XML application environment

Cocoon, from the Apache XML project, was chose as the XML application environment for the project. As an XML publishing framework based completely in Java, Cocoon allows any conformant XML parser to be used. In addition, Cocoon is based on the immensely popular Java servlet architecture. The cocoon framework is built to operate at an engine level rather than as another servlet in your engine.

Cocoon includes a number of processors. They are [5]:

- The XSLT processor – Applies XSLT transformations to the input document. XSLT allows you to solve your transformation needs as well as simple tag evaluation/processing due to its extensible and programmable nature. XSLT is a W3C Recommendation.
• The XSP processor – This processor allows you include programmatic logic into your pages as well as to separate the logic from the content.

• The SQL processor – Evaluates simple tags describing SQL queries to JDBC drivers and formats their result-set in XML depending on given parameters.

3.1.3 WAP development environment

Deploying WAP application within Tomcat is very easy; we only need to make one addition to the Jakarta-tomcat\conf\web.xml file, and Add WML, WMLScript and WBMP as new MIME types in web.xml file.

Cocoon is already WAP-applicable. one of the real powers of Cocoon lies in the use of wireless devices. Cocoon is able to understand which browser is requesting the page and applies a different stylesheet to the same XML page to render on the different clients. We only need to build a stylesheet for WAP device. The WML is used here.

UP.SDK [6], a software emulation of a wireless device that allows testing of WML pages, is used to test the WAP applications in this project. UP.SDK can be downloaded from http://updev.phone.com for free.

3.2 Architecture and Implementation

3.2.1 Overall architecture

The main focus of this project is to generate both web pages and WAP pages from the same content of a web site. Basically, there are two approaches applied here to implement the Web-WAP application. The first approach is a multiple pipeline approach, which mainly focus on JSP technology excusively. Another approach is to use JSP with
the web-publishing frame: Cocoon. JSP integrated with XML, XSLT, XSP and ESQL technology in the second approach, which basically is a combination approach. Three prototyping applications are designed and implemented in order to demonstrate these two approaches. There are stock quotes, web mail and coffee delivery. The multiple pipeline approach is used in implementing stock quotes and web mail, while the combination approach is used in implementing coffee delivery. Our main attention is focused on the coffee delivery. Figure 4 summarizes the whole logic architecture. The detailed discussion of these two approaches is in the following sections.

![Figure 4 Overall application logic](image)

3.2.2 Multiple pipeline approach

In the first approach, there are different JSP pages which are used to generate client-specific output from the same content. Our design allows for a compact and elegant
conversation between the clients, the main JSP page, the java bean, and output JSP pages. The idea is that the input page (either in html or wml format) contains a form whose ACTION attribute is the URL of the main JSP page functioning as a servlet. There are two functions for main JSP page: interacts with the main bean; determines the output template to send back the response (either in html or wml format). The main bean encapsulates the logic behind the web components, such as JSP, html, wml pages etc. The role of output JSP pages is to act as a transformer between the output of the attributes of the main bean and the response sent to the different user agent. Figure 5 illustrates this approach logic.

Figure 5. Application logic of the multiple pipeline approach

The application of Stock quotes will be used as example to illustrate the detailed implementation. We will present the stock quotes application in the following order:

- The entry pages
- The main JSP page
- The main bean
- The output JSP pages
The entry pages. The application’s entry pages are kept deliberately simple, there are two types of entry pages, one for html clients, another for wml clients.

```html
<html>
  <head><title>Stock Quotes</title></head>
  <body bgcolor="#FFFFFF">
    <center><table>
      <form action="quote.jsp" method="POST">
        <input TYPE="HIDDEN" name="outtype" value="htm">
        <tr>
          <td height="35" bgcolor="#9999FF" width="82%" align="left">
            Enter Symbol:<input name="symbol" value="YHOO">
          </td>
        </tr>
      </table></form></center>
      
      There is a variable named outtype with this input form (same as in html or wml entry page) to determine the output templates (output jsp pages) that the main JSP page selects for sending back the response. Above is the html entry page. When submitted, the form goes to the main JSP pages, process.jsp. If you point your browser at quote.htm you will see the following screen:

![Image of the web page with form input and submit button](image)
```
Above is the wml entry page. WML is an XML language, so the page begins with an XML declaration. The single <do> element of our page has three attributes, type, symbol and output. The accept type means that the element will wait to accept user action before it proceeds with its task. The label for this action is specified as Go! When submitted, go to the same main page, process.jsp, so does the html entry page. Left picture is what the wml page looks like in the UP.SDK. The <go> element can be used simply to jump to another URL, here to the main JSP page, but it can also function as a form. The destination is specified as an href attribute in either case.
The main JSP page, *quotes.jsp*

```jsp
<%@ page language="java" import="java.io.*; java.util.*; java.net.*;" %>

<% if ("html".equals (quotes.getOuttype ())) { %>
  out.clear();
  pageContext.forward("output.jsp");
%>

This page has no output elements, only Java code. It carries out the following tasks:

- Instantiate the main bean, *quotes*.
- Call the bean’s *getOuttype()* method
- Forward the request to the output jsp page determined by the *getOuttype()* method

One of the most powerful aspects of using JSP is the ability to use the JavaBeans component. A *jsp:usebean* action is used to associate a JavaBean with the JSP. A *jsp:setproperty* action is to set the values of properties in a bean, if property is set to "*", then the tag iterates over all the parameters in the ServletRequest, matching parameter names and value types to bean properties, and setting each matched property to the value of the matching parameter.

The main bean, *quotes*.

The central piece of the back end processing is the main bean, *Quotes.java*. The attribute name of the main bean is matched with the attribute name in the input form. This makes the exchange of information among tiers much easier. This main java bean uses Yahoo source to get the stock quoting price by java.net API.
public class Quotes {
    String symbol;
    String name;
    String price;
    String outtype;

    public void setOuttype(String outtype) {
        this.outtype = outtype;
    }

    public String getOuttype() {
        return outtype;
    }

    public void setSymbol(String symbol) {
        this.symbol = symbol;
        getSymbolValue(symbol);
    }

    public String getSymbol() {
        return symbol;
    }

    public String getName() {
        return name;
    }

    public String getPrice() {
        return price;
    }

    private void getSymbolValue(String symbol) {
        String urlString =
            "http://quote.yahoo.com/download/javasoft.beans?SYMBOLS=" +
            symbol + "&f=xformat=nl";
        try {
            URL url = new URL(urlString);
            URLConnection con = url.openConnection();
            InputStream is = con.getInputStream();
            InputStreamReader isr = new InputStreamReader(is);
            BufferedReader br = new BufferedReader(isr);
            String line = br.readLine();
            StringTokenizer tokenizer = new StringTokenizer(line,"","");
            name = tokenizer.nextToken();
            name = name.substring(1, name.length()-2); // remove quotes
            price = tokenizer.nextToken();
            price = price.substring(1, price.length()-1); // remove quotes
        } catch (IOException exception) {
            System.err.println("IOException: " + exception);
        }
    }
}

The output pages of result.

There are two output templates, one in html, and another in wml. The main JSP page will forward the request to relevant template determined by the value of the attribute named output. The output pages, at some level of generality, are the same: an XML
document of a specific document type (html or wml). Two different JSP pages are used to
generate those XML documents. One is to output html pages, another is to output wml
pages.

<html>
<head> <title>Stock Quotes Output</title> </head>
<body bgcolor="#FFFFF"> <center>
<p> <h2>The Quotes as following: </h2></p> <hr>
<table>
<tr> 
<td height="35" bgcolor="#9999FF" width="25%"><font face="Verdana,
Arial, Helvetica, sans-serif">Symbol</font></td>
<td height="35" bgcolor="#9999FF" width="25%"><font face="Verdana,
Arial, Helvetica, sans-serif">Name</font></td>
<td height="35" bgcolor="#9999FF" width="25%"><font face="Verdana,
Arial, Helvetica, sans-serif">Price & Time</font></td>
</tr>
</table>
</center></body></html>
The two JSP codes for output are shown above. The result set is obtained from the main bean, Quotes.java, and output directly either in html, wml format through the two JSP pages. The outputs, either for the html client or for the wml client, are shown here.

This completes the output of this application and, therefore, our first Web-WAP application. The approach used in our web mail application of this project is similar to the stock quotes application. We will not describe it in details.

We are now going to show a different approach to the same problem: how to generating both html and wml output from the same information source. An online coffee shop was implemented with a relational database as the source of information to demonstrate this approach.
3.2.3 A different combination approach

The another application of this project is a simulated online coffee shopping store. More complex features, such as session tracking, database connection and data sharing are needed to realize in an online store. How to handle these features under the Web-WAP context is really challenging. A different, more convenient approach needs to be developed for the complicated application.

This approach is the combination of the single and multiple pipeline methods. In this approach, the XSLT processor of Cocoon generates client-specific output by applying transformations to the same incoming XML data, which is called a single pipeline method. The multiple pipeline approach is also applied for this application. Figure 7 illustrates the application logic.

This online coffee shop allows users to buy a commodity through Internet or WAP-compatible device, where users have the ability to browse a catalog, choose items, and finally place an order. Our solution looks like that in the Figure 6. The modeling of
this problem is based on breaking it into the following parts, a four-tiers that separates the presentation from the logic and all other code:

- An entry XML page with different stylesheets either for wml or html client
- JSP pages to maintain application flow and to generate html or wml outputs.
- Business logic beans, and XML logicsheet
- Database to store the data

The important part is trying to figure what is going in a Bean, what is going in a JSP, what is going in a XML, and how to realize data flow among those objects.

![Application Architecture Diagram]

Figure 6 The architecture for the coffee application

We start off the illusion of the application with those business beans.

The JavaBeans

The Item class.
This bean abstracts an item in the shopping cart. Each item has the following attributes:

- String Cid, the ID of the item in the shopping cart
- String ItemName, the name of the commodity
- String price, the price of the commodity
- String comment, the textual detail about the item
- String quantity, the quantity of the item ordered
- String amount, the total amount in dollars of the item will be charged

![Diagram of Application Logic]

Figure 7 Application Logic of the combination approach

The Cid, ItemName, price are matched with correspondent attributes in the database; the comment, quantity, amount are the input from customers.

```java
package coffee;
public class Item {
    private String cid;
    private String itemname;
    private float price;
    private String comment;
    private float quantity;
    private float amount;
    public Item() {}
}
```
public string getCid() {
    return cid;
}

public void setCid(String aItemID) {
    cid = aItemID;
}

public float getPrice() {
    return price;
}

public void setPrice (float aPrice) {
    price = aPrice;
}

public float getQuantity() {
    return quantity;
}

public void setQuantity(float aPrice) {
    quantity = aPrice;
}

public float getAmount () {
    return amount;
}

public void CalAmount () {
    amount = quantity * price *100;
    int temp = (int) amount;
    amount = (float) (temp/100.0);
}

public String getComment() {
    return comment;
}

public void setComment (String aItemID) {
    comment = aItemID;
}

public String getItemName() {
    return itemname;
}

public void setItemname (String aItemID) {
    itemname = aItemID;
}

The *Order* Class.

This bean abstracts the concept of a client order, all the items in the shopping cart. Every order has an assigned ID, a vector of Items ordered.

```java
package coffee;
import java.util.Date;
import java.io.*;
import java.util.Vector;

public class Order {
```
private Vector items;
    private long orderId;

    public Order () {
        orderId = System.currentTimeMillis();
        items = new Vector ();
    }

    public void addItem ( Item item) {
        items.addElement (item);
    }

    public void removeItem (int i) {
        items.removeElementAt (i);
    }

    public Vector getItems () {
        return items;
    }

    public long getOrderId() {
        return orderId;
    }

The dbconn class.

We create the dbconn class to access coffee database on the linux MySQL server by JDBC. dbconn.java file encapsulates the basic connection and make up of a JDBC connection. The java bean not only hides the actual query and the result set handling from the JSP pages, but also hides the important connection information, such as user name and password from JSP pages, which is located in the /www folder and is available outside of the server.

package coffee;
import java.sql.*;
import java.io.*;
public class dbconn {
    ResultSet myResultSet = null;
    Statement stmt = null;
    String cid = "";
    private String myDriver = "org.gjt.mm.mysql.Driver";
    private String myURL = "jdbc:mysql://startcom.org/xdyang?user=xdyang&password=password";
    protected Connection myConn;
public void makeConnection() throws Exception
{
    class.forName(myDriver);
    myConn = DriverManager.getConnection(myURL);
}

public dbconn()
{
}
public void setCid (String str) {
    cid = str;
}
public String getCid () {
    return cid;
}
public String getColumn( String inCol) throws Exception
{
    return myResultSet.getString(inCol);
}
public boolean CoffeeDetail() throws Exception {
    String qry = 
    "SELECT * FROM coffees WHERE coffeoid = " + cid + ""
    stmt = myConn.createStatement();
    myResultSet = stmt.executeQuery(qry);
    return myResultSet.next();
}
public void takeDown() throws Exception
{
    stmt.close();
    myConn.close();
}

Database

The database in this application is trivial. There is only one table named coffee in
the database. We use it to demonstrate how to connect and retrieve information from
database with JSP and Cocoon. The database used in this application is MySQL server
3.22, the JDBC Driver used is mm Type 4 JDBC Driver 1.2b. The coffee table has the
structure as shown in Figure 8.

Figure 8. The schema of table coffee

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Null</th>
<th>Key</th>
<th>Default</th>
<th>Extra</th>
</tr>
</thead>
<tbody>
<tr>
<td>coffeoid</td>
<td>int(3)</td>
<td>YES</td>
<td>FRI</td>
<td>auto_increment</td>
<td></td>
</tr>
<tr>
<td>name</td>
<td>varchar(40)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>price</td>
<td>decimal(4,2)</td>
<td></td>
<td></td>
<td>0.03</td>
<td></td>
</tr>
</tbody>
</table>
The XML page and its stylesheets

In order to find a more convenient way to generate dynamic content for html and wml clients, we use Cocoon as part of implementation environment. Cocoon distribution includes a number of processors that implement common needs and situations. There are the XSLT processor, the XSP processor, the SQL processor and LDAP processor.

The entry page of this application is browse.xml, which queries the backend database and generate the dynamic XML file for further XSLT transformation. ESQL (Extended SQL taglib), which is evaluated by the SQL processor, is embedded into the browse.xml file to query the table coffee. Then, the XSLT processor applies XSLT transformations to the dynamic-generated XML file. The following figure is the browse.xml file:

```xml
<?xml version="1.0"?>
<?cocoons-process type="xsp"?>
<?cocoons-process type="xslt"?>
<?xml-stylesheet href="browse-html.xsl" type="text/xsl"?>
<?xml-stylesheet href="browse-wml.xsl" type="text/xsl" media="wml"?>

<xsp:page
language="java"
xmlns:esql="http://apache.org/cocoon/ESQL/1.0"
xmlns:xsp="http://www.apache.org/1999/XSP/Core"
>

<esql:connection>
<esql:driver>org.gjt.mm.mysql.Driver</esql:driver>
<esql:dburl>jdbc:mysql://startcom.org/xdyang</esql:dburl>
<esql:username>xdyang</esql:username>
<esql:password>password</esql:password>

<esql:execute-query>
<esql:query>select * from coffees</esql:query>
<esql:results>
<esql:row results>
```
Two instructions are used in `browse.xml` to control how Cocoon processes its XML content:

1) `<?cocoon-process type="xsp" ?>
    <xsp:page language="java"
    xmlns:esql = 'http://apache.org/cocoon/SQL/v2'
    xmlns:sp = 'http://www.apache.org/1999/XSP/Core'>

   These statements instruct Cocoon to generate a program that performs sql queries
   and serializes their results as XML.

2) The statements
   `<?cocoon-process type='xslt' >
   `<?xml-stylesheet href="browse-html.xsl" type = 'text/xsl' ?>
   `<?xml-stylesheet href="browse-wml.xsl" type = 'text/xsl' media = 'wap' ?>`

   instruct Cocoon to apply the `browse-html`, or `browse-wml` XSLT style sheet to the
   XML data resulting from the execution of the generated ESQL query program.

   Cocoon will detect whether the incoming web browser was an html browser or
   WAP browser, responding with the correct stylesheet [7]. The `browse-wml.xsl`
   will be discussed in the WML section; the `browse-html.xsl` stylesheets is listed
   here:
When users point their browser (whether a html browser or a WAP browser) to
browser.xml, the relevant attributes for each coffee item will be displayed, either as a row
of an html table in html page, or as a selection option contained in a <card> named buy in
the wml page. Here is the output of browse.xml transformed by browse-html.xsl. The
User can click the coffee name to continue shopping, and the cid of each coffee brand is
appended to the end of every hyperlink dynamically by browse-html.xsl.
There is also a wml output of browse.xml transformed by browse-wml.xsl. Since the coffee application is complicated, next we will describe the html version of this application, and will describe the wml version later in detail.

Html version

In html version, the browse.xml page will link users to JSP pages to continue their shopping.

The buy.jsp

The page creates a DBconn bean, which is used to retrieve from the database the detailed information about the coffee when a cid is given. The scope of this bean is defined as session, so once the DBconn is created, it will be available throughout the session. This bean also can be used to store order data when a user checks out. The buy.jsp page functions as a form for users to input a quantity they want to buy specific coffee.
<html><head></head><body bgcolor="#FFFFFF"><center>

<!-- DBCconn.makeConnection (); -->
String display = "";
String itemname = "";
String price = "";
String cid = "";

if (DBCconn.CoffeeDetail()) {

<h1><a href="browse.xml">Online Coffee Delivery</a></h1>
<p><h2><%
itemname = DBCconn.getColumn("name");
price = DBCconn.getColumn("price");
cid = DBCconn.getColumn("coffeeld");
%
>
Item: <%= itemname %><br />
Price (per lb): <%= price %>
</h2></p>

<% }) else { %>
<%
display = "No Coffee Listed";
%
<% } %>
</h2></p></center>
<form action="continue.jsp" method="post">
<table>
<tr><td height="35" bgcolor="#FFFF99" width="160" style="font-family:verdana,arial,helvetica,sans-serif">Quantity:</td></tr>
<tr><td height="35" bgcolor="#FFFF99" width="160" style="font-family:verdana,arial,helvetica,sans-serif">Comment:</td></tr>
<tr><td height="35" bgcolor="#FFFF99" width="160" style="font-family:verdana,arial,helvetica,sans-serif">Add to Cart</td></tr>
</table>
</form></body></html>
Here is the output of *buy.jsp*. The action of this form is *continue.jsp*

![Web Wap with JSP and XML](image)

**Online Coffee Delivery**

**Item:** French Roast Colombian Coffee  
**Price (per lb):** $8.99

<table>
<thead>
<tr>
<th>Quantity</th>
<th>0.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comment</td>
<td>ASAP</td>
</tr>
</tbody>
</table>

The *continue.jsp*  
This JSP page controls the main flow of the html version. It is responsible for instantiating the *Item* bean and *Order* bean using the input data from users of *buy.jsp*. The *Order* bean is available throughout all the session. From here, user can go to *browse.xml* to continue shopping, go to *vieworder.jsp* to view/modify the order object, or go to *checkout.jsp* to checkout. Since Cocoon and Tomcat is in the same servlet container, the session object is shared by both the XML page and the JSP pages. Therefore, the Order object, which is put into session or retrieved from session, is shared by these XML and JSP pages.

```jsp
<%@ page language="java" import="coffee.*, java.util.*" %>
<jsp:useBean id="item" class="coffee.Item" scope="page" />
```
Here is the output of *continue.jsp*
Web-Wap with JSP and XML

The Following item has been added to your cart

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Coffee</th>
<th>Amount</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25 pounds</td>
<td>French Roast</td>
<td>$2.24</td>
<td>ASAP</td>
</tr>
<tr>
<td></td>
<td>Columbian Coffee</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Continue Shopping? View/Modify Cart Checkout?

The vieworder.jsp

This JSP page shows the details for an order. It simply retrieves the order object from the session and displays its items in a formatted manner. The order stores the actual items. This is also the place for user to modify the order.

```html
<%@ page language="java" import="java.*,java.util.*" %>
<html>
<body bgcolor="#FFFFFF">
<hl> <center> The Following items are in your cart: </center> <hl>
<hr>
<p> <center> <h5> If you want to delete a item, just leave the Quantity column blank </h5> </p>
<p> <center> <table>
<tr>
<td action="updateorder.jsp" method="post"><br>
<table>
<thead>
<tr>
<th>Quantity (in lbs)</th>
<th>Coffee</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25 pounds</td>
<td>French Roast</td>
<td>$2.24</td>
</tr>
<tr>
<td></td>
<td>Columbian Coffee</td>
<td></td>
</tr>
</tbody>
</table>
</html>
```
And here's how it looks in the browser:
The *updateorder.jsp*

This page is the action of *vieworder.jsp*, and used to update the attributes of the items in the order object. It retrieves the order object from the session, and store the order object back into the session after the order is updated.
int i=0;
for (Enumeration e = (myorder.getItems()).elements();
e.hasMoreElements(); ) {
    Item incart = (Item) e.nextElement();
    try {
        String tempquantity = (String) request.getParameter("(i++"));
        incart.setQuantity(Float.parseFloat(tempquantity));
        if (incart.getQuantity() >= 0) {
            myorder.removeItem (i);
        } else {
            incart.setAmount (0);
        }
    } catch (Exception err) {
        myorder.removeItem (i);
    }
    ++i;
}
session.putValue("anOrder", myorder);
} catch (Exception eee) {
    %>
<tr><td height="35" width="30%"> <font face="Verdana, Arial, Helvetica, sans-serif">
    <a href="/browse.xml"> Continue Shopping? </a> </font> </td>
<td height="35" width="30%"><font face="Verdana, Arial, Helvetica, sans-serif">
    <a href="/checkout.jsp">Checkout? </a> </font> </td></tr></table></center></P></body></html>
The checkout.jsp and end.jsp

These are two simple JSP pages that just close the user's session by calling invalidate().

We know that Dbconn bean is available through the whole session. It is a easy task to store the Order data into the database. Since this is a trivial application, we just do nothing here but close the session object. We omit the two JSP codes here, the outputs are as follows:
The wml version

A WAP ‘page’ (actually not called page but decks), however, is typically much smaller, because of the screen sizes, and then the server often sends more than one of these ‘pages’ at once – so that the phone can cache multiple fragments of content for future use. Furthermore, data can be shared among those decks. Therefore, we can take advantage of this property to simplify the wml version of this application.
When the users point their WAP-enabled appliance to the same entry page of this application, http://xdyang.startcom.org:3080/coffee/browse.xml, the Cocoon will detect that the incoming web browser was WAP browser, then respond with browse-wml.xsl stylesheet, which is listed here:

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

    <xsl:template match="page">
        <xsl:processing-instruction name="cocon-
            format" type="text/wml"></xsl:processing-instruction>

        <wml>
            <card id="index" title="Wireless Coffee Delivery">
                <p align="center"> Wireless Coffee Delivery <br/></p>

                <p>
                    <a href="#details">Start Shopping</a><br/>
                    <a href="#about">About</a><br/>
                </p>
            </card>

            <card id="about" title="About">
                <onevent type="onTimer">
                    <prev/>
                </onevent>
                <timer value="25"/>
                <p align="center">
                    <br/>
                    <small>
                        Copyright © 2001<br/>
                        xiangdong Yang<br/>
                        All rights reserved.
                    </small>
                </p>
            </card>

            <card id="details" title="Please Select">
                <do type="accept" label="buy">
                    <go href="#quantity" />
                </do>
            </card>
        </wml>
    </xsl:template>
</xsl:stylesheet>
```
<xsl:text> Price: $$</xsl:text> 
</xsl:text>
</option>
</xsl:for-each>
</select>
</p>
</card>

<card id = "quantity" title = "Quantity">
   <do type = "accept">
      <go href = "#shippingmethod"/>
   </do>
   <p align = "center"> Add to Shopping Cart
       Quantity (lb); <input name="quantity" format="N.NN" maxlen="10" value=""/>
   </p>
</card>

<card id = "shippingmethod" title = "Shipping Method">
   <do type = "accept">
      <go href = "#modify"/>
   </do>
   <p>
      Shipping Method:
      <select name = "shippingcharge">
         <option value = "1.50"> US Mail -- $1.50 </option>
         <option value = "3.00"> UPS -- $3.00 </option>
         <option value = "4.50"> FedEx -- $4.50 </option>
      </select>
   </p>
</card>

<card id = "modify" title = "View Modify Order">
   <do type = "accept">
      <go href = "wmilour.jsp">
         <postfield name = "quantity" value = "$quantity"/>
         <postfield name = "shippingcharge" value = "$shippingcharge"/>
         <postfield name = "cid" value = "$cid"/>
         <postfield name = "output" value = "wml"/>
      </go>
   </do>
   <p>
      <do type = "options" label = "Modify?">
         <go href = "#quantity"/>
      </do>
   </p>
   <p>
      Submit Order?
      <br/>
      ------
      <br/>Coffee ID: $cid
      <br/>Quantity:$quantity Lbs
      <br/>shipping Charge:
      <br/>$shippingcharge
      <br/>
   </p>
</card>
</wml>
It is noticed that the browse-wml.xsl is much longer than its html counterpart: browse.html.xsl. In the Web version of this project, it is the browse-html.xsl page that lets users browse the coffee items; the buy.jsp page lets users to input the quantity they want to buy; the vieworder.jsp page and updateorder.jsp page combined together let users view/modify the order; and the checkout.jsp page lets users choose the shipping method. In the wml version, the single browse-wml.xsl page, whose actual content is packaged into a sequence of elements of type <card>, fulfills all the functions the browse-html.xsl, buy.jsp, vieworder.jsp and updateorder.jsp pages fulfill together. The details card in wml version functions similarly as browse-html.xsl in Web version, the quantity card functions similarly as buy.jsp page in Web version, the shippingmethod card functions similarly as checkout.jsp in Web version. Modifying an order is very easy; all you need to do is to revisit the relevant card again, since the data is shared among all the cards in the same wml page. The following is the wml output of those cards:
Wireless Coffee Delivery

1. Mountain Java
   Price: $9.99 (per lb)
2. French Roast
   Price: $10.99 (per lb)

Add to Shopping Cart
Quantity (lb): 1.25

Shipping Method:
1. USPS -- $4.50
2. UPS -- $3.60
3. FedEx -- $4.50
After the users click OK to submit the order, the order data will be sent to `wmlout.jsp`, which corresponds to the `checkout.jsp` and `end.jsp` pages in the Web version. It would be easy for `wmlout.jsp` page to retrieve the data from Request and store the order data into the database, since the `Dbconn` bean in this application, makes the database available to JSP pages, and is available for the whole session. `wmlout.jsp` simply just sends the order data back to WAP users. We list the `wmlout.jsp` here:

```xml
<wpml_page content_type="text/vnd.wap.wml; charset=ISO-8859-1"/>
<xml version="1.0"/>
</doctype wml PUBLIC "-//PHONE.COM//DTD WML 1.1//EN"
    "http://www.phone.com/dtd/wml11.dtd">
</wml>

<head>
  <meta http-equiv="Cache-Control" content="no-cache" forua="true"/>
</head>

<card id="output" title="coffee">
  <doctype accept="label="More?">
    <go href="browse.xml">
    <go>
    </go>
  </go>
  </doctype>
  <p align="center">Thank you for your order</p>
</card>
```
In the web version, the shopping cart is created and maintained easily by session tracking, which is an important feature that is provided by the servlet engine. In order to maintain session tracking, there are two common ways: Cookies and URL rewriting, cookies are used by default. But, the WAP browser does not support Cookies, and, the length of URL rewriting is also restricted in the WAP browser. Furthermore, there is another language, wml script, which support data sharing easily between wml pages. Therefore, we will not support the shopping cart in this WAP version, since the wml script does not belong to the scope of this project.

In this second approach, Cocoon generates client-specific markup by applying transformations to incoming XML data (browse.xml). This is called a single pipeline method. There is also a set of client-specific JSP pages (checkout.jsp, end.jsp for html clients, wmlout.jsp for wml clients) to generate client-specific outputs, same as the first approach: multiple pipeline approach. Therefore, our second approach is actually the combination of single pipeline and multiple pipeline methods.

4. Discussion and conclusion
The main purpose of this project is to generate both web pages and WAP pages from the same content on a web site. There are two approaches utilized here to fulfill the purpose.

The first one is a multiple pipeline approach. In this approach, different JSP pages are used to generate client-specific output from the same content. The output is either in html or in wml format. This approach is simple, straightforward, and efficient, it is applicable for small application implementation.

Another one is the approach that combines the single and multiple pipeline methods together. In this approach, JSP is integrated with XML technology (Cocoon as implementation environment) to generate client-specific markup languages by applying transformations to incoming XML data. Each type of client is applied with a different stylesheet. This approach is powerful, convenient, and ideal for complicated application implementation. But, the process of this approach is sophisticated and the response time of the approach is longer than the first one.

The Web-WAP project involves JSP, XML, XSLT, XSP, ESQL, Java, JDBC, MySQL, HTML, and WML languages. All those technologies cooperate very well under the same servlet container. A n-tier client/server architecture is easily realized using JSP/XML technologies to separate presentation from content thoroughly.

JSP and XML are natural partners for developing Web applications that use heterogeneous data sources and support multilingual clients. JSP is the good answer for the main problems facing web developers, such as scalability, integration of backend data and business logic, manageability, personalization etc. XML brings the same cross-platform benefits to information exchange as the Java programming has for processing.
Reference


[7] Java and XML, Brett McLaughlin, O'REILLY associates