Implementing OGC Web Map Service Client Applications Using JSP, JSTL and XMLC

Hao Ding , Richard Pascoe & Neville Churcher

Department of Computer Science
University of Canterbury. Christchurch, New Zealand
Phone: +64 3 364-2362 Fax: +64 3 364-2569
Email: hdi12@student.canterbury.ac.nz , {richard, neville}@cosc.canterbury.ac.nz

ABSTRACT

Java technologies are widely used in web application development. In this paper are described three approaches to developing Java-based web applications and our experiences with applying each to the development of client that interact with servers implementing the OGC (Open GIS Consortium) Web Map Service (WMS) specification. Also described is the installation and configuration of open source software that implements the WMS specification. The paper is concluded with some preliminary insights into when one of the three approaches to WMS client implementation is more suited to another.

Keywords and phrases: WMS, JSP, JSTL, XMLC, map layer, web map server

1.0 INTRODUCTION

Of the many technologies, such as Common Gateway Interface (CGI), Active Server Pages (ASP), JavaServer Pages (JSP), that are used to develop web applications, three are of particular interest to the research presented here. These three technologies or approaches to developing clients that utilise web services are JavaServer Pages (JSP), JSP with the use of tags from the JSP Standard Tag Library (JSTL), and the eXtensible Markup Language Compiler (XMLC).

JSP is a more convenient way to write Java servlets, and allows the insertion of Java code directly into static HTML (Hypertext Markup Language) pages. JSTL (SUN, 2002) is a new technology released in March 2002 by SUN. JSTL is based on JSP, but uses standard functional tags instead of Java code to manipulate dynamic content in a JSP page. Therefore, web designers who do not know Java can still develop JSP pages efficiently. Enhydra XMLC (Enhydra, 2002) provides an object-oriented mechanism for generating dynamic web pages from static template HTML pages, which entirely separates the web page appearance design from the development of dynamic content.

OGC Web Map Service (WMS) is an interoperable web mapping system. It provides common interfaces to connect with the client application and dynamically process geo-referenced data such as geographic maps and features coded using Geography Markup Language (GML) documents.

The objective of the research presented here is to evaluate the efficacy of three Java based technologies for implementing clients that interact with servers implementing the OGC WMS. GIS applications are often different from common web applications such as on-line stores, because they involve geographic data handling. For each approach, a client is implemented that achieves a series of tasks with different contexts in order to compare various aspects of the three approaches, including how easily they make the development of clients that handle geographic data and utilise the functionality provided by the WMS.

The remainder of this paper is structured as follows. In the next section, we introduce OGC WMS. The OGC WMS client application is described in section 3.0. A set of evaluation criteria and implementation tasks are described in section 4.0 and section 5.0. In section 6.0, the three technologies (JSP, JSTL and XMLC) are introduced with a simple example. The recent implementation, including building local systems and developing

client applications, are described in section 7.0. Finally, conclusions and future directions are presented in section 8.0.

2.0   OGC WEB MAP SERVICE

The OGC Web Map Service specification defines a set of functions that clients may use to interact with WMS providers (servers). Any client making requests that conform to the specification can interact with any server that implements the WMS service. In effect, this creates an interoperable, distributed web mapping system. A simple and typical example of the structure of a web mapping application is actually a web-based client/server architecture, as illustrated in Figure 1.

Figure 1: The typical structure of a web mapping application

In a typical web mapping scenario, the client application requests desired information from the web map server. The map server retrieves from the database the appropriate layers of geo-feature data for the specified spatial domain and generates a map, which is a simple graphic image (i.e., a GIF or PNG) that can be viewed directly in a graphical web browser or other pictorial software. The client and web map server interact using Hypertext Transfer Protocol (HTTP).

Before the creation of the OGC WMS specification (OGC, 2001), each map server might implement different functions: as a consequence, a client that successfully interacts with one map server would in most cases, be unable to interact with another. The OGC WMS specification offers a standard client-server interaction protocol that each map server implements as a common interface for accepting requests and returning responses. Thus, the same client is able to access to all available OGC web map servers over the Internet. The interoperable web mapping structure is illustrated in Figure 2, where each map server is accessed by the client through the common interface. In a distributed OGC WMS, a WMS server can also run as a WMS client that requests capabilities and maps from other WMS servers.
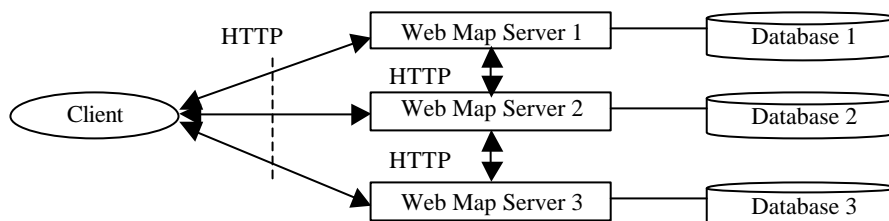

Figure 2: Standard interoperable web mapping structure

An OGC web map server implements three functions: GetCapabilities, GetMap, and GetFeatureInfo. The GetCapabilities function provides the client with a map server's service metadata, specifying its capabilities. The GetMap function specifies map request parameters that enable the client to request an image map. Finally, the GetFeatureInfo function allows the client to request more information about features at a specific location in the map.

3.0   OGC WMS CLIENT APPLICATIONS

An OGC WMS client application is a program that communicates with the OGC WMS servers using the three functions GetCapabilities, GetMap, and GetFeatureInfo, as noted earlier in section 2.0. More specifically, in a typical OGC WMS client/server interaction, the client firstly requests GetCapabilities from the map server in order to determine what the map server can do and what maps the map server can provide. The client then requests GetMap with the map server's capabilities information in order to get a map image. Finally, the client can request GetFeatureInfo by specifying a point on the map to receive more geographic feature information.

In response to a GetCapabilities request, the OGC web map server produces an eXtensible Markup Language (XML) document containing the web map server's service metadata, describing all the operations it supports, and providing information about the available maps. The client application has to parse the XML capabilities document to retrieve the necessary information used to request a map. The Document Object Model (DOM)

(http://www.w3.org/DOM/) is a widely used and efficient XML parser, which was utilised in our implementation. The DOM represents an XML document as a tree of nodes that can be easily traversed and edited with its standard interfaces.

With the capabilities information, the client can request a map image from the map server. The GetMap request usually includes some necessary parameters such as Layers, Styles, Bounding Box (BBox), Spatial Reference System (SRS), map output Width and Height, and the Format of the image. Layers specify the information to be shown on the map (for example, roads, rivers, towns, and so on). Styles are defined to depict Layers, for example, lines represent roads, circles represent towns, and so on. BBox is a set of four coordinate values (minX, minY, maxX, maxY) indicating a rectangular area on the earth to be mapped. SRS is the projected coordinate system to be used. (Please see OGC WMS specification (OGC, 2001) for a detailed description.)

After getting a map, the client can request feature data of a specific point on the map using the GetFeatureInfo function. The response from the WMS server will be one of the three output formats: a Geography Markup Language (GML) file, a plain text file, or a Hypertext Markup Language (HTML) file. GML was selected as the feature data output format for the research presented here. GML is a type of encoding based on the XML specifying the geographic feature information. Geographic features include points, lines, polygons, and so on, which are clearly defined in the OGC Simple Features Specification (OGC, 1999). The client application parses the GML and displays the feature information in the web browser.

4.0   EVALUATION CRITERIA

To evaluate the performance of three approaches to implementing the OGC WMS client applications, a set of criteria were developed:

?   **Ease of parsing XML/GML:** Parsing XML/GML documents is an important task to be achieved by these clients, since WMS and more significantly WFS (Web Feature Service), will utilise XML/GML to transfer information. The ease with which clients that process XML/GML can be implemented will be influential in determining which approach is adopted for client development.
?   **Map handling:** Retrieving and organising image maps is another key issue that needs to be addressed in a WMS client application. The maps could be retrieved from one server or different servers, could present a common area or different extents on the earth, or may be overlapped for display to the user. In addition, changing the order of overlapping maps may result in different effects.
?   **Multiple servers interaction:** In a distributed system, a client may wish to allow information from different servers to be retrieved and then merged into a single cohesive response for display to the user. The issue to be addressed here is the extent to which the implementation of such processing within a client is supported by the three approaches.
?   **Interface layout:** WMS client applications use the web browser as the user interface. How easily a client can generate dynamic web pages with the three approaches is an important issue need to be considered.
?   **Execution speed:** Speed is a factor that always been used to measure the efficiency of an application or a program. The issues to be addressed here include the speed of compiling, the speed of request handling, and the speed of pages loading.
?   **Ease of revision:** It is normal to modify an established page and application by adding or cutting some components and functions during the development. The issue addressed here is to critique how easily the client can make changes or upgrades based on the previous work with the three approaches.

5.0   IMPLEMENTATION TASKS

A series of OGC WMS client application implementation tasks were designed to compare the three approaches against the evaluation criteria described previously in Section 4.0. These tasks are briefly described next; the three approaches being investigated for implementing the client that performs these tasks will be described in the next section.

A map server typically provides geographic information for a specified spatial extent and this information is typically divided into thematic layers. The extent covered by the WMS server and the thematic layers available are described by the metadata within the map servers capabilities XML document.

In the first task, the client interacts with a single web map server. The client application requests the capabilities XML document using the GetCapabilities function, mentioned earlier, and parses this document to retrieve metadata describing the layers that users may select for retrieval using the GetMap function. The WMS server

will combine the requested layers into a single picture or map. The resulting map should be presented an HTTP image map so that the user can click on a specific point of the map to get more feature information using the GetFeatureInfo function.

In the second task, the client continues to interact with a single WMS, but multiple GetMap requests will be sent if more than one Layer is selected by the user — one request per Layer. Thus, if the user selects three Layers (for example, roads, lakes, and airports), three separate maps will be requested. Because the maps are requested for the same extent, they can be overlapped in a user specified order, unlike task one where the server determines the order, to make a new map. Each single-Layer map must be transparent so that one map will not be hidden by those on top of it when they overlap. Map overlapping is illustrated in Figure 3. The user can make a custom map using the single-Layer maps, and the map combination is performed by the client application.

In the third task, the client connects with multiple web map servers, and each map server has different capabilities. The client application must send GetCapabilities requests to each WMS and parse each capabilities XML document individually. When the user selects Layers belonging to different map servers, each map server will return a map, and these maps may represent different extents of the earth. Only maps of the same extent can be transparently overlapped by the client application, which is a variation from task 2. When the user clicks a specific point on the combined map, the feature data about that point could include data from multiple maps that are from different map servers. The client application must request GetFeatureInfo to each of those map servers, parse each GML document, and display all feature data in the web browser.
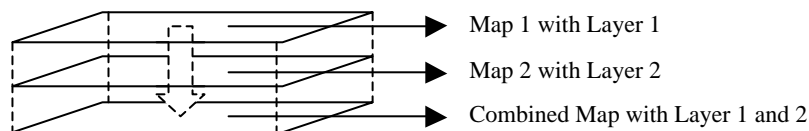
Figure 3: Map overlapping

6.0   JSP, JSTL, and XMLC

JSP, JSTL, and XMLC are all Java-based technologies for web application development. In this section, we briefly introduce each of them and demonstrate their differences using a simple example.

JSP is based on Java servlet technology, which enables you to mix static HTML with dynamically generated content. A JSP page is basically an HTML page with inline Java code that manipulates dynamical content and specific tags in addition to the regular HTML tags. The embedded Java code is expressed using script elements. A JSP page is processed by a JSP container. When the web browser makes a request for a JSP page, the JSP container in the web server first compiles the JSP page into a servlet, which is a Java program. Then the JSP container compiles the servlet with more business logic Java code (i.e. JavaBeans), and finally the JSP container executes the compiled servlet class to produce a generated web page to the web browser. Therefore, a JSP page is actually another way to write a servlet without having to be a Java programming expert (Bergsten, 2002).

JSTL is based on JSP technology, so a JSTL page is also a JSP page. JSTL is a collection of HTML-like tags that allow users to produce dynamic web pages without utilising Java code (Bayern, 2002). JSTL includes tags for many common tasks such as looping over data, performing conditional operations, importing and processing data from other web pages, simple XML manipulating, database accessing, and text formatting (SUN, 2002). With these standard tags, JSP page authors do not need to create their own custom tags to realise these functions, and web page designers without Java knowledge can also easily edit the JSP page. JSTL also supports an Expression Language (EL) that is inspired by both ECMAScript (JavaScript) and the XPATH expression languages, which means that JSP pages ought to have easier access to the data required. The JSTL expressions look like "${expression}". For example, instead of the Java/JSP expression, such as

<%= session.getAttribute("layer").getTitle() %>

JSTL can access the data using JSTL expression

<c:out value= "${sessionScope.layer.title}"/>

XMLC is a very different method from JSP and JSTL. XMLC provides a Java based object-oriented mechanism for creating dynamic content from static HTML (XML) documents (WebReference, 2000). Web page authors

can design attractive static HTML pages, adding "id" attributes and mock-up content into the elements that will be manipulated later. XMLC then compiles the template HTML pages and converts them to Java classes, where the HTML pages are represented using the DOM. The generated classes can be accessed, and the attributes, content, and nested tags of the elements with "id" attributes can be replaced or removed by Java programs using standard DOM manipulation APIs to create dynamic web pages (Enhydra, 2002).

In the following listings, we show a very simple page that will be generated using the three approaches. The page gets a parameter named "result" from a request. If the value of the parameter is "win", the page will print out "It is good news! We win." and the word "good" will be displayed in red font. If the parameter is "lose", the sentence displayed in the page will be "It is bad news! We lose." and the word "bad" will be displayed using a blue font. (Only the "body" code of the page is listed; other parts are omitted.)

The page generated using JSP and JSTL is shown in Listing 1. Each line of JSTL code in the right side corresponds to the JSP code in the left. In Listing 2, we show how to use XMLC to generate the same page. It is a template page with mock-up content. The elements <font> and <span> are identified with the "id" attribute.

Listing 1: JSP and JSTL example

```
……                                               ……
<body>                                            <body>
It is a                                           It is a
<%  String result = request.getParameter("result");   <c:set var="result" value="${param.result}"/>
     if(result.equals("win")) {                   <c:if test= "${result=='win'}">
%>                                                  <font color="red">good</font>
<font color="red">good</font>                     </c:if>
<%  } if(result.equals("lose")) {                 <c:if test= "${result=='lose'}">
%>                                                  <font color="blue">bad</font>
<font color="blue">bad</font>                     </c:if>
<%  }
%>
news! We <%= result %>                            news! We <c:out value="${result}"/>
</body>                                            </body>
……                                               ……
```

Listing 2: Template HTML page

```
……
<body>
It is a <font id="state" color="black">good or bad</font> news! We <span id="result">win or lose</span>
</body>
……
```

If the name of the page file in Listing 2 is *game.html*, XMLC will create a Java class named *game.class* after compiling the HTML file. This class presents the HTML page in the corresponding DOM, which can be used by other Java programs to modify the page content. Listing 3 is a manipulation Java program using a servlet to get the requested parameter, modify page content, and output the generated page. The highlighted code shows how to access the identified element and set its attribute and text content.

Listing 3: Manipulation class

```
public class gameMan extends HttpServlet{
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                 throws ServletException, IOException {

       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       String result = request.getParameter("result");     // Get request parameter

       game game = new game();        // Create an instance of the HTML page object
       HTMLFontElement state = game.getElementState();   // Get a reference to the <font> element

       if(result.equals("win") {
```

```
                state.setColor("red");            // Change the "colour" attribute of <font> element
                game.setTextState("good");   // Change text within <font> tags
            }
            if(result.equals("lose")) {
                state.setColor("blue");           // Change the "colour" attribute of <font> element
                game.setTextState("bad");        // Change text within <font> tags
            }
            game.setTextResult(result);          // Change text within <font> tags
            out.println(game.toDocument());   // Print out the modified page
        }
    }
```

7.0   IMPLEMENTATION

For the purpose of testing, a local web mapping system has been installed and configured using in part open source software and in part an OGC WMS client application which we have implemented three times, once for each approach being investigated, to accomplish the first task described in section 3.0. In this section, we briefly introduce both.

7.1   System Building

The local web mapping system architecture for this research is illustrated in Figure 4, where the WMS components and the message flows among them are described.
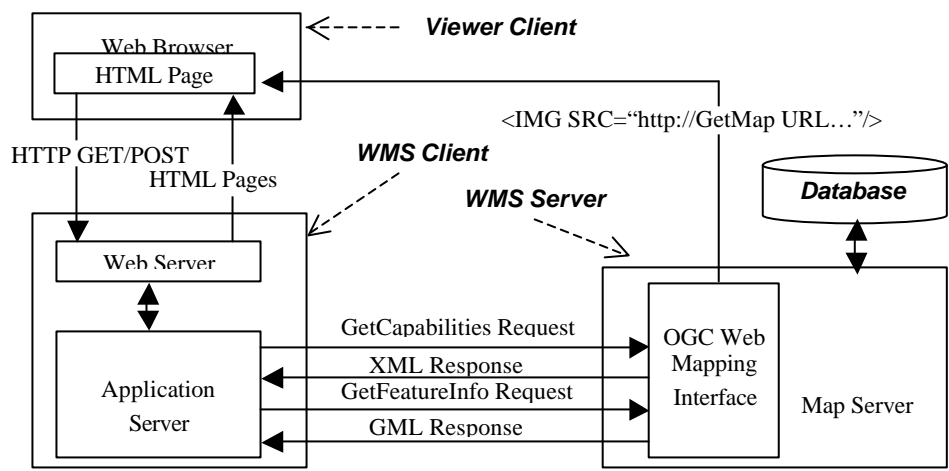

Figure 4: Local web mapping system architecture

Viewer client is actually a series of HTML pages running inside a Web browser that can interact directly with a map server via the HTTP profile of the OGC Web Mapping Interface. WMS client manages the interactions with OGC web mapping interfaces via HTTP, and dynamically generates HTML that can run in the Web Browser. WMS server is a map server that provides three OGC Web Mapping Interfaces (GetCapability, GetMap, and GetFeatureInfo). WMS server accepts requests from WMS client and viewer client in the form of HTTP URL strings, and returns results encoded as XML, GIF, GML, and so on. The database stores geo-feature data that can be accessed and utilised by the WMS server to generate GML documents or draw maps.

As illustrated in Figure 4, the user interacts locally with Viewer Client and submits HTTP GET/POST requests to the WMS client. The web server in the WMS client accepts user requests and parses them before forwarding them to the application server. The application server then processes these requests in an application, and returns dynamically generated HTML pages to the web browser. The details on how the application works were described in section 3.0. In real practice, GetCapabilities and GetFeatureInfo are requested from a Java program, and a GetMap request is embedded in the HTML pages in the form of <img src=GetMap request url\> to fetch an image map from the map server and display it directly in the web browser.

Based on the architecture described previously, the web mapping system can be built using some open source software. In this practice, the system is built as illustrated in Figure 5.

Any web browser supporting HTML 4.0, such as Internet Explorer or Netscape Navigator, can be used as client viewer. The WMS client uses Jakarta Tomcat (http://jakarta.apache.org/tomcat) as a web server and JSP/servlet container. The latest release, Tomcat 4.0, is able to implement the Java servlet 2.3 and JavaServer Pages 1.2 specifications.
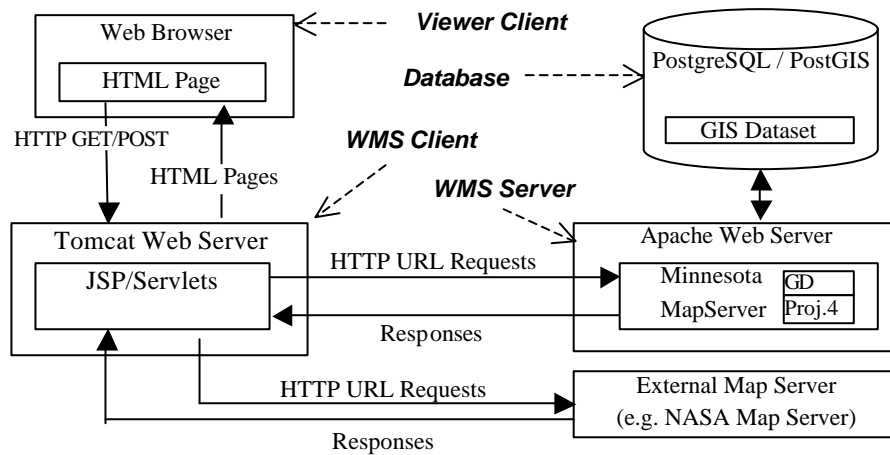

Figure 5: Practical web mapping system

The WMS server was set up using the University of Minnesota (UMN)'s MapServer 3.6 (http://mapserver.gis.umn.edu), which supports OGC WMS specification 1.1.0 (OGC, 2001). MapServer consists of only one executable file named "mapserv", which is a CGI program running in the Apache HTTP web server (http://www.apache.org) that knows how to handle WMS requests. In the practise, when multiple map servers are needed (such as in the third task described in section 4.0), several Apache web servers can simultaneously run MapServers on different ports. MapServer must be compiled together with the PROJ.4 (http://www.remotesensing.org/proj), which is a cartographic projections library, for OGC WMS compliance. MapServer also uses GD library (http://www.boutell.com/gd) to render GIFs or PNGs. Each MapServer needs a mapfile (a control file with the suffix .map), which is a configuration file defining display and query parameters and the data source to be used. A mapfile must include information about where to get the spatial data, how to draw the map, and what layer metadata and spatial information must be included in the WMS capabilities document. Listing 4 is part of the mapfile we set up for the MapServer with UMN MapServer demo dataset for Itasca country. The sample interfaces been implemented are shown in section 6.2.

Listing 4: Sample mapfile fragment

```
NAME DEMO
EXTENT 388013.643812817 5200395.13465842 500802.348432817 5313156.99196842
PROJECTION              # Project definition
   "init=epsg:26915"
END
LAYER                   # Layer definitions
   NAME lakespy2
   CONNECTIONTYPE postgis
   CONNECTION "user=hdi12 dbname=mygisdb host=localhost port=5432"
   DATA "the_geom from lakespy2"
   TYPE POLYGON
   STATUS OFF
   CLASS
      COLOR 49 117 185
   END
   DUMP TRUE             # allow GML export
   METADATA
      WMS_TITLE "Lakes and Rivers"
      WMS_ABSTRACT "DLG lake and river polygons for Itasca County."
      WMS_SRS "EPSG:26915"
   END
END              # lakes
……             # More layers definitions
```

MapServer uses PostGIS (http://postgis.refractions.net) as a vector database to store geo-feature data. PostGIS is an extension to the PostgreSQL (http://www.postgresql.org) object-relational database system that allows GIS objects to be stored in the database. PostGIS supports the "simple features" defined by the OGC (OGC, 1999). These simple features are Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeomCollection. Vector format data is a coordinate-based data structure that represents the location and shape of feature and boundaries precisely, and can be used directly by MapServer to draw maps or generate GML files.

The vector datasets to be stored in the PostGIS can be converted from shapefiles. The shapefile format is published by Environmental Systems Research Institute (ESRI) for simple vector data with attributes. An ESRI shapefile consists of three files with the same basename:

- ? .shp – the main file holding the actual vertices that describe shapes.
- ? .shx – the index file holding index data pointing to the structures in the main file.
- ? .dbf – the dBase table holding the feature attributes.

PostGIS provides a tool called "shp2pgsql" to create a SQL file from a shapefile. The SQL file holds SQL commands such as "create table" and "insert data", which can be executed directly by PostGIS. One shapefile corresponds to one PostGIS table, containing the points, lines, or polygons to display as map layers.

In the application described as follows, the shapefiles were acquired from the Minnesota MapServer demo application datasets, which are for Itasca County located in north central Minnesota. An external OGC web map server such as the NASA (National Aeronautics and Space Administration) map server (http://wmt.digitalearth.gov/cgi-bin/wmt.cgi?) was also used in the practice to test the client application.

7.2   Client Application

The client application we implemented connects to one web map server. The user can select one map server from the index page to connect with as shown in Figure 6. Once the user selects a map server (for example, UMN MapServer Itasca Demo), the GetCapabilities request is sent to that map server to get its service capabilities. In Figure 7, the map layers parsed from the UMN MapServer capabilities XML are listed on the left. Users can choose Layers they are interested in from the list and click the "submit" button, using the GetMap function to request an image map from the map server. The map created in Figure 7 displays country boundary, lakes and rivers, highways, and airports. The map displayed can be zoomed in, zoomed out, and scrolled. In Figure 8, the map is zoomed in and scrolled right and up. When users click the map itself, GetFeatureinfo is sent and the feature information on that point will be shown under the map (This function has not been added yet). The three clients implemented to demonstrate each approach to client development presented identical user interface (web pages).
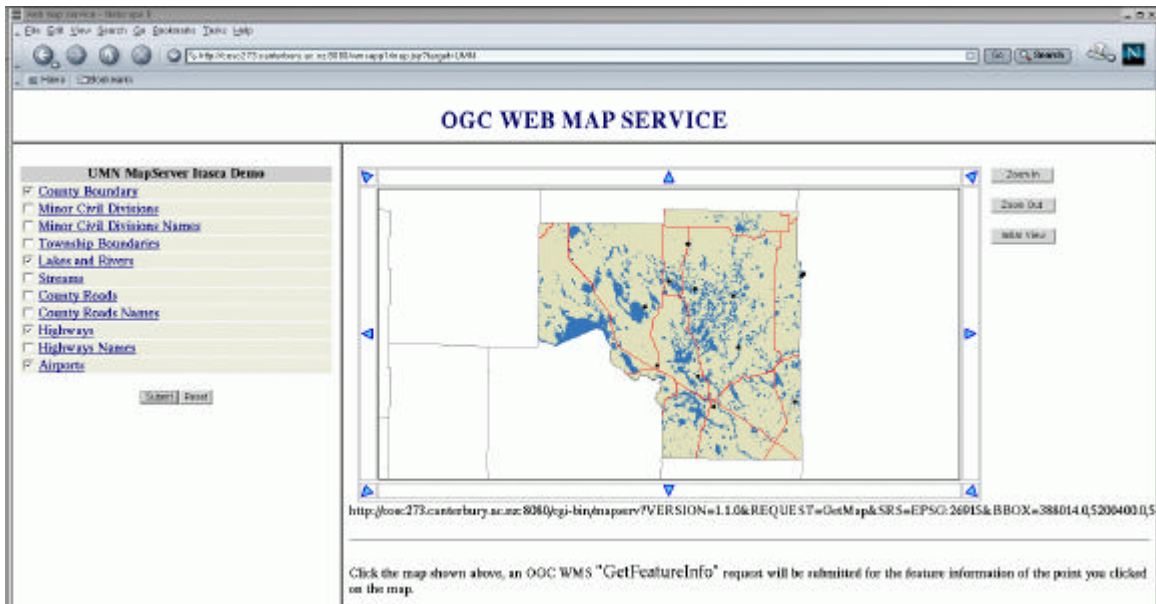

Figure 6: Index page
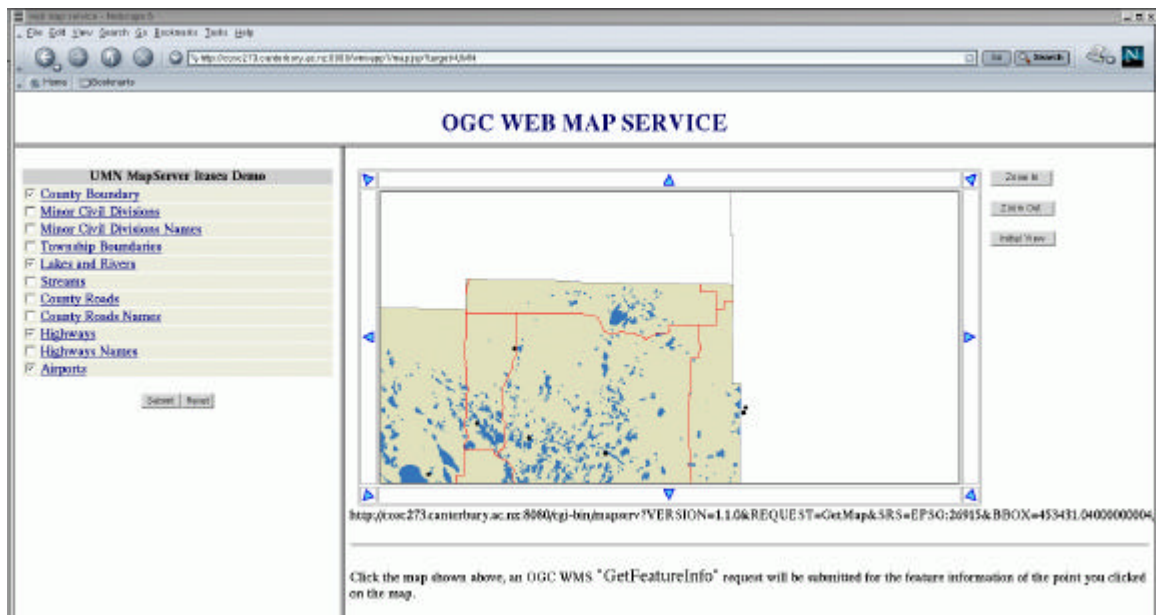
Figure 7: Layer list and map viewer

Figure 8: Zoom in and scroll the map

8.0   CONCLUSIONS AND FUTURE WORK

The primary comparison of the three technologies (JSP, JSTL and XMLC) is summarised in Table 1, which lists both some advantages and disadvantages of the three technologies. These factors affect the efficiency of the OGC WMS client application implementation, including web page interface design, realization of OGC WMS functionalities, and code debugging. Some of our findings during the implementation of the first task using the three technologies are summarised below.

For the XML/GML parsing issue, JSTL's XML tags enable XML documents to be parsed within web page to simply display XML-based data, but manipulation of those data still needs to be performed in back-end Java programs to prevent page complexity. When using traditional JSP and XMLC, all XML documents parsing tasks are performed in back-end Java programs.

Regarding map handling, one client request can retrieve a map with multiple layers from a single map server, and the order of the layers displayed in the map is determined by the order of the layer names in the list that

appeared in the request. Requesting such an image map and placing it into a web page are very simple tasks using each of the three approaches. Zooming in/out or scrolling through a map is realized by changing the Bounding Box values and submitting a new request, which are done in the back-end Java program. The map can then be inserted into the page using the query URL as the image resource. This strategy can be used in all three approaches.

Concerning interface layout, compared with JSP's mix of Java and HTML, JSTL makes the web page cleaner by using standard tags instead of Java code to control dynamical content. However, some functions such as method calling with arguments, which can easily be achieved using Java codes, are not supported in JSTL expression language. XMLC separates all data control from the page; the disadvantages of JSP and JSTL have mostly disappeared in XMLC.

In addressing the execution speed issue, a slight delay is encountered when a user requests a JSP page for the first time, because JSP and JSTL have to compile the JSP page before processing the request. In contrast, XMLC parses and interprets the page prior to run-time. The difference in the speed of request handling and page loading among the clients developed using the three approaches is very subtle.

Table 1: Primary comparison of JSP, JSTL, and XMLC

	JSP	JSTL	XMLC
Web page compilation	At run time	At run time	Prior to run time
Dynamical data control	Within the page	Within the page	Within Java program
Markup language and Java code	Mixed	Separated	Separated

Future work will include utilising the three approaches in implementing more complex tasks as described in section 5.0. Several main issues will be explored according to the evaluation criteria described in Section 4.0, including:

? How to make use of JSTL's XML tags in a WMS capabilities document and in GML document parsing and display?
? How do the three approaches deal with multiple overlapping image maps?
? Does one approach facilitate more than the others development of clients that simultaneously interact with multiple web map servers with different extent information?
? How does back-end business logic Java code support front-end web map interface development in the three approaches?
? Does one approach allows for easier debugging of clients during development?

The aims are to highlight advantages and disadvantages of three approaches (JSP, JSTL and XMLC) to help OGC WMS client application developers match appropriate technology to various client functionalities implementation.

REFERENCES

Bayern, Shawn (2002) JSTL in Action. August 2002, Manning Publications Co. Chapter 1 – Chapter 2

Bergsten, Hans (2002) JavaServer Pages, 2nd Edition, August 2002, O'Reilly, Chapter 3.

Enhydra XMLC (2002) Homepage of Enhydra XMLC. <http://xmlc.engydra.org>.

Open GIS Consortium Inc. (2001) Web Map Service Implementation Specification 1.1.0. *Open GIS project document: OGC 01-047r2*, June 2001.

Open GIS Consortium Inc. (1999) OpenGIS Simple Feature Specification For SQL Version 1.1. *Open GIS project document 99-049*, May 1999.

Sun Microsystems, Inc. (2002) JavaServer Pages™ Standard Tag Library (JSTL) Specification Version 1.0. March 2002.

WebReference.com (2000) Dynamically generating HTML pages with XMLC. November 9, 2000.
<http://webreference.com/xml/column23/>