Australian Government
Department of Defence
Defence Science and
Technology Organisation

DSTO

Investigating Web Map Service Client Technologies

Joel Blyth, Sean Geoghegan and
Damian O'Dea

DSTO-TN-0536

Australian Government
Department of Defence
Defence Science and
Technology Organisation

Investigating Web Map Service Client Technologies

Joel Blyth, Sean Geoghegan and Damian O'Dea

Command and Control Division
Information Sciences Laboratory

DSTO-TN-0536

ABSTRACT

Web Map Services may provide some assistance in alleviating the high cost of providing Geospatial Information System (GIS) capabilities to Defence users. This report discusses the development of several implementations of client applications of the Web Map Service interface. Problems encountered in the use of various Web technologies to construct the client applications are detailed. Suggestions are also made for future development efforts to improve client functionality.

Published by

DSTO Information Sciences Laboratory
PO Box 1500
Edinburgh South Australia 5111 Australia

Telephone: (08) 8259 5555
Fax: (08) 8259 6567

AR-013-009
January 2004

APPROVED FOR PUBLIC RELEASE

Investigating Web Map Service Client Technologies

Executive Summary

The cost of providing all Joint Command Support Environment (JCSE) users with a Geospatial Information System (GIS) capability and access to supporting data sets is prohibitively high. Web-based mapping services provide a cost saving by isolating GIS processing capabilities on a server and presenting the completed map product to a user as an image. Web mapping systems typically consist of one or more servers that provide map images to a range of client applications.

The investigation of different Web technologies in the development of Web Map Service (WMS) clients was undertaken under the auspices of the DSTO Edinburgh Vacation Student program.

Web technologies have evolved greatly from static HMTL-encoded pages, and each of these technologies offers different capabilities to develop client applications. In almost all cases the bulk of the computation of the client-server interaction is conducted in a user agent such as a Web browser. Determining the most suitable technology to be applied to the development of a client application requires studying the benefits that each approach presents.

In this effort, client applications were developed using a range of technologies. No formal experimentation was conducted to gather data regarding the merit of one technology over another. Instead the student conducting this implementation study made an assessment of the ease of applying each technology to the problem of developing a WMS client.

An assessment of the viability of applying the different technology to this problem domain is made in the conclusion.

Sample source code from two client applications is included as appendices. No formal design information is recorded.

Contents

1. Introduction

The concept of a Web Map Service (WMS) [1] has been developing for several years now, and today one effort is controlled and guided by the Open GIS Consortium (OGC). The OGC comprises of more than 230 companies, government agencies[1] and universities that are involved in a consensus process to develop publicly available interoperable geoprocessing specifications. Open interfaces and protocols defined by Open GIS Specifications offer solutions that "geo-enable" the Web, that is enable geographic maps to be displayed in a web browser, based upon criteria specified by a user.

The aim of this report is to discuss the available technologies and the methods for building web based mapping clients. A focus is placed on classifying the clients into groups, based upon the difficulty of development related to their technology, the problems encountered during development and how easy is it to implement varying degrees of functionality. It also provides a guideline to help developers to determine the most suitable technology to build web mapping clients.

2. Background to Web Map Services

A simple WMS consists of a client (typically a web browser interface) that sends requests to a server in the form of a Uniform Resource Locator (URL) [2]. The OGC WMS specification standardises the way in which clients request maps and the way that servers describe their data structure. Currently there are three operations, the first two of which are mandatory for any WMS implementation.

1. **GetCapabilities** – Obtain service-level metadata in the form of an extensible mark-up language (XML) [3] document that describes the content of WMS servers and acceptable request parameters.

2. **GetMap** – Obtain a map image whose geospatial and other parameters are well defined by the user.

3. **GetFeatureInfo** – Ask for information about particular features shown on a map.

The content of the URL submitted to the server depends upon the current activity of the client. If the client needs to know the content of the server then a *GetCapabilities* request is issued. This request returns a XML document with the available data. An example of a *GetCapabilities* request is below.

---

[1] DSTO has access to the OGC through the technical membership held by DIGO on behalf of Australian Defence.

e.g. http://demo.cubewerx.com/demo/cubeserv/cubeserv.cgi?Request=GetCapabilities&Service=WMS

If the user required a map to be generated within their browser then a *GetMap* request is submitted. The required parameters are summarised in Table 1. An example of a *GetMap* request is below.

e.g. http://demo.cubewerx.com/demo/cubeserv/cubeserv.cgi?Layers=
BARRIERL_1M:Foundation&SRS=EPSG:4326&BBox=-180,-90,180,90&Width=600&Height=400&
Format=jpeg&Styles=&Transparent=true&Version=1.1.1&Request=GetMap

Request Parameter	**Description**
VERSION=version	Request Version
REQUEST=GetMap	Request name
LAYERS=layer_list	Comma-separated list of one or more map layers.
STYLES=style_list	Comma-separated list of one or more styles.
SRS=namespace:identifier	Spatial Reference System (SRS).
BBOX=minx,miny,maxx,maxy	Bounding box corners (lower left, upper right) in SRS units
WIDTH=output_width	Width in pixels of map picture
HEIGHT=output_height	Height in pixels on map picture
FORMAT=output_format	Output format of map.

Table 1: Required Parameters of a GetMap Request

If the *GetMap* request is incorrectly formulated or requests a service that is unavailable, the server will return an exception in the form of an image or as an XML document. An optional *GetMap* request parameter can specify client preference for exception reporting.

The version number is an important factor in the URL request. It relates to the version of the WMS specification to which the client expects that the server will comply. The specification has been modified and updated several times, so it is essential that the correct version be selected. If the server is unable to support the requested version of the interface there is provision for negotiation of a mutually acceptable version between the client and server.

Given the two interface requests, a common role of the client is to simplify the creation of the URL requests that are sent to the server. A user should have the ability to easily select the parameters that are required and send the URL to the server without having to assemble it manually.

To find more details about the Web Map Service implementation specification see http://www.opengis.org/techno/implementation.htm.

3. Types of Clients

This section explores five different types of clients, namely *simple link*, *HTML form*, *servlet*, *JavaServer Pages* (JSPs), and *applet*. The types of client are characterised by the technology used to build them and the functionality they present to the user. Each implementation is characterised by a description of the technology and methods undertaken to implement a WMS client. Additionally, the functionality developed for the client and problems encountered in the implementation are reported.

3.1 Simple link

A link version of a WMS client can range from:

- a user typing in a request URL into the address bar of a browser,
- multiple links to services on a web page,
- thumbnails of images within a HTML page issue the appropriate request when selected.

3.1.1 Available Functionality

This method provides minimal functionality as the user may find it difficult or even impossible to change the parameters of the map. Presentation to the user of the link as a simple word (*e.g.* "map") or thumbnail image hiding the actual URL prevents the user from (anything but manually) configuring the URL. Complex URLs in the location bar of a web browser are not readily understandable and hence difficult to modify and customise. The URL for each map is fixed on the web page, and must be manually modified to customise the map displayed.

3.2 HTML Form

A form is a section of a HTML document containing not only normal content and mark-up but also special elements called controls (checkboxes, radio buttons, menus, etc.) that have labels. Users generally "complete" a form by modifying its controls (entering text, selecting menu items, etc.), before submitting the form to the server for processing. The content of the controls make up the content of the URL that is sent to the server.

3.2.1 Methods Undertaken

A simple HTML page with two frames has been developed. One frame contains a form and its controls, and the other provides browser space where the returned map is displayed (commonly known as a target frame).

A user types/selects different values for the controls and submits the form to the server by clicking a button labelled "Get Map". The values of the controls make up the

parameters of the *GetMap* request URL. The server returns a map, based on the input parameters in the opposite frame.

3.2.2 Available Functionality

There is minimal functionality because the user has to know what the available parameters of the map server are, before filling out the form. If one of the parameters is incorrect the server will either return an exception or provide an image that the user was not expecting.

3.2.3 Problems Encountered

A problem that was encountered with HTML forms was that some user agents (*i.e.* web browsers) silently performed URL encoding notation and certain web map servers were unable to deal with (or elected not to support) the notation.

The encoding notation replaces reserved characters with a three-character representation. This is a percent sign and two hexadecimal digits whose value corresponds to the position of the character in the ASCII character set. For example a space is converted to %20 or a comma is converted to %2C etc. The encoding is described in detail in [2].

The OGC specification states that a comma must separate each layer of the map request and each bounding box value. When sending a request from a form, these comma characters are replaced with their encoding by the user agent. In the event of the server not recognising the notation, it does not recognise the notation as a list element delimiter. When a server is incapable of dealing with these notations, an exception is returned for an otherwise legitimate request.

It was found that many of the web-mapping servers had very strict conformance to the OGC Implementation Specification and did not allow for the ability to deal with the encoding. One server that conformed strictly to the specification was CubeWerx. When trying to access a CubeWerx server from a form, an exception is always returned from the server stating that the bounding box's values are incorrect. The ESRI WMS connector to the ArcIMS server was capable of supporting requests from a HTML form. The ESRI server correctly parses the encoded URL and forwards the request on to the ArcIMS server for processing.

The OGC membership is aware of the issue with the WMS Version 1.1.1 and seeks to resolve it by revision of the relevant sections of the specification.

3.3 Servlet

A servlet is a server-side software component, written in Java, that dynamically extends the functionality of a web server. Unlike applets, servlets do not display a graphical interface to the user. A servlet's work is done "behind the scenes" on the server and only

the results of the servlet's processing are returned to the client (usually in the form of HTML).

The motivation to use a servlet is to provide dynamic web pages. In this case the available layers, bounding box values and other attributes of the map are automatically generated when the web page is first opened. This is done by parsing the document that is returned by the server in response to a *GetCapabilities* request, as discussed in Section 4.1.

For a servlet to generate HTML, it requires a Java PrintWriter within the Java source file, for example:

```
res.setContentType("text/HTML");
PrintWriter out  = res.getWriter();
out.println("<HTML>");
out.println("<BODY>");
out.println(layerArray[1]); // an array that contains the layers
out.println("</BODY>");
out.println("</HTML>");
```

A more complete listing of the code developed is included in Appendix A.2.

3.3.1 Methods Undertaken

Due to the fact that a servlet is difficult to modify because of the integration of Java and HTML code, it was found that it was best to first create and set-up the HTML page using a web site design program, with placeholders inserted into controls of the page for the WMS parameters.

After completing the design of the page the HTML code is copied into the servlet source file. Dynamic content generated by parsing the *GetCapabilities* XML document, replaced the placeholder elements.

3.3.2 Available Functionality

Servlets are capable of supporting all the functionality discussed in Section 4. Modifying an existing servlet leads to several problems as discussed below.

3.3.3 Problems Encountered

Changing the look and feel of the application, or adding support for a new type of client, requires the servlet code to be updated, recompiled and then redeployed on the server, which is an intensive process.

Taking advantage of web-page development tools when designing the application interface is difficult. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code. This process is time consuming and error prone.

3.4 JavaServer™ Pages (JSP)

JavaServer™ Pages [4] is another technology designed for the development of web based applications and is based on the servlet technology. The main distinction between a JSP and a servlet is that a JSP is an HTML page with Java code embedded within HTML tags, instead of a servlet being Java code that generates the web page. On initial access the JSP is converted into servlet source code and then compiled.

This distinction greatly simplifies the modification of the client, as a web page development tool can be used to create the web page. The HTML code can still be manipulated within the document, and that need not impact upon the Java code embedded within it.

3.4.1 Methods Undertaken

The JSP was created using a web page development tool. The design and layout of the web page is initially created, and then the dynamic content (as explained in Section 4.1) is added after by inserting the Java code that creates it.

Two methods were considered for the generation of the dynamic content: an Enterprise JavaBean™ (EJB, or simply Enterprise Bean) [5] and a standard Java Class. Both parse the *GetCapabilities* XML document and return the available parameters from the server by calling methods on them.

Due to the constraints of the unavailability of an EJB container and limited time there was no attempt to implement an EJB solution. Nonetheless there are theoretical benefits to such an approach and as such we describe it in detail below.

An EJB is a middle-tier component that encapsulates the business logic of an application. The business logic is the code that fulfils the purpose of the application. EJBs are implemented as Java classes that require a *container* within which to execute. The container intercepts requests made to and from the Enterprise Beans and automatically provides features such as transaction management, persistence and security. Requests are forwarded onto the Enterprise Beans by the container on behalf of the client making the request.

For several reasons, Enterprise Beans simplify the development of large, distributed applications. First, because the EJB container provides system-level services to Enterprise Beans, developers can concentrate upon solving business problems.

Second, because the beans and not the clients contain the applications business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules. As a result the clients are thinner.

3.4.2 Available Functionality

JSPs are capable of supporting all the functionality discussed in Section 4. What is ideal about using a JSP is that further functionality can be added without major code modifications such as a servlet. The HTML, Java and JavaScript code can be modified and tested within one file.

3.4.3 Problems Encountered

There were no significant problems encountered when implementing the Java class for the JSP version of the client. The problems faced in developing the servlet version were overcome because all the code could be modified and tested within one file. This made it very easy to build, deploy and maintain.

3.5 Applet

Applets are Java programs that are downloaded by web browsers and executed by a Java Plug-in within the web browser window. The full Java API is available to the programmer including the Graphical User Interface (GUI) Toolkits. This allows more complex user interfaces to be built than those available in standard HTML.

Applets differ from the other technologies in that all of the processing is performed on the client. While this can reduce server load, the processing requirements of the client are greatly increased when compared to HTML and JavaScript. On a slow client the time taken to start an applet can be unsatisfactory.

3.5.1 Methods Undertaken

The applet was developed using the Model-View-Controller [6] design pattern. Firstly, some classes representing the capabilities XML document were developed. These classes, along with the *GetMap* URL became the model part of the design pattern. The controllers would then manipulate the *GetMap* URL based on user input and the contents of the capabilities classes. When notified, the map display would update the map image by making a request to the WMS server using the modified URL.

3.5.2 Available Functionality

Applets are capable of supporting all the functionality discussed below. With a flexible design it is relatively easy to add new features.

3.5.3 Problems Encountered

The applet security model prohibits opening network connections to servers from which the applet did not originate. This poses a problem for a WMS client, which needs to be able to connect to a number of different servers to be truly useful. To solve this problem

the applet needs to be cryptographically signed using a digital certificate obtained from a Certificate Authority.

Applets depend on the Java Virtual Machine (JVM) for execution and this needs to be installed on each client machine. Additionally, not all Java features are available on all versions of the Java Plug-in. If an applet requires features not provided by the client Plug-in it needs to either provide the libraries itself or prompt the user to install the required Plug-in version.

In a configuration managed environment it may not be possible for the user to install an up-to-date JVM, so often the additional libraries will need to be provided for download from the applet's website. In some cases this can drastically increase the amount of code that needs to be downloaded and the time it takes for the applet to start on the client machine.

4. Functionality of Clients

4.1 Dynamic Content

Dynamic Content of a web page is defined as content that changes every time it is accessed. In the case of a web-mapping client, the dynamic content would be the data the client is able to access on the server and the restrictions that are placed on that data. e.g. available layers, minimum bounding box etc.

Dynamic content is created by determining the data available on the server and its structure. This is done by sending a *GetCapabilities* request to the server, which returns a XML document. The client is then able to parse the XML document and determine the HTML content to fabricate.

There are two types of XML parsers available, SAX (Simple API for XML) [7] and DOM (Document Object Model) [8]. The two types of parser provide applications with access to the information stored in XML documents. They both provide an API to access a document's structure (its markup tags) and content (the marked up information).

Each parser takes a very different approach towards accessing information. DOM creates a tree of nodes (based on the structure and information in the XML document) and can be accessed by traversing the tree. The textual information in the XML document is converted to leaf nodes. SAX, on the other hand, notifies the application of what is in the document utilising a stream of events to represent document elements. Either style of parser may validate the XML document against a schema represented as either a Document Type Definition (DTD) [9] or XML Schema [10].

Validation of the XML document returned from a WMS compliant server is unnecessary. It is reasonable to assume that any such server will respond to the *GetCapabilities* request with a valid XML document.

When parsing a XML document with a validating parser it is important to first determine which version of the WMS the server is implementing. The version determines which schemata the XML document conforms to and as such this will how the program parses the XML document. The WMS specification defines how version number negotiation should be carried out between a client and a server.

4.2 Zooming and Panning

Zooming and Panning are essential functions in web mapping, as it allows the user to tune the image displayed to suit their purposes.

In a HTML form version it requires an abundance of JavaScript. JavaScript is lightweight client side programming language that is usually embedded directly in HTML pages to add interactivity and power to web documents.

In the HTML implementation of the WMS client it was found that implementing the zoom box using JavaScript was a difficult task. It involved using multiple nested dividers (HTML elements), one that contained the map and the other was blank, but had a style sheet that gave it a border that represented the zoom box. The sample code for setting the zoom box, which is described further below, can be found in Appendix A.1.

When a user clicks a mouse over the map display, the setBorder() function is called to set the zoom box divider to become visible. The top and left attributes are set to equal to where the user clicked. As the mouse is moved over the map the mouseMoveHandler() function refreshes the dividers' width and height. A second mouse click sets the divider to be hidden again, and the position and the dimensions of the divider are utilised as the new bounding box values.

The positions of the mouse clicks are initially stored as the position of the mouse over the image in pixels, based upon the image size. These are converted to the coordinate system of the map, by calculating proportions from the original bounding box coordinates.

5. Other Problems when Building Clients

Other problems encountered when building and deploying WMS clients are summarised below. These problems are unrelated to how a client is specifically built.

5.1 Document Type Definitions

It is recommended that the capabilities document DTDs be stored on the local intranet. These DTDs are required by WMS clients to validate the result of the *GetCapabilities* request. If a client attempts to access a WMS service hosted locally and requires access to DTDs hosted outside the intranet that are unavailable, the XML cannot be validated and the client's parser would normally throw an exception.

Even in the event of a WMS client using a non-validating XML parser, it is necessary for the DTDs to be available to the parser, as they are loaded as a parser artefact. This has implications for the use of commercial WMS clients on Defence networks that do not have Internet connectivity. Using such clients requires that the DTDs location be masked or aliased in the domain naming service[2], or modification to the client code to circumvent hard coding of the DTDs' location.

5.2 JavaScript

JavaScript is used for most of the functionality of the HTML implemented WMS client. The JavaScript languages interpreted by Internet Explorer ("Jscript") and Netscape ("JavaScript") differ, so a programmer has to take this into account and determine what browser is being used and then apply the appropriate code. JavaScript has very poor debugging facilities and documentation so it is very hard to develop with no related experience.

Efforts to standardise the JavaScript language into a common language ECMAScript [11] may provide some hope in easing the development process.

5.3 Tomcat Server

Our WMS servlet clients were tested using the Tomcat servlet engine. It was discovered that the proxy settings for the servlet engine overrode the automatic proxy settings for web browsers in the development environment. Attempts to access WMS services hosted on internet servers outside the firewall were denied by settings permitting access to intranet hosted services. Each time a service was required outside the network, the proxy settings of the server had to be changed and then restarted. A means of configuring a solution into Tomcat to connect directly when accessing servers within the local network has not been discovered.

5.4 Image Size to Bounding Box Ratios

The image size ratio (width/height) to bounding box ratio (latitude/longitude) can create problems if not approximately the same. If the ratios vary by a large margin, the resulting image could be stretched, compressed or filled out with a background.

---

[2] Replicating DNS names in Intranet environments is not a recommended process.

When a server returns an image that has been padded out with excess background (strips of non-data), it is indicative of the bounding box ratio differing from the image size ratio. When this occurs the bounding box values calculated for zooming in or out become incorrect. When the user clicks on the image to start the zoom box, the position of the click is transformed from image pixels to coordinates. If the coordinates of the image corners do not correspond to the bounding box values the zoom function will be incorrect.

A solution is to determine the ratio of the bounding box when the *GetCapabilities* document is initially parsed. The map images width and height within the HTML document can then be adjusted to suit the ratio of the bounding box.

6. Future Work

6.1 Multiple Services

At the time of writing this document no implementation of client-side composition of the output from multiple services has been completed. The composition of output from multiple services means that map layers can be provided by distinct servers offering divergent data sources. The value of combining these data sources lies in being able to access data sourced from organisations specifically dedicated to the product they offer.

> *E.g. A client may seek to overlay current weather pattern information upon a reference map of an Operational Area.*

When requesting multiple layers from different services it is necessary to parse each server's *GetCapabilities* document separately, and then generate the dynamic HTML code. The implementation of a WMS client may need to consider mechanisms for layer management that allows a user to specify the order in which layers are displayed.

6.2 Scribble Layer

A "scribble layer" would be a great asset to have. The user would be able to add their own text, diagrams and other drawing objects similar to a Windows Paint program and then save the image in the desired format. Map annotations could be added to customise the server output to the purpose of a user.

6.3 Styled Layer Descriptor Enabled Client

Styled Layer Descriptors (SLDs) [12] allow a client to tell a server how it wants a layer to be rendered. This allows a user to customise the appearance of data, and to classify data based on their interests and see those classifications as graphical differences. Given the

alternative is for the customer to request that a map service provider modify the service itself, the advantage of the SLD-enabled WMS approach is clear.

SLDs also allow greater knowledge of the map on the client side. Instead of the client requesting a map and receiving an image defined by the service provider's conventions, it could know the exact details of the symbology and appearance of the map, allowing for greater control in construction of legends, user tools and other client features.

As services supporting SLDs become more numerous, clients that exploit this capability will need to be considered as the benchmark, rather than the exception.

7. Conclusion

The aim of this report was to discuss the available technologies and the methods for building web based mapping clients. The WMS specification provides a rigorous interface for a client to request maps from a service. Such rigour is beneficial in that it allows different (vendor's) client implementations to interact with services from different providers. Additionally clients can expect services to behave in a well-known manner, issues such as described in 3.2.3 notwithstanding.

Supplemental specifications in the Open GIS Web Services architecture space extend the limited expressiveness of the WMS interface. Exploitation of this additional capability extends the functionality available in WMS clients.

The technology used to implement WMS clients is broadly varied. Development of a WMS client, due to the complexity of some user interface operations (*e.g.* expressing pan and zoom actions within a user agent), benefits from the more sophisticated technologies applicable. There is a role for simple technologies, such as HTML-based clients, that provide all the complexity required to implement "thumbnails" or overview maps, without the computational overhead of the more advanced technologies.

References

1. Bruce, C., et al., *Web Map Service Interface Specification*, J.d.L. Beaujardière and A. Doyle, Editors. 2002, Open GIS Consortium Inc.
2. Berners-Lee, T., et al., *Uniform Resource Identifiers (URI): Generic Syntax*. 1998, The Internet Society: Reston, VA, USA.
3. *Extensible Markup Language (XML) 1.0 (Second Edition)*, T. Bray, et al., Editors. 2000, The World Wide Web Consortium (W3C).
4. Roth, M. and E. Pelegrí-Llopart, *JavaServer Pages™ Specification Version 2.0 (Proposed Final Draft 3)*. 2003, Sun Microsystems, Inc.: Santa Clara, USA. p. xxxiv+426.
5. DeMichiel, L.G., *Enterprise JavaBeans™ Specification, Version 2.1*. 2003, Sun Microsystems, Inc.: Santa Clara, California. p. 636.
6. Gamma, E., et al., *Design Patterns: Elements of Reusable Object-Oriented Software*. First ed. 1995, Reading, MA, USA: Addison Wesley Professional. xv+395.
7. Brownell, D., *SAX2*, ed. M. Brady. 2002, Beijing etc: O'Reilly. 240.
8. *Document Object Model (DOM) Level 2 Core Specification Version 1.0*, in *Document Object Model (DOM) Technical Reports*, A.L. Hors, et al., Editors. 2000, The World Wide Web Consortium (W3C).
9. McLaughlin, B., *Java and XML*. First ed. 2000, Beijing: O'Reilly. xv+509.
10. *XML Schema Part 0: Primer*, D.C. Fallside, Editor. 2001, The World Wide Web Consortium (W3C).
11. *ECMAScript Language Specification 3rd edition*. 1999, ECMA: Geneva, Switzerland. p. 188.
12. *Styled Layer Descriptor Implementation Specification*, W. Lalonde, Editor. 2002, Open GIS Consortium Inc.

Appendix A: Source Code Examples

A.1. JavaScript Sample Code

The code below demonstrates one mechanism for achieving the Zoom functionality for an HTML form-based client.

```
<SCRIPT>
// global variables
var initMouseX;  // inital x mouse position on page
var initMouseY;  // inital y mouse position on page
var finalMouseX;// final x mouse position on page
var finalMouseY;// final y mouse position on page
var initPixX;      // intial x position on image (in pixels)
var initPixY;      // intial y position on image (in pixels)
var finalPixX;          // final x position on image (in pixels)
var finalPixY;          // final y position on image (in pixels)
var firstClick = true;  // has the first click on the map happened
var isMoving = false; // is the mouse moving after the first click
var imgWidth = 600;  // image width
var imgHeight = 300; //image height
var initCoordx = 0;    // coordinates of first click on the map
var initCoordy = 0;
var finalCoordx = 0;  // coordinates of second click on the map
var finalCoordy = 0;
zoomin = true;
zoomout = false;
```

```
recentre = false;

// Bounding box values
var xmin = -180;
var ymin = -90;
var xmax = 180;
var ymax = 90;

// the differences between x min and x max, y min and y max
var xdiff = 0;
var ydiff = 0;

/*****************************************************************************************************************************************************/
/*   goZoomOut() - used to calculate the new coordinates of the bounding box for a zoom out.*/

function goZoomOut() {
xdiff = xmax - xmin;
ydiff = ymax - ymin;
var centrePixX = event.offsetX;  // determine the centre of the new map based upon where the user clicks (in pixels)
var centrePixY = event.offsetY;
var centreCoordx = xmin+((centrePixX/imgWidth)* xdiff); // convert pixels into coordinates
var centreCoordy =  ymax -((centrePixY/imgHeight)* ydiff);
var upLeftCoordx = centreCoordx - xdiff; // determine upper left coordinates
var upLeftCoordy = centreCoordy + ydiff;

// calculate minx, miny, maxx, maxy for the new bounding box
```

```
var newBBox = (roundAmount(upLeftCoordx)+","+roundAmount(upLeftCoordy - ydiff*2)+","+roundAmount(upLeftCoordx +
xdiff*2)+","+roundAmount(upLeftCoordy));

// set the BBOX text box in the right frame to the BBOX value
top.rightFrame.form2.BBox.value= newBBox;

// call the setURL method in the rightframe
top.rightFrame.setURL();
}

/*****************************************************************************************************************************************************/
/*   reCentre() - used to calculate the new coordinates of the bounding box for a zoom out. */

function reCentre() {
xdiff = xmax - xmin;
ydiff = ymax - ymin;
var centrePixX = event.offsetX; // determine the centre of the new map based upon where the user clicks (in pixels)
var centrePixY = event.offsetY;
var centreCoordx = xmin+((centrePixX/imgWidth)* xdiff); // convert pixels into coordinates
var centreCoordy =  ymax -((centrePixY/imgHeight)* ydiff);
var upLeftCoordx = centreCoordx - (0.5*xdiff);  // determine upper left coordinates
var upLeftCoordy = centreCoordy + (0.5*ydiff);

// calculate minx, miny, maxx, maxy for the new bounding box
var newBBox = (roundAmount(upLeftCoordx)+","+roundAmount(upLeftCoordy - ydiff)+","+roundAmount(upLeftCoordx +
xdiff)+","+roundAmount(upLeftCoordy));
```

```
// set the BBOX text box in the right frame to the BBOX value
top.rightFrame.form2.BBox.value= newBBox;

// call the setURL method in the rightframe
top.rightFrame.setURL();
}

/*****************************************************************************************************************************************/
/*  actionOption() - after the images is click this method is called and determines which option is required and actions it. */

function actionOption() {

    if (zoomin) {
    setBorder();
    }
    else if (zoomout) {
    goZoomOut();
    }
    else { //measure
    reCentre();
    //alert("testing the waters");
    }

}

/*****************************************************************************************************************************************/
/*  loadImg() - used to hide the "LoadImage.gif" during load a image */
```

```
function loadImg() {
document.all.maxControls.style.visibility= "hidden";
}

/*****************************************************************************************************************************************/
/*   unloadImg() - used to make visible the "LoadImage.gif" during load a image */

function unloadImg() {
document.all.maxControls.style.visibility= "visible";
}

/*****************************************************************************************************************************************/
/*   roundAmount() - used to round a value to 2 decimals places */

function roundAmount(n) {
 var s = "" + Math.round(n * 100) / 100
 var i = s.indexOf('.')
 if (i < 0) return s + ".00"
 var t = s.substring(0, i + 1) +
    s.substring(i + 1, i + 3)
 if (i + 2 == s.length) t += "0"
 return t
}

/*****************************************************************************************************************************************/
/*   setBorder() - set the border of the zoomIn box */
```

```
function setBorder() {

if (firstClick) {
      initMouseX=event.clientX;
      initMouseY=event.clientY;
      document.all.Layer1.style.left = initMouseX; // dividers left is equal to the inital mouse click
      document.all.Layer1.style.top = initMouseY;    // dividers top is equal to the inital mouse click
      document.all.Layer1.style.visibility= "visible"; // make the zoomIn box visible (divider)
      document.all.Layer1.style.width = 1;                    // start the zoombox at a 1 pixel x 1 pixel
      document.all.Layer1.style.height = 1;
      firstClick = false;                                     // the first click has occured
      isMoving = true;                                        // the mouse is now moving
      initPixX = event.offsetX;                               // determine the pixel position after first mouse click
      initPixY = event.offsetY;
      }
else {
      isMoving = false;          // second click has occured - mouse is no longer moving
      firstClick = true;         // first click set to true - waiting for another zoomin
      finalMouseX = event.clientX ;
      finalMouseY = event.clientY;
      finalPixX = event.offsetX;              // determine the pixel position after second mouse click
      finalPixY = event.offsetY;
      document.all.Layer1.style.visibility= "hidden"; // hide the zoomIn box (divider)
      setCoordinates();                  // calculate the new coordinates of the BBox
      }
}
```

```
/*****************************************************************************************************************************************/
/*  setCoordinates() - used to determine the new coordinates of the BBox */

function setCoordinates() {
xdiff = xmax - xmin;
ydiff = ymax - ymin;

initCoordx = xmin+((initPixX/imgWidth)* xdiff);
initCoordy =  ymax -((initPixY/imgHeight)* ydiff);
finalCoordx = xmin+((finalPixX/imgWidth)* xdiff);
finalCoordy =  ymax -((finalPixY/imgHeight)* ydiff);

var newBBox = (roundAmount(initCoordx)+","+roundAmount(finalCoordy)+","+roundAmount(finalCoordx)+","
    +roundAmount(initCoordy));

// set the BBox text box in the right frame equal to the new BBox value
top.rightFrame.form2.BBox.value= newBBox;

// call the method in the right frame setURL which determines the new URL for the image
top.rightFrame.setURL();
}

/*****************************************************************************************************************************************/
/*  mouseMoveHandler() - used to set the height and width of the zoomIn box (divider)        when the mouse is moving.  It also makes sure
the zoomIn box is within the the image. */
```

```
function mouseMoveHandler() {
    if (isMoving) {
        // do not allow x values to be negative
        if (event.clientX < initMouseX) {
        document.all.Layer1.style.width = 1
        document.all.Layer1.style.height = 1
        }
        else {
            // code used to keep the box within the image.
            // clientX has to be within the imgWidth + the position of the image on the page
            if (event.clientX > imgWidth + (event.clientX + document.body.scrollLeft)) {
            document.all.Layer1.style.width = imgWidth - initMouseX;
            }
            else {
            document.all.Layer1.style.width = event.clientX - initMouseX;
            }

        }
        // do not allow y values to be negative
        if (event.clientY < initMouseY) {
        document.all.Layer1.style.width = 1
        document.all.Layer1.style.height = 1
        }
        else {
            if (event.clientY > imgHeight + (event.clientY + document.body.scrollTop)) {
            document.all.Layer1.style.height = imgHeight - initMouseY;
            }
```

```
            else {
            document.all.Layer1.style.height = event.clientY - initMouseY;
            }
        }
    }
}

</SCRIPT>
```

A.2. Servlet Source Code

The following source code was developed to investigate the utility of a servlet implementation of a WMS client. A description of the client can be found in section 3.3 and subsections.

```
import java.io.*;
import java.text.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.net.*;
```

```
public class dmsolutions extends HttpServlet {

    public void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {

        String[] getMapPara = new String[5];
        getMapPara = xmlParse();

        //Details of the getMapPara array
        /*getMapPara[0] = Layers; getMapPara[1] = SRS; getMapPara[2] = BBox; getMapPara[3] = Format; getMapPara[4] = Exceptions;*/

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        out.println("<HTML><HEAD><TITLE>GetMap Test Template</TITLE>");
        out.println("<META http-equiv=Content-Type content=\"text/html; charset=windows-1252\">");
        out.println("<META content=\"MSHTML 5.50.4134.600\" name=GENERATOR></HEAD>");
        out.println("<BODY bgColor=#CCCCCC>");
        out.println("<DIV align=center><FONT color=black size=6><H7>GetMap Template <BR>");
        out.println("</font>");
        out.println("<form name=\"form1\" action=\"\">");
        out.println("<TABLE cellSpacing=2 cellPadding=2 align=center border=0 width=\"266\">");
        out.println("<TBODY> ");
        out.println("<TR> ");
        out.println("<TD width=\"122\">LAYERS</TD>");
        out.println("<TD width=\"130\"> ");
        out.println("<select name=\"Layers\" multiple size=\"5\">");
        out.println(getMapPara[0]);
```

```
out.println("</select>");
out.println("</TR>");
out.println("</TBODY> ");
out.println("</TABLE>");
out.println("</form>");
out.println("<FONT color=yellow size=6> </font> </DIV>");
out.println("<FONT color=yellow size=6>");
out.println("<SCRIPT LANGUAGE=\"JavaScript\">");
out.println("function getSelected(opt) {");
out.println("var selected = new Array();");
out.println("var index = 0;");
out.println("for (var intLoop = 0; intLoop < opt.length; intLoop++) {");
out.println("if ((opt[intLoop].selected) ||");
out.println("(opt[intLoop].checked)) {");
out.println("index = selected.length;");
out.println("selected[index] = new Object;");
out.println("selected[index].value = opt[intLoop].value;");
out.println("selected[index].index = intLoop;");
out.println("}");
out.println("}");
out.println("return selected;");
out.println("}");
out.println("function outputSelected(opt) {");
out.println("var sel = getSelected(opt);");
out.println("var strSel = \"\";");
out.println("var i = 0;");
out.println("for (var item in sel){      ");
```

```
out.println("strSel += sel[item].value;");
out.println("if (i<sel.length-1) {");
out.println("strSel += \",\";");
out.println("}");
out.println("i++;");
out.println("}");
out.println("form2.Layers.value= strSel;");
out.println("}");
out.println("</SCRIPT>");
out.println("<FORM name=\"form2\" target = \"mainFrame\" action =
    \"http://aldur.dsto.defence.gov.au/cgi-bin/dstomapserver.exe?\">");
out.println("<INPUT type=hidden value=\"\" name=Layers>");
out.println("<TABLE cellSpacing=2 cellPadding=2 align=center border=0>");
out.println("<TBODY>");
out.println("<TR>");
out.println("<TD>SLD</TD>");
out.println("<TD><INPUT name=SLD></TD></TR><!--- <tr>");
out.println("<td>SLD VERSION</td>");
out.println("<td><INPUT TYPE=\"RADIO\" NAME=\"SLDVersion\" VALUE=\"0.7.2\" >0.7.2</td>");
out.println("</tr> <tr>    <td></td>");
out.println("<td><INPUT TYPE=\"RADIO\" NAME=\"SLDVersion\" VALUE=\"0.7.1\" checked >0.7.1</td>");
out.println("</tr> <tr>    <td></td>");
out.println("<td><INPUT TYPE=\"RADIO\" NAME=\"SLDVersion\" VALUE=\"0.7.0\" >0.7.0</td>");
out.println("</tr>  --->");
out.println("<TR>");
out.println("<TD>SRS</TD>");
out.println("<TD>");
```

```
out.println("<select name=\"SRS\">");
out.println(getMapPara[1]);
out.println("</select>");
out.println("</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>BBOX</TD>");
out.println("<TD>");
out.println(getMapPara[2]);
out.println("</TD></TR>");
out.println("<TR>");
out.println("<TD>FORMAT</TD>");
out.println("<TD>");
out.println("<select name=\"Format\">");
out.println(getMapPara[3]);
out.println("</select>");
out.println("</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>TRANSPARENT</TD>");
out.println("<TD>");
out.println("<select name=\"Transparent\">");
out.println("<option value=\"false\">false</option>");
out.println("<option value=\"true\">true</option>");
out.println("</select>");
out.println("</TD>");
out.println("</TR>");
```

```
out.println("<TR>");
out.println("<TD>BGCOLOR</TD>");
out.println("<TD>");
out.println("<select name=\"BGColor\">");
out.println("<option value=\"0xffffff\">White</option>");
out.println("<option value=\"0xC0C0C0\">Silver</option>");
out.println("<option value=\"0x808080\">Grey</option>");
out.println("<option value=\"0x008000\">Green</option>");
out.println("<option value=\"0x000080\">Navy</option>");
out.println("</select>");
out.println("</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>EXCEPTIONS</TD>");
out.println("<TD>");
out.println("<select name=\"Exceptions\">");
out.println(getMapPara[4]);
out.println("</select>");
out.println("</TD>");
out.println("</TR>");
out.println("<TR>");
out.println("<TD>VERSION</TD>");
out.println("<TD>");
out.println("<select name=\"Version\">");
out.println("<option value=\"1.1.1\">1.1.1</option>");
out.println("<option value=\"1.1.12\">1.1.12</option>");
out.println("<option value=\"1.1.0\">1.1.0</option>");
```

```
        out.println("<option value=\"1.0.6\">1.0.6</option>");
        out.println("<option value=\"1.0.5\">1.0.5</option>");
        out.println("<option value=\"1.0.0\">1.0.0</option>");
        out.println("</select>");
        out.println("</TD>");
        out.println("</TR>");
        out.println("");
        out.println("<TD><INPUT type=submit value=\"Get Map\" ONCLICK= \"outputSelected(form1.Layers.options)\" ></TD>");
        out.println("<TD><INPUT type=reset value=Reset></TD></TR></TBODY></TABLE><INPUT type=hidden ");
        out.println("value=GetMap name=Request>");
        out.println("<INPUT type=hidden value=\"../msdemo/world1.map\" name=map>");
        out.println("<INPUT type=hidden value=600 name=Width>");
        out.println("<INPUT type=hidden value=400 name=Height>");
        out.println("<INPUT type=hidden value=wms_world name=ServiceName>");
        out.println("</FORM></FONT></BODY></HTML>");
        out.close();
    }

    public String[] xmlParse() {

        String[] getMapPara = new String[5];

        String Layers = "";
        String SRS = "";
        String BBox = "";
        String Format = "";
```

```
Document doc = parseXmlFile(false);

// Code Below Used to Get Layers, SRS and BBox
NodeList Layerlist = doc.getElementsByTagName("Layer");
Node first = Layerlist.item(0);
NodeList children = first.getChildNodes();

for (int i = 1; i < children.getLength(); i++) {
    Node theChild = children.item(i);

    if (theChild.getNodeName().equals("Layer")) {
        NodeList children2 = theChild.getChildNodes();
        Layers = Layers.concat(getLayers(children2));
    }

    if (theChild.getNodeName().equals("SRS")) {
        NodeList SRSNList = theChild.getChildNodes();
        Node SRSNode = SRSNList.item(0);
        SRS = getSRS(SRSNode);
    }

    if (theChild.getNodeName().equals("LatLonBoundingBox")) {
        String minx = ((Element) theChild).getAttribute("minx");
        String miny = ((Element) theChild).getAttribute("miny");
        String maxx = ((Element) theChild).getAttribute("maxx");
        String maxy = ((Element) theChild).getAttribute("maxy");
```

```
            String BBoxValues = minx + "," + miny + "," + maxx + "," + maxy;

            BBox = "<input type=\"text\" name=\"BBox\" value="+BBoxValues+">";
        }

}

// Code Below Used to Get Format
NodeList getMapNode = doc.getElementsByTagName("GetMap");
Node firstChildren = getMapNode.item(0);
NodeList getMapChildren = firstChildren.getChildNodes();

for (int i = 1; i < getMapChildren.getLength(); i++) {
    Node theMapChild = getMapChildren.item(i);

    if (theMapChild.getNodeName().equals("Format")) {
        NodeList formatChildren = theMapChild.getChildNodes();
        String formatChild = formatChildren.item(0).getNodeValue();
        Format = Format.concat("<option value=\""+formatChild+"\">"+formatChild+"</option>");
    }
}

// Code Below Used to Get Exceptions
NodeList getExceptionNode = doc.getElementsByTagName("Exception");
firstChildren = getExceptionNode.item(0);
NodeList getExceptChildren = firstChildren.getChildNodes();
```

```
        String Exceptions = "";

        for (int i = 1; i < getExceptChildren.getLength(); i++) {
            Node theMapChild = getExceptChildren.item(i);

            if (theMapChild.getNodeName().equals("Format")) {
                NodeList formatChildren = theMapChild.getChildNodes();
                String formatChild = formatChildren.item(0).getNodeValue();
                Exceptions = Exceptions.concat("<option value=\""+formatChild+"\">"+formatChild+"</option>");
            }
        }

        getMapPara[0] = Layers;
        getMapPara[1] = SRS;
        getMapPara[2] = BBox;
        getMapPara[3] = Format;
        getMapPara[4] = Exceptions;

        return getMapPara;

    }

    static String getLayers(NodeList children2) {

        String theName = "";
        String theTitle = "";
```

```
        for (int j = 1; j < children2.getLength(); j++) {

            if (children2.item(j).getNodeName().equals("Title")) {
                NodeList children3 = children2.item(j).getChildNodes();
                Node Title = children3.item(0);
                theTitle = Title.getNodeValue();
            }

            if (children2.item(j).getNodeName().equals("Name")) {
                NodeList children3 = children2.item(j).getChildNodes();
                Node Name = children3.item(0);
                theName = Name.getNodeValue();
            }
        }

        String layerNameTitle = "<option value="+"\""+theName+"\""+"selected>"+theTitle+"</option>";

        return layerNameTitle;
    }

    static String getSRS(Node SRSNode) {

        String SRS = "EPSG:4326";

        String SRSOptions = "<option value="+"\""+SRS+"\""+">"+SRS+"</option>";

        return SRSOptions;
```

```
}

// Parses an XML file and returns a DOM document.
// If validating is true, the contents is validated against the DTD
// specified in the file.
public static Document parseXmlFile(boolean validating) {
    try {
        // Create a builder factory
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(validating);

        // The URL where the GetCapabilities XML document is retrieved
        URL theURL = new URL("http://aldur.dsto.defence.gov.au/cgi-bin/dstomapserver.exe?map=" +
            "../msdemo/world1.map&SERVICE=WMS&REQUEST=GetCapabilities");

        // Create the builder and parse the file
        Document doc = factory.newDocumentBuilder().parse(theURL.openStream());
        return doc;
    }
    catch (SAXException e) {
        // A parsing error occurred; the xml input is not valid
    }
    catch (ParserConfigurationException e) {

    }
    catch (MalformedURLException e) {
```

```
    }
    catch (IOException e) {
        System.out.println("This file does not exist");
    }

    return null;

    }

}
```

Investigating Web Map Service Client Technologies

Joel Blyth, Sean Geoghegan and Damian O'Dea

AUSTRALIA

DEFENCE ORGANISATION

	No. of copies
Task Sponsor	
DGICD	1
S&T Program	
Chief Defence Scientist	shared copy
FAS Science Policy	
AS Science Corporate Management	
Director General Science Policy Development	
Counsellor Defence Science, London	Doc Data Sheet
Counsellor Defence Science, Washington	Doc Data Sheet
Scientific Adviser to MRDC, Thailand	Doc Data Sheet
Scientific Adviser Joint	1
Navy Scientific Adviser	Doc Data Sheet & Dist List
Scientific Adviser - Army	Doc Data Sheet & Dist List
Air Force Scientific Adviser	Doc Data Sheet & Dist List
Scientific Adviser to the DMO M&A	Doc Data Sheet & Dist List
Scientific Adviser to the DMO ELL	Doc Data Sheet & Dist List
Director of Trials	1
Information Sciences Laboratory	
Chief Command & Control Division	Doc Data Sheet
Research Leader Command & Intelligence Environments Branch	1
Research Leader Military Information Enterprise Branch	1
Research Leader Theatre Operations Analysis Branch	Doc Data Sheet
Head Virtual Enterprises	Doc Data Sheet
Head Systems Simulation and Assessment	Doc Data Sheet
Head Theatre Operations Analysis	Doc Data Sheet
Head Intelligence Analysis	Doc Data Sheet
Head Human Systems Integration	Doc Data Sheet
Head C2 Australian Theatre	Doc Data Sheet
Head HQ Systems Experimentation	Doc Data Sheet
Head Information Systems	Doc Data Sheet
Head Information Exploitation	1
Greg Chase (Task Manager) Information Exploitation C2D	1
Joel Blyth (Author) Information Exploitation C2D	3
Sean Geoghegan (Author) Information Exploitation C2D	3
Damian O'Dea (Author) Information Exploitation C2D	3
Publications and Publicity Officer, C2D/EOC2D	1 shared copy

DSTO Library and Archives

Library Edinburgh	1 copy & Doc Data Sheet
Australian Archives	1

Capability Systems Division

Director General Maritime Development	Doc Data Sheet
Director General Aerospace Development	Doc Data Sheet
Director General Information Capability Development	Doc Data Sheet

Office of the Chief Information Officer

Deputy CIO	Doc Data Sheet
Director General Information Policy and Plans	Doc Data Sheet
AS Information Structures and Futures	Doc Data Sheet
AS Information Architecture and Management	Doc Data Sheet
Director General Australian Defence Simulation Office	Doc Data Sheet

Strategy Group

Director General Military Strategy	Doc Data Sheet
Director General Preparedness	Doc Data Sheet

HQAST

SO (Science) (ASJIC)	Doc Data Sheet

Navy

SO (SCIENCE), COMAUSNAVSURFGRP, NSW	Doc Data Sht & Dist List
Director General Navy Capability, Performance and Plans, Navy Headquarters	Doc Data Sheet
Director General Navy Strategic Policy and Futures, Navy Headquarters	Doc Data Sheet

Army

ABCA National Standardisation Officer, Land Warfare Development Sector, Puckapunyal	e-mailed Doc Data Sheet
SO (Science), Deployable Joint Force Headquarters (DJFHQ) (L), Enoggera QLD	Doc Data Sheet
SO (Science) - Land Headquarters (LHQ), Victoria Barracks NSW	Doc Data & Exec Summ

Intelligence Program

DGSTA Defence Intelligence Organisation	1
Manager, Information Centre, Defence Intelligence Organisation	1 (PDF version)
Assistant Secretary Corporate, Defence Imagery and Geospatial Organisation	Doc Data Sheet

Defence Materiel Organisation	
Head Airborne Surveillance and Control	Doc Data Sheet
Head Aerospace Systems Division	Doc Data Sheet
Head Electronic Systems Division	Doc Data Sheet
Head Maritime Systems Division	Doc Data Sheet
Head Land Systems Division	Doc Data Sheet
Head Industry Division	Doc Data Sheet
Chief Joint Logistics Command	Doc Data Sheet
Management Information Systems Division	Doc Data Sheet
Head Materiel Finance	Doc Data Sheet
Defence Libraries	
Library Manager, DLS-Canberra	Doc Data Sheet
Library Manager, DLS - Sydney West	Doc Data Sheet
OTHER ORGANISATIONS	
National Library of Australia	1
NASA (Canberra)	1
UNIVERSITIES AND COLLEGES	
Australian Defence Force Academy	
Library	1
Head of Aerospace and Mechanical Engineering	1
Hargrave Library, Monash University	Doc Data Sheet
Librarian, Flinders University	1

OUTSIDE AUSTRALIA

INTERNATIONAL DEFENCE INFORMATION CENTRES	
US Defense Technical Information Center	2
UK Defence Research Information Centre	2
Canada Defence Scientific Information Service	e-mail link to pdf
NZ Defence Information Centre	1
ABSTRACTING AND INFORMATION ORGANISATIONS	
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
Materials Information, Cambridge Scientific Abstracts, US	1
Documents Librarian, The Center for Research Libraries, US	1
SPARES	5
Total number of copies:	**40**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)

2. TITLE

Investigating Web Map Service Client Technologies

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)

Document	(U)
Title	(U)
Abstract	(U)

4. AUTHOR(S)

Joel Blyth, Sean Geoghegan and Damian O'Dea

5. CORPORATE AUTHOR

Information Sciences Laboratory
PO Box 1500
Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-TN-0536	AR-013-009	Technical Note	January 2004

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
E9505/25/91	JTW 02/224	DGICD	36	6

13. URL on the World Wide Web

http://www.dsto.defence.gov.au/corporate/reports/ DSTO-TN-0536.pdf

14. RELEASE AUTHORITY

Chief, Command and Control Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for public release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT

No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DEFTEST DESCRIPTORS

Mapping
Comuter aided mapping
Computer generated imagery

19. ABSTRACT

Web Map Services may provide some assistance in alleviating the high cost of providing Geospatial Information System (GIS) capabilities to Defence users. This report discusses the development of several implementations of client applications of the Web Map Service interface. Problems encountered in the use of various Web technologies to construct the client applications are detailed. Suggestions are also made for future development efforts to improve client functionality.