12-19-2003

Enhancement of COTS GIS Web Publishing Software

Jin Chen
University of New Orleans

ENHANCEMENT OF COTS GIS WEB PUBLISHING SOFTWARE

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
The Department of Computer Science

by

Jin Chen

B.S., TongJi Medical University, 1998

December 2003

Dedicated to:

My parents, FaShun Chen and FanYu Meng

My Husband, Jie Cheng

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor, Professor Shengru Tu, for his guidance, support and encouragement. His kindness and generosity made me feel very comfortable working with him.

I also want to thank the members of my committee, Prof. Golden G Richard III, Prof. Ming-Hsing Chiu, for their priceless instruction, help and guidance on my graduate study.

I am also grateful to Mr. Alfred F. Daech for his kind help and financial support of my research. Without his support and help, I would never have been able to complete this project.

I deeply appreciate my husband, Jie Cheng, for his greatest support of my life and study. Greatest thanks for my parents, who have encouraged and guided me to independence, thanks for their support and patience while I pursued this master's degree.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In the modern geographic information systems, COTS software has been playing a major role. Customizing COTS software is inevitable because large organizations' needs usually exceed COTS built-in functions. However, this is often a challenge to COTS users, since the source code of COTS is rarely available.

In my thesis project, I have enhanced some functions of a GIS COTS product, ArcIMS, by taking advantage of this web publishing tool's well-thought architecture of services and its applications of the XML messaging technology. Multiple users now can access the same ArcIMS map service but with only constrained viewing power. Login dialog has been added to the map service, and query tools have been modified to ensure the viewing restraint.

Another aspect of this thesis project is to enhance the interactive features, which makes the HTML viewer be capable of carrying out all the functions that the ArcIMS Java plug-in viewer can do.

INTRODUCTION

The money spent on software development around the world has been skyrocketing. The total amount of software development cost (exclude related software services) was about $127 billion in 2002. It is estimated that at most 5% of all software projects worldwide were completed on time and within budget. During that same period, Over 75% of software projects initiated without a valuable completion [15]. The trend implied by the increasing dollars spent on software coupled with the consistently high rates of software project overruns and cancellations does not seem to diminish. In order to change this situation, one of the significant approaches widely adopted has been the use of Commercial-Off-The-Shelf (COTS) components as system elements.

The advantages of using COTS are numerous including market-tested reliability, market-approved features, and an opportunity for expanding software capabilities and improving system performance by the commercial marketplace. However, the claimed performance and functionality of the COTS based systems are too often not realized in practice. Indeed, there has been more failure than success in using COTS software products. The mounting experience from both successful stories and failed lessons shows that the effective use of COTS software products in major software system demands new skills, knowledge and different techniques and processes.

The challenge in developing COTS based systems comes due to two reasons. First, large organization's requirements often exceed the capability of COTS products. Secondly, the source code of COTS software is rarely available. Attempting to modify or add classes are usually impossible. For thin-client software, the source code of the web presentation layer is visible HTML or scripts such as JavaScript. For modern web service software, an XML API or a SOAP interface at the client side are often available. In addition, the database schemas are also visible. With these limited handles, enhancing COTS software is typically possible but difficult.

The geographic information system (GIS) is one of the fields where COTS software is more commonly used. A GIS is a computer system capable of capturing, storing, analyzing and displaying geographically referenced information. The data are identified according to location. The power of a GIS comes from the ability to relate different information in a spatial context and to reach a conclusion about this relationship. Most of the information we have about our world contains a location reference. For example, when rainfall information is collected, it is important to know where the rainfall is located. Comparing the rainfall information with the location of marshes across the landscape may show that certain marshes receive little rainfall and are likely to dry up. This observation can help us make the most appropriate decision about how human should interact with the marsh. Thus, the GIS reveal important new information that leads to better decision making.

The uses of GIS COTS software products usually achieve better results. This is because of at least three reasons. First, requirements of GIS are typically well defined in the specific GIS domain. Second, the traditional uses of GIS tools are for a group of

domain experts, a department or a project of controllable size. Third, the GIS COTS software products are often built based on rigorous mathematical models. However, as more and more geographic information systems become web enabled and enterprise-wide, the use of GIS COTS products encounters the same challenges that the general COTS based systems are faced with.

Even though there are many free GIS software products available, such as the FreeGIS Projects, GeoCommunity, and Directions Magazine Free GIS Tools. They are typically low-end with limited functionalities.

ArcGIS is one of the leading commercial GIS software product families. It is developed by ESRI. It supports users to build and manage maps, data and tools. Specifically, the ArcIMS (Arc Internet Map System) greatly eases web developers in making and delivering maps and data over the internet. As one of the leading COTS GIS web publishing software, ArcIMS has a well-thoughtful component architecture which consists of the spatial database, the application server, the manager component, and the web server along with the ArcIMS connectors (Figure 1). The details of these components will be described in Chapter 1. Using the ArcIMS, the GIS content provider designs and publish interactive web pages using the manager. The corresponds of the manager generate web pages that are stored in the web server. These web pages interact with the application server in XML requests/responses.

ESRI provides a very powerful Java plug-in for high-end web users (free of charge). Having this Java Plug-in, the web viewer is almost as powerful as ArcGIS's desktop applications. However, a serious drawback of this plug-in is that its installation is too complicated and too sensitive to the JVM's versions. Consequently, ESRI also let

ArcIMS designer deliver the ArcIMS HTML viewer that controls interactions in HTML and JavaScript, only by sacrificing many useful features.

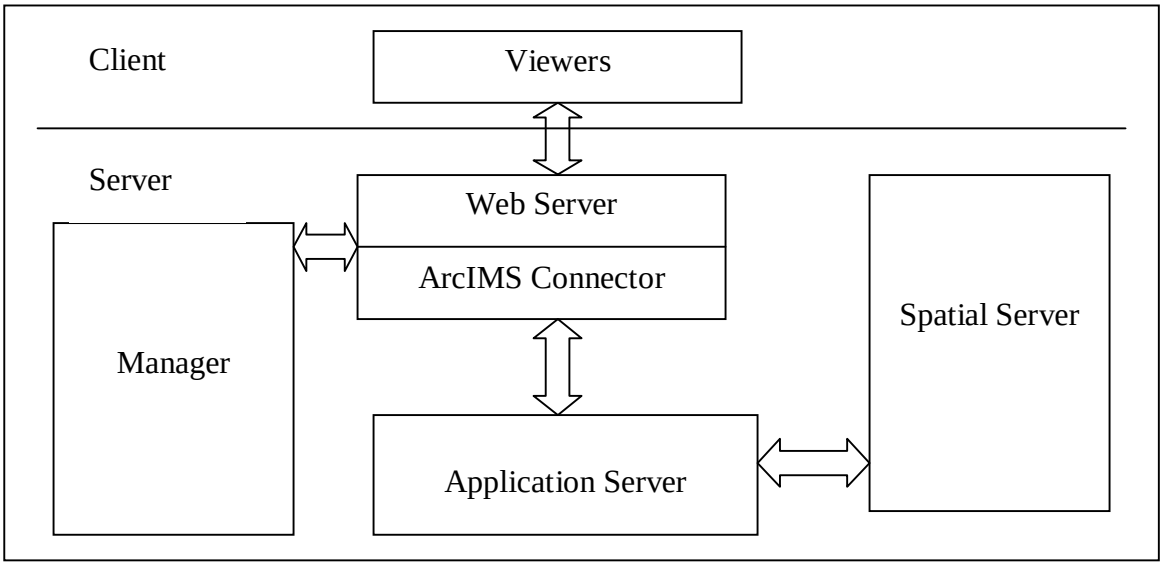

Figure 1 ArcIMS Architecture

In this thesis, I will report my experimental enhancement of the ArcIMS publishing tools. The additional features to ArcIMS include using authorization, authentication, and interactive function in ArcIMS HTML Viewers. I have made the ArcIMS HTML Viewer reclaims important features and become as powerful as the ArcIMS Java Plug-in viewer. My enhancement to ArcIMS is practically useful because I have automated the feature enhancement process. The ArcIMS users with limited programming skills will be able to enhance ArcIMS using my programs.

CHAPTER 1

BACKGROUND

1.1 Geographic Information System (GIS) and ArcGIS

A geographic information system (GIS) is a system that visualizes, manipulates, analyzes and displays spatial data. Since GIS has the power of revealing new information in a spatial context, a lot of software have been developed for GIS applications. The software that I enhanced in this thesis is the Internet Map Service product ArcGIS developed by the ESRI Company.

GIS can be used in many ways. For Emergency Services, GIS is widely used in Fire departments & Police stations; for environmental studying, GIS can be used to monitoring and modeling environmental parameter changing. In business area, GIS can be used to design site location, help customer locate the nearest store around them. Also it can be used by the delivery systems to locate their destinations. In industry, transportation, communication, mining, pipelines setup or healthcare services will need the GIS information. As for government use, GIS is a very important part for either the local, or State, or federal departments, especially in military uses.

A lot of computer databases that can be directly incorporated into a GIS are produced by Federal, State, tribal, and local governments, private companies, academia, and nonprofit organizations. Different kinds of data in map form can be entered into a GIS. A GIS can also convert existing digital information, which may not yet be in map

form, into the forms it can recognize and use. For example, census or hydrologic tabular data can be converted to a map-like form and serve as layers of thematic information in a GIS.

Many software systems have been used to develop GIS applications. GIS software provides the functions and tools needed to store, analyze and display information in a spatial context. The key components of GIS software are:

- Tools for entering and manipulating geographic information such as addresses or political boundaries.
- A database management system (DBMS)
- Tools that create intelligent digital maps you can analyze, query for more information, or print for presentation
- An easy-to-use graphical user interface (GUI)

GIS software ranges from low-end business-mapping software appropriate for displaying sales territories to high-end software capable of managing and studying large protected natural areas.

The ArcGIS system is an integrated geographic information system consisting of three key parts [7]:

- ArcGIS Desktop Software, an integrated suite of advanced GIS applications.
- ArcSDE gateway, an interface for managing geodatabases in a database management system (DBMS).
- ArcIMS software, internet-based GIS for distributing data and services.

ArcGIS provides a framework for implementing GIS can be deployed on a single desktop or distributed on a heterogeneous computer network of workstations and servers. Users can deploy various parts of this system to implement a GIS of any size – from a single-user system to large enterprise, and even societal GIS systems. It has the architecture as shown in Figure 1.

The ArcIMS Spatial Server is the backbone of ArcIMS. It processes requests for maps and related information. When a request is received, the ArcIMS Spatial Server performs one or more functions to generate response to the request.

The ArcIMS Application Server handles incoming requests and tracks which services are running on which ArcIMS Spatial Servers. It hands off a request to the appropriate Spatial Server. It is a Java application and runs as a Windows service or as an UNIX daemon process.

The ArcIMS Application Server Connectors connect the Web server to the ArcIMS Application Server. It is the standard connector used with ArcIMS. It supports the Open GIS Consortium (OGC) WMS 1.0.0 implementation specification. It communicates between the Web server and the ArcIMS Application Server using ArcXML.

ArcIMS Manager is a Web-based application that supports the three main tasks performed in ArcIMS—map authoring, Web site design and site administration. These tasks can also be completed using the three independent ArcIMS applications—ArcIMS Author, ArcIMS Administrator or ArcIMS Designer.

ArcSDE is the GIS gateway to relational databases. It allows user to manage geographic information in user's chosen DBMS and serve user's data openly to ArcGIS

desktop, ArcIMS and other applications. ArcSDE is a key component in a multi-user ArcGIS system. It provides an open interface to relational database management systems and allows ArcGIS to manage geographic information on a variety of different database platforms including Oracle, Microsoft SQL server, IBM DB2 and Informix.

ArcIMS is a very important part of the ArcGIS software system. It allows user to build and deliver a wide range of GIS maps, data, and applications to users on the World Wide Web. ArcIMS includes both client and server technology. It extends a web site by enabling it to serve GIS data and applications. The ArcIMS framework consists of clients, services, and data management all delivered from an out-of-the-box solution for creating, designing, and managing mapping and GIS capabilities within Web sites. The key features of ArcIMS include:

- Integration of locally held data with Internet data.
- Support for serving data as bit maps or with intelligent vector streaming.
- Facilitated access from multiple servers
- Support for DHTML, Java, or bespoke clients
- Easy installation, implementation, and administration with wizards and templates

  Integration with leading Database including Oracle, and SQL Server
- High-quality cartographic rendering

ArcIMS contains two different kinds of viewers: the HTML viewer and the Java viewer. In this thesis project, I enhanced the HTML viewer.

1.2 ArcXML

ArcXML is the protocol for communicating with the ArcIMS Spatial Server. As mentioned before, an ArcIMS Spatial Server is the backbone of the ArcIMS system and

provides the functional capabilities for accessing and bundling maps and data into the appropriate format before sending the data to clients. To understand ArcXML, it is necessary to know how configuration files, ArcIMS services, requests, and responses relate to each other and how they interact with the ArcIMS Spatial Server.

Figure 2 is a diagram showing the interaction between the ArcIMS Spatial Server and configuration files, services, requests and responses.

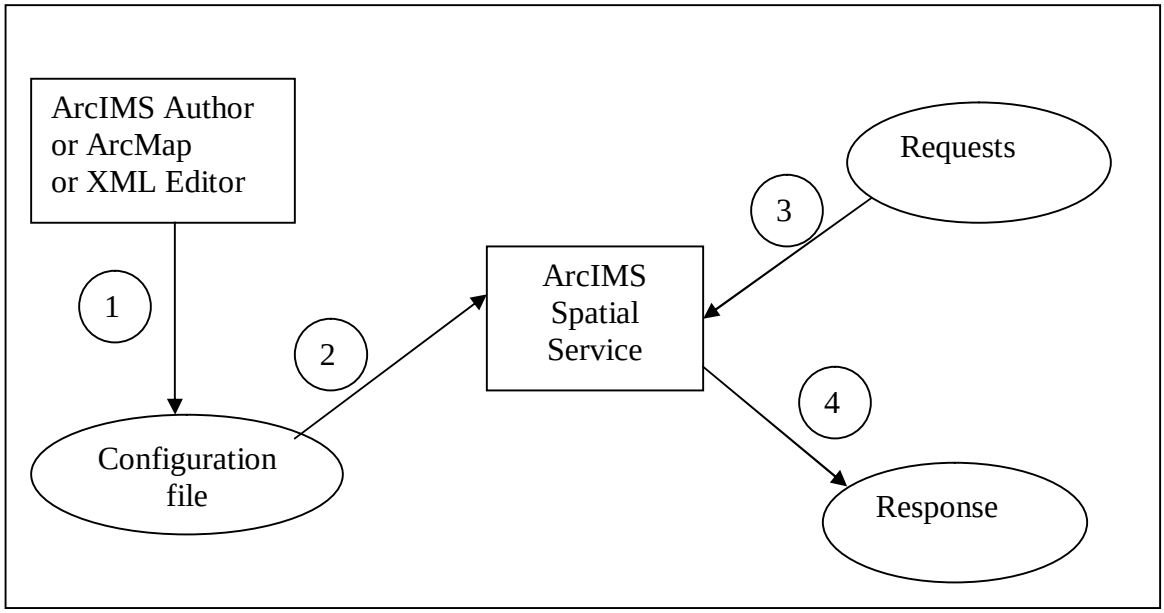

Figure 2 Interaction of ArcIMS Spatial Server with others

In Step 1, a configuration file is generated. Configuration files are used to define maps. User creates a map configuration file for Image and Feature Services by using ArcIMS Author or a text or XML editor. When this file is saved in ArcIMS Author, the result is a map configuration file, written in ArcXML, containing layer information of the map contents. (The file has an *.axl extension.)

Step 2 is to set up the map service. An ArcIMS service is a process that runs on the ArcIMS Spatial Server. A service can be thought as a portal to the Spatial Server. The spatial Server's functionality is accessible only through services running on the server. In this step, the map configuration file "*.axl" is the input to an ArcIMS service. It provides drawing instructions for each layer in a service. These instructions are the default state for the service. The naming of an ArcIMS service is independent of the name of the input map configuration file.

In Step 3, requests from internet users can be sent to the service once an ArcIMS service is running on the ArcIMS Spatial Server. These requests are generated by the ArcIMS HTML Viewer, Java Viewers, and any other viewers using the ColdFusion, ActiveX, or Java Connectors. The requests are:

- GET_IMAGE
- GET_FEATURES
- GET_GEOCODE
- GET_EXTRACT
- GET_SERVICE_INFO
- GET_RASTER_INFO
- GET_LAYOUT
- GET_METADATA
- PUBLISH_METADATA

The XML element that distinguishes a request from other types of ArcXML files is REQUEST. A request can also override some of the information in a service by asking for a new map at a different scale, turning layers on or off, requesting a subset of the attribute data, changing the projection, or adding acetate layers. With the Image Server, a request can also be used to change the rendering of a layer or add new data in dynamic layers. In this thesis work, this feature will be used to achieve the authentication goal.

In Step 4, the results are returned in a response when the ArcIMS Spatial Server processes a request. The XML element that distinguishes a response from other types of ArcXML files is RESPONSE.

In summary, ArcXML is the language used for sending requests and receiving responses through ArcIMS' spatial server. It is a specialized XML that follows ESRI's protocol.

1.3 COTS

According to the perspective of the Software Engineering Institute (SEI), a COTS product is: sold, leased or licensed to the general public; offered by a vendor trying to profit form it; supported and evolved by the vendor, who retains the intellectual property rights; available in multiple, identical copies; and used without source code modification [16].

Using "Commercial Off-the-shelf" (COTS) software components to build systems has been a means of developing software to reduce risk and cost and increase functionality and capability of the system. Building a system based on COTS components involves buying a set of pre-existing, proven components, building extensions to satisfy local requirements, and gluing the components together. The advantage claimed is that

the COTS components are honed in the competitive marketplace resulting in increased capability, reliability, and functionality for the end user over what would be available from custom built components. COTS software components from different vendors are expected to be integrated, applicable in a wide range of environments, and support extensions and tailoring to local requirements.

A significant research effort, the COTS-Based Systems (CBS) Initiative, was initiated at CMU in 2001. It is focused on improving the technologies and practices used for assembling previously existing components (COTS and other non-developmental items) into large software systems, and migrating existing systems toward CBS approaches. The CBS approach changes the focus of software engineering from traditional system specification and construction to one requiring simultaneous consideration of the system context (system characteristics such as requirements, cost, schedule, operating and support environments), capabilities of products in the marketplace, and viable architectures and designs.

In the last decade, the use of the (COTS) products as elements of larger systems is becoming increasingly commonplace. Shrinking budgets, accelerating rates of COTS enhancement, and expanding system requirements are all driving this process. The shift from custom development to COTS-based systems is occurring in both new development and maintenance activities [1].

ArcIMS is a leading GIS COTS product. I have experimented a method to enhance it without its source code. I have implemented an easy-to-use tool that can be used by the users with minimum programming skills.

1.4 Techniques used in this thesis project

1.4.1 JavaScript

JavaScript is a language developed strictly for the web sites to create interactive web page that can handle calculations, controls such as displaying variable information, validate forms without a lot of programming effort. In short description, java script is an easy programming language specifically designed to make web page elements interactive. An interactive element is one that responds to a user's input [2] [3].

1.4.2 Servlet

Servlets are the Java platform technology for extending and enhancing Web servers. Servlets provide a component-based, platform-independent method for building web-based applications. Servlets are server- and platform-independent, this leaves user free to select a "best of breed" strategy for their servers, platforms and tools.

Servlets are modules of Java code that run in a server application to answer client requests. Servlets are not tied to a specific client-server protocol but they are most commonly used with HTTP and the word "Servlet" is often used in the meaning of "HTTP Servlet".

Servlets make use of the Java standard extension classes in the packages “javax.servlet (the basic Servlet framework) and javax.servlet.http (extensions of the Servlet framework for Servlets that answer HTTP requests)”. Since Servlets are written in the highly portable Java language and follow a standard framework, they provide a means to create sophisticated server extensions in a server and operating system in an independent way [14]. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. Servlets can also access a library of HTTP-

specific calls and receive all the benefits of the mature Java language, including portability, performance, reusability and crash protection [5].

In this thesis project, I used a Servlet to access the database to achieve the authorization for users of ArcIMS Html viewer.

1.4.3 Cascading Style Sheet

Style Sheets allow user to control the rendering of a Web document without compromising its structure. CSS is a simple style sheet mechanism that allows authors and readers to attach style to HTML documents. It uses common desktop publishing terminology which should make it easy for designers to make use of its features. It addresses many of the problems of old-style HTML. With CSS, style information can be centralized. This centralization leads to increased power and flexibility.

With cascading style sheets, designers are able to use tags to reference a style rather than describe it at each instance. When a style needs to be changed, only the referenced declarations need to be changed, not all of the instances where it is used.

CHAPTER 2

ENHANCEMENT DESIGN

2.1 Refining the Authentication Features

As the ArcIMS viewers were originally designed, authorization and authentication are supported at the service level. Viewers with different reading rights are to use different map services. Any viewer of a service can see the whole map and query anything in the service. When a large number of users want to see their own individual data in the same viewer configuration, each of them must have an individual service. This will lead to large number of map services, and will cause high overhead to the system and management.

ArcIMS offers two methods for restricting access to map services through the ArcIMS Servlet Connector using an ACL (Access Control List):

1) Maintaining an ACL file in ArcXML

2) Maintaining an ACL in a database table

Maintaining a dynamic ACL using the file-based method is cumbersome since the servlet engine must be restarted after each change of the content of the ACL ArcXML file. The database method offers the ability to make dynamic changes to an ACL using the JDBC technique [11].

The ACL contains the user name and password for each authorized user. When a user opens a browser and access the HTML or the Java Viewer, a pair of

username and password is required before the site is loaded. A problem of this method is that every authorized user can see other authorized users' information if they shared the same map service. Since the source code of the ArcIMS Servlet Connector is unavailable, no modification can be made to the connector that performs the authentication feature. The HTML and JavaScript files produced by the ArcIMS designer and those of authentication and authorization page are the all source code I could get.

A desired security feature should begin with a login dialog followed by every service that ArcIMS provides under the condition that each viewer sees this viewer's own data only. In this way, many users can view the same map service without intruding into other user's private information.

Specifically, my ArcIMS project is for an agriculture information service company who collects, analyzes and delivers insects migration in farm lands. Farmers collect insects density data by reading bug trap tapes installed in their fields, recording the data with a GPS enabled PDA, and uploading the data with their computers. A central database gathers the raw data and support map services carried out by a ArcGIS installation. Each farmer user is allowed to view the user's own insects map and overall insects migration analyses from the company's web site.

My solution to the enhancement of the ArcIMS for this project consists of the following aspects:

- I used the ArcIMS built-in authorization feature but choose the ACL management that maintains the ACL in a database table

- It is assured that every piece of bug trap tape reading is associated with the field owner's identifier. This was consistent with the company's original data model, and implemented by the data uploading page.
- I added a login servlet to the web server that hosts the ArcIMS server. This servlet reacts to the URL when the user first attempts to view the company's service. Through the corresponding login dialog window, the servlet collects the user's id and password. The authentication is straightforward by carrying out a JDBC access to the ACL table. Upon a successful login, the servlet redirects the user to the main map service page. This map service page is a modified version of the JavaScript generated by the ArcIMS designer. The modification of the JavaScript will be described in next chapter.
- To limit the user's viewing scope to the user's own data, the user's web page provided by the above login servlet must only ask for the data associated with that user's id. In order to achieve this, we modified each database query corresponding to every supported function such as query, find, identify, measure, select points by rectangle and retrieve all data included in this rectangle. I could do so because the JavaScript of the ArcIMS HTML Viewer issues SQL query to the ArcIMS spatial service. By adding "userid=?" into the query's WHERE clause, the user's viewing scope is constrained. The value of the user id is inserted into the JavaScript by the above login servlet.

2.2 Interactive Features Enhancement

Two viewers are available in ArcIMS, which are the Java Viewer (including the Java Standard Viewer and the Java Custom Viewer) and the HTML Viewer. The same data are available in both the Java and HTML Viewers. Both viewers rely heavily on JavaScript. The HTML Viewer has limited simple functionality. The Java Viewer has more viewing and querying features, and is almost as powerful as the desktop ArcGIS software. However, the Java Viewer (Figure 3) requires over 10 MB downloading (once). Its installation is very sensitive to the JVM of the browser. And it requires higher network capacities. For many users, downloading and installing such a large java plug-ins and setting up the JVM runtime environment too much to handle.

The major reason to use the HTML Viewer (Figure 4) is its simplicity in use while ArcIMS designer outputs a full suite of functions. The HTML Viewer is supported on all platforms for both client and server. The developer only needs to know HTML and JavaScript to create a site using the HTML Viewer. The HTML Viewer's drawback is that it is thicker than other connector clients and its response time can be slow when accessing large datasets.

By analyzing differences between the Java Viewer and HTML Viewer and considering the common needs from the HTML viewer users, I have enhanced the HTML Viewer in three aspects.

2.2.1 Relocating the Overview Window

The overview window provides the user a bird-eye view of the whole map. It shows the entire map area in a smaller scale. With the overview map, the user will be able to easily tell which part of the map is currently zoomed in. The user can also

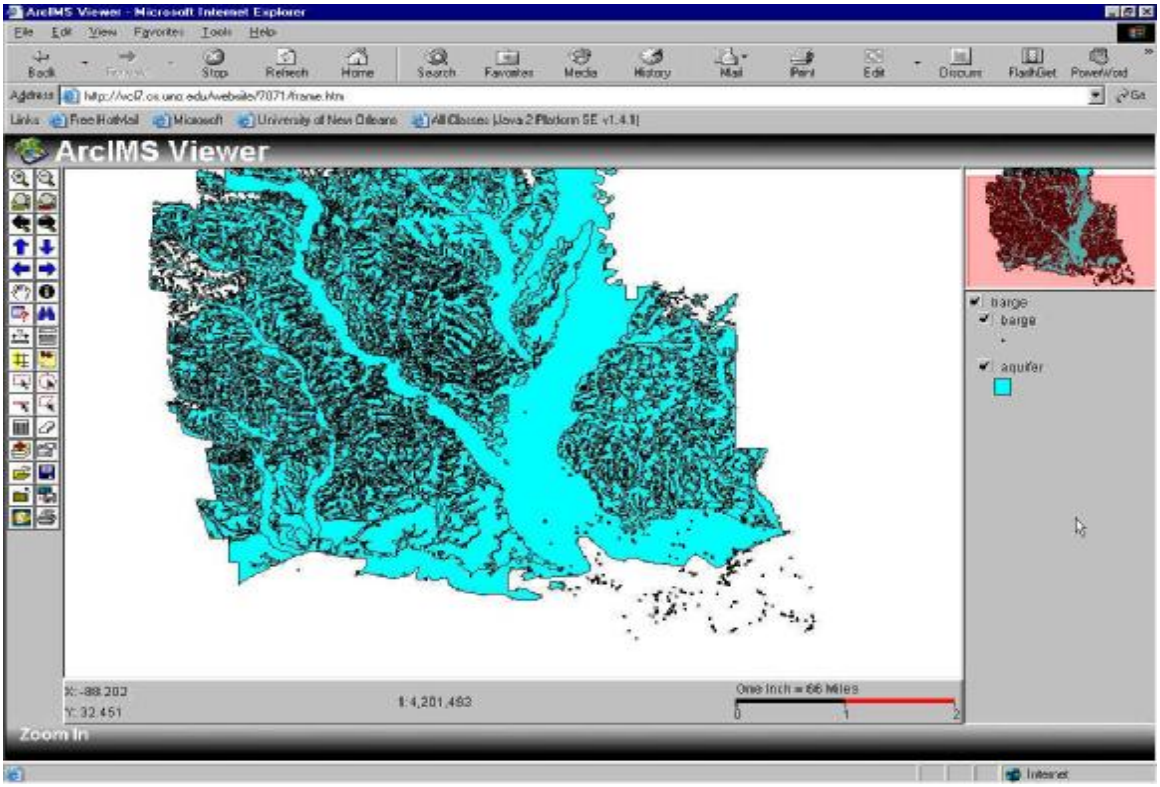

Figure 3 Java Viewer

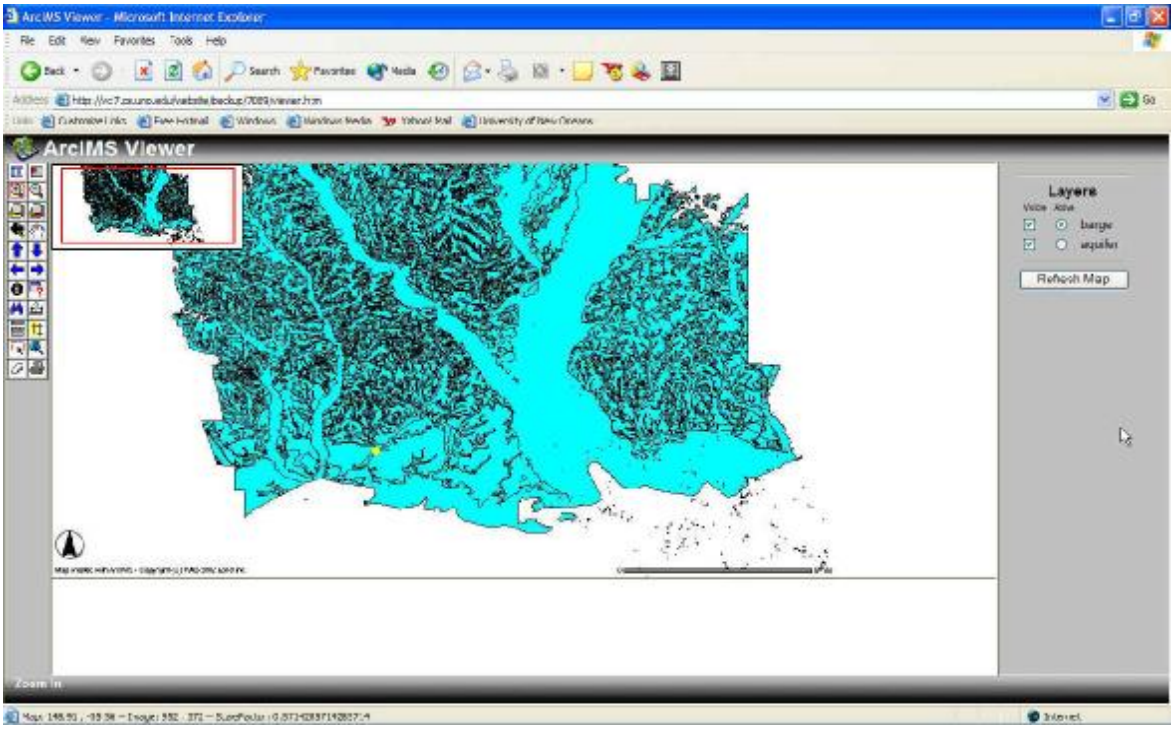

Figure 4 Html Viewer

navigate in the whole map by clicking on the overview mini-map. Both the Java Viewer and HTML Viewer provide users with an overview window.

In the Java Viewer, the overview window is placed in a separate window on the right hand side frame (Figure 3). The user can view the target area of the map and the overview map at the same time, which is convenient for the user. It also saves the space for the main map window. Whereas in the HTML viewer, the overview window is placed in the upper left corner of the main frame. That results that part of the main map is covered by that overview window (as shown in Figure 4), The data points which are covered by the overview window become un-clickable. Even though the user can use one of the tool bar button "Toggle Overview map" to disable the overview window, the user would loose the simultaneous view of the main map and the overview map.

My first simple modification to the ArcIMS HTML viewer was to place the overview window at a separate location without interferencing with the main map window.

2.2.2 Adding a Visual Scale Tool To the HTML Viewer.

To make a map useful, mapmakers typically establish and indicate a consistent relationship between the size on the map and the size in the real life. That is how map scale becomes to use. Map scale is the relationship between distances on a map and the corresponding distances on the earth's surface expressed as a fraction or a ratio (constant for a given map).

In ArcIMS, both the Java Viewer and the HTML Viewer provide the user with a scale bar located at the bottom of the main map. They also provide two tools for the users

to change the current scale, Zoom In or Zoom Out. Those are very useful tools for map users to view and measure their maps. But none of these two viewers provides any visual scale tool that is clickable for the user to view the scale more intuitively.

As requested by the clients, adding a visual scale tool becomes one of my goals in this thesis project. I need to design and develop a visual scale tool (shown in Figure 5). In this tool each bar should be clickable, and corresponding to its representative scale. Once the user click on any one of these bars, the main map should be zoomed in or zoomed out to the proper scale ratio with respect to that bar. Meanwhile the approximated scale ration will be shown under these bars.

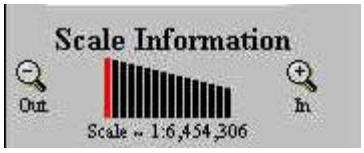

Figure 5 Scale tool

2.2.3 Adding a Graduating Symbol under Each Layer Name In the Layer List

As shown in figure 3, Java viewer put the graduating symbol just beneath each layer's name, so users can see the meaning of color in that layer. This is implemented by Java applets. As designed by the ArcIMS designer, when a Java Viewer is created, it will produce files named "default.axl" and "default.js" under the working directory. These files contain all the information needed for the java applets to construct the graduating symbols. By reading this file each time when the window is requested, applets can draw all symbols needed and create frames to put them all together. So in the Java viewer, each layer has its name and its symbols combined together and are put into a small frame.

Then all these small java applet frames are put into the larger java applet frame and shown as a whole set of layer list with their symbols in the TOC frame.

However, in the HTML viewer, layer names and their symbols are put in separate window. By using the tool “Toggle between legend and layer list” provided by the HTML Viewer, users can either show the layer list with only their names or show only the graduating symbols with no operations on the layer list at all. That would be inconvenient for the user to use the symbol. In order to know the meaning of each color, the user has to use the tool button located in the tool bar to toggle back and forth the legend to see the graduating symbol for each layer.

The cause of this problem in the HTML viewer is that after the HTML Viewer is created by the ArcIMS designer, each time the user operates on the map such as enable or disable layers of the map, a new ArcXML request will be sent to the server. Then the server will create a new image file for this newly specified map along with the image for the color legend. That means all the legends are drawn into one image file (usually a JPG file) by the server when server got the request from the client. Once it’s been created, the server will output this JPG file to a specified folder on the server side. Meanwhile a URL for this JPG file will be returned to the client by the server. Whenever the user clicks on the “toggle legend” button, this legend URL will be retrieved to show the legend image file. Since it is a whole image file, it cannot be broken down into several pieces for each layer, and that is the reason why the legend cannot be shown as a companion of the layer list.

To make the HTML Viewer as convenient as the java viewer to the users, a graduating symbol list is needed for each layer name.

CHAPTER 3

ENHANCEMENT AND AUTOMATION

In this chapter, all the enhancement designs has been implemented, an automation tool has been developed to easy the enhancement work. As show in the feature 6, all the source code we have are the HTML and JavaScript files produced by the ArcIMS designer. My work is focused on modifying them to make them perform enhanced features when they've been downloaded to the user's browser. Also a servlet has been added to the web server to meet the authorization purpose.

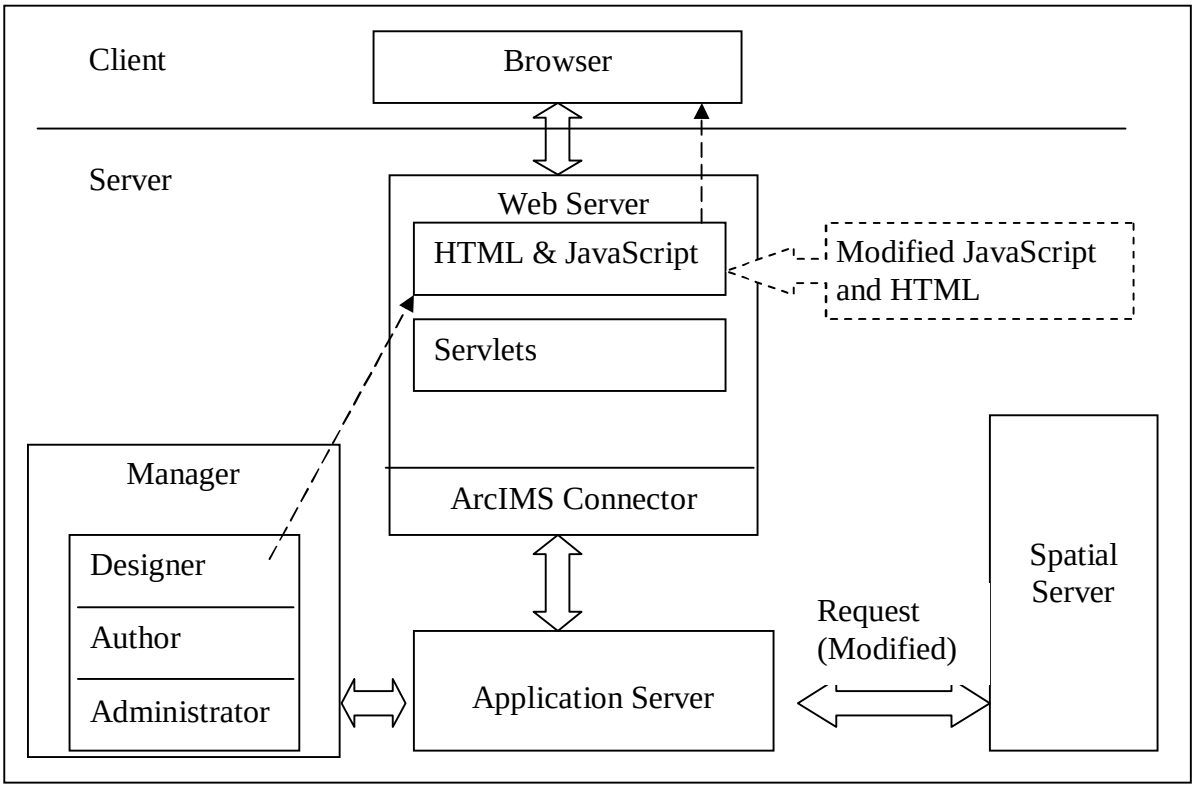

Figure 6 Enhancement to ArcIMS HTML Viewer

3.1 Reorganization of the Overview Window

In order for the HTML Viewer to have the overview map and main map shown at the same time, the best way to do is to move this overview map to a separate window that can be shown alone with the main map window. As shown in Figure 7, we can see that the whole window has been divided into 8 child frame. The main map will be put in the "MapFrame" and layer list in the "TOCFrame".

To move the overview window to a separate window, I need to add a separate frame to the whole frame set so it will be able to hold the new overview HTML file. We chose the TOCFrame since it only contains content of the layer list. All the Frames are defined in file "viewer.htm". Editing the source code of this file will generate a new frame (Figure 8). Also a new HTML file overview.htm (Figure 9) will be needed as the source code file for the new frame.

Another important step in changing the position of the overview window is to modify the lines of the Cascading Style Sheets (CSS) layers referencing the overview map and extent box in the mapFrame.htm.

The CSS lines for the overview map and its extent is defined in file "mapFrame.htm", code show as in figure 10.

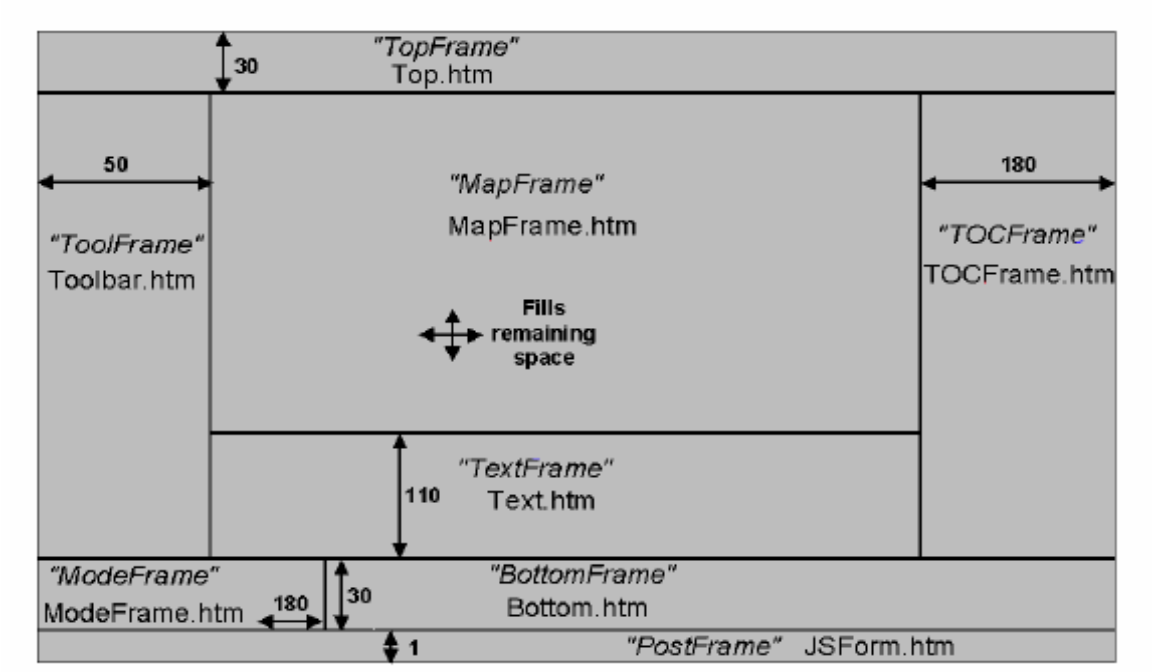

Figure 7 HTML viewer Frame Layout

```
…
document.writeln('<FRAMESET ROWS="' + (30+addNS) + ',*,30,0"
        FRAMEBORDER="No" FRAMESPACING="0" onload="doIt()" BORDER=0 '
        + moreStuff + '>');
…
document.writeln('                </FRAMESET> ');

////add a frame here
document.writeln('                <FRAMESET ROWS="150,*">          ');
document.writeln('                   <FRAME NAME="OverviewFrame"
        SRC="overview.htm" MARGINWIDTH="0" MARGINHEIGHT="0"
        SCROLLING="Auto" FRAMEBORDER="Yes" RESIZE="YES">');
document.writeln('                   <FRAME NAME="TOCFrame"
        SRC="TOCFrame.htm" MARGINWIDTH="0" MARGINHEIGHT="0"
        SCROLLING="Auto" FRAMEBORDER="Yes" RESIZE="YES">');
document.writeln('                </FRAMESET> ');
document.writeln('                <FRAME NAME="TOCFrame"
        SRC="TOCFrame.htm" MARGINWIDTH="0" MARGINHEIGHT="0"
        SCROLLING="Auto" FRAMEBORDER="Yes" RESIZE="YES">');
document.writeln('        </FRAMESET>');
…
 document.writeln('</FRAMESET>');
```

Figure 8. Add a frame to the whole frame set in viewer.htm

```
…
//over view map
<body BGCOLOR="White" LEFTMARGIN=0 TOPMARGIN=0
        RIGHTMARGIN=0>
<IMG SRC="images/locMap.gif" WIDTH=180 HEIGHT=150 HSPACE=0
        VSPACE=0 BORDER=0 ALT="Overview Map" ID="ovImage"
        name="ovImage" onmousedown="ovMap2Click(event)">
// overview extent box
<SCRIPT LANGUAGE="JavaScript1.2" TYPE="text/javascript">
var content = '<img name="zoomOVImageTop" src="images/pixel.gif" width=1
        height=1>';
createLayer("zoomOVBoxTop",0,0,180,150,false,content);
content = '<img name="zoomOVImageLeft" src="images/pixel.gif" width=1
        height=1>';
createLayer("zoomOVBoxLeft",0,0,180,150,false,content);
content = '<img name="zoomOVImageRight" src="images/pixel.gif" width=1
        height=1>';
createLayer("zoomOVBoxRight",0,0,180,150,false,content);
content = '<img name="zoomOVImageBottom" src="images/pixel.gif" width=1
        height=1>';
createLayer("zoomOVBoxBottom",0,0,180,150,false,content);
// set Overview map extent box color
setLayerBackgroundColor("zoomOVBoxTop", zoomBoxColor);
setLayerBackgroundColor("zoomOVBoxLeft", zoomBoxColor);
setLayerBackgroundColor("zoomOVBoxRight", zoomBoxColor);
setLayerBackgroundColor("zoomOVBoxBottom", zoomBoxColor);
</SCRIPT>…</BODY>
```

Figure 9. Code for overview.htm

:

```
// overview map and shadow
content = '<img name="ovShadowImage" src="images/gray_screen2.gif"  border=0 ';
content += '>';
createLayer("ovShadow",4,4,1,1,false,content);
if ((isNav4) || (isIE)) clipLayer("ovShadow",0,0,1,1);
content = '<img name="ovImage" src="images/locMap.gif" border=2
        width=' + 1 + ' height=' + 1 +'>';
createLayer("ovLayer",0,0,1,1,false,content);
setLayerBackgroundColor("ovLayer", "white");
// overview extent box
content = '<img name="zoomOVImageTop" src="images/pixel.gif"
        width=1 height=1>';
createLayer("zoomOVBoxTop",0,0,1,1,false,content);
content = '<img name="zoomOVImageLeft" src="images/pixel.gif"
        width=1 height=1>';
createLayer("zoomOVBoxLeft",0,0,1,1,false,content);
content = '<img name="zoomOVImageRight" src="images/pixel.gif"
        width=1 height=1>';
createLayer("zoomOVBoxRight",0,0,1,1,false,content);
content = '<img name="zoomOVImageBottom" src="images/pixel.gif"
        width=1 height=1>';
createLayer("zoomOVBoxBottom",0,0,1,1,false,content);
```

Figure 10 code for the CSS in MapFrame.htm

The first block of the code defines the style for the overview map and shadow; function “createLayer” is used here to create a layer for the overview window. In order to make the original overview window disappear in the up-left side of the main map, we changed the parameter of the function, putting the overview window to position (0,0) with the width and height of the window to (1,1). That means the window size will be set with 1 pixel width and 1 pixel height. Here the CSS layers are not removed but are resized to one pixel in width and height and moved off the visible page or “hidden”. We have to leave this layer there instead of completely remove it because errors will occur if these are removed from the viewer.

3.2 Adding a Scale tool to the HTML viewer

As explained in section 2.2.2, I'm to add a visual tool bar displaying the scale information of the current map and make it clickable for the user to zoom in or out at one time at any level.

To implement this tool, I need to convert from spherical coordinates to flat (Cartesian) coordinates. Normally to locate a point on earth, the easiest way is to give the latitude and longitude of the point. Latitude and longitude are spherical coordinates, based on recognition that the Earth is round. Their definition does not require that the Earth be exactly spherical; approximating the Earth as a sphere is satisfactory for most needs.

I applied the Haversine Formula in order to implement the scale tool. Presuming a spherical Earth with radius R, and that the locations of the two points in spherical coordinates (longitude and latitude) are lon1, lat1 and lon2, lat2, then the Haversine Formula [17]:

dlon = lon2 - lon1

dlat = lat2 - lat1

a = (sin(dlat/2))^2 + cos(lat1) * cos(lat2) * (sin(dlon/2))^2

c = 2 * atan2(sqrt(a), sqrt(1-a))

d = R * c

The intermediate result c is the great circle distance in radians. The great circle distance d is in the same units as R. Here radian is a unit of plane angle measure equal to the angle subtended at the center of a circle by an arc equal in length to the radius of the

circle. In another word, one radian is equal to 360°/2π, which is approximately 57° 17' 44.6".

In ArcIMS, all the data inputs are based on their geographic position. This scale tool will mainly use the zoom-to-Envelop, zoom-in and zoom-out function given by the ArcIMS designer. Meanwhile I used the Haversine formula to decide how deep the zoom should go in and out to fit the clients' requests.

The function to perform the Haversine formula is a JavaScript file as part of my tool. (Figure 11) The four coordinates are the two end points of a diagonal line across the image, here degreesLeft, degreesRight represent the min and max longitude values of the current map and degreesTop and degreesBottom show the latitude value.

```
function HaversineScale(degreesLeft, degreesBottom, degreesRight, degreesTop,
          imageWidthInPixels, DisplayDPI) {
…
    var deltaLat =  Math.abs(degreesTop - degreesBottom);
    var deltaLong = Math.abs(degreesRight - degreesLeft);
    // Calculate the scale based upon the Map Units.
    if ( MapUnits.toUpperCase() == "DEGREES" ) {
    // Decimal Degrees, change the difference in map units from degrees to radians
      var deltaLatRadians =  deltaLat * (Math.PI / 180);
      var deltaLongRadians = deltaLong * (Math.PI / 180);
      // Now do the Haversine equations
      var a = Math.sin(deltaLatRadians/2) * Math.sin(deltaLatRadians/2) +
          Math.cos(degreesBottom * (Math.PI / 180)) * Math.cos(degreesTop *
          (Math.PI / 180)) * Math.sin(deltaLongRadians/2) *
          Math.sin(deltaLongRadians/2);
     var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
     var d = RadiusEarth * c;
     var InchesPerPixel = d / imageWidthInPixels;
     //Return the scale using calculated inches per pixel
     ApproximateScale = Math.round( InchesPerPixel * MonitorDPI);
…
  }
}
```

Figure11 the Haversine function

By using this function we can draw a clickable tool to represent the current scale. Basically this tool makes use of the following methods defined in aimsMap.js:

function zoomToEnvelope(minXin,minYin,maxXin,maxYin) {…}

function zoomButton(zoomType) {…}

First we need to check what the current scale is, since the max scale would be the full extent map. The four coordinates of the full extent map given by the AXL file will be used to calculate the max scale. Without specifying exact area to zoom in, the default zoom in area will be the center of the full extent map. We divide the whole zooming range into 18 stages. From the max value to the min value, each stage has a scale that will be smaller then the previous scale by a ration of X. Here the number X can be defined by the map creator to make the scale increase or decrease rapidly or not. In this thesis work, We choose 1.25 as the default ration. By comparing the current map extent with the map scales for each stage based on the max scale, we can get the approximate current scale stage. (figure 12)

```
var ApproxScale = HaversineScale((eLeft + CenterAdjust), eBottom, (eLeft +
        CenterAdjust), eTop, iHeight, ScaleToolDPI);
//calculate the current scale
…
for (var j=18; j>0 ; j--) {
    CurrentStage = Math.abs(j-19);
    // adjust here by chen jin
    StageScale = theMaxScale/Math.pow(1.25,CurrentStage);
    NextStageScale = theMaxScale/Math.pow(1.25,CurrentStage+1);
    if ( ( ApproxScale > NextStageScale ) && ( CurrentStage <= 1 ) ) {
     ActualStage = 1;
    } else if ( (ApproxScale < NextStageScale ) && ( CurrentStage >= 18 ) ) {
     ActualStage = 18;
    } else if ( (ApproxScale > NextStageScale ) && (ApproxScale < StageScale ) ) {
     ActualStage = CurrentStage;
    }
  }…
```

Figure 12 : code to calculate the current actual scale stage.

```
…
var CenterX = parent.MapFrame.eLeft + ((parent.MapFrame.eRight –
        parent.MapFrame.eLeft)/2);
var CenterY = parent.MapFrame.eBottom + ((parent.MapFrame.eTop –
        parent.MapFrame.eBottom)/2);
  //chen jin
  for (var i=18; i>0 ; i--) {
    CurrentStage = Math.abs(i-19);
    if (CurrentStage < ActualStage) {
      //Zoom Out
      //adjust the base number of the power function to adjust the zoom deep
      //by chen jin
      xAdjustment = parent.MapFrame.xHalf * Math.pow(1.25,
          Math.abs(ActualStage - CurrentStage) - 1);
      yAdjustment = parent.MapFrame.yHalf * Math.pow(1.25,
          Math.abs(ActualStage - CurrentStage) - 1);
    } else {
      xAdjustment = parent.MapFrame.xHalf / Math.pow(1.25,
          Math.abs(ActualStage - CurrentStage) );
      yAdjustment = parent.MapFrame.yHalf / Math.pow(1.25,
          Math.abs(ActualStage - CurrentStage) );
    }
    StageEnvelopeX1 = CenterX - xAdjustment;
    StageEnvelopeX2 = CenterX + xAdjustment;
    StageEnvelopeY1 = CenterY - yAdjustment;
    StageEnvelopeY2 = CenterY + yAdjustment;
…
```

Figure 13 calculate envelop for each stage

Based on the current stage, the envelopes for other stage scales will be made. The base value for calculating these envelopes is the coordinators of the center point. For stages smaller than the current, which has bigger scale than the current scale, we use the base value multiplied by the factor of the power of 1.25. Thus each envelop will be 1.25 times bigger then the previous one. For the stages bigger than the current, the base will be divided by power of 1.25 (Figure 13). For different maps, according to the size of map, the base number can be various because the range of the model map is relatively small,

We chose the number 1.25, thus the envelop will not change rapidly. For larger maps, such as the map of the US country or the whole world, the base number can be 2 or larger.

After calculate these envelops, it is ready to draw scale pictures. By adjusting the width and height of image file “pixel.jpg”, we can draw the scale bars to represent different scales (Figure 14).

```
…
   if ( CurrentStage == ActualStage ) {
    ScaleToolHTML += ('<img src="images/pixel.gif" width="3" height="' + i + '"
          hspace="0" vspace="0" border="0" alt="Current Zoom Range"
          style="background-color: #FF0000;" />');
    ScaleToolHTML += ('<img src="images/pixel.gif" width="1" height="' + i + '"
         hspace="0" vspace="0" border="0" alt="Current Zoom Range" />');
    } else {
    ScaleToolHTML += ('<a href="javascript:void(0);"
         onclick="parent.MapFrame.zoomToEnvelope(' + StageEnvelopeX1 + ',' +
          StageEnvelopeY1 + ',' + StageEnvelopeX2 + ',' + StageEnvelopeY2 + ')"
          title="Zoom by factor of ' + (ActualStage - CurrentStage) + '">');
    ScaleToolHTML += ('<img src="usgs_images/pixel_black.gif" width="3"
         height="' + i + '" hspace="0" vspace="0" border="0" alt="Zoom by factor
          of ' + (ActualStage - CurrentStage) + '" style="background-color:
         #000000;" /></a>');
    ScaleToolHTML += ('<a href="javascript:void(0);"
         onclick="parent.MapFrame.zoomToEnvelope(' + StageEnvelopeX1 + ',' +
          StageEnvelopeY1 + ',' + StageEnvelopeX2 + ',' + StageEnvelopeY2 + ')"
          title="Zoom by factor of ' + (ActualStage - CurrentStage) + '">');
    ScaleToolHTML += ('<img src="images/pixel.gif" width="1" height="' + i + '"
          hspace="0" vspace="0" border="0" alt="Zoom by factor of ' +
         (ActualStage - CurrentStage) + '" /></a>');
   }
  }
  ScaleToolHTML += ('</td><td align="center" nowrap>');
  ScaleToolHTML += ('<a href="javascript:void(0);" title="Zooms the map In"
          onclick="parent.MapFrame.zoomButton(1);">');
  ScaleToolHTML += ('<img src="images/zoomin_1.gif" width="16" height="16"
          hspace="0" vspace="0" border="0" alt="Zooms the map In"
          name="zoomcenter" /><br /><small><small>In</small></small></a>');
  ScaleToolHTML += ('</td></tr>');
…
```

Figure 14 code to draw scale bars

To make each scale bar clickable, hyperlinks to JavaScript function zoom-To-Envelop will be attached for each bar. The current scale bar will be drawn in red to be distinctive. Each click on the scale bars will invoke the zoom-To-Envelop function with a corresponding envelop value, A new XML request containing the envelop information will be send to the server. The server then will return a new map image which is zoomed in/out to the user specified level. The scale bars will also got refreshed to mark the current scale with red color.

At this time we choose to locate the scale bar in the TocFrame. And in order to make the scale bar show in the window, we need to install it into toc.htm.

Since there are many functions that use this tool, the script code can be reused by have it stored in a separate JavaScript file. The JavaScript file is included in the JavaScript source code of MapFrame.htm. The DrawScaleTool function can be called in the toc.htm as shown in figure 15.

```
<BODY>
…
<SCRIPT TYPE="text/javascript" LANGUAGE="JavaScript">
…
document.writeln(parent.MapFrame.DrawScaleTool(true));
…
</BODY>
```

Figure15 insert code in toc.htm

3.3 Adding a graduating symbol to the HTML viewer

As mentioned before, it is desirable to make the HTML viewer have the graduating symbol immediately below each layer’s name as the Java Viewer does.

To draw the graduating symbol, we need to know the range and the color of every layer. Fortunately, these values are in the map service configuration. As mentioned in chapter 1, every map service is specified by an ArcXML (AXL) configuration document. Each of these ArcXML document is typically created by the ArcIMS Author tools, such as the ArcIMS designer with which the layer drawing order, the layer properties, and the range and color for the legends are chosen. My enhancement program reads in this kind of ArcIMS configuration file, and gets the information about the map legend's color and range. Figure 16 shows an example of the AXL file, which is a fragment that defines the range and color for one layer.

My enhancement program draws the color legends using colored buttons.

```
<LAYER type="featureclass"
        name="SDE.SDE.M123AMN3011453360321031"
        visible="true" id="0">
    <DATASET name="SDE.SDE.M123AMN3011453360321031"
        type="point" workspace="sde_ws-4" />
    <VALUEMAPRENDERER lookupfield="SPECIES_CO">
      <RANGE lower="0" upper="4" label="Less than 4">
        <SIMPLEMARKERSYMBOL color="255,255,0" />
      </RANGE>
      <RANGE lower="4" upper="8" label="4 - 8">
        <SIMPLEMARKERSYMBOL color="255,204,0" width="4" />
      </RANGE>
      <RANGE lower="8" upper="12" label="8 - 12">
        <SIMPLEMARKERSYMBOL color="255,153,0" width="5" />
      </RANGE>
      <RANGE lower="12" upper="16" label="12 - 16">
        <SIMPLEMARKERSYMBOL color="255,102,0" width="6" />
      </RANGE>
      <RANGE lower="16" upper="20" label="16 - 20">
        <SIMPLEMARKERSYMBOL color="255,51,0" width="7" />
      </RANGE>
    </VALUEMAPRENDERER>
 </LAYER>
```

Figure 16: Layer info in a map configuration file.

In the HTML viewer, the layer list is drawn by a “for-loop” in toc.htm in the order of the layer id. My program creates an array. In this array, each entry contains the HTML statements for the legend corresponding to the array index. Then in the “for-loop”, each array entry is inserted right after each layer name, the corresponding code is shown in figure 17 and the result in figure 18.

```
//////////insert legend array
var legendarray=new Array();
legendarray[0]="<tr><td ALIGN=\"left\" COLSPAN=\"3\">
     <style><!--.initial2{font-weight:bold;background-color:ffff00}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial2\"> less than 7<br>
     <style><!--.initial3{font-weight:bold;background-color:ffe600}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial3\"> 7-13<br>
     <style><!--.initial4{font-weight:bold;background-color:ffcc00}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial4\"> 13-19<br>
     <style><!--.initial5{font-weight:bold;background-color:ffb300}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial5\"> 19-25<br>
     <style><!--.initial6{font-weight:bold;background-color:ff9900}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial6\"> 25-31<br>
     <style><!--.initial7{font-weight:bold;background-color:ff8000}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial7\"> 31-38<br>
     <style><!--.initial8{font-weight:bold;background-color:ff6600}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial8\"> 38-44<br>
     <style><!--.initial9{font-weight:bold;background-color:ff4d00}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial9\"> 44-50<br>
     <style><!--.initial10{font-weight:bold;background-color:ff3300}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial10\"> 50-56<br>
     <style><!--.initial11{font-weight:bold;background-color:ff1a00}//--></style>
     <Input TYPE=\"button\" style=\"font-size: 10px\" class=\"initial11\"> 56-63<br>
     </tr></td>";
…
for (var i=0;i<theCount;i++) {
   if ((!t.hideLayersFromList) || ((t.hideLayersFromList) && (!t.noListLayer[i]))) {
     if ((t.listAllLayers) || ((t.mapScaleFactor>=t.LayerMinScale[i]) &&
          (t.mapScaleFactor<=t.LayerMaxScale[i]))) {
…
      document.writeln('<td><font face="Arial" size="-1">' + t.LayerName[i] + '</font></td>');
       //insert legend here
      document.writeln(legendarray[i]);
…
}
```

Figure 17 insert legend right beneath the layer name in toc.htm

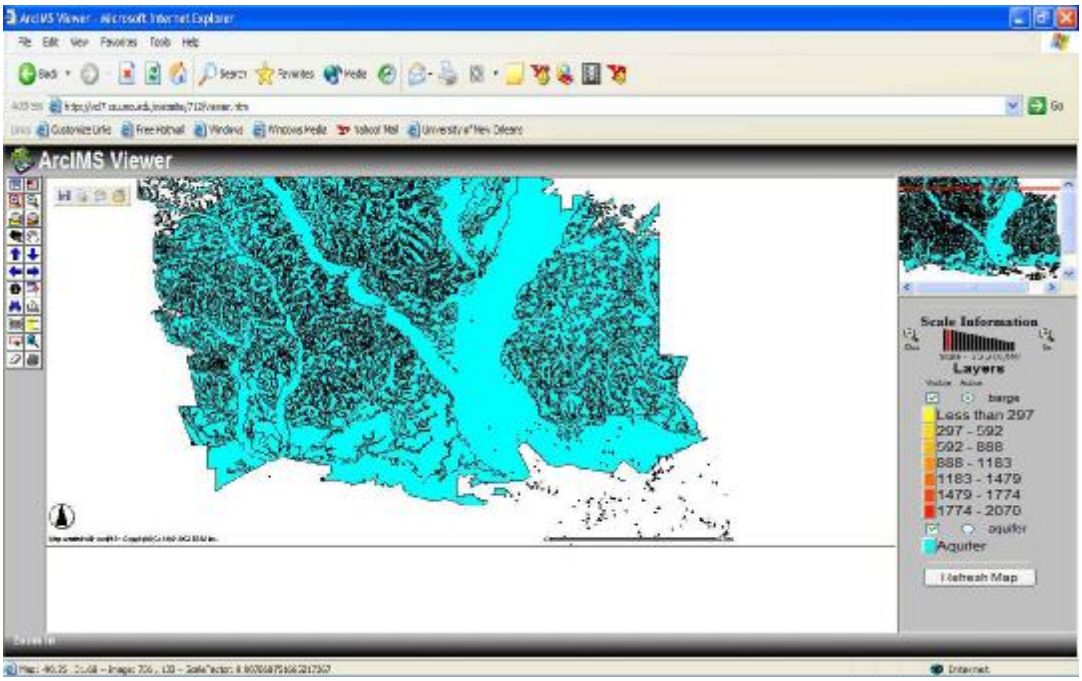

Figure 18 show color legend beneath each layer name in Html Viewer

3.4 Preserving the User Identification Information

As described in section 2.1, we want to restain each user's viewing scope to the user's own data while all the users used the same map service generated by the ArcIMS designer. To do so, I have added a login dialog to the user viewing process, as shown in Figure 19.

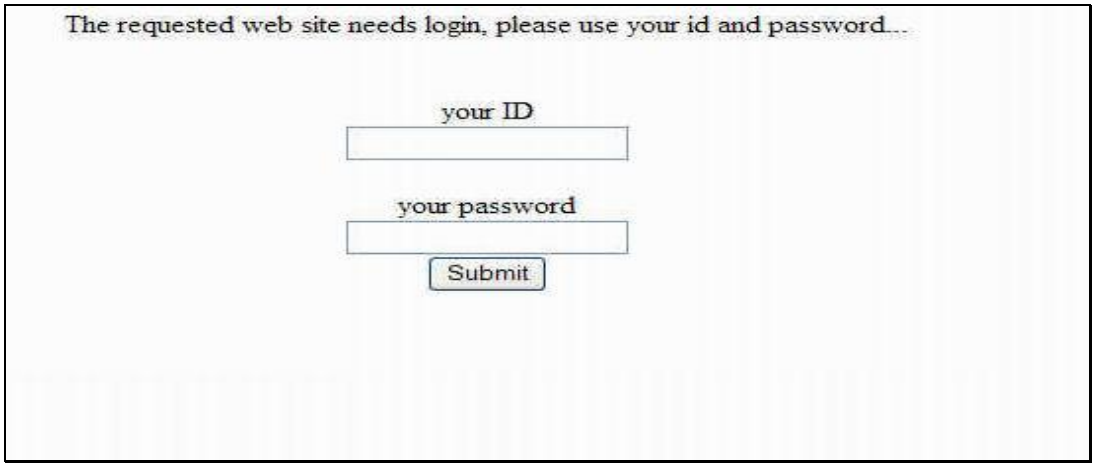

Figure 19: Screen shot of log in page

In my implementation, a table of pairs of user id and password is created in the database. During login, a servlet is called to connect to the database and search for the matching pair. If the table contains of the combination of the given user-id and password, the login is successful. As a result, the servlet delivers the viewer page to the user. Otherwise, an error message will be shown on the user's screen and the user will be redirected back to the log in page. The login process is depicted (Figure 20).

The above apparently simple login process is not adequate for our real needs because the user id and the password will have to be used in every query in the same session. A common Java solution would be to use a "Session bean" to preserve the identification information and control the entire session effectively. [18] However, this J2EE solution is not applicable since we do not have the source code of the server and we cannot afford rewriting the server program. Without modifying the server's code, the user identification information has to be preserved by the client side. Another commonly used approach, the URL rewrite method, suggests to attach the user id and password to the URL that is returned to the browser. That is obviously not secure because the user name and password are exposed to the visible URL. We have to hide the user identification information from being displayed in the visible URL.

In order to solve this problem, I applied a hierarchical frame structure in my web page, same browser window by using frames. It also give us a very good feature to use, that is, once the parent frame is loaded, changing the content of its child frame won't make any change to the URL shown in the address bar.

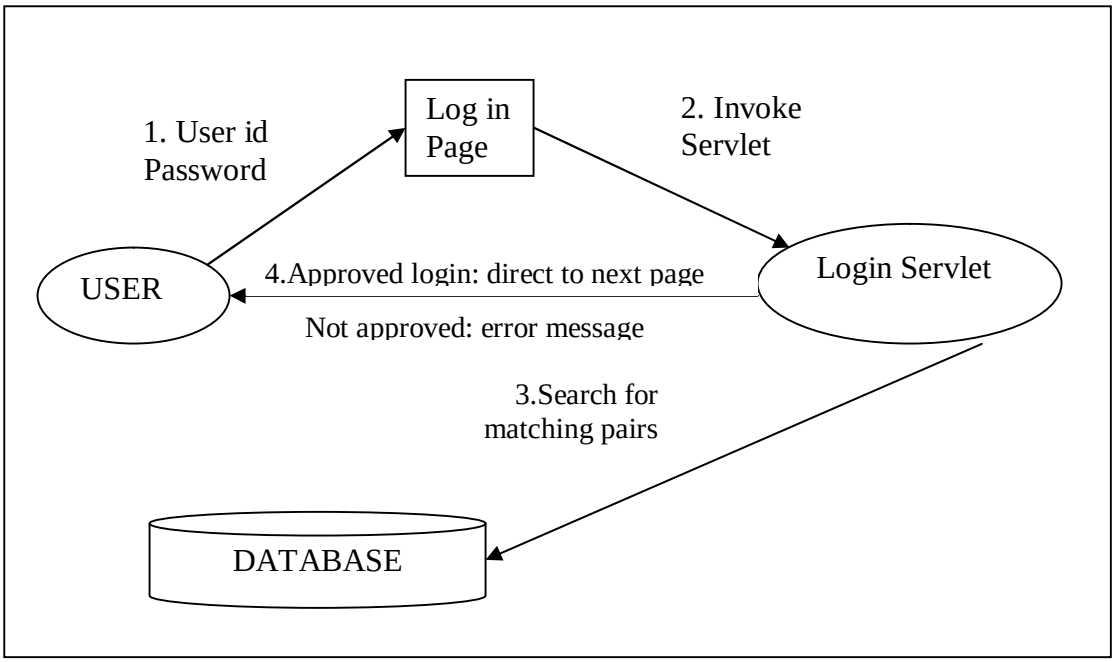

Figure 20 Login procedure

In my frame structure, the sole parent frame contains only one column and one row. The “login.htm” will be the source HTML file to be loaded. When a user logs in through the login page, the user id and password will be passed to the servlet by a form method. Upon a successful login, the servlet will dynamically create the HTML statements to create eight child frames into the viewer page. Since the parent frame is loaded once and will never been changed in the session, the URL of the page will not show the output of the servlet parameters including the user id and the password. Figure 21 illustrates the JavaScript code of this top-level frame, figure 23 the login form. The code in the servlet to carry out the login process is listed in figure 22.

```
…
document.writeln('<FRAMESET ROWS="100%,*" FRAMEBORDER="No"
FRAMESPACING="0" BORDER=0 '+ '>');
document.writeln('<FRAME NAME="loginframe" SRC="login.htm"
MARGINWIDTH="0" MARGINHEIGHT="0" SCROLLING="No"
FRAMEBORDER="Yes" RESIZE="YES">');
...
document.writeln('</FRAMESET>');
…
```

Figure 21 Parent frame to contain the log in page

```
String user_name = request.getParameter("user_name");
 String password = request.getParameter("password");
 boolean allowed=false;
 PrintWriter out = response.getWriter();
Connection sqlConn=null;
// connect to SQL server:
   try{
      Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
      sqlConn = DriverManager.getConnection
       ("jdbc:microsoft:sqlserver://olawin:1433;User=moritor;Password=moritor");
    }catch(ClassNotFoundException e){
     e.printStackTrace();
    }catch(SQLException e){
      e.printStackTrace();}
  try{
        Statement statement=sqlConn.createStatement();
        String query="SELECT * FROM authentication WHERE
                User_Name='"+user_name+"' and password='"+password+"'";
        ResultSet rs=statement.executeQuery(query);
        if(rs.next())
          allowed=true;
        rs.close();
        sqlConn.close();
    }catch(SQLException e){
      e.printStackTrace();}
…
```

Figure 22 Log in servlet

```
…
<form name="myForm" action="/servlet/LoginHandler"><br>
<p>your ID<br>
<input name="user_name" type=TEXT size="20"><br>
: password<br>
<input name="password" type=TEXT size="20"><br>
<input name="Sybmit" type="Submit" value="Submit">

…
```

Figure 23 Form method in login.htm

3.5 Restraining user's viewing scope

In our project, the user's data are points associated with numbers (the bug counts on tapes). In ArcIMS image services, maps are drawn layer by layer by order of layer ids. In ArcIMS, all the request and response communication between server and client are in ArcXML. In ArcXML, element PROPERTIES provides the framework for defining properties about an ArcIMS service, such as the layer info and start extent coordinates of the map. A LAYER used in a map configuration file has several child elements. The purpose of these elements is to identify the data source and to render the data. The transparency of a layer can be defined by setting the transparency value from 0.0 to 1.0, which is the percentage of transparency. 1.0 is 0 percent transparent and 0.0 is 100 percent transparent.

My stategy to retain to user's viewing ability is to set the entire points to be invisible (100% transparent). Then I add a layer to the map which contains only the points belonging to the user. Apparently, the invisible data set wastes computing resource. However, this way minimizes the code modification for the ArcIMS HTML Viewer. The original ArcIMS HTML Viewer assumes that every viewer should see

everything specified by the web map designed by the ArcIMS Author tool. Therefore all the layers are specified by the Author tool. The code for most of the eight query buttons in the left pane of the viewer refers to the active layer, the bug count points. By keeping this layer but turning it invisible, we can use almost all the code except for a few small changes. I experienced a different method in witch the bug count layer was taken away from the initial map but added through a “Get_Map” action dynamically. That method results in rewriting the Javascript code for most of the buttons. On the other hand, the saved computing time was negligible.

While the browser loads the viewer, an ArcXML request will be sent to the spatial server for the map image. In order to make the spatial server add an additional layer to the map, we have added a layer element to the request XML document. A original ArcXML request is shown in figure 24.

```
<ARCXML version = “1.1”>
 <REQUEST>
  <GET_IMAGE>
   <PROPERTIES>
    <ENVELOPE minx=”-94.4919395398005” miny=”29.142727525”
                    maxx=”-89.325330578” maxy=”32.762353291”/>
    <IMAGESIZE height=”426” width=”1042”/>
    <LAYERLIST>
     <LAYERDEF id=”0” visible=”true”/>
     <LAYERDEF id=”1” visible=”true”/>
    </LAYERLIST>
   </PROPERTIES>
   <LAYER type=”acetate” name=”theCopyright”>
    <OBJECT units=”pixel”>
     <TEXT coords=”3 3” label=”Map created with ArcIMS-Copyright©1992
            -2002 ESRI INC.”>
       <TEXTMARKERSYMBO fontstyle=”Regular” fontsize=”8”
            fontcolor=”0,0,0” antialiasing=”true” blockout=”255,255,255”
            overlap=”false”/>
     </TEXT>
    </OBJECT>
   </LAYER>
   <LAYER type=”acetate” name=”theNorthArrow”>
    <OBJECT units=”pixel”>
     <NORTHARROW type=”4” size=”15” coords=”20 35” shadow=”32,32,32
            angle=””0” antialiasing=”True” overlap=”False”>
    </OBJECT>
…..
</GET_IMAGE>
</REQUEST>
</ARCXML>
```

Figure 24 ArcXML request send to server before modification

This request specifies the number of layers to be drawn and their drawing order. The ArcIMS server will then make up a map according to this request. To make the server draw an additional layer, I added a layer element to the original ArcXML request as shown in figure 25.

```
<ARCXML version = “1.1”>
 <REQUEST>
  <GET_IMAGE>
   <PROPERTIES>
    <ENVELOPE minx=”-94.4919395398005” miny=”29.142727525”
                  maxx=”-89.325330578” maxy=”32.762353291”/>
    <IMAGESIZE height=”426” width=”1042”/>
    <LAYERLIST>
     <LAYERDEF id=”0” visible=”true”/>
     <LAYERDEF id=”1” visible=”true”/>
    </LAYERLIST>
   </PROPERTIES>
   <LAYER type=”featureclass” name=”selected features” visible=”true”>
    <DATASET fromlayer=”0”>
     <SPATIALQUERY where=
          "SDE.SDE.TRAPLOACTION.USER_NAME='ywu1’"/>
      <SIMPLERENDERER>
        <SIMPLEMARKERSYMBOL color="227,27,227"
             type="Cycle" width="10"/>
       </SIMPLERENDERER>
   </LAYER>
   <LAYER type=”acetate” name=”theCopyright”>
    <OBJECT units=”pixel”>
     <TEXT coords=”3 3” label=”Map created with ArcIMS-Copyright©1992
          -2002 ESRI INC.”>
      <TEXTMARKERSYMBO fontstyle=”Regular” fontsize=”8”
          fontcolor=”0,0,0” antialiasing=”true” blockout=”255,255,255”
          overlap=”false”/>
     </TEXT>
    </OBJECT>
   </LAYER>
   <LAYER type=”acetate” name=”theNorthArrow”>
    <OBJECT units=”pixel”>
     <NORTHARROW type=”4” size=”15” coords=”20 35” shadow=”32,32,32
          angle=””0” antialiasing=”True” overlap=”False”>
    </OBJECT>
…..
</GET_IMAGE>
</REQUEST>
</ARCXML>
```

Figure 25 Modified ArcXML request send to server

In ArcIMS, all the ArcXML requests are generated by function writeXML() in JavaScript aimsXML.js. By editing this function, I accomplished the necessary

modification to query the data belonging to the user only and to make up the additional layer. Thus, each time a request is sent to server asking for a map image, the ArcXML request for this layer will be added to the generated ArcXML request. In the map returned by the spatial server, a layer with all the points that are associated with the user id will be drawn. (Figure 26)

The default ArcIMS HTML Viewer has a set of query tools such as the “identify” tool, the “select by rectangle” tool, the “find” tool, the “buffer” tool, the “select by line/polygon” tool, and the “user-defined query” tool. Having the user identification information preserved by the top-level frame of the viewer window, we have to restrain the result data to each user’s own data. This section describes how to modify the JavaScript functions that enforce these tools.

Although we made all other users’ points invisible to the current user, those points are still there and may actually cause errors if we don’t modify the tool functions. For example, without modification of the tool “select by rectangle”, this function tool will display all the data information it get from the server including that of the invisible points. Other tools such as the identify tool and the find tool also have similar problems.

```
…
// prepare the request in xml format for Main Map
function writeXML() {
…
        theString += '</PROPERTIES>\n';
        // add extra layer here
        var myaddlayer = '<LAYER type="featureclass" name="selected Features"
                visible="true">\n';
        myaddlayer += '<DATASET fromlayer="1"/>\n';
        myaddlayer += '<SPATIALQUERY where=
              "SDE.SDE.MORITOR.USER_NAME=\"+parent.user_name+\"'/>\n';
        myaddlayer +='<VALUEMAPRENDERER lookupfield=
                "SPECIES_COUNT">\n';
        myaddlayer +='  <RANGE lower="1" upper="15" label="Less than 15">\n';
        myaddlayer +='    <SIMPLEMARKERSYMBOL color="255,175,175"
                 width="6"/>\n';
        myaddlayer +='</RANGE>\n';
        myaddlayer +='<RANGE lower="15" upper="29" label="15 - 29">\n';
        myaddlayer +='<SIMPLEMARKERSYMBOL color="255,140,140"
                width="7" />\n';
        myaddlayer +='</RANGE>\n';
        …
        myaddlayer +='</VALUEMAPRENDERER>\n';
        myaddlayer += '</LAYER>\n';
        theString = theString+myaddlayer;
        …
        return theString;
}
```

Figure 26: code in writeXML() to add an extra layer

```
…
//our condition statement
var quyid = parent.gettheid;
var quyquystring ='ID = \'"+quyid+\'";
…
// process query
function sendQueryString(newString) {
        //add conditional statement here
        newString = newString+" AND "+quyquystring;
…
        var theString = writeQueryXML(newString);
        sendToServer(imsQueryURL,theString,queryXMLMode);
}
…
```

Figure 27 modification for the function for “user-defined query”

The “user-defined query” tool is for the user to search for features based on a query expression. Once the query string is input by the user, an ArcXML request is constructed and sent to the server. The searching result will be displayed in the TextFrame. When the user sends query request to the server, an additional condition must be added to the query to restrict the search range. Since this query (in ArcXML) is dynamically generated by the JavaScript function sendQueryString() in aimsQuery.js, I have modified it by inserting a condition statement into the request as shown in figure 27.

The “Identify” tool is used to display attribute information for the feature that the user clicked. The reason to modify this function is that in case the user clicks on an invisible point which belongs to other users, the coordinates of this point will be sent to the sever and in return, feature of this point will be shown to the current user. So restriction should be added to the query request so that in this circumstance the server won’t return other person’s information to the current user. The function that write xml requests for this “Identify” tool is defined as writeIdentifyXML() in aimsIdentify.js.

```
…//conditional statement
var idenid = parent.gettheid;
var idenquery = "ID = \"'+idenid+"\"";
…
function writeIdentifyXML
        (theLayer,theLayerType,theFields,leftX,bottomY,rightX,topY,
        maxReturned,hasLimit) {
        var theString = '<ARCXML version="1.1">\n<REQUEST>\n
                <GET_FEATURES outputmode="xml" envelope="false"
                checkesc ="true" geometry="false" featurelimit="' +
                maxReturned + '">\n';
        theString += '<LAYER id="' + theLayer + '" />';

        //add conditional statement here
        theString += '<SPATIALQUERY subfields="' + theFields + '"
                where="' +      idenquery + '">';
…
        return theString;
}
```

Figure 28 modify function for “Identify” tool

The “Find” tool is used to find map features with an attribute value matching a string that user typed in. The server will search the whole database to find result that has the matching string in any attribute fields. The function that write ArcXML requests for this “Find” tool is defined as writeFindRequest() in aimsQuery.js. The modification in this function is shown in Figure 29

```
var quyid = parent.gettheid;
var quyquy ='ID = \"+quyid+'\";
…
// write out find form
function writeFindRequest(findQuery,fieldList) {
        var theString = '<ARCXML version="1.1">\n<REQUEST>\n
                <GET_FEATURES  outputmode="xml" geometry="false"
                envelope="true" checkesc ="true"';
        …
        if (useLimitExtent) {
        theString += '<SPATIALQUERY subfields="' + fieldList + '"
                where="('+quyquy+') AND (' + findQuery + ')" />';
        …
        } else {
                //add condition here
                theString += '<QUERY subfields="' + fieldList + '"
                        where="('+quyquy+') AND (' + findQuery + ')" />';
…       }…
        return theString;
}
```

Figure 29 modify function for “Find” tool

The “buffer” tool is used to select the features of one layer that are within the specified buffer distance of selected features of another layer. The “Select by Rectangle” tool is used to select the group of features contained by or in contact with a rectangle user draw on the map. And the “Select by Line/Polygon” tool is used to select the group of features contained by or in contact with a line or polygon user draw on the map. They are fulfilled by the following JavaScript functions:

I have modified each of these functions by adding a conditional statement to the code as part of the spatial query in a way similar to the code shown in figure 28.

Tools	**JavaScript Function**	**JavaScript File**
QUERY	function sendQueryString(newString) {…} function writeFindRequest(findQuery,fieldList) {…}	aimsQuery.js
Identify	function writeIdentifyXML (theLayer,theLayerType,theFields,leftX,bottomY, rightX,topY, maxReturned,hasLimit) {…}	aimsIdentify.js
Select	function writeEnvelopeXML (theLayer,theLayerType,theFields, maxReturned,startRec,theEnvelope,hasLimit) {…} function writeShapeSelect(theType) {…} function addSelectToMap(){…}	aimsSelect.js
Buffer	function addBuffertoMap(){…} function writeQueryBufferXML(){…} function writeShapeBufferXML(theType){…} function writeEnvelopeBufferXML(){…}	aimsBuffer.js

Table 1 modified JavaScript functions

3.6 Automation of the enhancement

In the previous sections, I have described the method to enhance the ArcIMS HTML Viewer. Since manually modifying the ArcXML documents and the JavaScript code is error prone. I have programmed a graphic user interface (GUI) automate the

modification (Figure 30). Since adding color legend needs to read the configuration file, the add legend buttons is initially disabled. The user will have to specify the ArcXML configuration file using the “map to be edited” button to active it.

Figure 30 GUI for automation tool

Each feature enhancement such as adding color legend, adding scale tool and moving overview window should only be carried out once for a web site. Once a button is pressed, it will be disabled after the corresponding enhancement is done. The “enhance all” button is to perform every enhancement altogether. When this button is pressed, all buttons except the “exit” button will be set to disabled. The code that automates enhancements is organized in two packages.

The first package is for the interface editing, it contains four java class, one JavaScript file and one HTML file as listed in table2.

For the user who needs no login addition, this is the only package they need.

The second package is for the login addition. It contains one servlet program, five JavaScript files and two HTML files as listed in table 3.

Java Class	JavaScript file	HTML file
Top.class	Scaletool.js	Overview.htm
Mod.class		
ToolGUI.class		
Axlchooser.class		

Table 2 files for package one

Servlet program	JavaScript files	HTML files
LoginHandler.class	aimsXML.js	viewer.htm
	aimsQuery.js	login.htm
	aimsSelect.js	
	aimsBuffer.js	
	aimsIdentify.js	

Table 3 files for package two

Use this package will allow user to add authorization and authentication feature to the web site. The user will need to do three steps to achieve this goal.

Step one, copy all HTML and JavaScript files into working directory. And adjust variables of those files.

Step two, create a user name and password table in the database.

Step three, adjust the variables of the servlet file and compile it, put the class file into the web server's working directory, and restart the web server.

Modification of the GUI features must be carried out before adding the login process.

CHAPTER 4

CONCLUSIONS

In the modern geographic information systems, COTS software has been playing a major role. Customizing COTS software is inevitable because large organizations' needs usually exceed COTS built-in functions. However, this is often a challenge to COTS users, since the source code of COTS is rarely available.

In my thesis project, I have enhanced some functions of a leading GIS COTS product, ArcIMS, by taking advantage of this web publishing tool's well-thought architecture of services and its applications of the XML messaging technology. Specifically, my enhancements are in the aspects. First, I have made it possible for multiple users to access the same ArcIMS map service but with constrained viewing power. Not only I have added a login dialog to the map service, also have I modified all the query tools such that each user can view the user's own data from the same map service. Secondly, I enhanced the features of the GUI of the ArcIMS HTML viewer such that the HTML viewer can carry out all the functions that the ArcIMS Java plug-in viewer can do. Using my enhanced ArcIMS HTML viewer, users can enjoy all the powerful tools that the Java plug-in viewer has and avoid the complicated installation of the plug-in and the possible JVM version mismatch problems. In order for the ArcIMS end users to carry out the above enhancements, I have further implemented a program that automates the customization.

The problems that I encountered in the above enhancements (customizations) are very interesting. Implementing these enhancements requires a careful design, too. However, the design goals of COTS customization are very different from developing a home-made system. For example, minimizing changes is a very important design criterion in COTS customization. This is important because minimum changes to the COTS will maximize the chance for the customized system to survive in COTS upgrades. I have also learned that the XML messaging technology can greatly ease software integration and customization.

Some of my customizations can be improved. For instance, I have added the graduating legend feature to the HTML viewer. However, the legends' range and color spectrum are predefined by the map service author. When new data lying different range, the fixed legends will not reflect the new data. It would be better if the range and the color spectrum are dynamically determined according to the data. This is an example of my future works.

REFERENCES

1. Dean Gravel, *COTS-Based Software Systems, First International Conference, ICCBSS 2002*
2. Jerry Bradenbaugh; *Javascript Application Cookbook*
3. William H. Murray; Chris H. Pappas; *Javascript & HTML 4.0 User's Resource*
4. Michael Floyd; *Building Web Sites with XML*
5. Servlets White page: http://java.sun.com/products/servlet/whitepaper.html
6. *ESRI ArcXML Programmer's Reference Guide*
7. *ESRI Using ArcIMS*
8. ESRI. ArchIMS online http://www.esri.com/software/arcims/index.html
9. ESRI Support Center http://support.esri.com
10. ESRI User Forum http://support.esri.com/index.cfm?fa=forums.gateway
11. ESRI Support – *Technical Articles: HowTo: Use the JDBC-ODBC bridge to implement ArcIMS authentication with an MSAccess database*

http://support.esri.com/index.cfm?fa=knowledgebase.techarticles.articleShow&d=21458

12. DHTML Tutorial http://w3schools.com/dhtml/default.asp
13. MS SQL Server online help http://www.microsoft.com/sql
14. Marty Hall; *More Servlets and JavaServer Pages.*
15. Chris Abts, *COTS-BASED System (CBS) Functional Density – A Heuristic for Better CBS Design*

16. Maurizio Morisio, Marco Torchiano; Definition and Classification of COTS: A Proposal

17. R. W. Sinnott; *Virtues of the Haversine* Sky and Telescope, vol. 68, no. 2 1984, p.159

18. Tutorial of J2EE programming.

VITA

The author was born in Wuhan, Hubei Province, China on May 14, 1976. She graduated from the Hubei Province Experimental School at Wuchang in July of 1993 and then began undergraduate study at TongJi Medical University, China in the Fall of 1993. After 5 years' study at TongJi Medical University, she earned the B.S degree in Medical Science in July of 1998. After one year's experience as resident physician, she came to the University of New Orleans, USA, to pursue her M.S. degree in Computer Science, under the supervision of Prof. Shengru Tu.

THESIS EXAMINATION REPORT

CANDIDATE: Jin Chen

MAJOR FIELD: Computer Science

TITLE OF THESIS: Enhancement of COTS GIS Web Publishing Software

APPROVED:

Major Professor & Chair –Dr. Shengru Tu

Dean of the Graduate School

EXAMINING COMMITTEE:

Dr. Shengru Tu

Dr. Ming-Hsing Chiu

Dr. Golden G. Richard III

DATE OF EXAMINATION: August 18, 2003