THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato
Research Commons

http://waikato.researchgateway.ac.nz/

Research Commons at the University of Waikato

Copyright Statement:

The digital copy of this thesis is protected by the Copyright Act 1994 (New Zealand).

The thesis may be consulted by you, provided you comply with the provisions of the Act and the following conditions of use:

- Any use you make of these documents or images must be for research or private study purposes only, and you may not make them available to any other person.
- Authors control the copyright of their thesis. You will recognise the author's right to be identified as the author of the thesis, and due acknowledgement will be made to the author where appropriate.
- You will obtain the author's permission before publishing any material from the thesis.

A Web-based Geographical Information System for Low Bandwidth Access

Shi Zhennan

This thesis is submitted in partial fulfillment of the requirements for the Degree of Master of Science at the University of Waikato.

July 2007

© Shi Zhennan 2007

ABSTRACT

The Geographic Information Systems (GIS) have become popular tool, used in different fields. The launching of Google Maps offered a new approach of building web based GIS systems; making it possible to integrate external geographically referenced data with the powerful map service supplied by Google.

This thesis demonstrates the design and implementation of creating a web based geographic information system. The system is built by adapting the Google Maps API library and building a web server to display and explore agricultural data.

ACKNOWLEDGMENTS

My thanks to my supervisor Bill Rogers for his wonderful ideas, great patience and kind support during the thesis. My thanks also go to the TSG people in the department for the technical support.

My personal thanks to my parents for also being there for me, also to my girlfriend Lan Peng(Paula) for her continuous care, encouragement and help during the whole time.

CONTENTS

ABSTRACT .... A

ACKNOWLEDGMENTS .... B

CONTENTS C

LIST OF FIGURES .... F

LIST OF TABLES I

CHAPTER 1 INTRODUCTION **1**

1.1 THESIS STRUCTURE **3**

CHAPTER 2 LITERATURE REVIEW **5**

2.1 THE USE OF THE GEOGRAPHIC INFORMATION SYSTEM (GIS) 5

2.2 MAPPING TOOLS IN CONSTRUCTING GIS SYSTEMS 7

2.3 GOOGLE MAPS IMPLEMENTATIONS 8

2.3.1 Geospatial visualisation of student population using Google Maps 8

2.3.2 Mapping crimes in Chicago City 10

2.3.3 Weather Bonk showing real time weather information using Google Maps 11

2.4 CONCLUSION 14

CHAPTER 3 WEB BASED APPLICATION **16**

3.1 CLIENT SIDE SCRIPTING 17

3.2 DOCUMENT OBJECT MODEL (DOM) 18

3.2.1 Background History 18

3.2.2 DOM examples 19

3.3 DYNAMIC HYPERTEXT MARKUP LANGUAGE (DHTML) 27

CHAPTER 4 WEB APPLICATION MODELS, AJAX AND GOOGLE MAPS **36**

4.1 CLASSIC WEB APPLICATION MODEL (SYNCHRONOUS)........ 37
4.2 AJAX WEB APPLICATION MODEL (ASYNCHRONOUS)........ 38
4.3 HISTORY OF THE AJAX........ 40
4.4 AJAX COMPONENTS........ 41
4.5 A SIMPLE AJAX EXAMPLE........ 44

CHAPTER 5 GOOGLE MAPS, THESIS PROJECT **58**

5.1 INTRODUCTION TO GOOGLE MAPS........ 58
5.2 THE THESIS PROJECT........ 60
5.3 SETTING UP THE EXPERIMENT ENVIRONMENT........ 63
5.4 FINDING CLASS DEFINITIONS........ 67
5.5 MAIN API CLASSES AND THE IMAGE TILING SYSTEM........ 68
5.6 THE COORDINATE SYSTEM........ 74
5.6.1 The Map Projection function........ 76
5.6.2 Coordinate system used in agricultural data........ 76
5.6.3 The transformation between two spaces........ 77

CHAPTER 6 THE IMPLEMENTATION **83**

6.1 THE NEW MAP TYPE........ 83
6.2 ADDING THE CUSTOM IMAGE DATA........ 83
6.2.1 Preparing the Image Tiles........ 85
6.2.2 Setting the Image Tiling System........ 87
6.3 SETTING THE COORDINATE SYSTEM........ 89
6.3.1 Determining the Parameters for the Transformation between Two Spaces........90
6.3.2 The Transformation Functions........ 92
6.4 ADDING THE CUSTOM MAP TO THE API LIBRARY........ 97
6.5 THE AGRICULTURAL DATA........ 102
6.5.1 Adding Circular/Triangle Overlays on the Map........ 103
6.5.2 Fetching Data in the AJAX Fashion........105

CHAPTER 7 CONCLUSION **115**

7.1 SUGGESTIONS FOR FURTHER WORK 117

7.2 FINAL REMARKS 117

References........a

LIST OF FIGURES

FIGURE 2. 1 VISUALISATION OF STUDENT POPULATION. 10
FIGURE 2. 2 A MAP SHOWING HOMICIDE CRIME SCENES. DETAILS OF THE CHOSEN CASE ARE SHOWN IN THE INFO WINDOW. 11
FIGURE 2. 3 USING CUSTOM MARKERS TO DISPLAY TEMPERATURES OVER PLACES . 12
FIGURE 2. 4 CUSTOM MARKERS SHOWING THE WIND DIRECTIONS AND TEMPERATURE OVER LOCATIONS. 13
FIGURE 2. 5 AN INFO WINDOW SHOWING DETAILED WEATHER INFORMATION IN CLEVELAND/OHIO/USA. 14

FIGURE 3. 1 THE DOM TREE OF THE EXAMPLE HTML DOCUMENT 20
FIGURE 3. 2 THE LOOK OF THE PAGE BEFORE THE CLICK ON THE BUTTON, THE SIZE OF THE IMAGE IS 200X200. 29
FIGURE 3. 3 THE LOOK OF THE PAGE AFTER THE BUTTON CLICK. THE SIZE OF THE IMAGE IS NOW 500X500 PIXELS. 30
FIGURE 3. 4 WEB PAGE SHOWING THE IMAGE IN ITS ORIGINAL SIZE (200 X 200 PIXELS).. 34
FIGURE 3. 5 ENLARGED IMAGE BY ADJUST THE SLIDER BAR CONTROL.................... 35

FIGURE 4. 1 CLASSIC WEB APPLICATION MODEL 37
FIGURE 4. 2 AJAX WEB APPLICATION MODEL... 40
FIGURE 4. 3 THE EXAMPLE APPLICATION USER INTERFACE 46
FIGURE 4. 4 A TABLE ON THE INTERFACE SHOWING THE SELECTED PERSONAL INFORMATION.. 51
FIGURE 4. 5 THE EXAMPLE USER INTERFACE 53
FIGURE 4. 6 INFORMATION OF SELECTED PERSON DISPLAYED ON THE INTERFACE .. 57

FIGURE 5. 1 GOOGLE MAPS USER INTERFACE.. 58
FIGURE 5. 2 A SIMPLE GOOGLE MAPS EXAMPLE. A PAGE WITH A GOOGLE MAPS DISPLAY EMBEDDED 61
FIGURE 5. 3 A MAP WITH THE MAP TYPE CONTROL AND ZOOM CONTROL ENABLED 63
FIGURE 5. 4 A MAP SHOWING THE SATELLITE VIEW OF AN AREA............................. 69

FIGURE 5. 5 ONE OF THE IMAGE TILES USED TO CREATE THE MAP SHOWN IN FIGURE 5.4 70
FIGURE 5. 6 A MAP SHOWING THE WORLD AT THE LOWEST RESOLUTION IN GOOGLE MAPS 71
FIGURE 5. 7 THE SINGLE IMAGE TILE USED TO CREATE THE MAP SHOWN IN FIGURE 5.6 72
FIGURE 5. 8 A MAP OF WORLD AT THE SECOND LOWEST RESOLUTION IN GOOGLE MAPS 72
FIGURE 5. 9 THE FOUR IMAGE TILES USED TO CREATE THE MAP SHOWN IN FIGURE 5.8 73
FIGURE 5. 10 DIAGRAM OF LATITUDE, LONGITUDE GRID ON A SPHERICAL EARTH (AFTER CHARTON, 1988) 75

FIGURE 6. 1 IMAGE DATA FOR THE CUSTOM MAP TYPE. 84
FIGURE 6. 2 THE 4 IMAGE TILES FOR RESOLUTION LEVEL "4" OF THE CUSTOM MAP 86
FIGURE 6. 3 FOUR IMAGE TILES USED FOR THE CUSTOM MAP AT RESOLUTION LEVEL"4" 88
FIGURE 6. 4 THE GEOGRAPHIC RANGE COVERED IN THE CUSTOM MAP 89
FIGURE 6. 5 MAP SWITCH BUTTONS ON THE GOOGLE MAPS INTERFACE 97
FIGURE 6. 6 ADAPTED GOOGLE MAPS INTERFACE WITH SWITCH BUTTON FOR OUR CUSTOM MAP 101
FIGURE 6. 7 THE CUSTOM MAP 101
FIGURE 6. 8 MAP WITH A FILLED CIRCULAR OVERLAY ADDED ON 105
FIGURE 6. 9 A DROPDOWN MENU ON THE MAP INTERFACE FROM WHICH DIFFERENT ATTRIBUTE OPTIONS CAN BE SELECTED 106
FIGURE 6. 10 THE SLIDER BAR CONTROL SHOWS WHEN OPTION "DRYMATTER" IS SELECTED 107
FIGURE 6. 11 LOCATIONS WITH VALUES GREATER THAN 18.314% FOR ATTRIBUTE "DRY MATTER" 110
FIGURE 6. 12 MAP SHOWING LOCATIONS WITH VALUES GREATER THAN 16.622% FOR ATTRIBUTE "DRY MATTER" 111
FIGURE 6. 13 MAP SHOWING LOCATIONS WITH THE VALUE "HW" FOR ATTRIBUTE "VARIETY" 112

FIGURE 6. 14 MAP SHOWING LOCATIONS WITH VALUE "GK" FOR THE ATTRIBUTE "VARIETY" 113
FIGURE 6. 15 MAP SHOWING LOCATIONS WITH EITHER VALUES FOR ATTRIBUTE "VARIETY" 114

LIST OF TABLES

TABLE 4. 1 METHODS OF THE *XMLHTTPREQUEST* OBJECT 43

TABLE 4. 2 PROPERTIES OF THE *XMLHTTPREQUEST* OBJECT 44

TABLE 4. 3 PERSONAL DATA USED IN THE EXAMPLE 45

TABLE 4. 4 SAME SET OF PERSONAL DATA USED IN THE EXAMPLE 52

TABLE 6. 1 TABLE SHOWING THE RELATIONSHIP BETWEEN THE RESOLUTION LEVELS, THE PIXEL SPACE SIZES AND THE NUMBER OF IMAGE TILES 85

TABLE 6. 2 A SINGLE RECORD IN THE AGRICULTURAL DATABASE 103

Chapter 1 INTRODUCTION

Since the Internet became widely available, there have been all sorts of web based applications which bring a great deal of convenience and pleasure to people's lives.

The web based geographical information system (GIS) is one of the popular and useful web based applications in current use. It is a computer system that is capable of storing, managing, and presenting geographically referenced information. GIS systems are used in many fields, such as environmental control, tourism, scientific researches, resource management, marketing, and sales.

GIS systems can be used to provide support in agricultural production. For example, data collected from agricultural sources is often geographically referenced and may include a number of agricultural attributes, such as climatic features in the areas (e.g. temperature, humidity, and air pressure), soil quality, variety of agricultural products and so forth. Often, farmers have analysis done on samples of soil or produce taken from different places at their farms. Usually, samples are taken back to a laboratory for processing. This data may also be the basis used for different analysis and modelling. The machine learning group at Waikato University is actively involved in such work.

Since the agricultural data is geographically referenced (sampled at locations), a GIS system can be used to geospatially present the patterns of the agricultural production and the results of the analysis on the agricultural data.

A web based system can offer farmers the convenience of rapid access to their test results as soon as the tests have been completed. There is also a possibility that farmers could try different management scenarios on models of their farms based on their test data. For this reason, the machine learning group is interested in providing an interactive web based system to allow farmers to visualise and manipulate data, both experimented as a result of modelling.

This thesis project was to build such a web based GIS system to provide support in presenting and exploring agricultural information for people living in rural areas.

The whole task was divided into two major subtasks. Firstly, we needed to find a mapping tool, using which we could build a map to give an aerial view of the agricultural area. Secondly, we need to build a database to store all the relevant geographical and agricultural information that was to be graphically displayed on the map.

There have been a number of web mapping tools released. In particular, Google Maps, a product by Google, provides people with smooth, highly responsive visual aerial views at various resolutions. Also, Google Maps provides developers with an open API so people can build their own map applications by embedding the mapping services on their websites.

Because of the advantages that Google Maps had, it was the best choice for the web mapping tool to adopt to build the map in our GIS system. However, there were certain obstacles to using Google Maps directly. Since the image resolution on Google Maps of the area we focus on is not high enough for the requirements, we have to make our own custom image data to replace the Google Maps data for the area. The geographic reference used in the agricultural data is specified in a coordinate system that is not supported by Google Maps, which means we have to change the coordinate system applied in Google Maps as well. The solution chosen was to create a custom map using our own custom map imagery data and defining our own custom coordinate system by adapting the Javascript API library source code of Google Maps. Also, using Google Maps requires high speed Internet connection which is not usually available in rural areas in New Zealand. In order to make the system practically usable for people in rural areas through a low speed Internet connection, we decided to store all the map imagery data in the users' local file systems to reduce the network workload.

Creating such a custom map requires a great deal of work studying how Google Maps and the technology applied behind it works. This involved an intensive analysis on the Javascript source code of Google Maps API library.

For the second subtask, a mySQL database was to be built and maintained on an experimental server to store all the agricultural data. PHP scripts were developed to interface the database with the

To sum up, the structure of the whole system is as follows, the system is a web based GIS application, which involves both server side and client side. An adapted Google Maps display is used to build the map in the GIS system. A database is maintained on the server to store all the geographically referenced agricultural data. Users can interact with the application to send requests to the server to obtain agricultural data from the database. The queried data is then sent back to the client side (web browser) and geospatially presented on the map.

1.1 Thesis Structure

There are totally 7 chapters included in this thesis. Chapter 2 gives a review of some web based GIS applications based on making use of Google Maps API.

Chapter3 gives a description of the web based applications and some major web technologies popularly used in building web based applications: the Document Object Model and Dynamic HTML. Examples are presented and discussed for each of the technologies.

Chapter 4 describes the two data transfer models applied in the web based applications. Of these, the AJAX model (asynchronous data transfer model) has become very popular in building web based applications (e.g. the Google Maps). An AJAX application example is also provided in this chapter.

Chapter 5 describes the task of this thesis project. An analysis of the image tiling system and the coordinate system adopted in Google Maps is included as the

project application is built on the Google Maps API. Also the process of setting up the experiment environment is described in the chapter.

Chapter 6 describes the implementation of the thesis project. It explains in details how the web GIS application was built and how the agricultural data was presented.

Chapter 7 gives the conclusion of the thesis project.

Chapter 2 LITERATURE REVIEW

The thesis project was to design and build an interactive web based GIS system (geographic information system). The system consists of two major components. One is a web mapping application which covers an agricultural area in New Zealand. The other is a database of the agricultural information which is geographically referenced to the agricultural area covered by the map application. The task was to graphically display the agricultural information on the web mapping application depending on user's queries.

2.1 The use of the Geographic Information System (GIS)

The term geographic information system has been used to describe a variety of software systems and sometimes their associated hardware [7, 17, 23, 43]. A definition which is useful for the purposes of this thesis is:

"*A geographic information system is a system for capturing, storing, analyzing and managing data associated attributes which are spatially referenced to the eart*h" [34].

A GIS system can be viewed from different angles:

- A GIS system can be viewed as a database storing geographic information of the world, an information system for geography. The whole system is fundamentally based on a structured database that describes the world in geographic referenced features.

- A GIS system is most often associated with maps. A GIS system can be viewed as a set of map views that graphically present the geographic features on the surface of the earth. Also, other kinds of geographic referenced information can be displayed on the maps. For example, a map

can be used to give a view of certain area. Then graphic markers (lines, coloured dots, arrows…) can be placed on the map to give indications (tourist attractions, driving directions, business addresses…).

Moreover, the maps in a GIS system often work as the user interface through which users can work with the geographic data in the system.

Nowadays, given the power to integrate all sorts of geographic referenced information into a graphic presentation, the geographic information systems have been used in all kinds of different fields:

- Environmental control – environmental issues can be described by a list of environmental variables, such as water, air, soil, noise. Geographically referenced data can be extracted from the measurements of these environmental variables. GIS systems are then used to assist in the analysis and assessment of the all kinds of environmental problems, such as park usage, pollution assessment, and situations in natural disaster control.

- Public security – GIS systems have been utilised by police forces for use in a variety of operational situations. GIS systems such as crime mapping tools are used in analysing crime patterns in working towards reducing crime rates. Also, GSI systems are used to provide accurate guidance to crime scenes so that responding officers can save valuable time.

- Transportation – GIS systems have been playing an important part in managing, planning, evaluating and maintaining transportation systems. A real time traffic analysis using GIS systems provides assistance in the development of high way infrastructures.

2.2 Mapping tools in constructing GIS systems

There are two key components to a GIS system. One is the database that contains the geographically referenced information. The other is the set of maps on which the geographical referenced data are presented. An important part of the implementation of a GIS system is constructing the base maps. The base maps in the GIS systems can be constructed using desktop mapping programs or web mapping applications.

Maps are an important component in GIS systems. Many GIS systems use maps as their user interfaces. Through maps GIS system users obtain a way to work with the geographic data in the GIS system. Maps in GIS systems link the GIS data to geographic locations. Also the product of a GIS system most often takes the form of a map (a graphical presentation of the geographically referenced data).

Implementation of a GIS system often involves mapping programs for constructing the set of maps in the system. Available web mapping services offer a straightforward way to build maps for web based GIS systems. External data sources can be integrated into the web mapping services to build complete GIS systems.

With the launching of Google Maps, GIS developers are provided with a powerful web mapping service for constructing GIS base maps. Google Maps offers three types of maps (the standard street map, the satellite map, and the hybrid map) of the world at various resolutions. Also, the Google Maps provides a very interactive user interface – navigation on the map can simply be done by performing “drag and drop” on the map using the mouse.

Being built in the AJAX web application model (discussion of the AJAX application model is included in the chapter 4 in this thesis), Google Maps performs with high responsiveness. Unlike with most of other online mapping applications, Google Maps users do not experience “blank browser” when

directing the map to a new area. And for fact, the demanded part of the map always shows on the screen quickly and users still can interact with the application during the loading of the new map. The functionality of the mapping application is based on a number of Javascript modules, which are loaded from the Google's site then executed on the client side (web browser). The fact that Google Maps functions on the client side makes it even possible for web developers to add in their own map imagery.

Among all the advantages Google Maps has, the most meaningful one for web developers is the application programming interface (API) offered by Google Maps. The Google Maps API [15] library consists of a number of Javascript modules. The API library offers methods which enable web developers to embed a Google map in their own web applications. Also, the Google Maps API supports a set of graphical map overlays. This provides a graphic toolkit for creating visualisations of the external geographic referenced data to be associated with the map. And, even nicer, the Google Maps API is freely available.

Having all the advantages and convenience mentioned in the last paragraph, Google Maps was chosen to be the development platform for the implementation of this thesis project.

2.3 Google Maps implementations

Since the Google Maps API was published, a number of web applications have been developed that integrate external custom data on Google Maps. These Google Maps implementations are not much discussed in the research literature, but are available as websites.

2.3.1 Geospatial visualisation of student population using Google Maps

The article [21] describes a web GIS application that presents a visual geospatial overview of the student population distribution using Google Maps.

The implementation of the project involved mainly three components:

- The user interface – a Google Maps display was embedded into the user interface of the application to construct the base map of the visualisation. On top of the base map, graphic overlays supported in the Google Maps API (GMarker and GIcon) were drawn to present the distribution of the student population.

- Student database – ZIP codes of students' locations were collected then translated into corresponding latitudes and longitudes as the Google Maps API is based on the latitude/longitude coordinate system. Then a MySQL database was built and maintained to store the location information.

- PHP file – the file was stored and running on the experimental server. Instructions in the file established a connection to the students' location database and ran a query against it. Also, the queried data were written into the form of an XML file as the Google Maps application took XML as the acceptable format of external datasource.

The information of the students' location was queried from the database then written into an XML file by the instructions included in the PHP file. The data in the XML package was then sent to the client side (web browser). Markers created using the Google Maps API class "GMarker" were then placed on the map to indicate the locations of the students.

Markers on the map were colour coded. Different colours of a marker denoted the different number of students from the location pinned by marker. Therefore, combining the location where a marker was pinned and the colour of the marker, users are able to tell how many students were from the indicated location. A map with all the markers in different colours pinned on then sufficiently described the distribution of the students' population.

The article also includes a snapshot of the visualisation of the students' population on the application map:

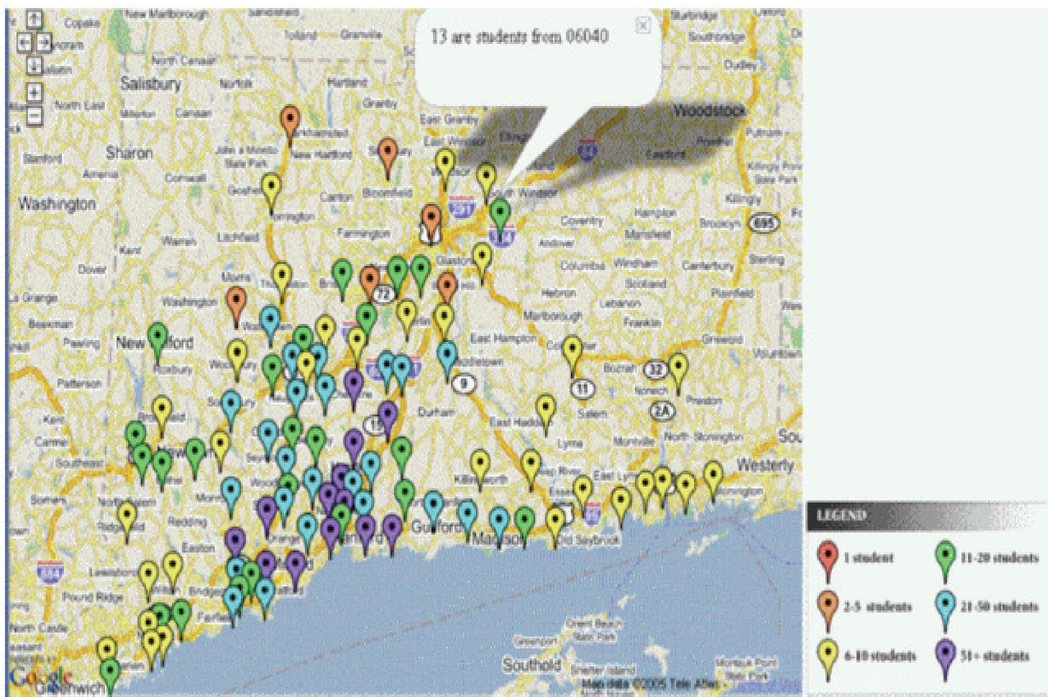

Figure 2.1 visualisation of student population.

2.3.2 Mapping crimes in Chicago City

This is an example of police use of the GIS system. The Chicago Police Department publishes this web application [6] built on the Google Maps API to serve as a regular bulletin of crimes that have taken place within the city for the general public. Google Maps was embedded in the application to provide the base map.

The application gives a geospatial visualisation of criminal data extracted from the Chicago crime database. Locations of crime scenes are indicated by markers on the map. Also, markers are in different sorts and colours to reveal information other than locations. For example, a red coloured marker with a black circle tells that the crime made damaged a person's life. If the mouse is moved over a marker, an info window (an object of the Google Maps API "GIcon") will pop up with a more detailed description of the crime that happened on the location indicated by the marker. In addition, the user interface provides a set of menus on

which users can select the attributes of the crimes (e.g. crime type, time) to obtain a filtered display of crimes.

The snapshot (figure 2.2) shown below shows the homicides happened between Nov 2, 2005 and June 5, 2007 in city Chicago. The info window over location tells that there was a first degree murder.

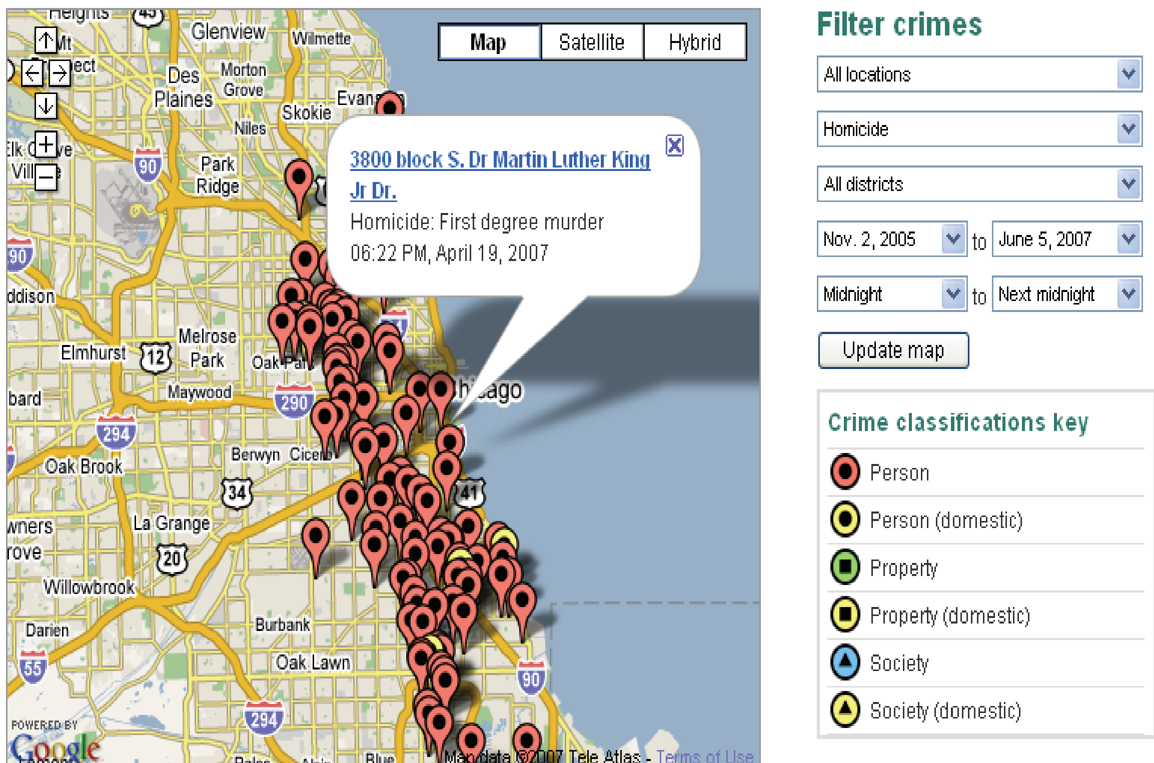

Figure 2.2 a map showing homicide crime scenes. Details of the chosen case are shown in the info window.

From the geospatial visualisation of this criminal data, people can identify dangerous areas. Also, criminal patterns may be revealed on the application map as well.

2.3.3 Weather Bonk showing real time weather information using Google Maps

The website [31] uses Google Maps to geospatially display real time weather information over areas.

The web site is equipped with a searching function, with which users can search for the weather information at a certain location. The real time weather data comes from the combination of various weather data sources, from personal weather stations run at people's houses, to some official weather information web services. This implementation demonstrates a new approach (called a mash-up) of building web based GIS systems, that is, to "mash" several different available web services together.

In this case, Google Maps were mashed in to build the base map of the application providing the web mapping service. The other weather web services were employed to provide the weather data which is integrated with the Google Maps API to make the geospatial display of the weather data.

The interesting part of the application is the way the weather data is presented on the map. Custom markers are used to present the weather data on the map.

Figure 2.3 using custom markers to display temperatures over places

From the snapshot (Figure 2.3), we can see that custom markers made in the form of numbers are placed on the map. In this way, the map gives a clear overview of the temperatures over different locations.

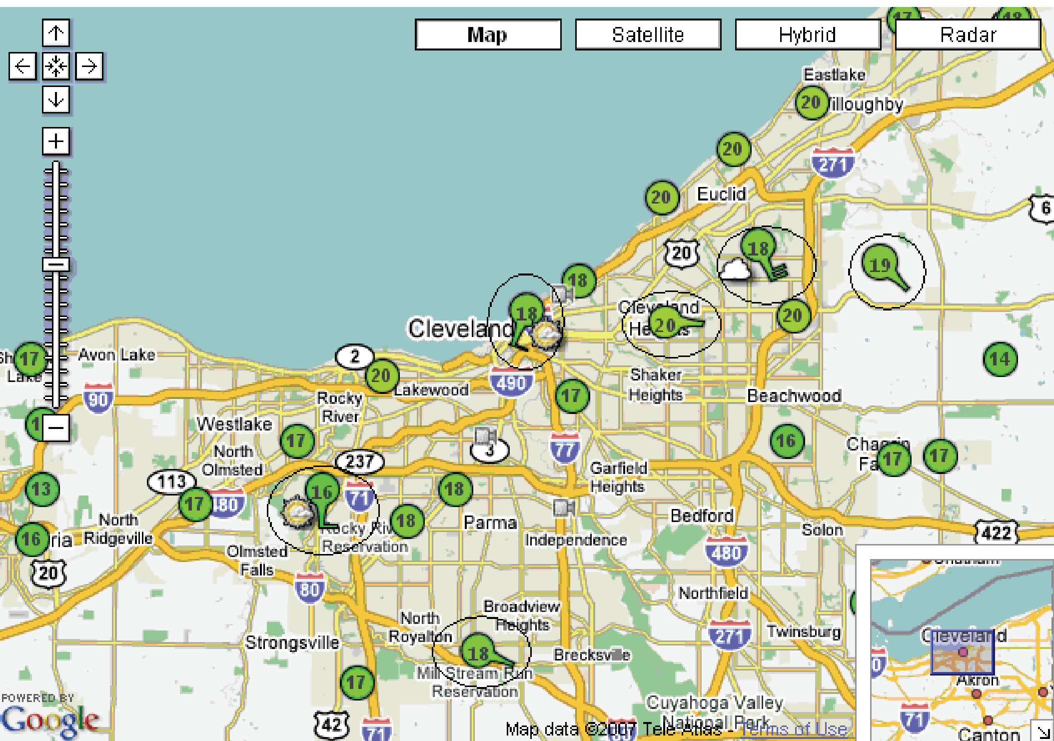

Figure 2.4 custom markers showing the wind directions and temperature over locations.

The snapshot above (figure 2.4) shows another type of custom marker used in the application. Filled circles with differently oriented tips are placed on the map to show the directions of the wind over different locations. If we look harder, we can see on the "tip" on some of the custom markers, there are bar(s) revealing the speed of the wind. Also, numbers are put in these markers showing the temperatures over the locations.

There are some other kinds of custom markers used in the application. E.g. from the figure above, markers in different images (a piece of cloud, sun with some cloud on) showing the kinds of weather at the indicated locations.

Besides, info windows pop up as a result of clicking on these markers giving more detailed weather information at the indicated locations.

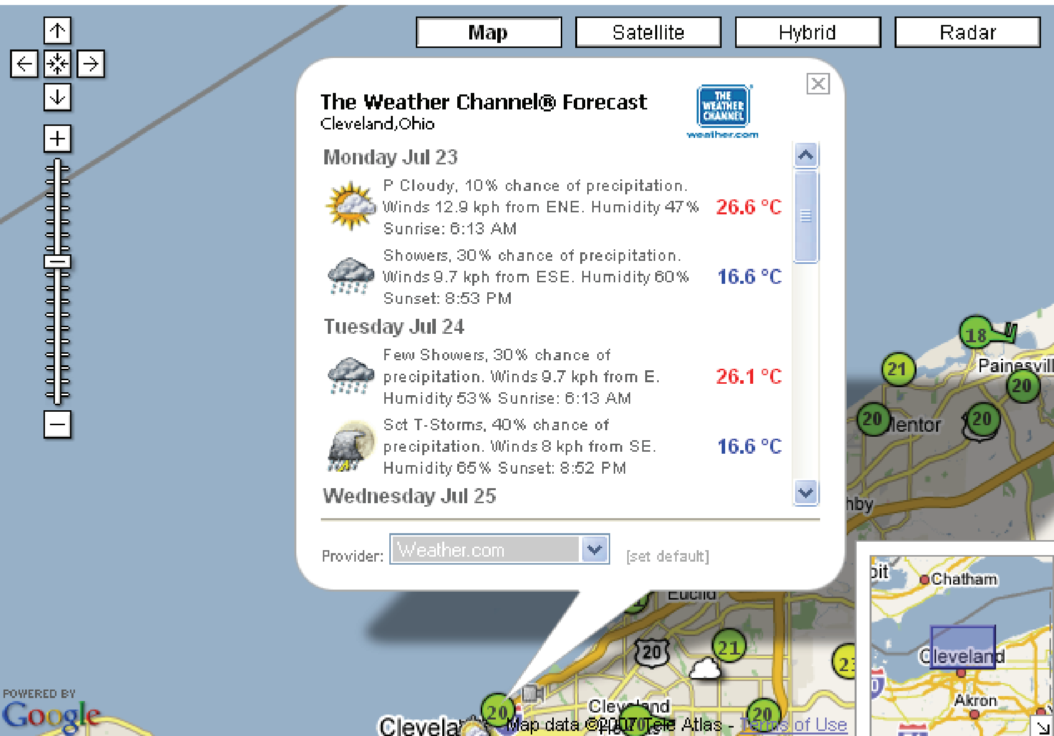

Figure 2.5 an info window showing detailed weather information in Cleveland/Ohio/USA.

2.4 Conclusion

Google Maps offers a convenient web mapping service because they have virtually unconditionally published its application programming interfaces. Web based GIS systems can be constructed combining geographically referenced data sources with the mapping service. From the example Google Maps implementations, we see that geographically referenced data can be in the form of a database or data serving web services. Besides serving map data, the Google Maps API also offers a set of graphic indicators (marker, info window) to support the graphic display of all sorts of geographically referenced data. Custom graphic indicators can be added to the map to optimise the graphic presentation of geographically referenced data.

As Google Maps is a web mapping service, high speed Internet connection is required for its use, because all the map data is served through the network. In the

context of this thesis project, the high speed Internet requirement was an issue because the application was being developed for people living in rural areas who usually do not have high speed Internet connection. Also, the imagery on Google Maps for the particular agricultural area this thesis project was focused on is not in a high enough resolution. The two problems were resolved by creating a custom map on Google Maps. Sufficiently high resolution image data for the focused area was used to construct the map data of the custom map. Moreover, the custom map was designed to store its map image data on each of the end users' local machine, thus the image serving for the custom map is no longer a network issue.

Chapter 3 WEB BASED APPLICATION

From a point of view of computer software engineering, computer applications can be divided into two major categories, desktop application software and web based application [40] software. Desktop applications are those computer applications that the users have to install properly on their personal computers before they can be up and running. Desktop applications constitute a large part of the whole application software world, and there have been all sorts of desktop application, such as the word processor, most of the multimedia software, computer games and so forth. However, along with the development of the high speed Internet and more and more advanced technologies being applied to the web browsers, the other kind, web based application software has gained more and more popularity.

In early forms of client-server applications, a piece of software had to be installed on the client computer to serve as the user interface. Any changes to data structure on the server side would require the updates on the client side software.

Nowadays, more and more advanced technologies and new standards have been applied to web browsers, like dynamic HTML, Javascript, Flash. Also, other application functionality like drawing graphs on the screen, accessing input devices, playing multi-media have become feasible in web browsers, which makes it possible for applications to use web browsers as the standard client interface.

Web based applications use the web browser as the client user interface, which enables those applications to be maintained and upgraded without installing software or disturbing the settings on the client computers. This is probably the biggest advantage that web application software has over the desktop kind. Nowadays many pieces of the computer software are implemented as web based applications; online trading services; webmail services; discussion message boards; weblogs and many other kinds of online services. One well known example is Amazon, the famous online book store. Book shoppers do not need to install any software on their personal computers to enjoy the convenience that the

application brings. All the functions required, like purchasing and browsing on the items can be done only through the web browser.

The following few sections discuss some technologies involved in web based applications.

3.1 Client Side Scripting.

Colourful and fancy web pages can be made just by using plain HTML, but it will just stay static. What if we require some interaction with the page, or alternatively, to make the page responsive to the user actions? The solution to the issue is scripting. By scripting, web pages can function dynamically and be very responsive.

There are two kinds of scripting, server side scripting [39] and client side scripting [38, 5]. Server side scripts run on the web servers, normally written in PHP, PERL. Server side scripts are executed by the web servers when there are requests for server side documents from the client side. The server generates the documents in the right format that is interpretable by the web browsers, and then returns them back to the client side to be presented on the browsers.

As for the context of this thesis project, we are to be more focused on client side scripting.

Client side scripting is the type of computer program that exist on the web but to be loaded and executed on web browsers on the client's computer. By performing client side scripting, web pages can be very changeable in response to user input or changes in environment conditions. Client side scripting enables displaying and hiding of elements on web page, or changing the location, size, font or colour of page elements in response to user action, which can make the pages very interactive and responsive.

The most commonly used client side scripting language is Javascript. The option to make programs written in Javasript executable has become part of the technical standard of modern web browsers. Javascript is an object oriented scripting language which runs inside a web browser. It manipulates the elements on a web page by interacting with the objects of the document, which is also referred as the Document Object Model (DOM).

Scripts/programs written in Javascript can be embedded in an HTML document. Alternatively they can be included in an external file, as long as they are referenced in the HTML document that uses it. The scripts file is sent to the client computer by the web server when a request is executed by the web browser. By applying client side scripting, we are capable of changing the presentation of a page which has already been loaded, we can add more text, images or other kinds of html elements to the page and change the properties of the current elements on the page.

3.2 Document Object Model (DOM)

"Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats."[41]

3.2.1 Background History

Javascript[9] was first introduced to the web browser in Netscape Navigator 2. It gave web developers a method to access elements on a web page – forms, links, images and other features through scripting in Javascript. Microsoft implemented its version of Javascript in Internet Explorer 3 (called JScript) to keep up with its competition. Each of the browser manufacturers developed their own standard for the scripting interface, which caused a cross browser issue – web developers had to write different code for each method in their scripts in order for the scripts to be able to work on both browsers. Then the W3C stepped in and released the first specification for the scripting interface – the Document Object Model 1, which specified an outline of standard methods of accessing the various parts of an XML

document via Javascripting, which also applied to a valid HTML document. Both web browser manufacturers then implemented DOM level 1 specification [44] in their new products – Internet Explorer 5 and Netscape 6. Since then, the W3C has also released DOM level 2 [45] and DOM level 3 [46].

From the brief history of the DOM, we see that the DOM is a technology standard and also a development interface that is supported by modern web browsers, through which developers can access and manipulate the elements on web pages via scripting in Javascript [18].

When a web browser loads a web page, all the HTML elements are represented in a hierarchical tree–like structure (also referred as the HTML tree). Elements, attributes of the elements and some other document objects are represented as nodes in the document tree structure. In the DOM representation of a document, each element is treated as an object which has its own property and methods. The document object model provides the web developers with a programming interface to access and manipulate those document objects by giving methods for the developers to retrieve and set the properties of the objects. With the methods defined in the document object model, developers can also add or remove the document objects, by which dynamic changes to web page content can be achieved. Moreover, the document object model provides an interface for web developers to deal with the window events. It makes it feasible for the developers to capture and respond to user actions on the browsers.

3.2.2 DOM examples

Next, let's go through a set of examples to get a better idea about how an HTML document is represented in the DOM model, and how the elements on a page can be manipulated through the interaction with the DOM model via scripting.

Here is one simple HTML document:

```
<html>
<head>
	<title>DOM example</title>
</head>
```

```
<body id="body">
      <p id="p1">paragraph 1</p>
      <p id="p2"></p>
      <p id="p3">
      <a href=http://www.google.com/  id="link">google</a>
      </p>
</body>
</html>
```

The <body> tag is the root of the HTML tree, it has four child nodes: three <p> tags which divide the document into 3 paragraphs. In an HTML tree, each node is either an HTML tag or a text entry. This Html document will be used through the next few examples showing some DOM methods.

The DOM tree for the HTML document looks like this:

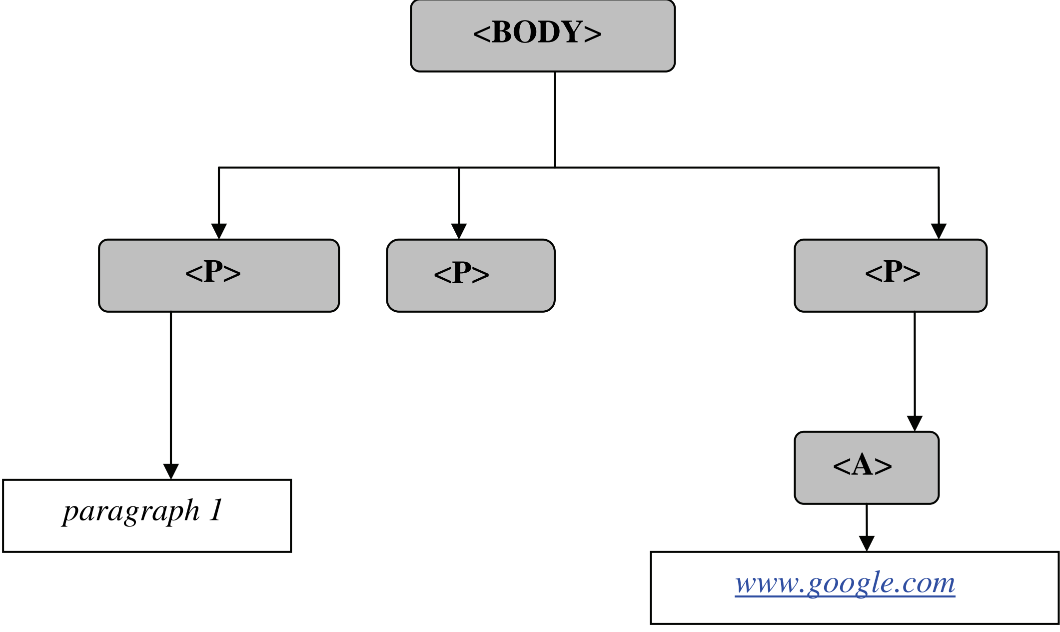

Figure 3.1 The DOM tree of the example HTML document

The root of the tree is the node <BODY>, which has three child nodes: Three <p> nodes and an <A> node which contains a text string. As we can see in the tree structure presentation of the HTML document, every pair of tags is represented by a node in the tree, and there are other nodes for the attributes of the nodes or the character data.

Accessing the Elements

In the DOM model, all elements are treated as objects organised in a tree structure. The object *document* stands for the root element of the HTML document. There is only one *document* object for an HTML document. The object *document* also implements the DOCUMENT interface, which provides methods for accessing and creating other nodes in the document.

There are two main DOM techniques that can be used to get a reference to a particular element or a set of elements of the same kind in a document.

The first technique, *document.getElementById*, is very popularly used in Dynamic HTML code. In an HTML document, any tag can be assigned a unique id. For example, in the example document, the first <P>tag is assigned an id "p1":

```
<p id="p1">paragraph 1</p>
```

Elements having unique ids, can be identified and then accessed by using method *document.getElementById*. For example, we can get a reference to the first <P> element using the following Javascript statement:

```
var p_1 = document.getElementById('p1');
```

The second technique, *document.getElementByTagName*, is used to retrieve all the elements of the same type (same tag name). The method returns an array with all the elements with the same tag name. For example, we can get the reference to an array of all the <P> elements in the document by the following Javscript statement:

```
var p_array = document.getElementByTagName('p');
```

For our example, it will return an array containing three entries as there are three <P> tags in the document.

Changing Element Attributes

Now we have found a way to get references to the element(s) in the document by using the methods *document.getElement* and *document.getElementByTagName*. Once we have obtained references to the document elements, we can access and make changes to their attributes.

For example, we can first get a reference to the <A> element by using the Javascript statement:

```
var link = document.getElementById('link');
```

Then we can change the HREF attribute of this element by the statement:

```
link.href = "http://www.waikato.ac.nz";
```

After the script has run, clicking on the link will take it to the website of the university of Waikato (New Zealand) rather that the Google site. The HTML document has been effectively updated as:

```
<html>
<head>
      <title>DOM example</title>
</head>

<body id="body">

      <p id="p1">paragraph 1</p>

      <p id="p2"></p>

      <p id="p3">

      <a href=http://www.waikato.com/  id="link">google</a>

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

Changing Text Node

One thing to be noticed from the DOM tree in the example is that the textual part inside an element is not part of the element or an attribute of the element. Instead it is a child of the element node.

For example, to change the text inside the first <P> element, this is currently set as “paragraph 1”:

First we get a reference to the <P> element:

```
var p1 = document.getElementById('p1');
```

Then we get a reference to the child of the node <p1> by visiting its *childNodes* property (an element’s *chileNodes* gives an array of the element node’s child nodes, and in this example the <p> node only has one child - its first child):

```
var txt = p1.childNodes[0];
```

Now we can change the value of the text node by modifying its *nodeValue* property:

```
txt.nodeValue = 'new paragraph 1';
```

After this, the example HTML content will be effectively updated to:

```
<html>
<head>
	<title>DOM example</title>
</head>
<body id="body">
	<p id="p1">new paragraph 1</p>
	<p id="p2"></p>
	<p id="p3">
	<a href=http://www.waikato.com/  id="link">google</a>
```

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

Moving Elements

Method *appendChild* can be used to add an element as a child node to an existing node in the document. In this example, we will move the <a> element inside the third <p> tags to be inside the second pair of <p> tags:

First, we get the reference to the link element by the using the Javascript statement:

```
var link = document.getElementById('link');
```

Then we get the reference to the second <P> element with the Javascript statement:

```
 var p2 = document.getElementbyId('p2');
```

Next, we use the DOM method *appendChild* to attach the <a> element to the second <P> element by the Javascript statement:

```
p2.appendChild(link);
```

Note that, we don't have to detach the <a> element from the third <p> element and throw it away to complete the moving action of the element. Once the <a> element has been attached to its new parent node, it will not exist at its old location any more. Now the HTML content is effectively updated like below:

```
<html>
<head>
	<title>DOM example</title>
</head>
<body id="body">
	<p id="p1">new paragraph 1</p>
```

```
        <p id="p2">
              <a href=http://www.waikato.com/  id="link">google</a>
        </p>
        <p id="p3">
         </p>
</body>
</html>
```

Removing Element from the document

The DOM method *removeChild* is used to remove an element from an HTML document. In other words, detach the element node from its parent node. Therefore, before we can possibly remove an element we need to check if the element has a parent node. We can access a certain element's parent node through the property *parentNode*. Here we try to remove the <a> element under the second <p> in the HTML document:

First, we get the reference to the <a> element, again, by using method *getElementById*:

```
var link = document.getElementById('link');
```

Then we check if the element has a parent node, and remove it from the document if that is the case:

```
if(link.parentNode){
      link.parentNode.removeChild('link');
}
```

As the <a> does have a parent node, it will be removed from the document, which will change the HTML content to that shown below:

```
<html>
<head>
      <title>DOM example</title>
```

```
</head>
<body id="body">
      <p id="p1">new paragraph 1</p>
      <p id="p2">
      </p>
      <p id="p3">
       </p>
</body>
</html>
```

Creating an element and inserting it to a document

A brand new element can be created and inserted into a document by using DOM methods. Note that the textual part on a page resides in a text node. Thus if we need to create an element that contains text we have to create both the new element node and a new text node to contain the text, which involves the use of two DOM methods: *document.createElement* and *document.createTextNode*.

In this example, we are going to create a new <a> element which has a text string inside the element and then attach it to the third <P> element in the document.

First, we create the new <a> element by using the method *document.createElement*:

```
var newLink=document.createElement('a');
```

Then we set the point this element links to by setting its property HREF:

```
newLink.href='http://www.yahoo.com';
```

Then we create the text node to contain the text which will appear inside the link:

```
var linkTxt=document.createTextNode('The Yahoo Site');
```

At this stage, the text node is still separate from the <a> we just created, so we add the text node as a child of the <a> element by using the method *appendChild*:

```
newLink.appendChild(linkTxt);
```

Now we need to add the new <a> element to the third <P> element in the document, so first we get a reference to the <P> element by Javascript statement:

```
var p3 = document.getElementById('p3');
```

Then, add the new <a> element to it:

```
P3.appendChild(newLink);
```

Now the third <p> element has a child node which is a link containing the text "The Yahoo Site". The HTML content is now effectively like:

```
<html>
<head><title>DOM example</title></head>
<body id="body">
      <p id="p1">new paragraph 1</p>
      <p id="p2"></p>
      <p id="p3">
            <a href="http://www.yahoo.com/">The Yahoo Site</a>
       </p>
</body>
</html
```

3.3 Dynamic Hypertext Markup Language (DHtml)

Dynamic HTML is not the name of a certain language, nor is it the name of a kind of technology. It is a collection of several web technologies that are used together in order to make web pages interactive and animated, in brief, to make dynamic web pages. It is the combination of the static markup language (HTML), presentation style definition language (Cascading Style Sheet), client side

scripting (client side Javascripting) and the Document Object Model [18]. HTML is used to create a page. A CSS sheet defines the display style of the page; Javascripting is applied on the client side with the DOM model to manipulate the elements on the page to equip the page with dynamic features.

As a concept, DHtml has actually been around for quite a period of time, however early web browsers were not able to accomplish the idea. Since the newer technologies have been introduced to and built into the modern web browsers (MicroSoft Internet Explorer 5, Mozila Firefox and other 4th generation web browsers), the idea of dynamic HTML has become all feasible and it now probably is the most common used tool in creating all sorts of web application.

Next we are to look at some DHtml examples.

Resizing an image on the page

This example shows how to resize an image on a web page.

The Html code:

```
1 <html>

2     <head>
3        <title>scripting example</title>
4        <meta http-equiv = "Content-Type" Content = "text/html;
         charset = utf-8">
5     </head>
6     <script src="client-side-script.js"
      type="text/javascript"></script>
7     <body>
8       <img id = "s-pic" src = "s.jpg" alt = "spiderman"
          width=200  height=200 align = "center">

9       <input type="button" style="width: 100px; height: 50px"
        value="Hi" onClick="resize()">
```

```
10     </body>
11  </html>
```

In the example, the client side scripts – Javascript functions - are contained in an external file “client-side-script.js”, which is referenced in the *<sciprt>* section at line 6 in the HTML code.

Here is the Javascript code contained in the file “client-side-script.js”:

```
function resize(){
            var it = document.getElementById("s-pic");
            if(it.width == 200){
                  it.width = 500;
                  it.height = 500
            }else{
                  it.width=200;
                  it.height=200;
            }
}
```

DOM method *document.getElementById* is used to get the handle of the image object so that we are able to change the attributes-width and height in this example.

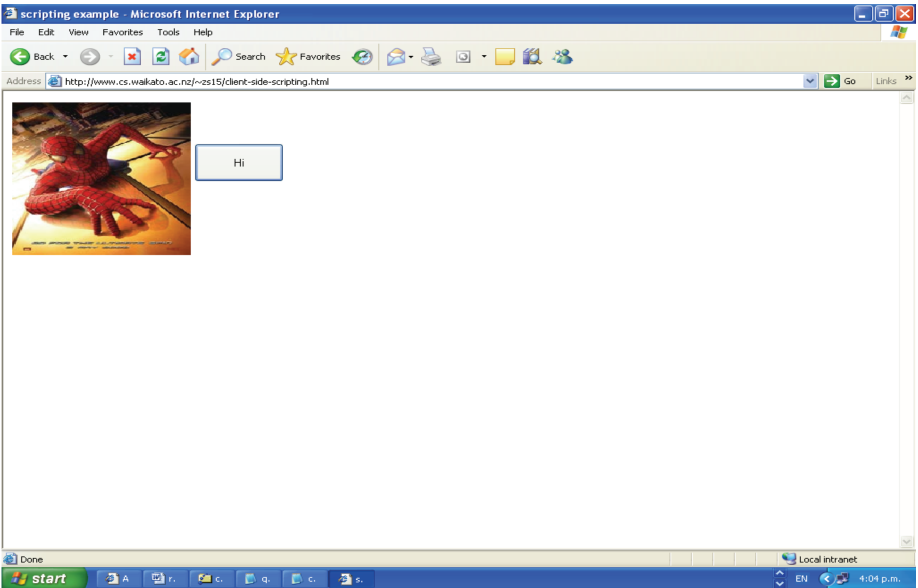

Figure 3.2 the look of the page before the click on the button, the size of the image is 200X200.

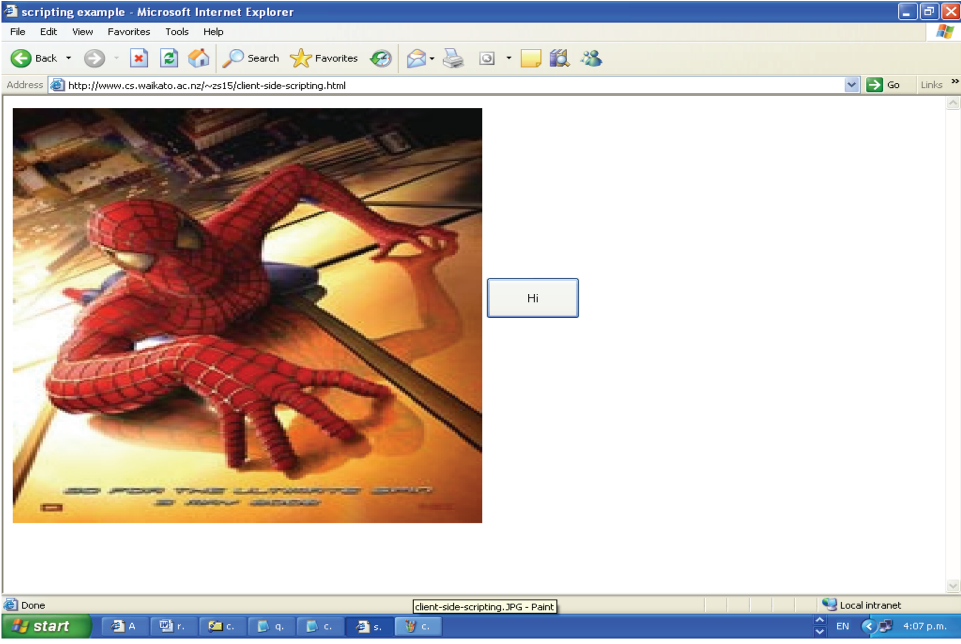

Figure 3.3 the look of the page after the button click. The size of the image is now 500X500 pixels.

Slider bar control

Surprisingly, an appropriate slider bar control is not available in the HTML system. The slider bar control used in this project was built from primitive HTML components.

This example explains how to create a slider bar and use it to control the size of an image on the page. The Javascript source for this example is adopted from the WebFX DHtml Demos [4].

This is the HTML document:

```
1 <html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-
        microsoft-com:vml">
2       <head> <meta http-equiv="content-type" content="text/html;
        charset=UTF-8"/>
3       <title>Test_4</title>
4       <script src="example2.js" type="text/javascript"></script>
5       </head>
6       <body>
```

```
7      <img id = "s-pic" src = "s.jpg" alt = "spiderman"
       width=200   height=200 align = "center">
8      <div style="width: 160; height: 20; background:
                                           buttonface;">
9            <div id ="bar" class="sliderHandle" type="x"
                  onChange="update(this)"
                  style="position: relative;
                  left: 55; width: 63; height: 100%;
                  background: red;">
10           </div>
11     </div>
12     </body>
13 </html>
```

Again, all the scripting functions are contained in an external file “example2.js”, which is referenced in the <script> section at line 4.

At line 7, an image is placed on the page. The original size of the image is set as 200X200.

A slider bar widget is actually a draggable <div> element that is contained inside another <div> element. In the HTML source code, two <div> elements are placed one inside anther to build the sliderbar (from line 8 to 11).

The slider bar control implementation consists of several event handlers that determine the slider value depending on the event arguments.

Catching the *OnMouseDown* Event

The event handler for event onMouseDown is defined in function “slidebar_onMouseDown()”. When the mouse is pressed down on the inner <div> element – the thumb of the slider, it sets the “drag” connection between the mouse and the inner <div> element and calculates the distance from the left edge of the container (the outer <div> element) to the left edge of the slider thumb (the inner <div> element).

```
function slidebar_onMouseDown() {

            var temp = getReal(window.event.srcElement,
"className",            "sliderHandle");

            if (temp.className == "sliderHandle") {
                     slidebarDragObj = temp;

                  onChange =
                  slidebarDragObj.getAttribute("onChange");

                  if (onChange == null) {
                        onChange = "";
                  }

                  SlidebarType =
                  slidebarDragObj.getAttribute("type");

                  slidebarPosX = (
                  window.event.clientX -
                  slidebarDragObj.style.pixelLeft
                  );

                  window.event.cancelBubble = true;
                  window.event.returnValue = false;
                  }
                  else {
                        slidebarDragObj = null;
                  }
            }// end of function slidebar_onMouseDown
```

Catching the *onMouseUp* event

Function “slidebar_onMouseUp” catches the *onMouseUp* event. It simply releases the connection between the mouse and the dragged object (the inner <div> element) when the mouse is released.

```
function slidebar_onMouseUp() {

                  if (slidebarDragObj) {
                        slidebarDragObj = null;
                  }
            }//end of function slidebar_onMouseUp
```

Catching the *onMouseMove* event

The event *onMouseMove* is handled by function “slidebar_onMouseMove()”. When the inner <div> element is connected to the mouse, it checks if the mouse pointer is outside of the draggable object – the inner <div>. If not, then it updates the position of the inner <div> element in its container – the outer <div> element. The value of the slider is also calculated and an *onChange* event is fired.

```
function slidebar_onMouseMove() {
	if (slidebarDragObj) {

		if (event.clientX >= 0) {

			if ((event.clientX  - slidebarPosX >= 0) &&
			(event.clientX - slidebarPosX <=
			slidebarDragObj.parentElement.offsetWidth -
			slidebarDragObj.offsetWidth))
			{

				slidebarDragObj.style.left =
				event.clientX - slidebarPosX;
			}

			if (event.clientX - slidebarPosX < 0) {
					slidebarDragObj.style.left = "0";
			}

			if (event.clientX - slidebarPosX >
			slidebarDragObj.parentElement.clientWidth -
			slidebarDragObj.offsetWidth - 0)
			{

				slidebarDragObj.style.left =
				slidebarDragObj.parentElement.clientWidth
-				slidebarDragObj.offsetWidth;
			}

			slidebarDragObj.value =
			slidebarDragObj.style.pixelLeft /
			(slidebarDragObj.parentElement.clientWidth -
				slidebarDragObj.offsetWidth);

			eval(
				onChange.replace(
					/this/g, "slidebarDragObj"
				)
			);

		}
			window.event.cancelBubble = true;
			window.event.returnValue = false;
	}
}//end of function slidebar_onMouseMove
```

Dynamically changing the image size with the value of the slidebar control

Function "update()" catches the *onChange* event on the slide bar control. It dynamically changes the size of the image on the page.

```
function update(el){
            var num = Math.round(10*bar.value);
            var img = document.getElementById("s-pic");
            img.width = num;
            img.height = num;
}
```

The two snapshots (figure 3.4, figure 3.5) below show the change of the image size by adjust the slider bar control:

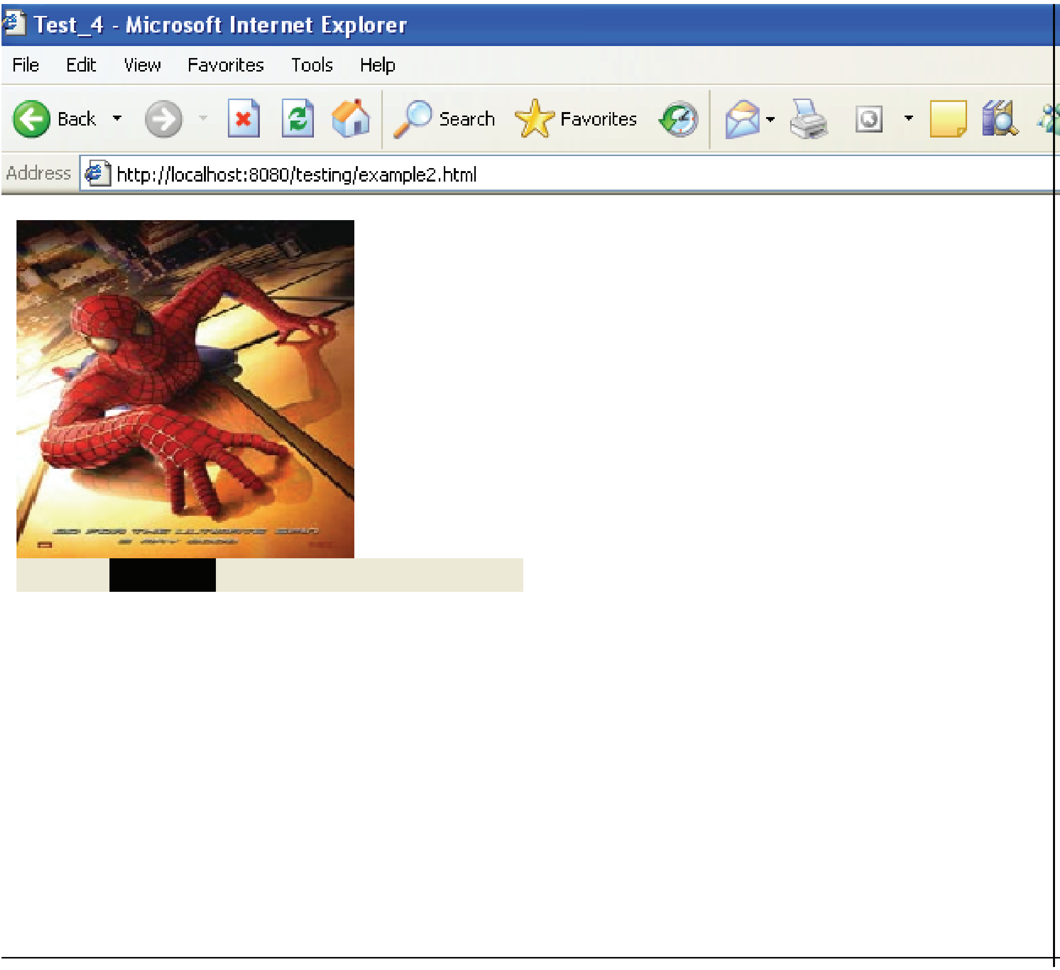

Figure 3.4 web page showing the image in its original size (200 X 200 pixels)

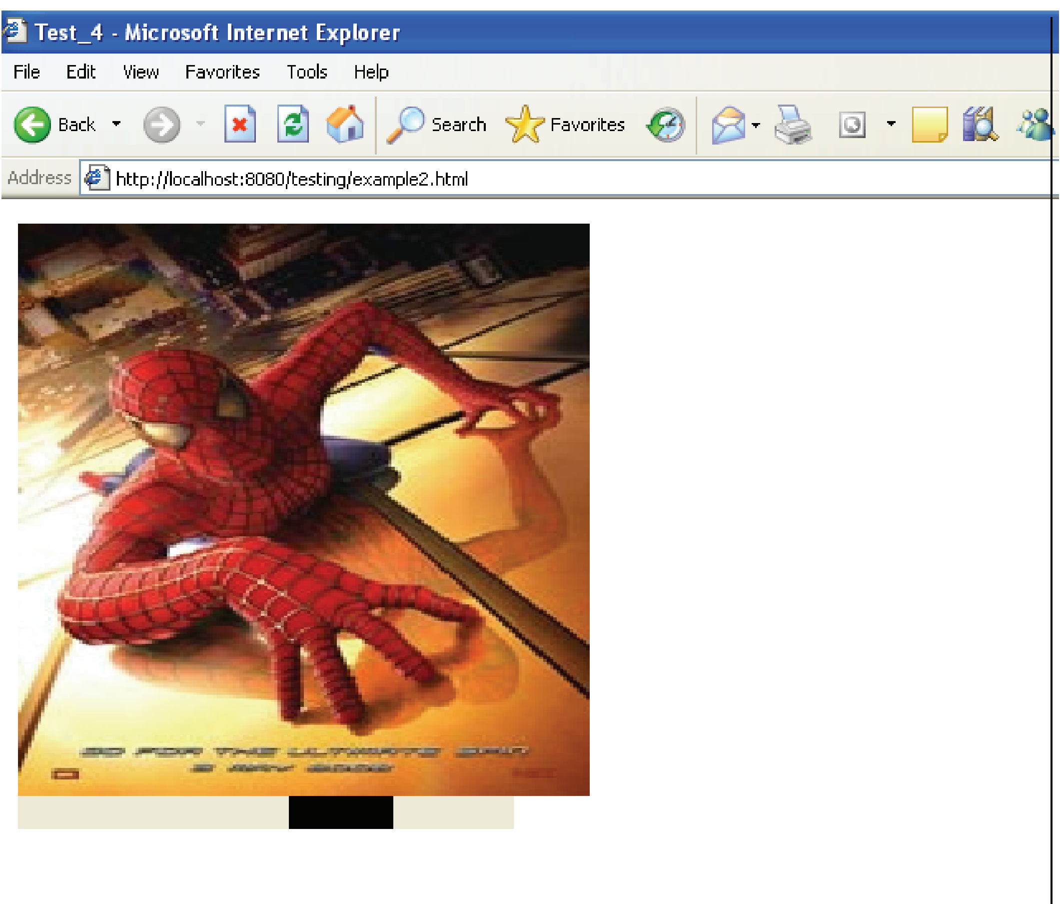

Figure 3.5 image resized by adjusting the slider bar control

This completes the description of the aspect of DHtml that were used in the project. The next chapter discusses communication between client and server.

Chapter 4 WEB APPLICATION MODELS, AJAX MODEL

Let us have a quick review of the two major kinds of computer software application: desktop applications and web based applications. The two kinds each have their own advantages and disadvantages.

Desktop applications usually run quickly. Users do not have to wait on their Internet connection (although in some cases, applications download things from websites). Web applications run on web servers and users access them through web browsers. Time delay resulting from the Internet connection can become a major concern (time for the servers to respond, time for the screen to become refreshed). In terms of the user interface, desktop applications give a better performance than the web applications. User interfaces of desktop applications are usually very dynamic, they provide users with a great user experience, giving all sorts of functionality-users can type, click, pull up all sorts of menus, cruise around and most importantly, users always get their responses quickly.

On the other hand, web applications are always up to date, and with the amount and variety of the data/information on the web, they provide the users with great services that desktop applications are not capable of. This advantage is of overwhelming significance, as we are in an age of information.

Therefore, from a point of view of computer software engineering, the ideal solution probably is to equip a web application with a highly interactive and responsive user interface so that the users can have all the interactive and fancy usability that they can find in a desktop application while still enjoying the pleasure from the powerful services that the web application offers. This is exactly where and why the idea of AJAX came into being and found its position.

4.1 Classic Web Application Model (synchronous)

In the traditional and classic web application model, a client computer makes a request to the server in the form of a submission of the whole page. A user action on the user interface causes a submission of the web page in the form of an HTTP request to the server; the server receives the request and does the necessary processing (e.g. fetch certain data) and then returns the data in a form of an html page back to the client browser as a response. For the whole time between the user's action that triggers the HTTP request to the server till the moment the html page generated by the server arrives back at the client computer, the user has to wait. And due to other factors, like the number of the requests that the server has to cope with at the time, the amount of data that has to be sent to the client side on the request, or the traffic situation on the Internet, the time for the user to wait can be rather long, and even worse, of a quite unpredictable length.

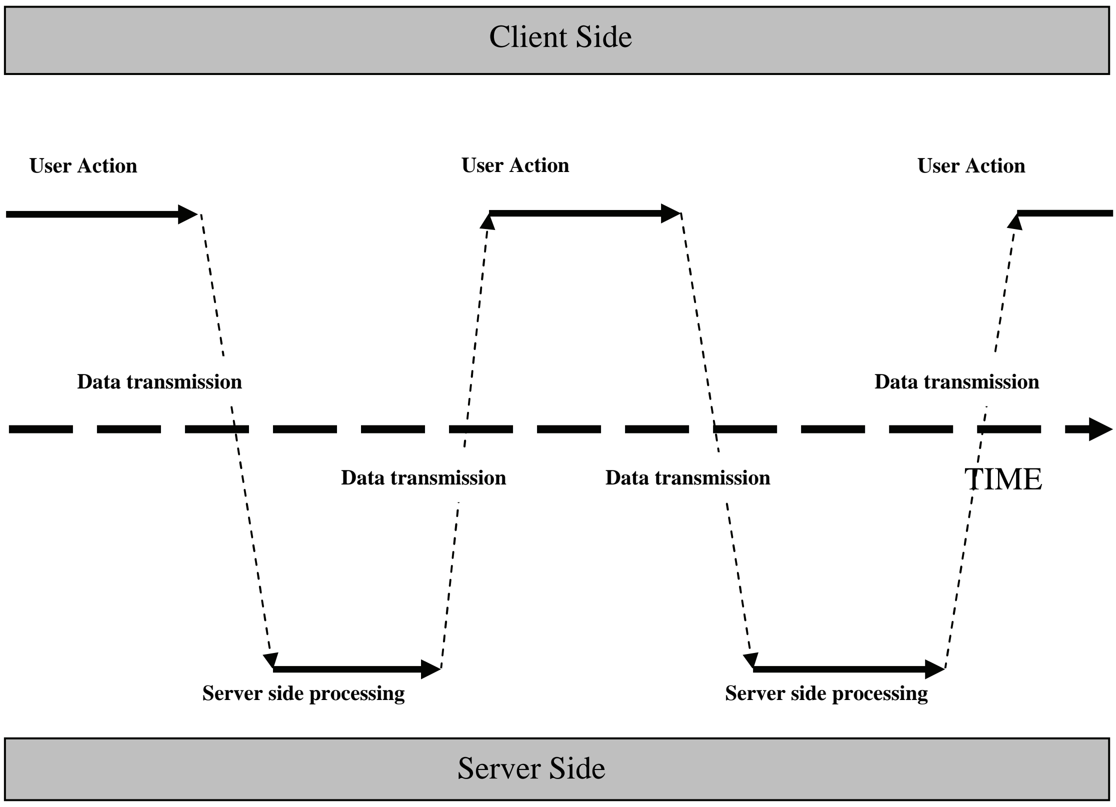

Figure 4.1 classic web application model

Some of the online map applications that work in this classic model can be used as examples here for a vivid description of the case. On the user interface of an online map application, there is, of course, a map view of certain area, and then direction controls for the users to move the view, and probably a scale that user can use to adjust the resolution level of the map. Every time the user drags the map or adjusts the resolution, they will have to wait for a time with a blank browser window, before the expected view comes back on the screen. As the size of the image data requested on user actions is relatively large, the data delivering can take a relatively long time. Even worse, when there is a great deal of traffic on the Internet, the time for the user to wait will be variable, which makes the map application really a patience test.

As we have discussed in the two paragraphs above, a traditional web application works in a start-stop (wait)-start-stop (wait) mode. Because the whole web page has to be submitted to the web server whenever there is a user request (in another sense, a request is only triggered on the form submission). The screen will usually go blank for a period of time during which user interaction with the application is interrupted. The application is not responsive at all and the user is able to do nothing but stare at a blank screen while waiting for the feedback from the server side to arrive. This synchronous data communication model surely makes a clear technical sense, but can bring considerable annoyance to the user.

4.2 AJAX web application model (asynchronous)

An AJAX [1, 12, 13, 20, 29, 33] web based application works around the problem by transferring data between the client side and the server side in an asynchronous fashion. It exchanges data with the server behind the screen-without the user's knowledge.

In an AJAX application, extra software is added on the client side, which is practically a piece of script written in Javascript and is referred to as the AJAX engine.

In the AJAX application model, user interaction does not require the submission of the entire web page to interact with the web server, only relevant data on the page gets updated when necessary. When there is a user action which would normally trigger an HTTP request to the server in a classic web application model, an AJAX application makes a Javascript call to the AJAX engine. The AJAX engine takes the data from form fields on the user interface by the Javascript call and sends a request to the server, all of this is happening without the user's awareness. The user interface does not flash or even disappear. Even better, the AJAX engine sends the requests in an asynchronous mode, which means the AJAX engine does not have to wait for a response from the server before processing further user interaction events.

The server sends back the data to the AJAX engine instead of directly to the client interface. The AJAX engine takes the data and updates the page without reloading it. As we can see from the whole process, with the asynchronous way the AJAX engine sends requests to the server, the interaction between the user and the application is independent of data communication with the server, which means the user interaction with the application doesn't get interrupted by the page reloading or the time delay results from the data transmission. Of course, if the user's input requires data from the server, they must wait before it becomes visible. The important part is that this delay does not block all interaction with the user interface. Caching programming of the application is necessary to give a smooth user experience. For example, when scaling, Google Maps immediately shows an expanded low resolution view of an area, then update it with a better map as data arrives. This means that the user gets immediate feedback on their scaling operation, even though there is a delay in final completion.

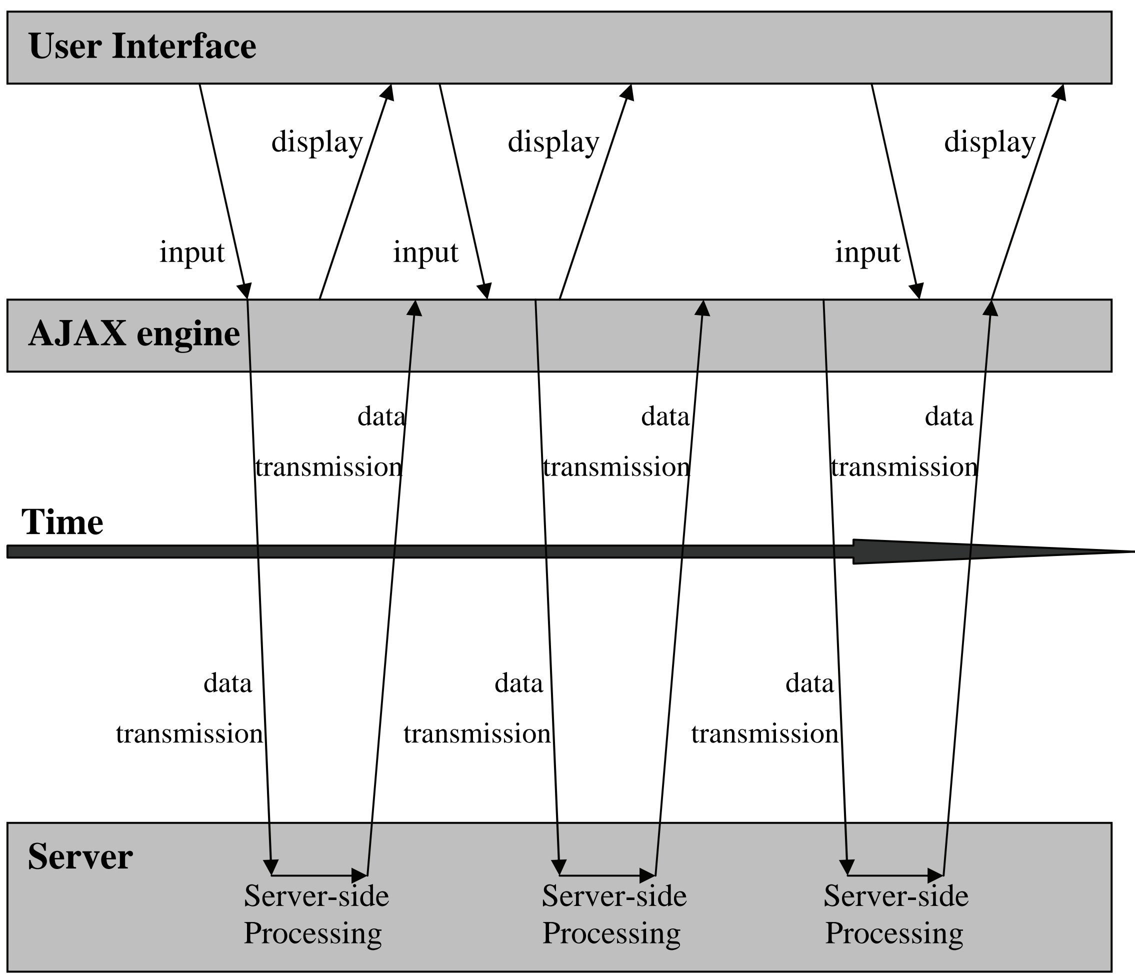

Figure 4.2 AJAX web application model

4.3 History of the AJAX

AJAX is shorthand for Asynchronous Javascript and XML, a development technique for creating highly interactive and responsive web based applications. The term does not stand for one technology but a combination of several web technologies that working collectively to allow a web application to function in a much more efficient fashion.

Although the term was coined by Jesse James Garrett in 2005, the technologies that make AJAX possible have been in existence for years. The idea of asynchronously loading of content on an existing web page without a full page reload can be traced back to the IFRAME element type which was introduced into Microsoft Internet Explorer 3 in 1996 and the LAYER element type introduced into Netscape 4 in 1997. Both element types could achieve AJAX-like effect by loading an external web page containing Javascript code that could manipulate the parent web page. The client side technologies involved were grouped under the term DHTML. In 1998, Microsoft introduced its Remote Scripting (MSRS); data could be obtained by a Java applet in which the communication between the client side and the server side was made using Javascript. In Microsoft Internet Explorer 5, Microsoft first created the *XMLHttpRequest* object which has become the key part of the AJAX pattern, and applied the technology making use of the object in Outlook Web Access.

4.4 AJAX components

Like DHTML, the AJAX is not a technology in itself, but a term describing the use of a combination of a set of web technologies:

- HTML/XHTML and the cascading style sheet are used for the marking up and specifying the style of the page content.

- The Document Object Model provides a platform on which a client side scripting language – Javascript can be used to dynamically display and interact with the data presented on the web page.

- The *XMLHttpRequest* object is used for asynchronous data communication with the web server.

- XML works as the format for transferring data between the client side and the server side.

The core of AJAX is the *XMLHttpRequest* object [2, 8, 42, 47]. It enables web browsers to make requests of web servers asynchronously without having to unload and reload a web page.

The very original concept of *XMLHttpRequest* was developed by Microsoft as part of OUTLOOK WEBACCESS 2000. It was a server side API call at the time, not yet client side feature. In Internet Explorer 5.0, Microsoft implemented the *XMLHttpRequest* as an *ActiveX* object, called XMLHTTP, which could be accessed via Javascipt. Then in 2002, the Mozilla project group developed its compatible native implementation of *XMLHttpRequest* in Mozilla 1.0. Also, this implementation was later followed by Apple in Safari 1.2.

Since there are different versions of implementations of *XMLHttpRequest* object across different browsers, creating an *XMLHttpRequest* object requires branching syntax:

For Mozilla and Safari, the object can be created by simply calling the construction method:

```
var xmlHttp = new XMLHttpRequest();
```

For Internet Explorer 5+:

```
var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
```

The *XMLHttpRequest* object has a list of concise but powerful properties and methods [8, 47], the tables below shows the object methods and properties that are supported by Mozilla, Safari and IE 5+:

Methods

Method	**Description**
abort()	Cancel the current request.
getAllResponseHeaders()	Returns the complete set of the response headers as a string.
getResponseHeader(headername)	Returns the value of the specified http header.
open(method, url) open(method, url, asyn) open(method, url, asyn, username) open(method,url,asyn,username,password)	Specifies the request method and the destination the request is sent to, also some other optional attributes. The first parameter *method* can be set to a value of "GET", "POST", "HEAD", "PUT", "DELETE" or any other HTTP methods. The second parameter *url* can either be an absolute or relative URL. The third parameter *asyn* can be set to either TRUE or FALSE. When it is set to TRUE, the request is handled asynchronously, which means the scripts processing can carry on after the request has been sent without waiting for the response. When it is set to FALSE, the script processing will not carry on till the response for the request arrives.
send(content)	Sends the request.
setRequestHeader(label, value)	Adds a label to the http header to be sent.

Table 4.1 methods of the *XMLHttpRequest* object

Properties

Property	**Description**
onreadystatechange	Event handler for an event that fires at every state change of the request.
readyState	Indicates the state of the request object. 0 = object uninitialized. 1 = open. 2 = request sent. 3 = receiving response. 4 = completed.
responseText	The plain text version of the data returned from the server side.
responseXML	The XML version of the data returned from the server side.
status	Returns the HTTP status code as a number. Such as, 404 for 'NOT FOUND', 200 for 'OK'.
statusText	Returns the text version of the status. (e.g. 'NOT FOUND' or 'OK')

Table 4.2 properties of the *XMLHttpRequest* object

4.5 A simple Ajax example

Next, let us walk through a simple example to see how to make use of the *XMLHttpRequest* object to create an AJAX application. The example demonstrates how a web page can fetch information from a mySQL database using AJAX technology. The example application involves four parts:

- a mySQL database
- a web page
- a Javascript file
- a PHP file.

The database, maintained on the server side, contains some personal information of few individuals. All the information is maintained in one table called *users* which has 5 fields, the person's user id, name (first name and last name), age, hometown and occupation.

There are currently five records stored in the database:

user_id	FirstName	LastName	Age	HomeTown	Job
1	Joe	Harris	33	L.A	actor
2	Michael	Robinson	53	Auckland	school teacher
3	Paul	Lee	28	N.Y	bus driver
4	Jeffery	Morris	35	Wellington	police officer
5	Griffin	King	34	N.J	pilot

Table 4.3 personal data used in the example

The HTML page:

```
1. <html>
2.    <head>
3.       <script src="ajax_ex.js"></script>
4.    </head>
5.    <body>
6.       <form>
7.       Select a User:
8.          <select name="users" onchange="showTable(this.value)">
9.            <option value="1">Joe Harris</option>
10.           <option value="2">Michael Robinson</option>
11.           <option value="3">Paul Lee</option>
12.           <option value="4">Jeffery Morris</option>
13.           <option value="5">Griffin King</option>
14.         </select>
15.      </form>
16.      <p>
17.            <div  id="table"><b>User  info  will  be  listed
here.</b></div>
18.      </p>
19.    </body>
20. </html>
```

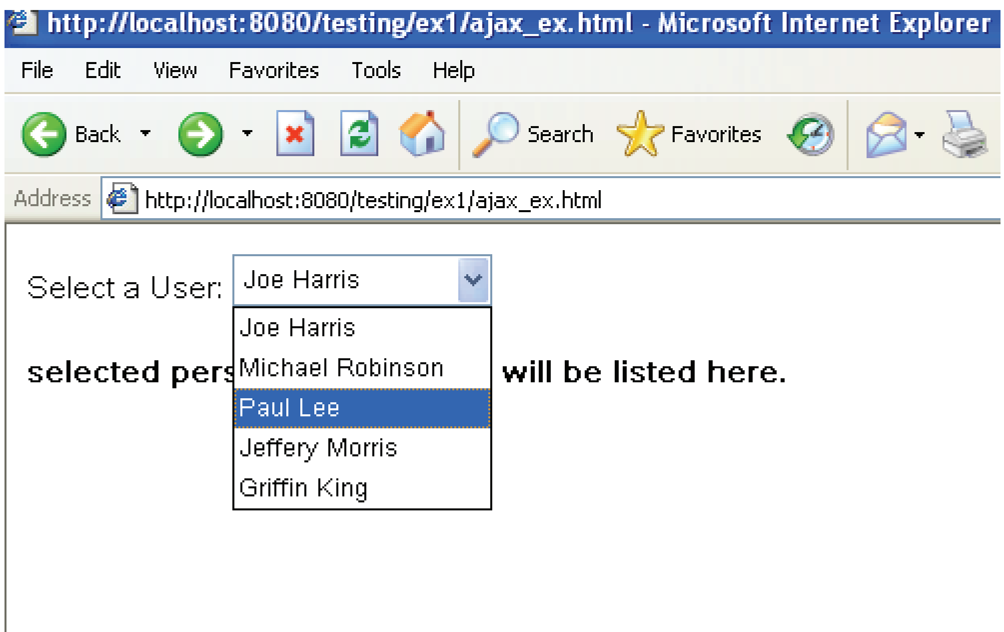

Figure 4.3 the example application user interface

The web page serves as the user interface. It is just a simple HTML form containing a dropdown menu called "users" with the names of the persons and values set to the user_id from the database (as we can see in the snapshot, the option "Paul Lee" is to be selected, and the value of the option is the user_id "3").

At the bottom of the page, there is a <div> element called "table" (line 17 in the source code listed above), which is used as a placeholder to display the data retrieved from the database on the server side.

An "onChange" event is caused when a user selects from the dropdown menu, which triggers the execution of the Javascript function "showTable(str)" (the function takes the selected value of the dropdown menu as a parameter). All the relevant Javascript functions are stored in an external Javascript file called "ajax_ex.js", which is linked to in the <*script*> section (line 3 in the HTML source code listed above).

The Javascript file "ajax_ex.js":

```
var xmlHttp;

function showTable(str) {
   xmlHttp = GetXmlHttpObject()
   if (xmlHttp == null) {
            alert ("error in creating HTTP Request")
            return
   }
   var url = "ajax_ex.php";
   url = url+"?q="+str;
   url = url+"&sid="+Math.random();
   xmlHttp.onreadystatechange = stateChanged;
   xmlHttp.open("GET",url,true);
   xmlHttp.send(null);
}


function stateChanged() {
   if (xmlHttp.readyState == 4 ){


document.getElementById("table").innerHTML=xmlHttp.responseText;

   }
}


function GetXmlHttpObject() {
   var xmlHttp = null;

   try {
      // Firefox, Opera 8.0+, Safari
      xmlHttp = new XMLHttpRequest();
   }
   catch (e) {
      //Internet Explorer
      try {
      xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
      }
      catch (e) {
      xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
   }
   return xmlHttp;
}
```

There are in total three functions included in the Javascript file. A global variable "xmlHttp" is declared at the beginning of the file to hold the reference to the *XMLHttpRequest* object to be created.

When an execution of the function "showTable()" is triggered from the "onChange" event caused by the user action on the HTML page, it first calls the

function “GetXmlHttpObject” to create an *XMLHttpRequest* object. As we discussed before, due to the different versions of the implementation of the object across different web browsers, branching syntax is used in the function to order to successfully create the object on each browsers.

Once the *XMLHttpRequest* object is successfully created, the object’s method “open” is used to determine the HTTP method to be used for the request and the destination the request is to be sent to. In this case, the method is set to “GET”, and the request destination is our PHP page (ajax_ex.php) maintained on the server. In addition, a parameter “q” with the selected value from the dropdown menu on the HTML page and a random generated session id are added to the request URL. The session id is intended to prevent the server from using a cached file. As seen in the snapshot, the option “Paul Lee” is selected, and the request destination in this case is set to “ajax_ex.php?q=3&sid=0.5341513556740984”. In this way, the whole HTML form does not need to be submitted in order to send the value from the dropdown menu. The system avoids the reloading of the HTML page every time user makes an action that triggers a request to the server.

The function “stateChanged()” defines the request object’s property “onstatechange” as the event handler. The function executes every time the state of the *XMLHttpRequest* object changes. As we see in the function body, when the state of the request object equals to “4”, namely, the data returned from the server has been successfully received, it writes the returned data to the place defined by the <div> element “tablet” on the HTML page. In this case, we use the request object’s property “responseText” to obtain the string version of the returned data from the server (here we use the plain text as the format of the returned data from the server, later we will discuss on using XML as the format of returned data).

The PHP file

The PHP file is maintained on the experimental server.

```
1. <?php
2. $q=$_GET["q"];

3. $con = mysql_connect('mysql.scms.waikato.ac.nz', 'zs15',
                        '0220104xing');
4. if (!$con)
 {
5.      die('couldn't connect: '.mysql_error());
 }

6. mysql_select_db("zs15", $con);

7. $sql="SELECT * FROM users WHERE user_id = '".$q."'";

8. $result = mysql_query($sql);

9. echo "<table border='1'>
10. <tr>
11. <th>Firstname</th>
12. <th>Lastname</th>
13. <th>Age</th>
14. <th>HomeTown</th>
15. <th>Job</th>
16. </tr>";



17. while($row = mysql_fetch_array($result))
 {
18. echo "<tr>";
19. echo "<td>" . $row['FirstName'] . "</td>";
20. echo "<td>" . $row['LastName'] . "</td>";
21. echo "<td>" . $row['Age'] . "</td>";
22. echo "<td>" . $row['HomeTown'] . "</td>";
23. echo "<td>" . $row['Job'] . "</td>";
24. echo "</tr>";
 }
25. echo "</table>";

26. mysql_close($con);
?>
```

There are basically two parts in the PHP file, one part takes charge of the connection and query to the database on the server, and the other is responsible for putting the results from the database query into the right format and then return them to the client.

At line 2, the value of the dropdown menu is received and held in the variable "$q" (recall that in the Javascript function "showUser(str)", we include a parameter "q" which was set to the value of the dropdown menu item as part of the request destination). From line 3 to line 8, the instructions open a connection to the database by specifying the name of the database, the username and the

password. Then the value of the dropdown menu is used to form a query against the database. The result from the query is stored in the variable “$result”.

From line 9 to line 25, based on the results from the database query, a piece of text is generated and sent to the client browser as the response. The text holds the query result and also instructions to display a table.

At line 26, connection to the database is shut down.

As we have seen in the snapshot, the option “Paul Lee” has been selected, the PHP page receives the request and gets the value of the dropdown menu item, which is “3”. Having the value of the dropdown menu, a SQL query “SELECT * FROM users WHERE user_id = ‘3’;” (at line 7 in the PHP source code listed above) is run against our database, which returns one record. Then based on the record returned from the query, a piece of text is generated and sent back to the client.

Here is what the text looks like when the option “Paul Lee” has been selected:

```
<table border='1'>
   <tr>
      <th>Firstname</th>
      <th>Lastname</th>
      <th>Age</th>
      <th>HomeTowm</th>
      <th>Job</th>
   </tr>
   <tr>
      <td>Paul</td>
      <td>Lee</td>
      <td>28</td>
      <td>N.Y</td>
      <td>bus driver</td>
   </tr>
</table>
```

This text will be interpreted at the client browser to display a table on the interface page.

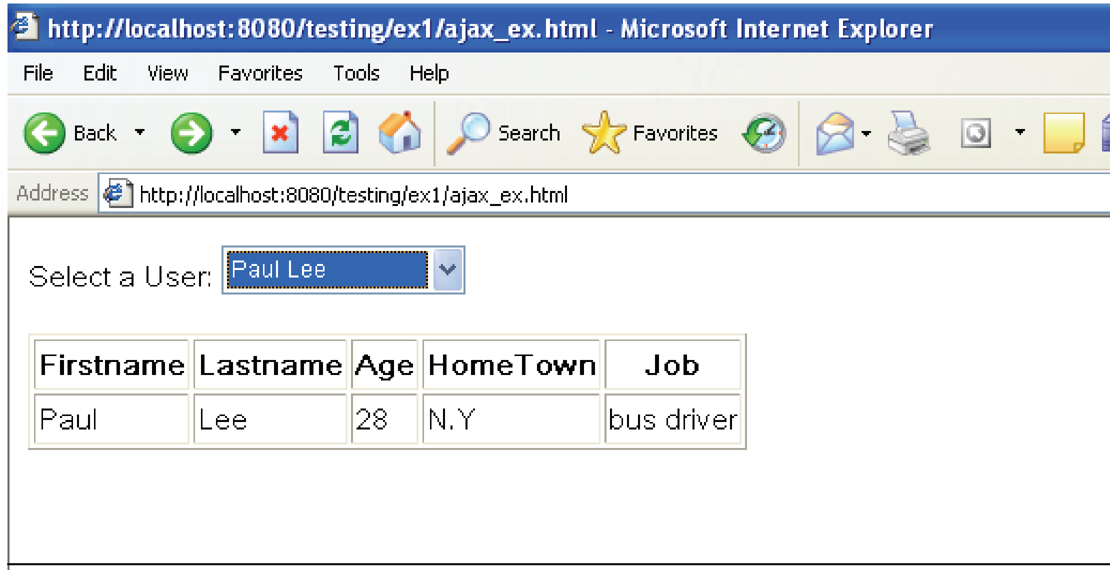

Figure 4.4 a table on the interface showing the selected personal information

As we have seen, using the *XMLHttpRequest* object, data retrieved from the server can be represented in the form of text and obtained by accessing the *XMLHttpRequest* object's property "responseText". However, a more powerful way is to represent the retrieved data as an XML document object, which can be obtained by accessing the *XMLHttpRequest* object's property "responseXML".

Now, let us alter the example we have seen above to make a version using an XML document to transfer the retrieved data from server. The new version of the example also involves four parts:

- A database
- A web page
- A Javascript file
- A PHP file

The database

No change needs to be made to the database, the contents in the database stays the same:

user_id	FirstName	LastName	Age	HomeTown	Job
1	Joe	Harris	33	L.A	actor
2	Michael	Robinson	53	Auckland	school teacher
3	Paul	Lee	28	N.Y	bus driver
4	Jeffery	Morris	35	Wellington	police officer
5	Griffin	King	34	N.J	pilot

Table 4.4 same set of personal data used in the example

The HTML web page

```
1. <html>
2.    <head>
3.      <script src="ajax_ex_xml.js"></script>
4.    </head>
5.    <body>
6.       <form>
7.       Select a User:
8.         <select name="users" onchange="showTable(this.value)">
9.            <option value="1">Joe Harris</option>
10.           <option value="2">Michael Robinson</option>
11.           <option value="3">Paul Lee</option>
12.           <option value="4">Jeffery Morris</option>
13.           <option value="5">Griffin King</option>
14.        </select>
15.      </form>

16.      <form>
17.       <div id="firstname"></div>
18.       <div id="lastname"></div>
19.       <div id="age"></div>
20.       <div id="hometown"></div>
21.       <div id="job"></div>
22.      </form>
23.   </body>
24.</html>
```

There are two forms on the page. The first form is to create a drop down menu, which is exactly the same as the one used in the page in the last version of this example. The second form has five <div> HTML elements, which are used to display the contents of the XML document returned from the server side.

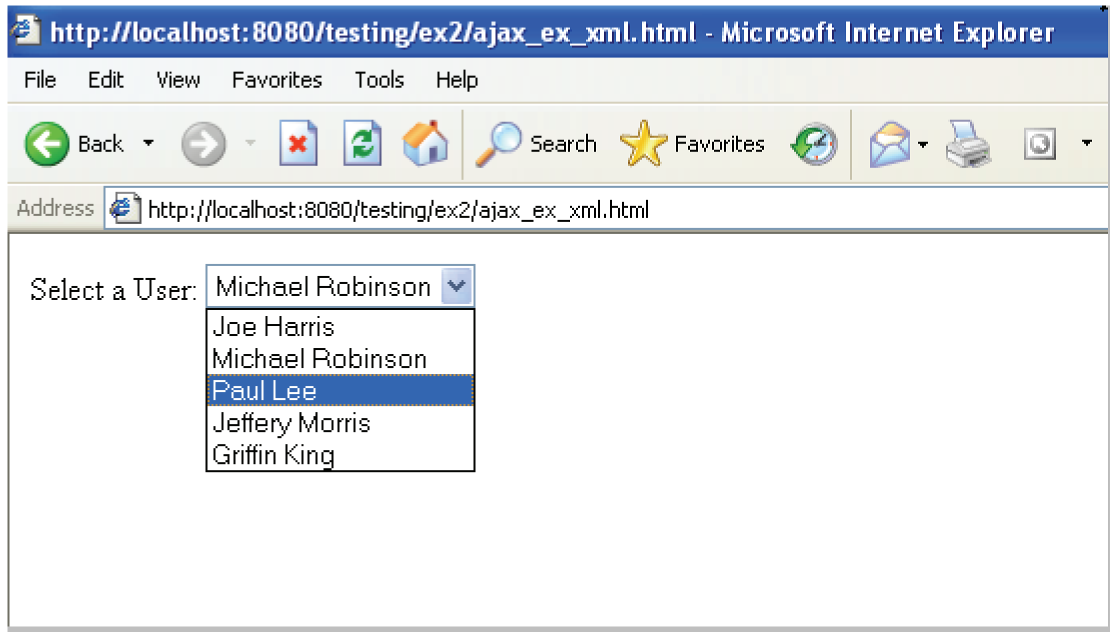

Figure 4.5 the example user interface

Figure 4.5 is a snapshot of the HTML page. As can be seen, the third option "Paul Lee" is to be selected. When user selects data on the dropdown menu, an "onchange" event is caused, which will again trigger the execution of the Javascript function "showTable(str)". Again the function takes the selected value from the dropdown menu as the parameter. As before, all the Javascript functions are included in an external file – "ajax_ex_xml.js", which is linked to in the HTML source code (at line 3).

The Javascript file:

```
var xmlHttp

function showTable(str) {
   xmlHttp=GetXmlHttpObject();
   if (xmlHttp==null) {
      alert ("Browser does not support HTTP Request");
      return;
   }
    var url="ajax_ex_xml.php";
    url=url+"?q="+str;
    url=url+"&sid="+Math.random();
    xmlHttp.onreadystatechange=stateChanged;
    xmlHttp.open("GET",url,true);
    xmlHttp.send(null);
 }
```

```
function stateChanged() {
   if (xmlHttp.readyState == 4 ) {
      xmlDoc=xmlHttp.responseXML;

      document.getElementById("firstname").innerHTML
             = "FIRSTNAME:" +
               xmlDoc.getElementsByTagName("firstname")[0]

.childNodes[0].nodeValue;

      document.getElementById("lastname").innerHTML
             = "LASTNAME: " +
             xmlDoc.getElementsByTagName("lastname")[0]

.childNodes[0].nodeValue;

      document.getElementById("age").innerHTML
             = "AGE:" +
             xmlDoc.getElementsByTagName("age")[0]
                                   .childNodes[0].nodeValue;

      document.getElementById("hometown").innerHTML
             = "HOMETOWN: " +
             xmlDoc.getElementsByTagName("hometown")[0]
                                        .childNodes[0].nodeValue;


      document.getElementById("job").innerHTML
            = "JOB: " +
            xmlDoc.getElementsByTagName("job")[0]
                                  .childNodes[0].nodeValue;

  }
}



function GetXmlHttpObject() {
   var objXMLHttp = null

   if (window.XMLHttpRequest) {
      objXMLHttp = new XMLHttpRequest();
   }
   else if (window.ActiveXObject) {
      objXMLHttp = new ActiveXObject("Microsoft.XMLHTTP");
   }
   return objXMLHttp;
 }
```

There are three functions in the Javascript file. Again, a global variable "xmlHttp" is declared and used to hold the reference to the *XMLHttpRequest* object to be created.

Function “GetXmlHttpObject()” is responsible for creating the *XMLHttpRequest* object.

Function “showTable(str)” executes when user selects an option on the dropdown menu. It first makes a call to function “GetXmlHttpObject” to create an *XMLHttpRequst* object, then sends the request to the PHP page “ajax_ex_xml.php” running on the server along with the selected value from the dropdown menu.

As in the last version of this example, the *XMLHttpRequest* object’s property “onstatechange”, the event handler, is defined to be function “stateChanged()”. As we can see in function “stateChanged()”, the request object’s property “responseXML” is used to obtain the response data from the server side (in this case, the data is an XML document). The next lines in the function simply use DOM methods to locate and get the information in the XML document and write it to the HTML page.

The PHP page “ajax_ex_xml.php”

Since in this case, an XML document is used to package the data retrieved on the sever and then sent to the client as the response, certain changes have to be made in our PHP page in order to create the response data in the right format. Below is the altered PHP source code:

```
1. <?php
2. header("Content-Type: text/xml");
3. header("Cache-Control: no-cache, must-revalidate");
4. header("Expires: Mon, 26 Jul 1997 05:00:00 GMT");
5. $q=$_GET["q"];
6. $con = mysql_connect('mysql.scms.waikato.ac.nz', 'zs15',
'0220104xing');
7. if (!$con)
 {
 8. echo 'Could not connect: ' . mysql_error();
 }
9. mysql_select_db("zs15", $con);
10. $sql="SELECT * FROM users WHERE user_id = ".$q."";
11. $result = mysql_query($sql);
```

```
12. echo '<?xml version="1.0" encoding="ISO-8859-1"?>
13. <person>';
14. while($row = mysql_fetch_array($result))
 {
15. echo "<firstname>" . $row['FirstName'] . "</firstname>";
16. echo "<lastname>" . $row['LastName'] . "</lastname>";
17. echo "<age>" . $row['Age'] . "</age>";
18. echo "<hometown>" . $row['HomeTown'] . "</hometown>";
19. echo "<job>" . $row['Job'] . "</job>";
 }
20. echo "</person>";
21. mysql_close($con);
22. ?>
```

In the same way as in the PHP source code used for the last version of this example, the PHP code in this file includes two parts: one part (from line 6 to line 11) sets up a connection to the database, forms a query taking the value from the request sent to the page (line 10) and then run the query against the database. What is to be noticed is, at line 2 the header of the PHP file is set to be "`Content-Type: text/xml`", which tells that the return document type is XML. Line 3 prevents the file from being cached.

In the second part (from line 12 to line 19), results from the query against the database are inserted into an XML document, which will be sent back to the client side as the response data.

In our case, as the third option "Paul Lee" is selected, the XML document returned from the server will look like this:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<person>
  <firstname>Paul</firstname>
  <lastname>Lee</lastname>
  <age>28</age>
  <hometown>N.Y</hometown>
  <job>bus driver</job>
</person>
```

This XML document will then be received and processed in the Javascript function "stateChanged()" to update the HTML page:

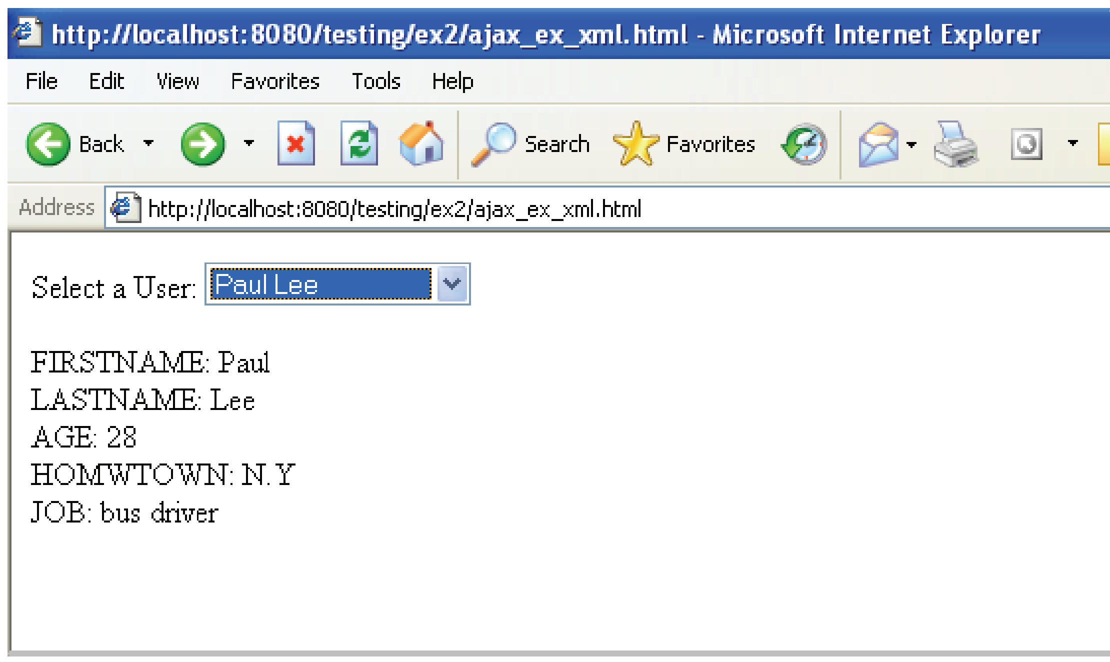

Figure 4.6 information of selected person displayed on the interface

Chapter 5 GOOGLE MAPS, THESIS PROJECT

5.1 Introduction to Google Maps

Google Maps is probably one of the most sophisticated AJAX web applications to be developed so far. It provides aerial views in various resolutions of the world, and permits smooth navigation by dragging the mouse. Associated with Google's powerful search engine, Google Maps offers the service of finding a specific area on the map - users can quickly find a location by entering an address or the name of general area. Also, like other online map services, Google Maps provides the users with a driving direction service. They can get a step by step list of how to get to the destination and an estimated travel time.

There are different viewing modes available in Google Maps: Map (street view), Satellite (aerial photographs taken by satellites) and Hybrid (a view of the last two modes of views overlaid), users can switch between the three modes by clicking on the textual links on the top right corner of the map.

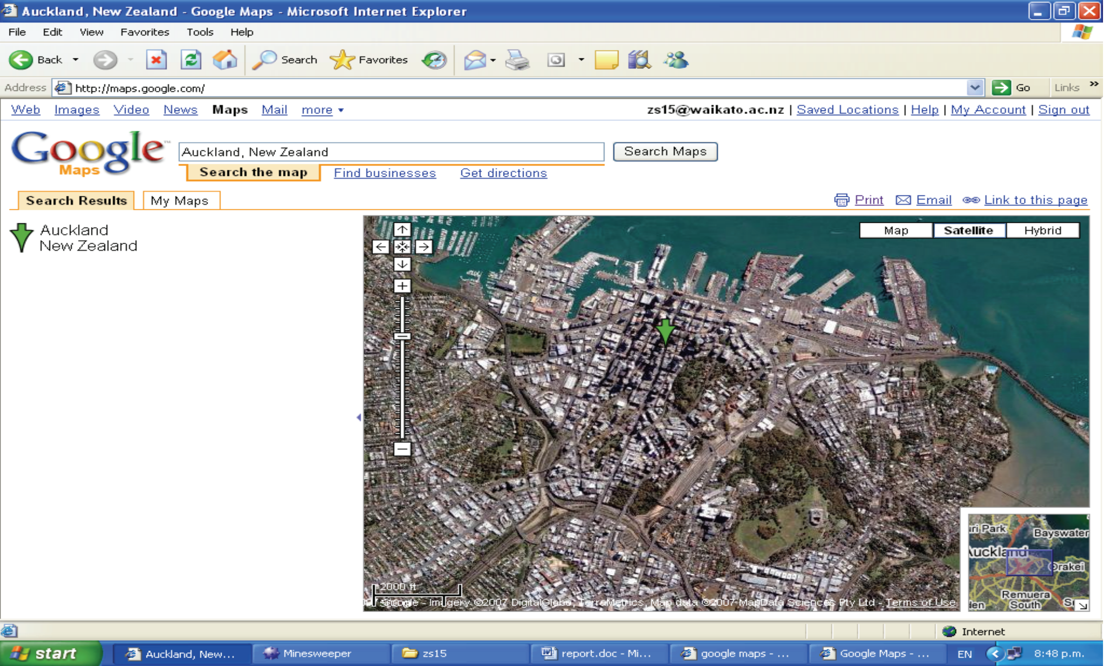

Figure 5.1 Google Maps user interface

The search function works in a classic web application mode, when a certain location query has been entered it refreshes the entire web page. The view of the map is implemented in the AJAX fashion.

The most interesting and noticeable feature of the Google Maps interface is the panning feature. Users can drag on the map surface to shift the area of view by using the mouse and this panning of the map does not cause a reload of the page at all. The whole map view is composed of small image tiles, when the map is panned to a place far enough to expose some new image tiles, they are asynchronously downloaded from the server. There is a noticeable time during which blank white areas displaying initially before the new tiles have been loaded and filled in spaces. However, during the downloading of the tiles, users can still continue to drag on the map, which will trigger a request for more image tiles from the server. Also, those image tiles already loaded are cached by the application for the extent of the session so that it makes the program operates quickly when the map is panned back to parts already viewed.

Moreover, Google Maps makes use of the browser overflow to serve as an "image tile buffer" to make the map really responsive. When a map view is initialised, not only the image tiles immediately required for the display are loaded, but also some extra outskirt tiles are loaded. These are kept visibly hidden and shown when necessary. For example, when a map of size 4 tiles is loaded, the application does not only load 4 tiles, instead it may load 16 tiles to the browser; 4 of the 16 tiles will be used to fill in the view to compose the map, other 12 will be "hidden". When the user drags on this map, some new tiles have to be exposed to provide the requested view, and then there are two cases:

1. All the requested tiles are already loaded to the browser-the 12 hidden tiles include those required to compose the updated view. All it needs to be done is to make them visible. The requested map view will show almost

instantly and at the same time the browser starts to download some fresh image tiles from the server to fill the tile buffer.

2. the 12 hidden tiles are not enough to compose the required view (the user has panned the map far enough), some of the tiles needed are still to be downloaded from the server plus addtional ones to fill the buffer, in which case the new map will not be filled up instantly. For either of the two cases, the tile buffering helps improve the responsiveness of the application.

Google Maps provides web developers with an API library as a free service. This allows the map service to be integrated into the developers' own websites. In order to access the Google Maps API, a developer needs to sign up an API key first which will be bonded to a specific URL. A Google maps API key can be registered at http://www.google.com/apis/maps/signup.html. The Google Maps API comes up with classes that allow users to create map view, add different kinds of overlays on the map – polylines, markers, points, display info windows, add map controls, and load XML files from the server and so on.

5.2 The Thesis Project

The task for this thesis project was to build a web GIS system, on which some given agricultural data was to be graphically displayed at users' requests. Because of the advanced features and the convenient and free API that Google Maps offered, we chose to use Google Maps to build the map for the GIS system.

First thing to do was to sign up for a Google Maps API key. Along with the API key, access to the Google Maps API documentation [15] was also provided. The documentation gave a description of the API and also examples on how to create a map then add different sorts of overlays on the map.

I started with the examples from the tutorial and experimenting with them. This is an example download from the tutorial:

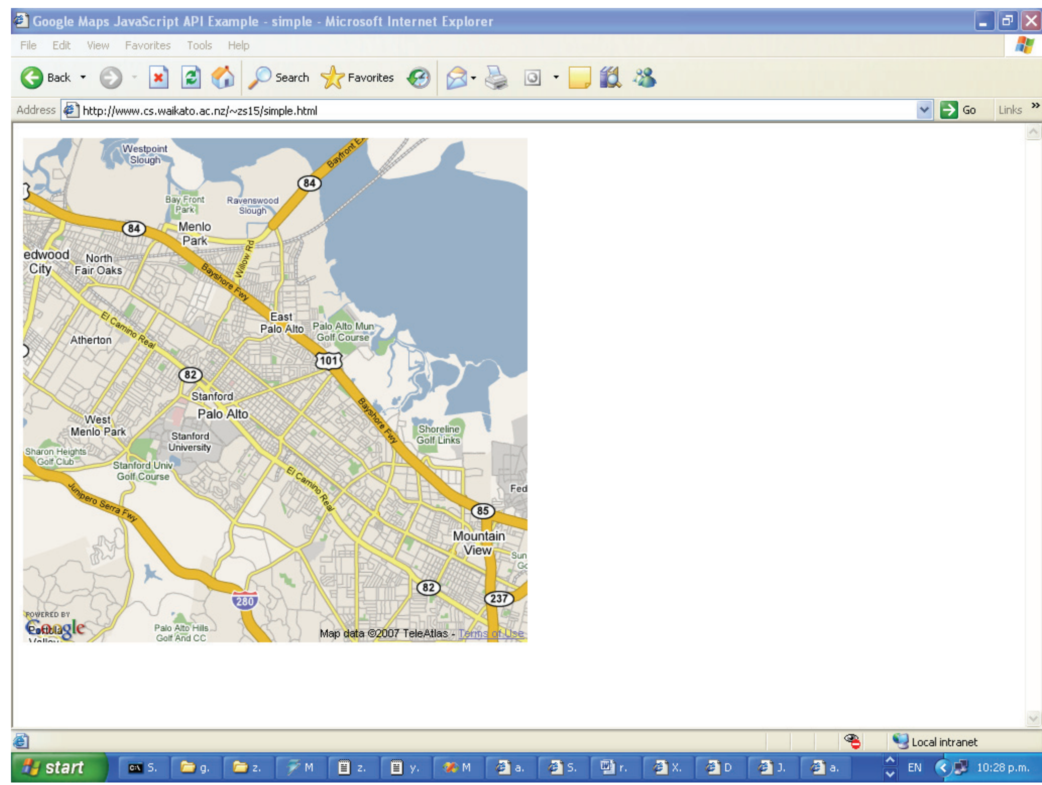

Figure 5.2 a simple Google Maps example. A page with a Google Maps display embedded in

Here is the source code of this example:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

1.<html xmlns="http://www.w3.org/1999/xhtml" xmlns:v="urn:schemas-
                                                microsoft-
com:vml">

2.   <head>

3.     <meta http-equiv="content-type" content="text/html;
charset=UTF-8"/>

4.     <title>Google Maps API Example - simple</title>

5.     <style type="text/css">
    v\:* {
      behavior:url(#default#VML);
    }
    </style>

6. <script
      src="http://maps.google.com/maps?file=api&v=1&key=ABQIAAAA8R
      Yg82pr7vMNIlCaxAU5mBRzPz2OzvJpWBiMy4DZ5ovONhiulRTlWa_7CeE6p6
      e7w93qM0ssxIaqOQ " type="text/javascript">
```

```
   </script>
   <script type="text/javascript">
    //<![CDATA[

7.    function onLoad() {
      // The Basics
      //
      // Creates a map and centers it on Palo Alto.

8.       var map = new GMap(document.getElementById("map"));

9.       map.centerAndZoom(new GPoint(-122.141944, 37.441944), 4);
    }

    //]]>
    </script>
</head>

10.  <body onload="onLoad()">

11.    <div id="map" style="width: 500px; height: 300px"></div>
12.    <div id="message"></div>
13.  </body>

14.</html>
```

It was a very simple map display (probably the simplest form of a Google Maps application). At line 14, a <div> element was specified on the HTML page in order to contain the map. The view of the map was represented by an object of the API class “GMap”, which was created in the Javasscript function “onLoad()”. Also, at line 12, by calling the map object’s method “centerAndZoom”, the map view was specified to be at the resolution level “4” and centred at the location specified in latitude and longitude (-122.141944, 37.441944).

As we can see in the snapshot of the example, there was only one type of map view on the page, and there were no map controls attached to it. Map controls and more map types were added to the example by adding two lines to the Javascript function “onLoad()”:

```
map.addControl(new GLargeMapControl());
map.addControl(new GMapTypeControl());
```

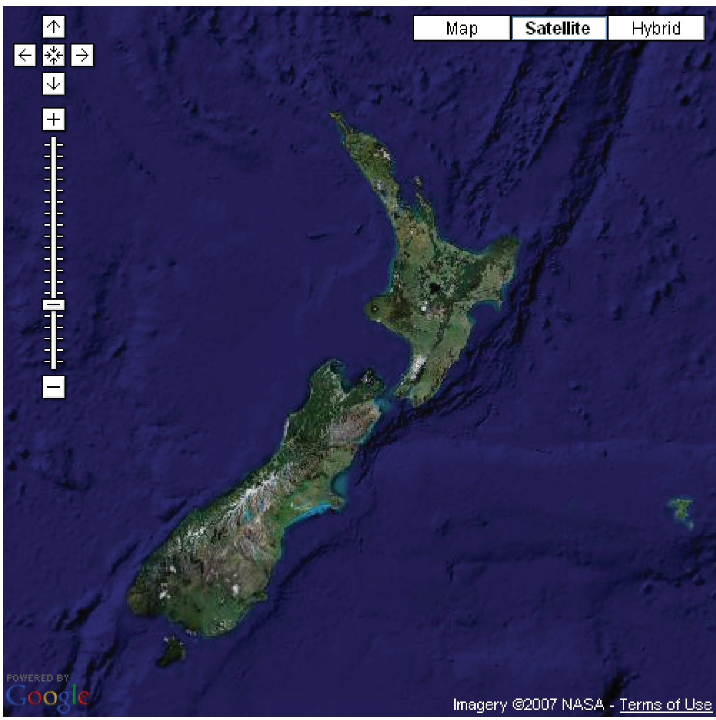

Figure 5.3 a map with the map type control and zoom control enabled

As the project was to display agricultural data in New Zealand, the modified map was panned and zoomed to give an aerial view of the entire country:

5.3 Setting up the experiment environment

There were a few things that learnt from the example so far. First, a view of a Google Maps map on the page was an instance of the class “GMap” in the Javascript library. Second, the coordinate system adopted by the Google Maps was specified in latitude and longitude. Unfortunately, the geographical locations related to the agricultural information for this project were specified in a different coordinate system in the data provided. Also, the resolution of the image data that Google Maps had for New Zealand was not good enough to meet the requirement of the project. Therefore, the focus of the work was set on adding the custom map data to Google Maps and changing its coordinate system. Both of the tasks depended on an analysis of the Google Maps Javascript library.

The library is referenced in the <script> section at line 6 of the HTML source code, an external Javascript file reference was specified by the URL:

“http://maps.google.com/maps?file=api&v=1&key=ABQIAAAA8RYg82pr7vMNIlCaxAU5mBRzPz2OzvJpWBiMy4DZ5ovONhiulRTlWa_7CeE6p6e7w93qM0ssxIaqOQ”. The text string after “key=” was the key I was allocated when I registered for the API.

A Javascript file was obtained from the URL above (saved as “first.js”). Looking into the code in the file, there were a number of variable definitions and functions and basically what it did was to check the compatibility of the browser it was running on. One of the functions caught my attention:

```
.
.
.

function GLoadMapsScript(){
      if (_havexslt()){

          _script("http://maps.google.com/mapfiles/maps.30a.js");
      } else
      if(_ua('safari')){

      _script(http://maps.google.com/mapfiles/maps.30a.safari.js
              );
      }
       else{

      _script("http://maps.google.com/mapfiles/maps.30a.xslt.js");
       }
}
.
.
```

The function “_script(URL)” has the same purpose as writing a <script> section in the HTML document which put a link to the external Javascript source file located at the specified URL. In this case, as the Internet Explorer was used for the experiment, the file actually loaded was from the URL:

http://maps.google.com/mapfiles/maps.30a.js.

A much larger Javascript file called “maps.30a.js” was then downloaded from the URL above. The contents were all squashed together and a considerable amount of time was required to tidy it up in order to make it readable.

The file was large, there were a total of 8, 000 lines of code in it after the tidying up. In order to test if this file constituted the Google Maps library implementaion, I used the two Javascript files obtained from the Google's site to build up my experiment environment for the project:

Both of the two Javscript files and the HTML page were moved onto a server running a website for the project experiment.

First, a change was made to the function "GLoadMapsScript()" in the Javascript file "first.js" (the smaller file obtained from the web). The file URL for script loading was replaced by the file path of the saved file "maps.30a.js" (the large Javascript file downloaded from the Google site).

```
.
.
.

1.function GLoadMapsScript(){
2.     if (_havexslt()){
3.           _script("maps.30a.js"); //changed to use the local
                                   //file
4.     } else
5.     if(_ua('safari')){
      6.     _script(
               http://maps.google.com/mapfiles/maps.30a.safari.js
             );
      }
7.      else{
8.           _script(
               http://maps.google.com/mapfiles/maps.30a.xslt.js
             );
       }
}
.
.
```

At line 3, the parameter for function "_script()" was changed to the path of the local file "maps.30a.js". Since Internet Explorer was used to run the experiment, no changes were made to the statements dealing with other web browsers.

Also, a change was made to the HTML page where the map was embedded. In the <script> section where the Javascript file was linked, the file link was changed to the saved local file "first.js".

Here is the updated HTML source code with more map controls added and the external Javascript file link changed:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

1.<html xmlns="http://www.w3.org/1999/xhtml">

2.  <head>

3.     <title>Google Maps JavaScript API Example - simple</title>

4.      <script src="first.js" type="text/javascript"></script>

5.  </head>
6.  <body>

7.      <div id="map" style="width: 500px; height: 500px"></div>

8.      <script type="text/javascript">

    //<![CDATA[

9.      if (GBrowserIsCompatible()) {

10.         var map = new GMap(document.getElementById("map"));

11.         map.addControl(new GLargeMapControl());

12.         map.addControl(new GMapTypeControl());

13.         map.centerAndZoom(
               new GPoint(-122.141944, 37.441944), 5
            );

         }
       //]]>

14.       </script>
15.  </body>
16. </html>
```

At line 4, the reference to the external Javascript file was changed to link to the Javscript file “first.js”, which was maintained on the experiment website so that the Javascipt file “first.js” was to be imported when loading the HTML page. Also, since the change made to the function “GLoadMapsScript()”, the file “maps.30a.js” maintained on the experiment site was to be loaded instead of a file from the Google site.

The updated HTML page turned out showing no difference to its previous version, and the map object on the page was fully functioning with perfect

controllability as before (the navigation of the map, switch between different map types, and the zooming function). This proved the idea that the Javascript file "maps.30a.js" was the file that responsible for building the Javascript library for the Google Maps API.

5.4 Finding Class Definitions

As seen in the HTML page source code, the view of the map was an object of the class "GMap", so I continued my analysis by searching for the definition of the class "GMap".

The search for the class definition was begun by finding the key word "GMap" in the whole source file, which led to an interesting part of the document:

```
.
.
function Rb(a){

1.    var b=a||window;
2.    b._MapsApplication=s;
3.    b._VPage=xa;
4.    b._Point=v;
5.    b._Size=G;
6.    b._Bounds=E;
7.    b._Map=n;
8.    b._Icon=X;
9.    b._Marker=H;
10.   b._Polyline=I;
11.   b._LargeMapControl=fa;
12.   b._SmallMapControl=Ea;
13.   b._SmallZoomControl=Fa;
14.   b._MapTypeControl=aa;
15.   b._GoogleMapMercSpec=C;
16.   b._KeyholeMapMercSpec=M;
17.   b._HybridMapSpec=Q;
18.   b._GOOGLE_MAP_TYPE=null;
19.   b._SATELLITE_TYPE=null;
20.   b._HYBRID_TYPE=null;
21.   b._DocumentTransport=va;
22.   b._XmlHttp=Va;
23.   b._Xml=Ka;
24.   b._Xslt=U;
25.   b._Event=j;
26.   b._Timer=T;
27.   b._Log=P;
28.   b._makePasteBox=Sb;
29. b._getElementsByClassName=xd;
30.   b.GMapsApplication=s;
31.   b.GVPage=xa;
32.   b.GPoint=v;
33.   b.GSize=G;
34.   b.GBounds=E;
35.   b.GMap=n;
36.   b.GIcon=X;
37.   b.GMarker=H;
38.   b.GPolyline=I;
39.   b.G_DEFAULT_ICON=S;
40.   b.GControlPosition=ca;
41.   b.G_ANCHOR_TOP_LEFT=0;
42.   b.G_ANCHOR_TOP_RIGHT=1;
43.   b.G_ANCHOR_BOTTOM_LEFT=2;
44.   b.G_ANCHOR_BOTTOM_RIGHT=3;
45.   b.GLargeMapControl=fa;
46.   b.GSmallMapControl=Ea;
47.   b.GSmallZoomControl=Fa;
48.   b.GMapTypeControl=aa;
49.   b.GScaleControl=ma;
50.   b.G_MAP_DEFAULT_COPYRIGHTS=Tc;
51.   b.G_MAP_API_COPYRIGHTS=Qc;
52.   b.GGoogleMapMercSpec=C;
53.   b.GKeyholeMapMercSpec=M;
54.   b.GHybridMapSpec=Q;
55.   b.GDocumentTransport=va;
56.   b.GXmlHttp=Va;
57.   b.GXml=Ka;
58.   b.GXslt=U;
59.   b.GEvent=j;
60.   b.GTimer=T;
61.   b.GLog=P;
62.   b.GMakePasteBox=Sb
}
.
.
```

At line 35 in the function body, it stated “b.GMap=n”. I had an idea it was a reference of the name used in the actual coding for the “GMap” class (i.e. the class document in the API as “GMap” was actually coded with name “n”). Also note the statements at lines from 32 to 38 and lines from 45 to 49, in which several other names of API classes (GSize, GPoint,.., GMarker… GLargeMapControl,.., GMapTypeControl) appeared in the same manner, which was adding some more confidence to the idea.

A followed up search with the key word “n” brought the focus of attention in the document to the “n section” - a section in the document begining with the function “n( )” followed by a list of related functions. The section was proved to the definition of the API class “GMap” by adding alerting windows in the body of function “n”. The function “n” was the constructor method of class “GMap” and functions following behind it in the section were the class methods. In the same way, definitions of other API classes were located in the document as well, e.g. The “G” section for API class “GSize”, the “V” section for API class “GPoint”.

Having located the definitions of the API classes, the focus of my work was moved on to analysing the source code. Alerting windows were inserted into the lines to order to trace the work flow and considerable time was spent on reading the obfuscated code. Also, some articles and posts [3, 10, 11, 14, 16, 19, 22, 24, 30, 32, 35] on the web regarding the subject were found to be very helpful in finding clues during the process.

5.5 Main API Classes, the Image Tiling System

A view of the map on the page is represented by an object of the API class “GMap”. The “GMap” class has a list of class methods that respond to all kinds of user activity on the map (dragging on the map, clicking on the map, switching between different map types, zooming in and out). A map can have one of three different map types: the standard street view map type, the satellite view map type and the hybrid view map type. These three types of map shown on a web page are objects of three API classes, which are:

- Class “GGoogleMapMercSpec” – defined in the “C” section and represents the standard map type.

- Class “GKeyholeMapMercSpec” – defined in the “M” section and represents the satellite map type.

- Class “GHybridMapSpec” – defined in the “Q” section and represents the hybrid map type.

When a “GMap” object is created, objects of the three map type classes are created as well and inserted into an array which is held as a data member of the map object. One of them is set to be the default map type which will show on the screen initially. Basically, the “GMap” object works as a container of the map type objects providing methods through which the properties of the map type objects can be accessed and methods of the map type objects can be executed.

The class definitions for the three map types have the same structure with same set of class methods and properties. Methods responsible for handling the imagery data are contained in the map type classes. The graphic view of a map is created by numbers of square image tiles stitching together. For example, in the map we see below (figure 5.4):

Figure 5.4 a map showing the satellite view of an area

The view of the map shown above is made up of 16 image tiles. As mentioned before, only parts of the 16 tiles are displayed, the rest of them are buffered in the memory of the browser to enable a fast map navigation response.

One of the image tiles looks like below(figure 5.5):

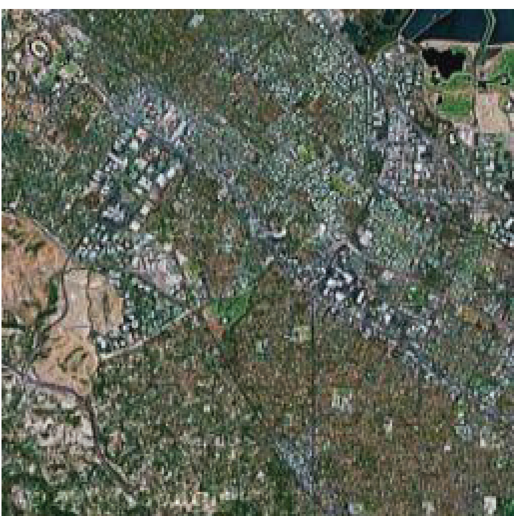

Figure 5.5 one of the image tiles used to create the map shown in figure 5.4

The image tiles are pre-rendered with a size of 256 by 256 pixels and stored on the image servers at 18 resolution levels (level 0 to level 17). The size of the image tiles and the number of the available resolution levels are specified in the constructor method of the map type class.

In the body of function "C()"–the constructor method of class "GGoogleMapMercSpec"- the standard street map type:

```
.
.
this.tileSize=256; //specifies the image tile size
.
.
this.numZoomLevels=18; //specifies the number of resolution levels
.
.
.
```

Level 17 gives the lowest resolution and level 0 the highest. From one resolution level to its next higher level, the number of the image tiles used to create the map is doubles in both vertical and horizontal directions. In other words, at a certain resolution level, four image tiles are used to cover the same area that is covered by one single image tile at the next lower resolution level.

At level 17, which gives the most distant view of the world, the whole world is fitted into a single image tile.

Figure 5.6 a map showing the world at the lowest resolution in Google Maps

The map shown above gives a view of the world centred at the intersection of the equator and prime meridian (point (0, 0) in latitude/longitude) at resolution “17”- the lowest resolution level defined in Google Maps. The tiling function of Google Maps makes the world look quite different – we see the world tile repeated because the “size” of the map is bigger than a size of a singe image tile.

This is the one singe image tile used to display the view of the world shown above:

Figure 5.7 the single image tile used to create the map shown in figure 5.6

At level “16”, the resolution twice that at level “17. The entire view of the world is covered in four image tiles.

This is the world looks like at resolution “16” in Google Maps:

Figure 5.8 a map of world at the second lowest resolution in Google Maps

The four image tiles used to cover the world are shown in the table below:

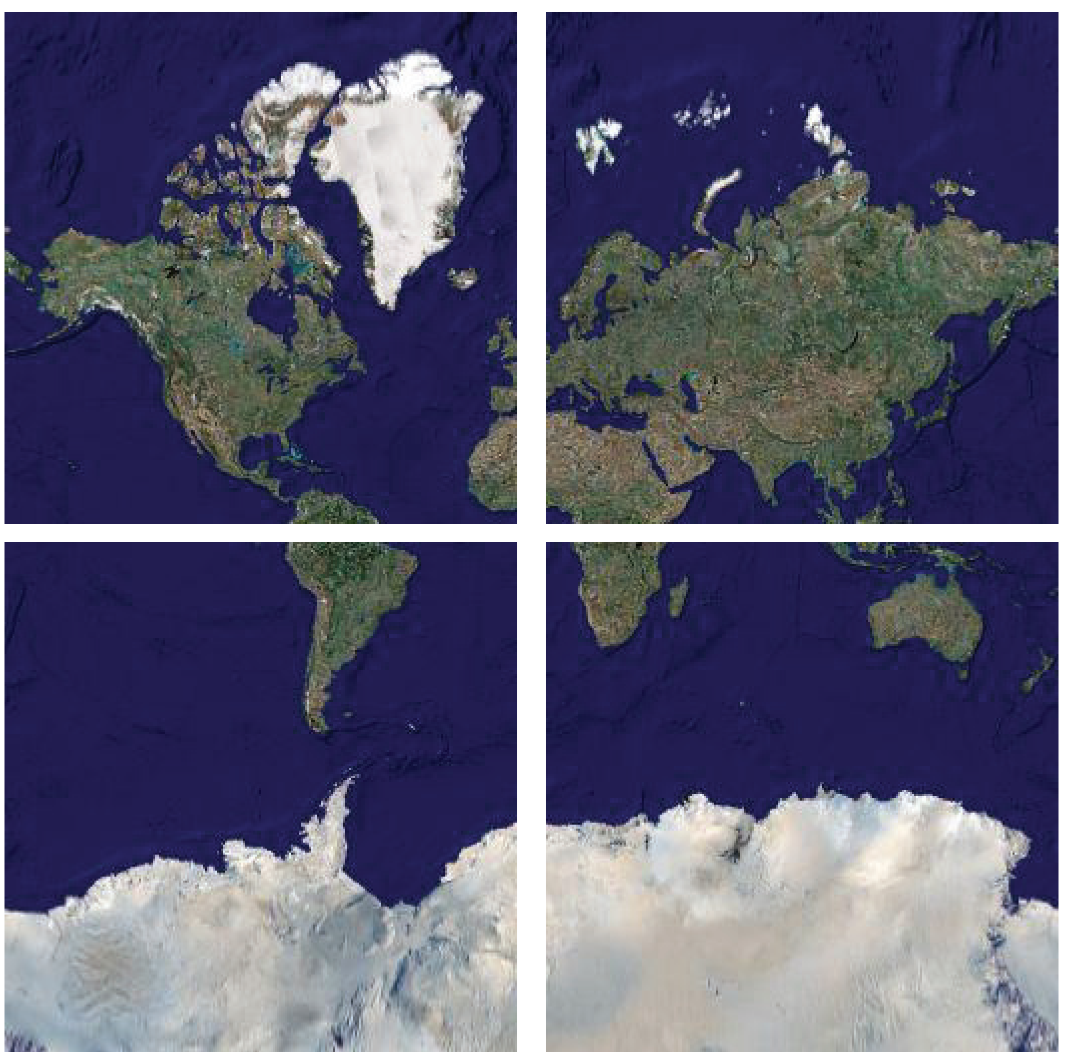

Figure 5.9 the four image tiles used to create the map shown in figure 5.8

Then At level “15”, 16 image tiles are required to complete the view of the entire earth. The relation between the resolution level and the number of image tiles used to cover the view of the earth can be described in the function:

Number of tiles at resolution X= Math.pow (4, (17 - X)).

Keeping calculating in this way, in order to give the view of the whole world at level 0, there are totally 17,179,869,184 image tiles needed!

Requesting an image tile from the image servers turns out to be not such a difficult job. What it takes is to figure out the URL of the image tile. An image tile request URL takes the form:

```
http://mt.google.com/mt?n=404&v=tap.2&n=404&x={x tile index}&y={y
tile index}&z={resolution level}
```

The function that calculates the request URLs of the image tiles is defined in each of the map type classes (named "getTileURL()"). The calculation of the request URL of an image tile is based on the resolution level the map is currently at and the absolute position on the view the tile is meant to fill in.

5.6 The Coordinate System

Locations on the surface of the earth can be specified by geographical coordinates. There are different kinds of geographical coordinates in use.

The most common way to locate a point on the earth is by using latitude and longitude. The latitude value represents the angular distance measured in degrees between a point on the surface of the earth and the equator. Locations on the northern hemisphere are given positive latitude values and those on the southern hemisphere are given negative ones. Points on the equator are at 0 degree in latitude, the north pole of the earth is at positive 90 degrees in latitude and the south pole is at minus 90 degrees in latitude.

Longitude represents the angular distance measured in degrees from a point on the surface of the earth to the prime meridian (the longitude line that runs through Greenwich, England, internationally accepted as the 0 degree longitude line). The whole globe is divided equally into 360 degrees of longitude, points that located east of the prime meridian have positive longitude values and those on the west of

the prime meridian have negative longitude values. Combining the values of the latitude and longitude, any point on the globe can be located.

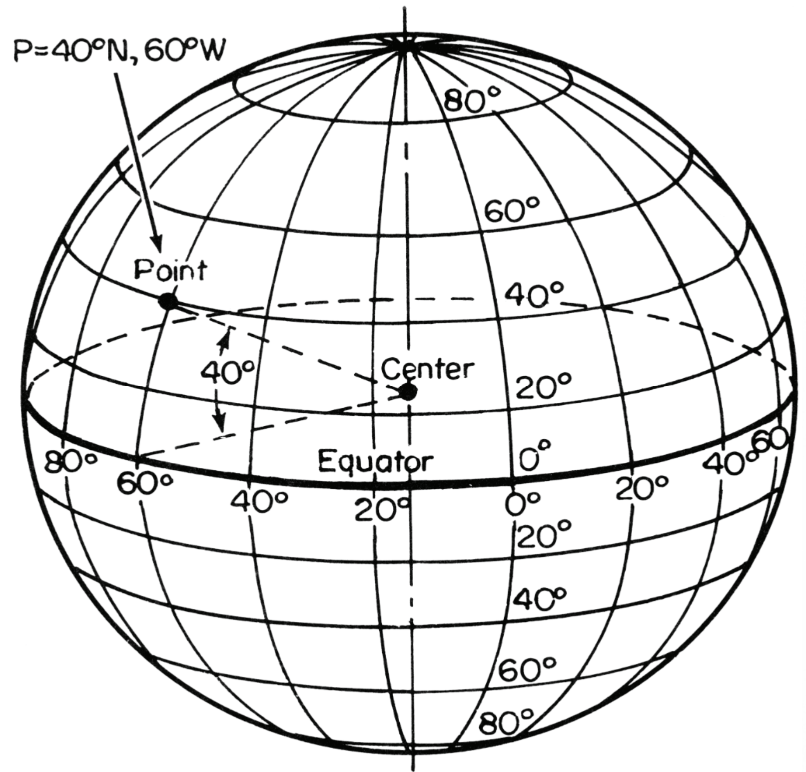

Figure 5.10 Diagram of latitude, longitude grid on a spherical earth (After Charton, 1988)

Locations on Google Maps are specified in latitude and longitude. For example, when creating a map object, the location that the view of the map is to be centred on is specified in latitude and longitude. Also, when adding overlays onto a map (e.g. polylines, markers), locations at which those overlays are to be drawn are specified in latitude and longitude as well.

However, the view of map is presented in a space of pixels on the computer monitor. How is a location specified in latitude and longitude interpreted and then located in the pixel space, or the other way around? A geodetic coordinate system [26, 27] is defined in Google Maps to make the transformation between the latitude/longitude space and the pixel space [24, 25].

The geodetic coordinate system involves several components:

- the map projection function
- the unit system
- the origin

5.6.1 The Map Projection Function

In order to map the globe shaped surface of the earth onto a flat plain (the computer screen), a map projection method is needed. “Flat maps couldn’t exist without map projections, because a sphere cannot be laid flat over a plane without distortions”. [36] A projection method defines the translation function from the geographic coordinates (latitude/longitude) to the ones in the unit system that the flat map is presented in, which in our case is the pixel space. Or in other words, a projection method defines the method of graphical distortion when making a flat map for a spherical surface.

The projection method adopted in Google Maps is the Mercator projection [37]. In other words, the pre-rendered flat image (a presentation of the world in pixels) of the earth (broken into tiles) was produced under the rule of the Mercator projection function. Evidence of the adoption of the Mercator projection can be found in the relevant functions in the source code. These functions specify the mathematical relationship between the latitude/longitude space and the pixel space.

5.6.2 Coordinate System Used in Agricultural Data

As mentioned before, the locations contained in the agricultural data for this thesis project are specified in a different sort of geographical coordinates, Easting and Northing. The values of the coordinates (easting/northing) were defined under a

unique mapping projection function adopted in New Zealand called the "New Zealand Map Grid" (NZMG). By applying that mapping projection, the land area of New Zealand is fitted into a rectangle 1,000,000 metres from east to west by 1,500,000 metres south to north. The origin of the projection is the spot at latitude 41 degrees South and longitude 173 degrees East.

5.6.3 The Transformation Between Two Spaces

In Mercator projection, for the real world – a space specified in latitude and longitude, the origin is the intersection of the equator and the prime meridian (0 degree in latitude and 0 degree in longitude). However, in the pixel space – the modern computer monitor device, the origin is the top left corner of the screen, also the Y values increase downwards. Therefore, in order to calculate the transformation between the two spaces, the origin has to be shifted.

As we know, there are in total 18 resolution levels (based on Google Maps version 1) pre-defined in Google Maps, and at different resolution levels, the world is presented in different sized pixel spaces – composed of different numbers of image tiles. An individual projection is calculated for each of the resolution level.

Following is the source code of the function "iniMercator()" defined in each of the map type classes:

```
.
var kd = 2*Math.PI;
.
C.prototype.initMercator=function(){
1.     this.pixelsPerLonDegree=[];
2.     this.pixelsPerLonRadian=[];
3.     this.bitmapOrigo=[];
4.     this.numTiles=[];
5.     var a=256;
6.     for(var b=this.numZoomLevels-1;b>=0;--b){
7.           this.pixelsPerLonDegree[b]=a/360;
8.           this.pixelsPerLonRadian[b]=a/kd;
9.           var c=a/2;
10.          this.bitmapOrigo[b]=new v(c,c);
11.          this.numTiles[b]=a/256;
12.          a*=2
       }
};
```

The function shown above initialises the values of the parameters of the projection for each of the 18 resolution levels for a map type. These parameters are:

- The origin of the world specified in pixel space

- The surface distance (in pixels)between two one-radian-away latitude parallels - radius of the globe – measured in pixels

- The distance (measured in pixels) that covers one degree in longitude

The Array named "bitmapOrigin" stores the pixel coordinates of the origin of the world for each the resolution levels (at line 3). The Array named "pixelPerLongDegree" records the distance (measured in pixels) that a single longitude degree covers at different resolution levels. And the array named "pixelPerRadian" contains the distance (in pixels) on the global surface between any two one –radian-different longitude parallels for each resolution level.

The origin of the world map at a certain resolution level is the centre of the pixel space that covers the whole world:

```
var tile_size = 256 pixels

var number_of_tiles = Math.pow(2, (17-resolutionLevel)) -note,
this is only the number of tiles along the horizontal axis.

var origin_x = (256 * number_of_tiles ) / 2

var origin_y = (256 * number_of_tiles) / 2
```

The circumference of the globe is just the width of the pixel space.

```
var circumference = tile_size * number_of_tiles
                  = 256 pixels * number_of_tiles
```

Then the surface distance (in pixels) between two one-radian-away latitude parallels on the globe is not hard to work out:

```
var pixel_per_radian = circumference / (2 * Math.PI)
```

The number of pixels that represents a single longitude degree is:

```
var pixel_per_longitude = circumference / 360
```

For example, at level 17 (the lowest resolution level), only one single tile is used to present the view of the world:

```
Number_of_tiles (17) = Math.pow(2, (17-17)) = 1
```

And the size of a single image tile is 256 by 256 pixels.

So the origin – the centre of the pixel space at resolution level 17 is:

```
Origin (17) = (256 pixels / 2, 256 pixels / 2) =
(128 pixels, 128 pixels)
```

The circumference of the globe is:

```
circumference (17) = 256pixels * 1 = 256pixels
```

Therefore, the surface distance between two one-radian-away latitude parallels on the globe is:

```
pixels_per_radian (17) = 256pixels / (2 * Math.PI)
                       ≈ 40.7437 pixels
```

Also, the number of pixels that covered by a single longitude degree is:

```
Pixels_per_longitude (17) = 256pixels / 360
                          ≈ 0.7111 pixels
```

Following the same method, the corresponding parameters for the projection at level 16 are:

```
number_of_tiles (16) = Math.pow(2, (17-16)) = 2

orgin(16) = ((256pixels *2) / 2,  (256pixels *2) / 2)
          = (256pixels, 256pixels)

circumference (16) = 256pixels * 2
                   = 512 pixels

pixels_per_radian (16) = 512pixels / (2*Math.PI)
                       ≈ 81.4873 pixels

pixels_per_longitude (16) = 512pixels / 360
                          ≈ 1.4222 pixels
```

Having figured out the values of the sets of parameters for each of the projection functions at each resolution level, methods defined in each of the map type classes are used to translate coordinates specified in latitudes/longitudes to the ones in pixels, and vice versa.

Javascript function “getBitmapCoordinate()” defined in the map type classes translates the pixel coordinates to latitude/longitude:

```
.
.
var Eb=Math.PI/180;
.
function Za(a){return a*Eb};
.
.

C.prototype.getBitmapCoordinate=function(a,b,c,d){

1.     if(!d){d=new v(0,0)}

2.     d.x=
       Math.floor(
             this.bitmapOrigo[c].x + b*this.pixelsPerLonDegree[c]
       );
```

```
3.     var e=Math.sin(Za(a));

4.     if(e>0.9999){e=0.9999}

5.     if(e<-0.9999){e=-0.9999}

6.     d.y=
       Math.floor(
             this.bitmapOrigo[c].y +
             0.5*Math.log((1+e)/(1-e))*-this.pixelsPerLonRadian[c]
       );

7.     return d
};
.
.
```

The function above takes the location's latitude/longitude coordinates ("a" the longitude and "b" the latitude) and the resolution level ("c") as arguments. The last argument "d" is an object of the API class "GPoint" which is used to hold a two-dimensional point either in latitude/longitude space or in pixel space. The pixel coordinates of the map origin are contained in "bitmapOrigi[c]". "pixelsPerLonDegree[c]" gives the pixel distance that covers one singe longitude degree at the given resolution "c". "pixelsPerLonRadian[c]" gives the distance on the global surface (in pixels) between the two one-radian-different longitude parallels at the resolution "c". The given location's horizontal pixel coordinate is calculated through the equation at line 2 and the vertical pixel coordinate is calculated at line 6.

The reverse-function of the function "getBitmapCoordinate()" translates a location's pixel coordinates to its coordinates in the latitude/longitude space. This function is also defined in the map type classes – function "getLatLng()":

```
.
.
var Eb=Math.PI/180;
.

C.prototype.getLatLng=function(a,b,c,d){

1.     if(!d){d=new v(0,0)}

2.     d.x=(a-this.bitmapOrigo[c].x) / this.pixelsPerLonDegree[c];

3.     var e=(b-this.bitmapOrigo[c].y) /
       -this.pixelsPerLonRadian[c];
```

```
4.     d.y=(2*Math.atan(Math.exp(e))-Math.PI/2) / Eb;
5.     return d
};
.
.
.
```

The function takes a location's pixel coordinates (a, b), the resolution level ("c") and a "GPoint" object ("d") as arguments. The location's longitude coordinate is calculated at line 2, and its latitude coordinate is calculated at line 4.

The two functions are the key functions of the coordinate system in Google Maps defining the transformation between the pixels space and the latitude/longitude space, which are relied on by all the other functions that involve the coordinate's translation (e.g. Defining the where the map is centred at, panning map to a given location...).

A crucial part of adapting Google Maps for the thesis project was to implement alteration to these functions to operate with data specified in Northing/Easting instead of latitude/longitude.

Chapter 6 THE IMPLEMENTATION

As explained in the last chapter, we know that the image tiling system and the coordinate system of the map object are both defined in the map type classes. Therefore, in order to add in custom map data and apply a different coordinate system, the most sensible way is to define our own map type class and add it to the Google Maps API library.

6.1 The New Map Type

In my implementation, the new map type class was named "GCsMapMercSpec" following the API class naming convention. All the relevant Javascript functions that related to the class definition of the new map type were included in "CS section" added to the Javascript API source file "maps.30a.js".

The new map type was defined with exactly the same structure as the other three map type class definitions with the same set of class methods with the same method names.

6.2 Adding the Custom Image Data

The image data for the new map type was obtained by scanning printed maps marked both in latitude/longitude and Easting/Northing coordinates. The resulting image data used for the new map type covered the geographical area from 2,830,000 metres to 2,870,000 metres Easting (values increase from east to west) and 6,330,000 meters to 6,370,000 metres Northing (values increase from south to north) in the North Island in New Zealand. So in other words, the new "whole world" that our new map type covers is the area described above. The origin of this new "world" was the bottom left of the "world", which was the spot located at 2,830,000 metres in easting and 6,330,000 metres in northing. This is a farming area in New Zealand for which we had agricultural data.

The image shown below is the image acquired from the scanning the paper maps (shrunk to fit in the page):

Figure 6.1 image data for the custom map type.

For the new map type, we have created 5 resolution levels (level "4" the lowest resolution level and level "0" the highest one). In the same way as the other three map types, the view of the whole area is made up of numbers of image tiles which are also in the size of 256 squared pixels.

The new map type differs from the other three originally defined map types in that, at the lowest resolution level of the new map type, the area is covered by 4 image tiles instead of one single image tile. After that the increment in the number of image tiles follows in the same way that applied to the other map types. The number of image tiles doubled up the one at next lower resolution level in both

the vertical and horizontal directions. The relation between the number of image tiles required to build the view and the resolution level that the view was at can be described by the equation as below:

```
Number of tiles at level X = Math.pow(4, 4-X)
```

6.2.1 Preparing the Image Tiles

From the equation defining the relation between the resolution level and the number of image tiles that are required to build the view at that resolution level, the sizes of the pixel spaces that views of the map at different resolution levels stand for could be worked out:

```
Size_of_pixelspace at level X = area_of_imagetile *
                                number_of_tiles at level X
```

The table below gives the relationship between the resolution levels, the pixel space sizes and the number of image tiles:

Resolution level	Number of image tiles	Pixel space size
1	4	512pixels X 512pixels
2	16	1024pixels X 1024pixels
3	64	2046 pixels X 2048 pixels
4	256	4096 pixels X 4096 pixels
5	1,024	8192 pixels X 8192 pixels

Table 6.1 Table showing the relationship between the resolution levels, the pixel space sizes and the number of image tiles

For each of the 5 pre-defined resolution levels, the image tiles were produced using the following procedure, starting with high resolution scanned image of the printed map:

1. Resize the original image from the map scanning to the size of the pixel space that the view at the resolution level was meant to cover. E.g. at level

"4", the image was to be resized to 512 pixels X 512 pixels; at level "3", resize the image to 1024 pixels X 1024 pixels.

2. According to the number of image tiles for the resolution level, cut the image obtained from step 1 into tiles using the graphic manipulation software Photoshop. E.g. at level "4", cut the 512 pixels X pixels image into 4 256 pixels X pixels image tiles; at level "3", cut the 1024 pixels X 1024 pixels image into 16 256 pixels X 256 pixels image tiles.

   The following shows the 4 image tiles cut for resolution level "4" of the custom map type:

Figure 6.2 The 4 image tiles for resolution level "4" of the custom map type

6.2.2 Setting the Image Tiling System

For the map types originally defined in the Google Maps API, image tiles required for a map were acquired by sending request URLs to the image servers then downloading them through the network. However, our project application was orientated to users from rural areas, who usually do not have a high speed internet connection. In this case, downloading a number of image tiles could be quite time consuming, thus resulting in poor responsiveness of the application. Therefore, it was decided for the new map type that all the image tiles were to be installed on the client side computer so that loading image tiles would not be a network concern any more.

In the same way as for the other three map type classes, Javascript function "getTileURL()" was responsible for finding locations of the necessary image tiles. As all the image tiles for the new map type were to be stored on the client machine, instead of working out a tile request URL to a server, the function worked out the file path of the requested image tile in the local file system.

Since the image tiles were absolutely positioned on the map, each image tile was named after the horizontal and vertical indices of its position in the whole map and the resolution level the tile was responsible for. Therefore, based on the resolution level of an image tile and its position on the map at that resolution level, the file name of the image tile could be determined. This was how function "getTileURL()" calculates the file path of the image tiles.

An image tile file name takes the form:

[*X index*]_[*Y index*] _ [*resolution level*].JPG

For example, at the resolution level "4" (the lowest resolution level), the map of the whole area consists of 4 image tiles. The top left image tile has the file name "1_1_4.JPG", and the bottom right one has the file name "2_2_4.JPG". At resolution level "3", the view of the whole area consists of 16 image tiles, so the X index of the tiles has the range from 1 to 4 and Y index from 1 to 4. Therefore,

the top left tile of the whole map is the one named “1_1_3.JPG” and the bottom right one “4_4_3.JPG”.

The figure (figure 6.3) shows the image tiles for resolution level “4” and their corresponding file names:

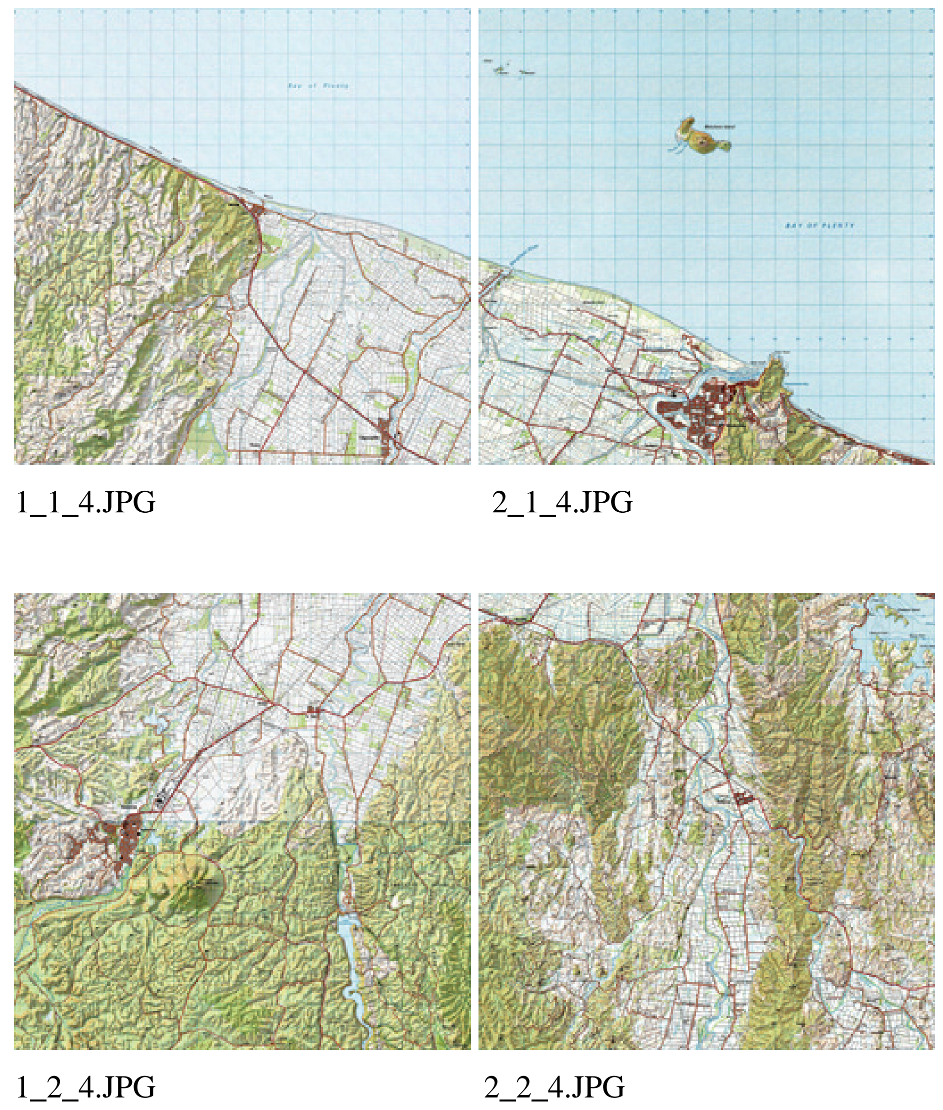

Figure 6.3 Four image tiles used for the custom map at resolution level “4”

6.3 Setting the Coordinate System

As mentioned before, the custom map we built to add to Google Maps covers the geographical area of New Zealand from easting 2,830,000 metres to 2,870,000 metres and northing from 6330000 metres to 6370000 metres (New Zealand Map Grid):

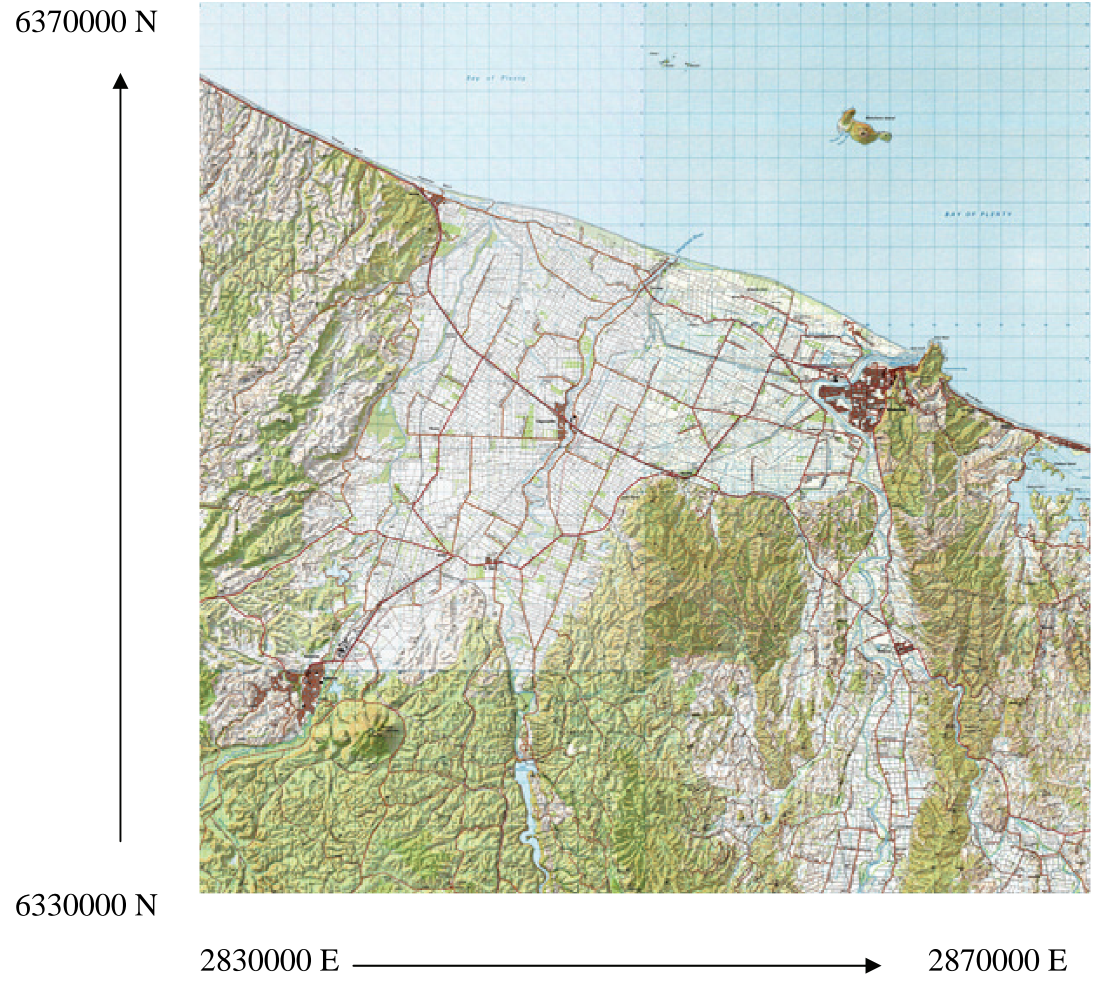

Figure 6.4 the geographic range covered in the custom map

It was found from observation on the printed maps from which the image data were obtained, that the Easting values increased from east to west and the Northing values increased from south to north. In both directions, values were equally distributed over the distance. The map only covered a very tiny range of

area on the earth so that the graphical distortion from the map projection was small enough to be ignored.

6.3.1 Determining the Parameters for the Transformation between Two Spaces

The origin of the "world" for the custom map was set at the bottom left corner of the image square, which was the point (283000E, 6330000N). On a computer monitor screen, the origin of the pixel space is the top left corner, and values increase downwards in the vertical direction. Therefore, at resolution level "4" (the lowest level), as 4 image tiles were used to cover the view of the whole area, the pixel coordinates of the map origin were (0 pixels, 512 pixels). Then, at the next higher level, the origin was (0 pixels, 1024 pixels). In the same way, the pixel coordinates of the map origin at each resolution level were defined.

Also, the pixel distance that covered by a single Easting unit (metre) could be determined. As the Easting values were equally distributed over the distance, the translation function between the pixels and Easting values was linear. So the number of pixels covered by a single Easting unit could be determined by the equation below:

pixels_per_northing = pixel_distance / northing _distance

For example, at resolution level "4" (the lowest resolution level), as 4 image tiles were used to make up the map, the pixel distance is 2 X 256 pixels which covers the distance from 2830000 metres to 2870000 metres in Easting. The number of pixels covered by a single metre in Easting is then:

2 * 256 pixels / 2870000 – 2830000 = 512 pixels / 40000 = 0.0128 pixels

Following the same method, the number of pixels covered by a single Easting unit at each resolution level could be determined.

In the vertical case, the Northing values were also equally distributed over the distance. So the number of pixels covered by a single Northing unit could be determined by the equation below:
pixels_per_northing = pixel_distance / northing _distance

The Northing distance covered in the map is:

6370000 metres – 6330000 metres = 40000 metres

Therefore, at resolution level "4" (at which the whole area was covered by 4 image tiles), the number of pixels covered by a single Northing unit is:

2 * 256 pixels / 40000 = 0.0128 pixels

The pixel distance that covered by a single Northing unit at each resolution level could be determined by the same method.

The calculation of the three parameters required for the transformation between the pixel space and the one specified by Easting and Northing on the map is performed in the Javascript function "iniMercator()" defined in the custom map class:

Following is the source code of the Javascript function "iniMercator()" defined in the custom map type class "GCsMapMercSpec" :

```
CS.prototype.initMercator=function(){
1.     this.pixelsPerNorthing=[];
2.     this.pixelsPerEasting=[];
3.     this.bitmapOrigo=[];
4.     this.numTiles=[];
5.     var a=256*2;
6.     for(var b=4;b>=0;--b){
7.            this.pixelsPerNorthing[b]=a/40000;
```

```
8.          this.pixelsPerEasting[b]=a/40000 ;
9.          this.bitmapOrigo[b]=new v(0,a);
10.         this.numTiles[b]=a/256;
11.         a*=2
      }
};
```

From line 1 to line 3 in the source code shown above, three arrays are declared. The pixel distances covered by a single Northing unit at each resolution level are stored in the array "pixelsPerNorthing"; the pixel distance covered by a single Easting unit at each resolution level is stored in the array "pixelsPerEasting"; Array "bitmapOrigo" contains the pixel coordinates of the map origin at each resolution level. The calculation of these three parameters for each resolution level is defined from line 5 to line 11.

We used the name "iniMercator" for this function for consistent with the other map types, even though it is misleading in this case.

6.3.2 The Transformation Functions

Having determined the values of three key parameters (a. the number of pixels covered by a single Northing unit - pixels_per_northing; b. the number of pixels covered by a single Easting unit - pixels_per_easting; c. pixel coorinates of the origin – bitmapOrigo) for each resolution level, the translation between the coordinates in the Easting/Northing space and ones in the pixels space could be defined.

Since the values in both Easting and Northing directions were equally distributed over distance on the map, the translation between pixels coordinates and the Easting/Northing coordinates were linear functions in both horizontal and vertical directions.

In the horizontal case, to translate a location's Easting value to its X pixel coordinate value; first, we find the distance between the location and the origin in Easting with:

$$\text{distance_easting} = \text{location_E} - \text{origin_E} \quad (1)$$

Then translate this Easting distance into pixels by:

$$\text{distance_pixel_X} = \text{distance_easting} * \text{pixels_per_easting} \quad (2)$$

Since the values increase from left to right (same as the Easting values) in the pixel space, the location's horizontal pixel coordinate is:

$$\text{location_pixel_X} = \text{origin_pixel_X} + \text{distance_pixel_X} \quad (3)$$

Vertically, first we find the Northing distance between the location and the origin:

$$\text{distance_northing} = \text{location_N} - \text{origin_N} \quad (4)$$

Then translate this Northing distance into pixels:

$$\text{distance_pixel_Y} = \text{distance_northing} * \text{pixels_per_northing} \quad (5)$$

In the pixel space, the vertical values increases downwards which is opposite to the way that Northing values increases (upwards). Thus the vertical pixel coordinate of the location is:

$$\text{location_pixel_Y} = \text{origin_pixel_Y} - \text{distance_pixel_Y} \quad (6)$$

An example will complete the description of the coordinate translation functions. For example, we have a location, (2850000 E, 635000N) in Easting/Northing. The map is at the resolution level "4", composed of 4 image tiles, thus the size of the pixel space is 512 pixels X 512 pixels.

The pixel coordinates of the map origin (the bottom left corner) are (0 pixels, 512 pixels):

origin_pixel_X = 0

origin_pixel_Y = 512

The number of pixels covered by a single Easting unit is:

pixels_per_easting = 512 pixels / 40000 = 0.0128 pixels

The number of pixels covered by a single Northing unit is:

pixels_per_northing = 512 pixels / 40000 = 0.0128 pixels

So, for the horizontal coordinate, the Easting distance from the location given to the map origin is (applying equation (1)):

distance_easting = location_E – origin_E = 2850000 – 2830000 = 20000

We can get the pixel distance from the location to the map origin by translating the Easting distance in to pixels (applying equation (2)):

distance_pixel _X = distance_easting * pixels_per_easting

= 20000 * 0.0128

= 256 pixels

Then the location's horizontal coordinate in the pixel space is (applying equation (3)):

location_pixel_X = origin_pixelX + distance_pixel_X

= 0 pixels + 256 pixels

= 256 pixels

In the vertical case, the Northing distance between the location and map origin is (applying equation (4)):

distance_northing = location_N – origin_N

= 2850000 – 283000

= 20000 metres

Translatig the Northing distance into pixels (applying equation (5)):

distance_pixel_Y= distance_northing * pixels_per_northing

= 20000 * 0.0128

= 256 pixels

So the location's vertical pixel coordinate is (applying equation (6)):

location_pixel_Y = origin_pixel_Y - distance_pixel_Y

= 512 pixels – 256 pixels

= 256 pixels

Therefore, at resolution level "4", the corresponding coordinates in the pixel space of the given location (285000 E, 6350000 N) – the centre of the Easting/Northing range covered in the map is (256 pixels, 256 pixels) – also the centre of the pixel space that composes the view of the map.

The translation of the Easting/Northing coordinates to pixel coordinates was implemented in the Javascript function "getBitmapCoordinate" in the custom map type class.

Following is the source code of the function "getBitmapCoordinate":

```
CS.prototype.getBitmapCoordinate=function(a,b,c,d){

1.     if(!d){d=new v(0,0)}
2.     d.x=Math.floor(
       this.bitmapOrigo[c].x + (b-2830000)*this.pixelsPerEasting[c]
       );

3.     d.y=Math.floor(
       this.bitmapOrigo[c].y -
       (a-6330000)*this.pixelsPerNorthing[c]
       );
4.     return d
};
```

The function takes 4 parameters:

“a” – the given location’s Northing value.
“b” – the given location’s Easting value.
“c” – the current resolution level of the map.
“d” – an object of the “GPoint” class, which is used to hold a two-dimensional point.

The translation of the given location’s horizontal coordinate is defined at line 2; at line 3, the translation of the location’s vertical coordinate is defined.

The reverse-function of function “getBitmapCoordinate” translates a given location’s pixel coordinates to its coordinates in the Easting/Northing space. In each of the other map type classes, the method that responsible for translating the pixel coordinates to latitude/longitude was named “getLatLng()”, the method was also named “getLatLng()” to follow the map type interface specification:

```
CS.prototype.getLatLng=function(a, b, c, d){

1. if(!d){d=new v(0, 0)}

2. d.x=(a-this.bitmapOrigo[c].x) / this.pixelsPerEasting[c]
        +2830000;

3. d.y=(this.bitmapOrigo[c].y-b) / this.pixelsPerNorthing[c]
        +6330000;

4. return d
};
```

The function takes the current resolution level, the given location’s pixel coordinates and a “GPoint” object as parameters. The translation of the location’s horizontal pixel coordinate to its Easting value is defined at line 2; the translation of the location’s vertical pixel coordinate to its Northing value is defined at line 3.

6.4 Adding the Custom Map to the API Library

Having set up the image tiling system and the coordinate system, the custom map type class was almost completed. However, before adding this custom map to the map interface, the map type had to be assigned a link name which was to show a button on the map interface. Clicking on the button, the custom map would be switched on.

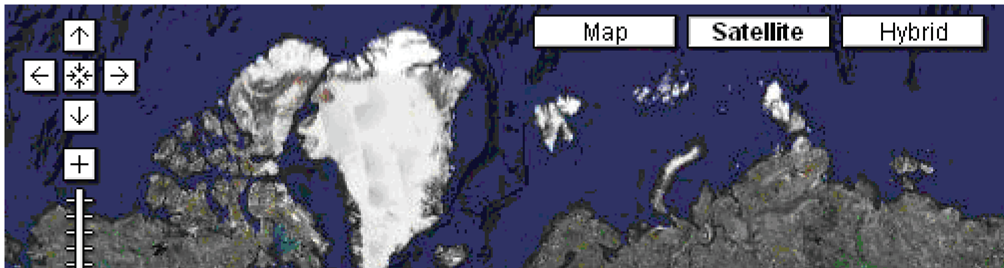

Figure 6.5 map switch buttons on the Google Maps interface

As we can see from the snapshot (figure 6.5)shown above, there are currently three buttons on the map with text on them linking to the three originally defined map types in the Google Maps API library ("Map"– map type "GGoogleMapMercSpec"; "Satelite" – map type "GKeyholeMapMercSpec"; "Hybrid" – map type "GHybridMapSpec"). Users can switch between the three map types by clicking on these three link buttons.

The custom map type was given a link name "CsMap". Two variables were declared in the Javascript file "first.js" (the first Javascript file obtained from the Google site used to set up our experiment environment) stored on the server to reference this link name:

```
.
.
var _mCSMap = 'CsMap';
var _mCSMapShort = 'CsMap';
.
.
```

Also, in the same way that the other three map types declare their link names, two Javascript functions were defined in the class definition of the custom map type:

```
.
.
CS.prototype.getLinkText=function(){
      return _mCSMap
};

CS.prototype.getShortLinkText=function(){
      return _mCSMapShort
};
.
.
.
```

Now the custom map type was ready to be added in the Google Maps API library. Recall the process of locating the definition of the Google Maps API classes in the source file "maps.30a.js". Links between the names used for the classes in the actual coding and the names that appear in the API library were found in the body of function "Rb()":

```
.
function Rb(a){
.
.
b.GGoogleMapMercSpec=C;
b.GKeyholeMapMercSpec=M;
b.GHybridMapSpec=Q;
.
.
}
```

The partial source code shown in the function body maps the names of three originally defined map types that appear in the API to the names of the three classes that are actually used in the code.

In the same way, our custom map type class "GCsMapMercSpec" which was defined under the name of class "CS" was registered to the API library by adding a line to the function "Rb()":

```
.
function Rb(a){
.
.
b.GGoogleMapMercSpec=C;
b.GKeyholeMapMercSpec=M;
b.GHybridMapSpec=Q;
.
b.GCsMapMercSpec=CS; // add in the custom map type
.
.
}
```

As mentioned before, objects of the three originally defined map types are created, and then pushed into an array which is held by the "GMap" object as its data member. The "GMap" objects then works as a framework holding and manipulating those three map type objects to perform all sorts of functions (switching between the map types, navigation on the map, zooming functions …). Therefore, all it takes to add in the custom map is to create an object of the custom map type, and then push it into the array which holds the map type objects. This part of the work was done by adding extra statements in the function "createMapSpecs()" in the Javascript file "first.js".

Following is the modified source code in function "createMapSpecs()":

```
function createMapSpecs() {
.
.
.
1. if (!arguments.callee.mapSpecs) {
2.     var mapSpecs = [];
       .
       .
       .
3.     _GOOGLE_MAP_TYPE = new GGoogleMapMercSpec (mt, mtd,
       tileVersion, mapCopy, lrTileVersion);

4.     mapSpecs.push(_GOOGLE_MAP_TYPE);
```

```
5. if (!kdisable) {
6.     _SATELLITE_TYPE = new GKeyholeMapMercSpec (
             kmt, kmtd, kdomain, ktv,khauth, kjapandatumhack
                         );

7.     mapSpecs.push(_SATELLITE_TYPE);

8.     if (hybrid) {
9.           _HYBRID_TYPE = new GHybridMapSpec (kmt, kmtd, kdomain,
             ktv,  khauth,kjapandatumhack,  hmt,  hmtd,hTileVersion,
             hybridCopy,lrHTileVersion
                               );

10.          mapSpecs.push(_HYBRID_TYPE);
                   }

   11.    _CS_MAP_TYPE = new GCsMapMercSpec (
                   mt, mtd, tileVersion, mapCopy, lrTileVersion
                         );

12.    mapSpecs.push(_CS_MAP_TYPE);
             }

13. arguments.callee.mapSpecs = mapSpecs;
      }
14. return arguments.callee.mapSpecs;
}
```

At line 2, an array named “mapSpecs” is declared in order to contain the map type objects. An object of class “GGoogleMapMercSpec” (the street map type) is created at line 3 and then pushed into the array “mapSpecs” at line 4. An object of class “GKeyholeMapMercSpec” (the satellite view map type) is created at line 6 and then pushed into the array at line 7. An object of class “GHybridMapSpec”(the hybrid map type) is created at line 9 and then pushed into the array at line 10.

An object of our custom map type “GCsMapMercSpec” is created at line 11, and then pushed in the container array at line 12.

The snapshot shown below gives the map interface with our custom map type added (the fourth button links to the custom map):

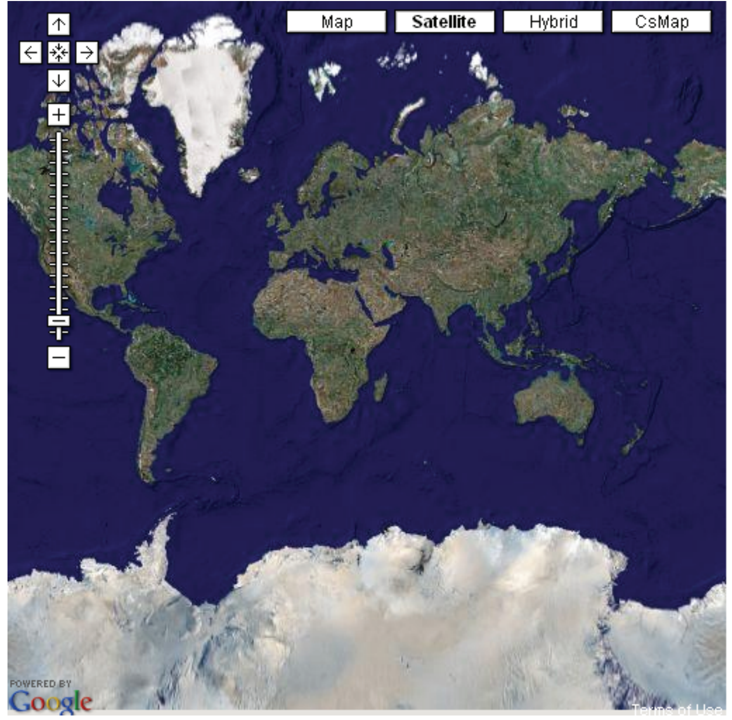

Figure 6.6 adapted Google Maps interface with switch button for our custom map

Clicking on the link button “CsMap” switches the view to our custom map:

Figure 6.7 the custom map

6.5 The Agricultural Data

As mentioned before, this thesis project was divided into two tasks. First task was to build an online map application using Google Maps API. The second task was to graphically display some agricultural data on the application The first task was completed by creating a custom map on Google Maps. The second task was achieved by drawing graphic overlays on the map.

The agricultural data used for the project contains the records of a set of agricultural features sampled at a number of geographical locations in New Zealand. A record in the dataset is described by values of a list of data attributes. Five of the data attributes were chosen for graphical presentation. These 5 attributes are:

- "Xcoord" – a numeric attribute, which indicates a sample point's horizontal coordinate – specified in Easting values.

- "Ycoord" – a numeric attribute, which indicates a sample point's vertical coordinate – specified in Northing values.

- "Variety" – a binary type attribute, describing one of the agricultural features, which has two possible values: "HW" or "GK".

- "GrowthMethod" – also a binary type attribute, describing one of the agricultural features, which has two possible values: "CK" or "OB".

- "DryMatter" – a numeric type attribute, with values range from 10% to 20%.

A mySQL database was built and maintained on the server side to store the agricultural data. The database had only one table named "dryMatter", the five data attributes were made into 5 table fields.

The following table (table 6.2) shows a single record in the table “dryMatter”:

id	easting	northing	variety	growthMethod	dryMatter
456	2839184	6351135	HW	CK	16.215560 %

Table 6.2 a single record in the agricultural database

In the implementation, overlays in different shapes (trangle, circular) filled in different colours were put on the map to present the data records. For example, the attribute “variety” is of binary data type. The agricultural records are divided into two subsets by the two values of this attribute. Overlays in two different shapes (triangle, circular) can be used to distinguish the subsets of records. For the numeric attribute “dryMatter”, a threshold value can be set by the user input to divide the data into two groups. In which case, coloured circular shapes can be drawn on the map to indicate the sample spots with attribute values greater than the threshold.

The combination of the values of the first two data fields (“easting” and “northing”) of a data record was used to determine the location on the map at which an overlay should be placed.

6.5.1 Adding Circular/Triangle Overlays on the Map

Polygon (triangle)/circular shaped overlays are not supported in the Google Maps API library. Searching on the internet, an extension to the Google Maps overlay library was found. The API extension is called “XMaps”, created by Chris Smoak. The extension library includes a set of new features. By using this extension library, some new sorts of overlays (e.g. polygons, circles, arrowed lines) can be created and added to a map. The extension library Javscript source file “xmaps.1c.js” can be obtained from the web address http://xmaps.busmonster.com [28].

In order to use the extension library, a link to the library source file needs to be added to the <script> section in the map user interface HTML file:

```
<script src="xmaps.1c.js" type="text/javascript"></script>
```

A circle, or other kinds of polygons can be created by using the polygon drawing method “createRegularPolygonFromRadius” in the API extension library. The method takes 6 parameters. These parameters are:

- The coordinates of the centre of the circle/, in a form of a “GPoint” (a two-dimensional point class) object.

- The radius of the circle, specified in one of various unit systems defined in the extension library.

- The number of points, specifies how many points are used together to approximate a circular shape. Usually a number larger than 30 gives a good circle shape. When set as 3, a triangle shape is created.

- The starting angle, at which angle from the centre the first point should be placed, usually set as 0.

- The overlay’s outline style- set as “null” in the actual implementation as the overlays are not in an outlined style.

- The overlay’s fill style - specifies the weight, opacity and colour of the overlay.

For example, the following Javascript statements put a circle at the centre of our custom map (2850000E, 6330000N):

```
var centre = new GPoint (2850000, 6330000);
var radius = new XDistance(1, XDistance.KM);
var numPoints = 36;
var startAngle = 0;
```

```
 var outlineStyle = null;
 var fillStyle = {color: "fuchsia", weight: 1, opacity : 0.8};
 var circle = new XPolygon.createRegularPolygonFromRadius(
                    centre, radius, numPoitns, startAngle, outlineStyle, fillStyle
                    );
map.addOverlay (circle);
```

The following (figure 6.) shows the map with the filled circle added at the map centre:

Figure 6.8 map with a filled circular overlay added on

6.5.2 Fetching Data in the AJAX Fashion

The database that contains the agricultural data is maintained on the server side. Users query the data from the database by actions on the user interface (selecting the agricultural attribute, determining the attribute threshold). The data is requested in the AJAX fashion, by which the whole web page (user interface)

does not need to be submitted and reloaded so that the user experience with the application does not get interrupted. We will explain the process by walking through a use case on the application. The user interface of the application looks as shown below:

Figure 6.9 a dropdown menu on the map interface from which different attribute options can be selected

There is a dropdown menu is placed at the top of the user interface, from which user can select between the agricultural attributes. As seen in the snapshot, there are 5 options in the dropdown menu:

- Option "variety" – presents the binary attribute "variety" by drawing two different kinds of overlays on the map to indicate locations with two different attribute values ("HW" and "GK").

- Option “variety – “HW” ” – indicates the locations with value “HW” for the attribute “variety”.

- Option “variety – “GK” ” – indicates the locations with value “GK’ for the attribute “variety”.

- A similarly structured set of options, “growth method”, “growth method – “CK” ” and “growth method – “OB” ” were assigned for presenting the attribute “growth method”.

- Option “dry matter” – presents the numeric attribute “dry matter” of the dataset. When chosen, a slide bar shows on the interface. The user can use the slide bar to set a threshold for the attribute, locations with values greater than the threshold for the attribute will be indicated on the map.

Figure 6.10 shows the option “dry matter” selected, and the slider bar at the bottom of the web page.

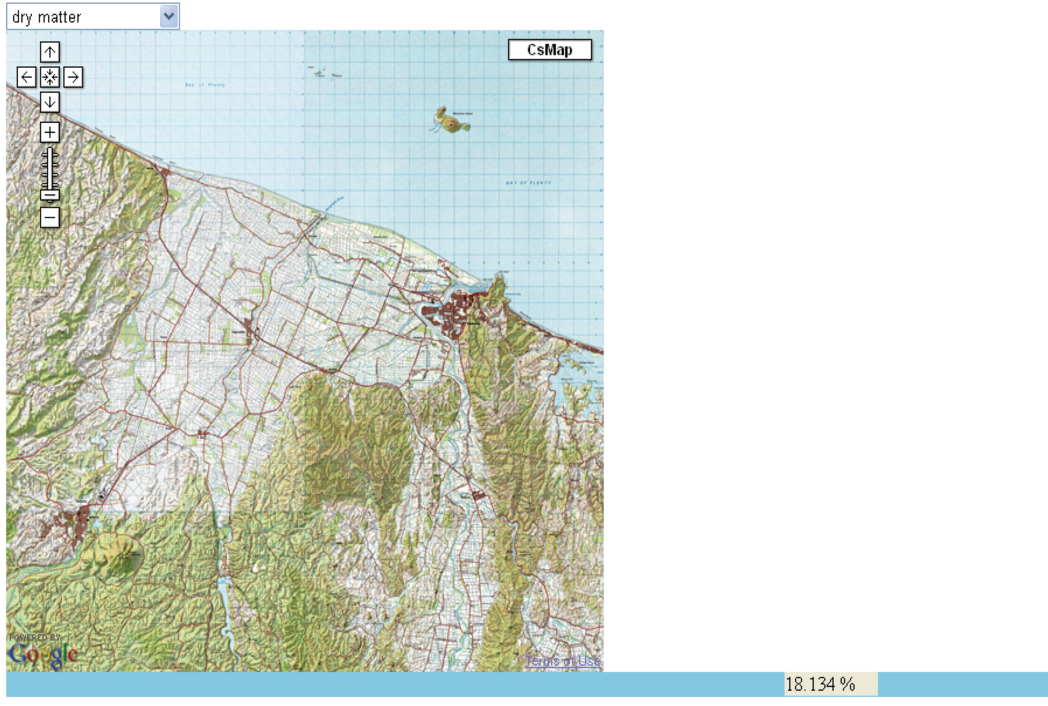

Figure 6.10 the slider bar control shows when option “dryMatter” is selected

Users can set the threshold value for the attribute "dry matter" by adjusting the slider bar. From the snapshot, we can see that the threshold is to be set as 18.134% as shown on the slide bar handle.

Recall in the AJAX example explained in chapter 4, the key element in an AJAX application is the XMLHttpRequest object. In the implementation, an XMLHttpRequest object is created by the following Javascript statement using the "GXmlHttp" class in the Google Maps API library:

```
var xmlHttp = GXmlHttp.create();
```

Instead of submitting the whole web page to the destination PHP file, the threshold value set by the user action is sent to the PHP file ("PHP.php") maintained on the server side. In this case, the destination URL consists of three parts: the URL of the PHP file, the value of the user data, a randomly generated session id.

```
var url = "PHP.php?q=18.134&sid="+Math.random();
```

The request is sent to the server by the following Javascript statement using the method "open" of the XMLHttpRequest object:

```
xmlHttp.open("GET", url, true);
```

On the server side, the instructions in the PHP file establish a connection to the database which contains the agricultural data. The PHP file gets the threshold value from the destination URL then uses the threshold value to generate a SQL query against the database:

```
SELECT * FROM dryMatter where dryMattter >= 18.134;
```

The queried data is written in the form of an XML file in the PHP code, then returned to the client side:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<markers>
.
.
<marker easting="2844464" northing="6346119" variety="HW"
growthMethod="CK" dryMatter="18.359633" />

<marker easting="2841188" northing="6359835" variety="HW"
growthMethod="CK" dryMatter="18.437424" />

<marker easting="2847419" northing="6354311" variety="GK"
growthMethod="CK" dryMatter="19.701045" />
.
.
</markers>
```

Back at the client side, the property “readyState” of the XMLHttpRequest is used to check if the request has been completed (completed when the readyState equals “4”). Once the request is completed, the property “responseXML” of the XMLHttpRequest object is used to receive the XML file:

```
var xmlDoc = xmlHttp.responseXML;
```

The following Javascript statements strip off the XML package and get the values of the five data fields for each record in the queried data:

```
var markers=xmlDoc.documentElement.getElementsByTagName("marker");
for(var i=0; i<markers.length; i++){

  var easting = parseFloat(markers[i].getAttribute("easting"));
  var northing = parseFloat(markers[i].getAttribute("northing"));
  var variety = markers[i].getAttribute("variety");
  var growthMethod = markers[i].getAttribute("growthMethod");
  var dryMatter =parseFloat(markers[i].getAttribute("dryMatter"));

}
```

What we actually need in this case is the Easting/Northing coordinates in the queried data records. The filled circles are therefore drawn on the map based on the geographical coordinates in the queried data to indicate the locations with greater "dry matter" values than 18.134%. Figure 6 11 shows these locations indicated by filled circles:

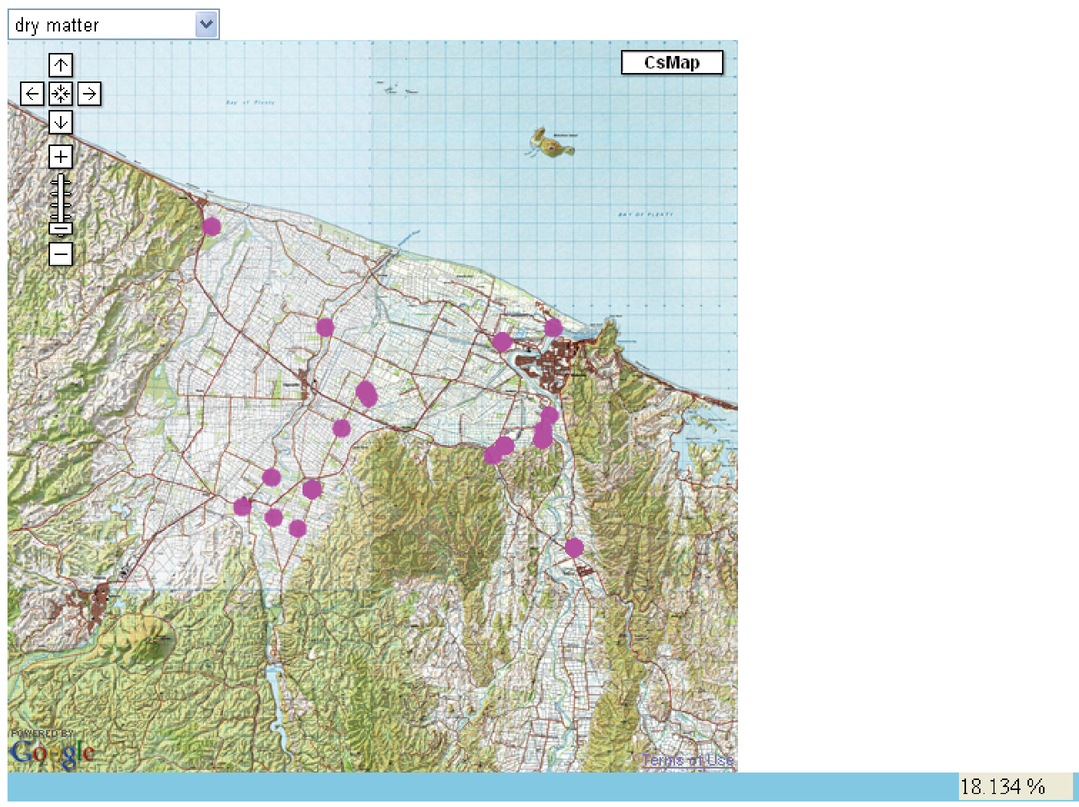

Figure 6.11 locations with values greater than 18.314% for attribute "dry matter"

Figure 6.12 shows the locations with values greater than 16.622% in attribute "dry matter":

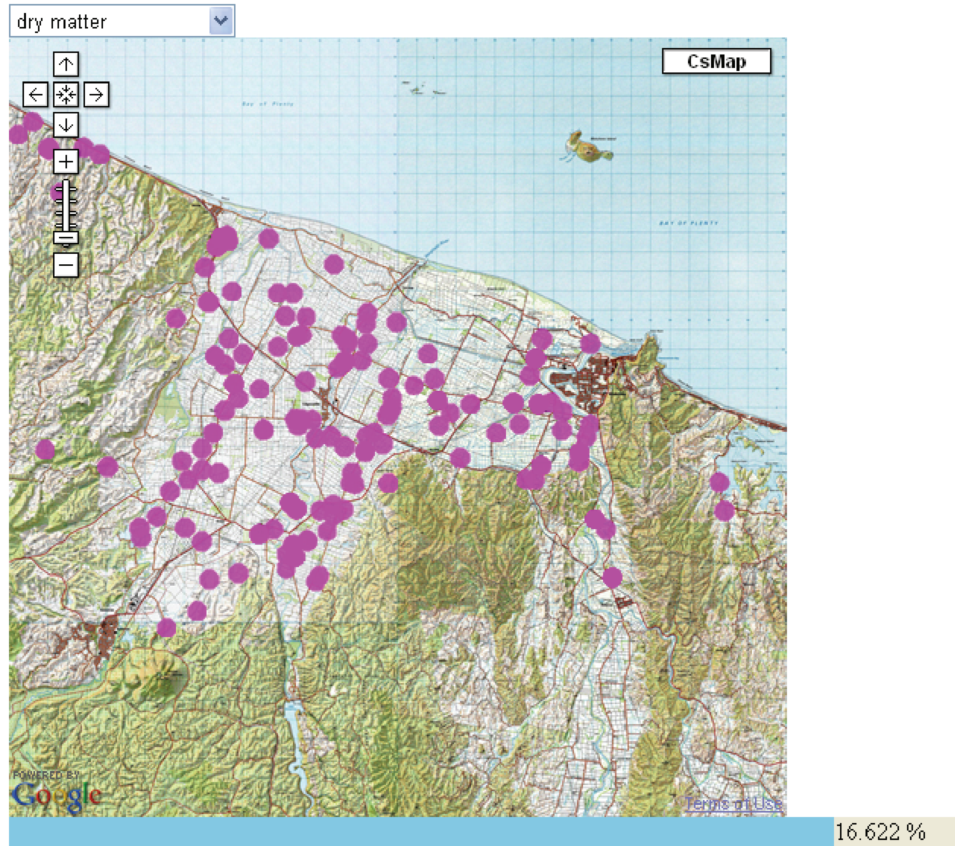

Figure 6.12 map showing locations with values greater than 16.622% for attribute "dry matter"

As seen in the figure 6.12, a smaller threshold value for the attribute "dry matter" leads to more locations being shown on the map.

For the presentation of the other attributes ("variety" and "growth method"), data is queried from the database on the server side and displayed on the map following the same process.

Figure 6.13 shows the locations with the value "HW" in attribute "variety":

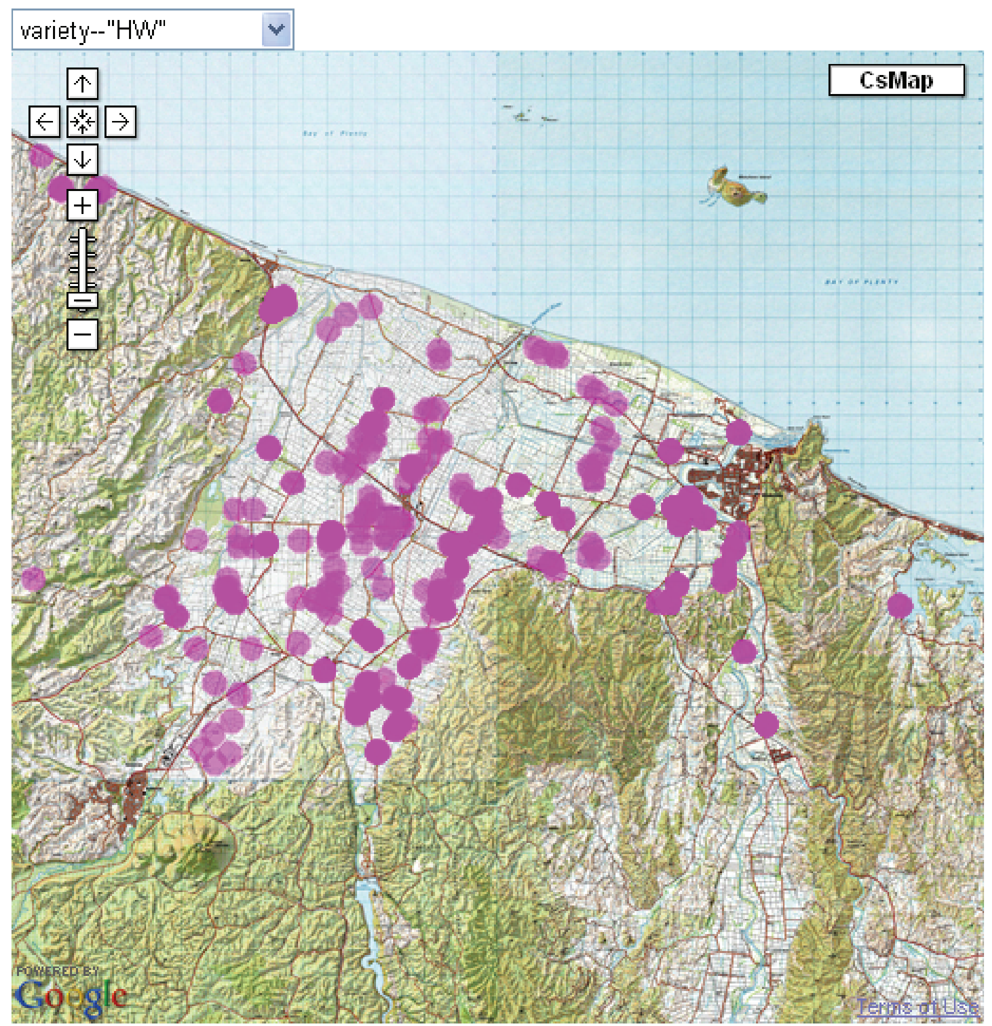

Figure 6.13 map showing locations with the value “HW” for attribute “variety”

Some of the locations shown in figure 6.13 are very close to each other. The circles indicating these locations are mostly overlapped. Because the overlays are translucent, these overlapped circles are making dark spots on the map (more overlays overlapped, the darker the spot becomes). In this way, the distribution of the locations is reasonably presented.

Locations with value “GK” for the attribute “variety” (locations are indicated by triangle shaped overlays):

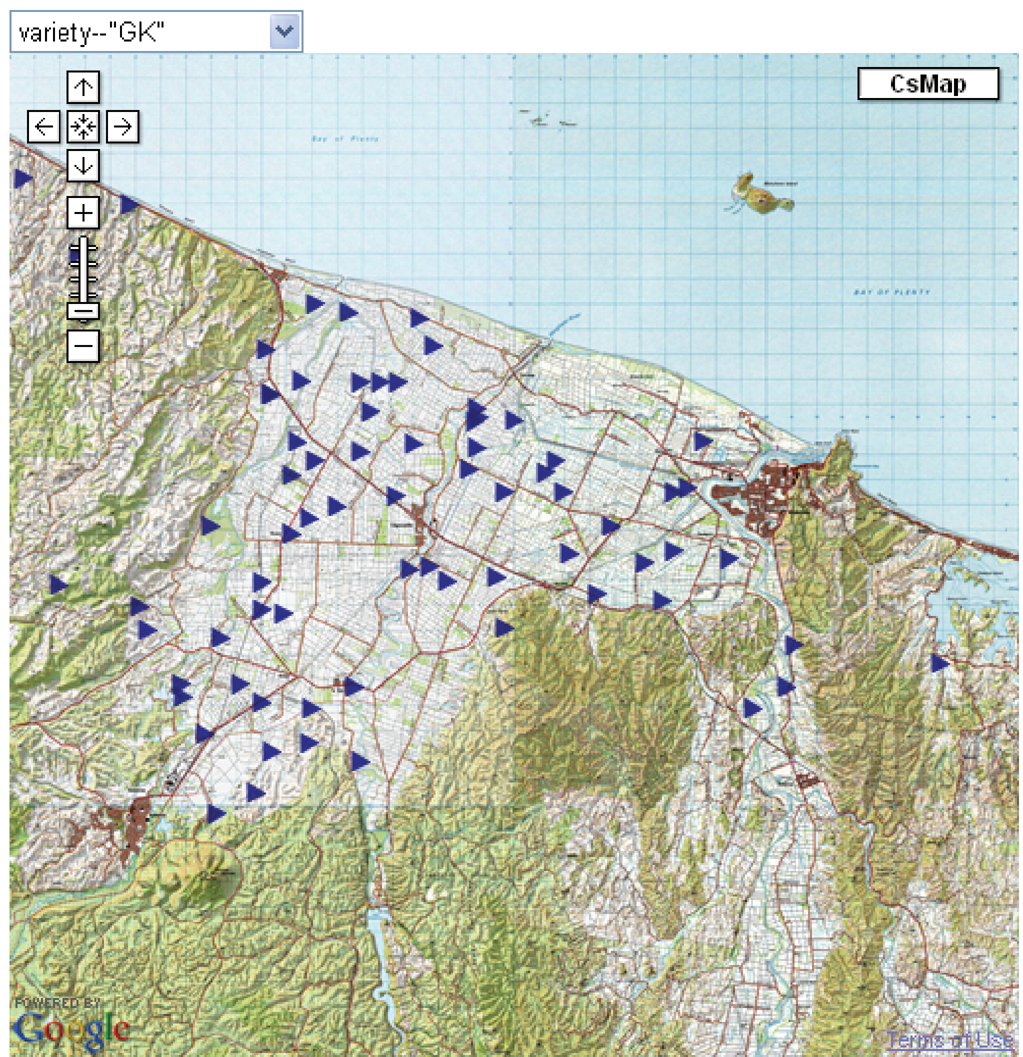

Figure 6.14 map showing locations with value “GK” for the attribute “variety”

Selecting the option “variety” shows locations with either attribute values, which gives a comparative visual effect (locations with attribute value “HW” are indicated by circular shaped overlays, locations with the attribute value “GK” are indicated by triangle shaped overlays, see figure 6.15 below):

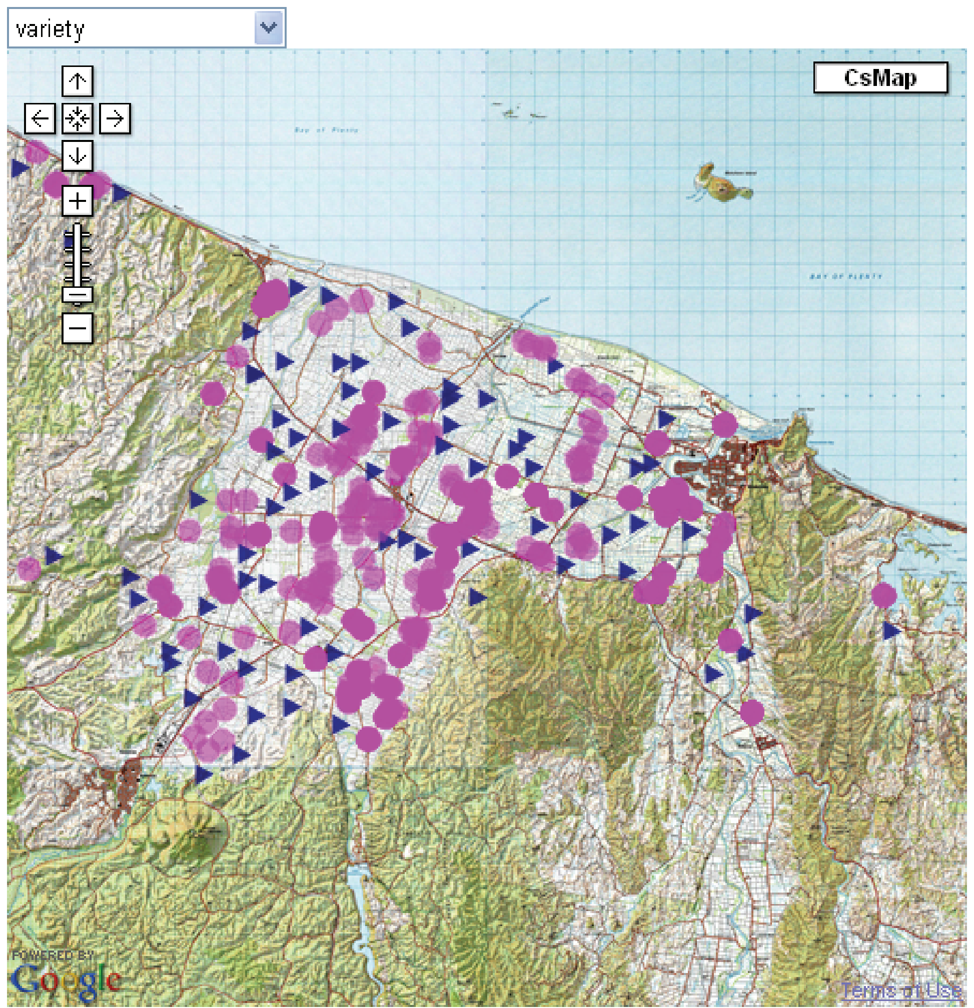

Figure 6.15 map showing locations with either values for attribute "variety"

To sum up, the implementation of the whole system consists of three parts:

- The online map application - serving as a painting plate to present agricultural data, built by creating a custom map using the Google Maps API.

- The mySQL database – maintained on the server side, storing agricultural data.

- PHP files – maintained and running on the server side, responsible for the connection and queries to the agricultural database, and packaging the queried data from the database in XML.

Chapter 7 CONCLUSION

Chapter 1 and chapter 2 described the thesis project task of building a web based GIS system for geospatially displaying agricultural data. The system was orientated to serving people living in rural areas. The task was divided into two parts. One was to build a base map for the GIS system using Google Maps. The other was to build a database to store the agricultural data which was to be displayed on the maps. For three reasons, the base map could not be built directly using the web mapping service that Google Maps offered:

- Google Maps did not provide high enough resolution imagery for the agricultural area where the data was collected.
- The coordinate system used to specify the data sample locations in the agricultural data was not supported by the API of Google Maps.
- Using Google Maps requires high speed Internet connection. As mentioned before, our application was to serve people from rural areas in New Zealand. However, high speed Internet connection is usually not available in rural areas. Therefore, there was a network issue.

The approach to the base map construction was to create a custom map type in Google maps by adapting the map image serving system and the coordinate system originally used in Google Maps. This part of the task required a detailed analysis and understanding of the Google Maps Javascript library source code and then modification to it, which took considerable time and effort.

Chapter 3 talked about the advantages of the web based applications. The main technologies involved in building web based applications were described with examples.

Chapter 4 described the two models in web based applications, the classic model (synchronous data transfer model) and the AJAX model (asynchronous data

transfer model). Because Google Maps functions using the AJAX model, the discussion then focused on AJAX. A simple AJAX application example was presented and discussed in the chapter as well.

Chapter 5 narrated the process of obtaining the Javascript source code, defining Google Maps API classes, and the process of setting up our experimental environment with this Javascript source code. Based on the Javascript source code, analysis of the image tiling system and the coordinate system applied in Google Maps was described in detail in this chapter.

The process of actual implementation of the system was described in chapter 6. The system base map was constructed by inserting a new map type class which used its own image data and defined its own coordinate system as an extension to the Google Maps API library. A mySQL database was built on the server side to maintain agricultural data. The agricultural data was stored, ready to be fetched from the database and sent to the client side in the AJAX fashion. Also, some testing results were shown in the chapter.

The custom map implemented by adapting Google Maps Javascript library performs well, as expected. Because all the imagery data for making up the map is stored on the end users' local file systems, only the agricultural data needs to be transferred through the network. The agricultural data is in relatively small amounts, which do not require a high speed network to transfer. In this way, the network issue is efficiently avoided. The system is practically usable for people from rural areas who do not have a high speed Internet connection. With the custom overlays developed, the geospatial agricultural data is clearly presented on the map. However, when there are a large number of instances in the agricultural dataset to be displayed (more overlays to be drawn on the map), the system functions with poor response. For example, it takes up to 10 seconds to draw 500 overlays on the map. This delay is not caused by the network. It is a result of poor graphic performance in the Google Maps system.

7.1 Suggestions for Further Work

The system could be extended to present more analysis. A bigger variety of custom graphical indicators could be created and applied for data presentation. For example, if there was sufficient geographical reference supplied in the agricultural data (for example, geographical information of boundaries), polygons could be used to used on the map to give a more detailed and vivid data presentation.

7.2 Final Remarks

The project has demonstrated how to adapt the web mapping service Google Maps to create a custom map, and on top of which a web based GIS system can be built.

Follow the same strategy, various kinds of imagery data and coordinate systems can be applied to create various custom maps with Google Maps for different kind of uses. The thesis carefully documents all aspects of the system, and should serve to assist others in making such extensions.

The analysis and implementation is based on the version 1 of Google Maps Javascript API source code (maps.30a.js).

References

[1] Adams, C. (2005). *AJAX: Usable Interactivity with Remote Scripting*.
Retrieved March 2006, from:
http://www.sitepoint.com/article/remote-scripting-ajax

[2] Ajax Patterns (2006). *XMLHttpRequest Call*.
Retrieved March 2006, from:
http://ajaxpatterns.org/XMLHttpRequest_Call
#Creating_XMLHttpRequest_Objects

[3] Ajax Patterns (2006). *Google Maps.*
Retrieved March 2006, from:
http://ajaxpatterns.org/Google_Maps

[4] Arvidsson, E. & Eklind, E. (2002). *WebFX DHtml Demos*.
Retrieved April 2007, from:
http://www.siteexperts.com/tips/webfx/page1.asp

[5] Canter, S. (2004). *Understanding Client - Side Scripting.*
Retrieved October 2006, from:
http://www.pcmag.com/article2/0,1759,1564972,00.asp

[6] Chicago crime map (n.d.). *Chicago Police Department.*
Retrieved June 2007, from:
http://www.chicagocrime.org/map/

[7] Delaney, J. (1999). *Geographical Information Systems An Introduction.*
Oxford University Press.

[8] Developer Connection (n.d.). *Dynamic HTML and XML:*
The XMLHttpRequest Object.
Retrieved September 2006, from:
http://developer.apple.com/internet/webcontent/xmlhttpreq.html

[9] Duffy, S. (2003). *How to do everything with JavaScript.*
McGraw-Hill/Osborne

[10] Dunck, J. (2006). *GoogleMapsHacking*.
Retrieved April 2006, from:
http://dunck.us/collab/GoogleMapsHacking

[11] Dunck, J. (2005). *Simple Analysis of Google Map and Satellite Tiles.*
Retrieved December 2005, from:
http://dunck.us/collab/Simple_20Analysis_20of_20Google_20Map_20and
_20Satellite_20Tiles

[12] Eernisse, M. (2006). *Build Your Own Ajax Web Applications.*
SitePoint Pty.Ltd.

[13] Garrett, J. J. (2005, February). *Ajax: A New Approach to Web Applications*.
Retrieved December 2005, from:
http://www.adaptivepath.com/publications/essays/archives/000385.php

[14] Gilmore, J. (n.d.). *Integrating Google Maps in to Your Web Applications.*
Retrieved March 2006, from:
http://www.developer.com/tech/article.php/10923_3528381_1

[15] Google (2005). *Concepts and Examples*.
Retrieved October 2005, from:
http://www.google.com/apis/maps/documentation

[16] Gottipati, H, (2005). *Hacking Maps with Google Maps API*.
Retrieved March 2006, from:
http://www.xml.com/pub/a/2005/08/10/google-maps.html

[17] Hearnshaw, H. M., & Unwin, D. J. (1994). *Visualization In Geographical Information Systems.* The Association for Geographic Information. / Wiley

[18] Langridge, S. (2005). *DHTML Utopia: Modern Web Design Using JavaScript&DOM.* SitePoint Pty.Ltd.

[19] Mapki (2006). *Custom Map in Version 1.*
Retrieved July 2006, from:
http://mapki.com/wiki/Custom_Map_in_Version_1

[20] McLaughlin, B. (2005). *Mastering Ajax, Part 1: Introduction to Ajax.*
Retrieved May 2006, from:
http://www-128.ibm.com/developerworks/web/library/wa-ajaxintro1.html

[21] Podnar, H., Gschwender, A., Workman, R., and Chan, J. (2006). *Geospatial visualization of student population using Google™ Maps*. J. Computing Small Coll. 21, 6 (Jun. 2006), 175-181.

[22] Pimpler, E. (2006). *Introduction to Developing with Google Maps.*
Retrieved March 2007, from:
http://www.directionsmag.com/article.php?article_id=2120&trv=1

[23] Ralston, B. (2004). *GIS and Public Data.* Canada: Delmar Learning.

[24] Savage, C. (2006, May). *Google Maps Deconstructed.*
Retrieved May 2006, from:
http://cfis.savagexi.com/articles/2006/05/03/google-maps-deconstructed

[25] Savage, C. (2006, July). *Mouse Coordinates To Lat/Long.*
Retrieved July 2006, from World Wide Web:
http://cfis.savagexi.com/articles/2006/06/30/mouse-coordinates-to-lat-long

[26] Savage, C. (2006, May). *Coordinate Systems – Putting Everything Together.* Retrieved May 2006, from World Wide Web: http://cfis.savagexi.com/articles/2006/05/02/coordinate-systems-putting-everything-together

[27] Savage, C. (2006, April). *Geodetic Coordinate Systems.* Retrieved May 2006, from: http://cfis.savagexi.com/articles/2006/04/29/geodetic-coordinate-systems

[28] Smoak, C. (2005). *XMaps Library – A Google Maps API Extension*. Retrieved December 2005, from: http://xmaps.busmonster.com/

[29] Stefanov, S. (2005). *Take Command with AJAX*. Retrieved March 2006, from: http://www.sitepoint.com/article/take-command-ajax

[30] Udell, J. (2005). *Google Maps pushes the envelope*. Retrieved December 2005, from: http://www.infoworld.com/article/05/02/18/08OPstrategic_1.html

[31] Weather Bonk – Live Weather, Forecasts, WebCams, and more on a Google Map (2006). Retrieved June 2007, from: http://www.weatherbonk.com/

[32] Webber, J. (2005). *Mapping Google.* Retrieved December 2005, from: http://jgwebber.blogspot.com/2005/02/mapping-google.html

[33] Wikipedia, The Free Encyclopedia. (2005). *AJAX (programming).* Retrieved December 2005, from: http://en.wikipedia.org/wiki/AJAX

[34] Wikipedia, The Free Encyclopedia (2005). *Geographic information system.* Retrieved December 2005, from: http://en.wikipedia.org/wiki/Geographic_information_system

[35] Wikipedia, The Free Encyclopedia (2005). *Google Maps.* Retrieved December 2005, from: http://en.wikipedia.org/wiki/Google_maps

[36] Wikipedia, The Free Encyclopedia (2006). *Map projection*. Retrieved July 2006, from: http://en.wikipedia.org/wiki/Map_projection

[37] Wikipedia, The Free Encyclopedia (2006). *Mercator projection*. Retrieved July 2006, from: http://en.wikipedia.org/wiki/Mercator_projection

[38] Wikipedia, The Free Encyclopedia (2007). *Client-side scripting.* Retrieved April 2007, from: http://en.wikipedia.org/wiki/Client-side_scripting

[39] Wikipedia, The Free Encyclopedia (2007). *Servert-side scripting*. Retrieved April 2007, from: http://en.wikipedia.org/wiki/Server-side_scripting

[40] Wikipedia, The Free Encyclopedia (2007). *Web application.* Retrieved April 2007, from: http://en.wikipedia.org/wiki/Web_application

[41] Wikipedia, The Free Encyclopedia (2007). *Document Object Model.* Retrieved April 2007, from: http://en.wikipedia.org/wiki/Document_Object_Model

[42] Wikipedia, The Free Encyclopedia (2007). *XMLHttpRequest*.
Retrieved April 2007, from:
http://en.wikipedia.org/wiki/XMLHttpRequest

[43] Worboys, M. F. (1995). *GIS-A Computing Perspective.* Taylor & Francis

[44] W3C Working Draft (2000). *Document Object Model (DOM) Level 1 Specification (Second Edition)*. Retrieved May 2007, from:
http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929/

[45] W3C Recommendation (2000). *Document Object Model (DOM) Level 2 Core Specification*. Retrieved May 2007, from:
http://www.w3.org/TR/DOM-Level-2-Core/

[46] W3C Recommendation(2000). *Document Object Model (DOM) Level 3 Core Specification*. Retrieved May 2007, from:
http://www.w3.org/TR/DOM-Level-3-Core/

[47] W3Schools (n.d.). *The XMLHttpRequest Object.*
Retrieved September 2006, from:
http://www.w3schools.com/xml/xml_http.asp