UNIVERSITY OF WAIKATO

594 Dissertation

Development of a map service

Student: Xiaotie Huang
Supervisor: Annika Hinze

A DISSERTATION SUBMITTED TO

COMPUTER SCIENCE DEPARTMENT

MARCH 2007

Acknowledgements

I give my sincerest thanks to a number of people without whom this dissertation could not have been completed.

Firstly, to my supervisor, Annika Hinze, for your kind support, interests, precise advice and patience with me through the whole dissertation process. Secondly, to all the members of Information Systems and Databases Group at the Department of Computer Science at the University of Waikato for participating my presentations and giving valuable comments and suggestions. Thirdly, to my friends, Aya, Teresa and David, who have helped me with the stress of carrying out this study, I sincerely appreciate your support and encouragement. Lastly, to my parents, who have supported me in many ways during my stay in New Zealand. I am grateful to you.

Abstract

Geographic Information Systems (GIS) a computerized mapping system for capturing, storing, retrieving, analysing, and displaying spatial data (the type of data that has a geographic location). It is one of the fastest growing high tech fields and has been widely used in my areas where analysis of spatial referenced data is needed.

In this paper we developed and implemented a GIS-based personalized Travel Planning application. Our goal is to provide a map based information system that gives recommendation information (e.g. closeby sights, must-see sights) to travellers according to their destinations, locations and preferences. Furthermore, the TP application was developed as a client-side service under the Tourist Information Provider System. It runs on client machines and uses the spatial referenced data stored at the TIP server-side database.

Contents

Chapter 1 Introduction

There is no doubt that mobiles have become the centre communication tools in many people's lives. It is hard to imagine living in the modern world without having a mobile. Nowadays, mobile manufacturers focus on improving performance and adding exclusive functions to their mobile devices. Recent mobile devices have a fast CPU, massive memory and storage, a lightweight operating system (WinCE, Symbian, etc), a digital camera, wireless internet, Bluetooth, integrated GPS receiver and so on. With these new features, Geographic Information Systems (GIS), which require a lightweight hardware, a positioning system and wireless internet, are able to run on mobiles devices. GIS is a computer-based system that is used to capture, storage, retrieval, analysis and display geographic information. In a more generic sense, GIS is a tool that provides users with a graphical view of geographic information. It allows users to create interactive queries (user created searches), analyzes the geographic information.

The use of GIS on mobile devices is called Mobile GIS. Mobile GIS is a fast growing technology. By the time this paper is written, various applications that use mobile GIS technology will have been published ([11], [12]). However, the most commonly used GIS applications are the Digital Map systems. A Digital Map system is a computer system that displays and links map images and

spatial data (location defined), such as sightseeing places, travel routes and airports. The GIS-based map system can provide the user with a map view based on his/her real location.

The aim of this paper was to develop an application that makes use of mobile GIS and Digital Map technologies to assists tourists in planning their vacations. While there are already a number of travel planning systems, such as travel web sites and airline-specific web sites, in most cases such systems are limited to only specifying means of travel and routes. In this paper, our focus is on location-based recommendations. In a Location-based recommendation tourists are provided with recommendations based on their location. This might include nearby sight-seeing locations, the nearest accommodations and information about a specified geographical location. The geographical location would be either the real-time position of the tourist detected by the GPS receiver or a virtual location on the earth where a user assumes he/she is visiting. Furthermore, our application is developed as part of the Tourist Information Provider System originally proposed by A. Hinze and A. Voisard in [3]. Detailed information about the Tourist Information Provider System will be given later in this section.

The remainder of this section introduces the TIP system and its system architecture (Section 1.1), describes limitations of the TIP system (Section 1.2), and outlines the basic design principles of the Travel Planning application as a functional enhancement to the TIP system (Section 1.3), followed by a general description of the structure of this document (Section 1.4).

1.1 Tourist Information Provider (TIP)

The Tourist Information Provider (TIP) is a location-based mobile tourist information system. TIP was introduced by Hinze and Voisard in [3]. It was designed to provide users with sight-related information based on their current location (location-aware) in combination with the user's profile (pre-defined personal preferences stored in a database). For example, a TIP user can acquire information about sights to see which are relevant to his/her personal interests on the go. The architecture of the TIP system is illustrated on Figure 1: TIP system architecture.

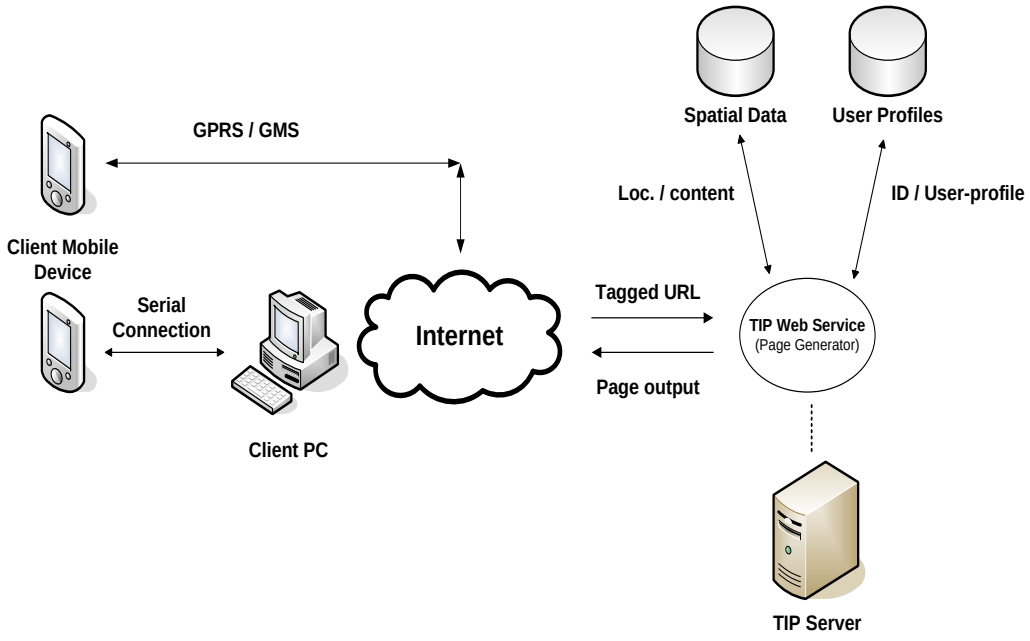


Figure 1: TIP system architecture

The TIP system contains a web service that runs on a web server. It provides users with content (i.e. HTML webpage) that contains travel information (such as accommodation, transport, activities, events, tours, adventure, and recommended sight-seeing destinations) related to their location area and personal interests. In order to connect to the TIP web service, clients must have internet access on their mobile devices through either GPRS or through a serial

connection via a PC. When a client requests information from TIP via a web browser, a URL containing ID tags (e.g. location tag, client_id tag) will be automatically created and sent to the TIP web server. When the TIP web server receives a request from a client, it knows not only the client_id of the requesting client, but also the location of the client at the time the request was sent. Through the client_id, the system can find the profile of the client which specifies his/her personal interests. Through the known location the system can locate the spatial data which is related to the client's location. Therefore, the TIP web service can dynamically generate a user-specified page and output it to the client.

The TIP system contains several information services specially tailored to meet different aspects of tourists' information need; this differentiates it from other mobile information systems. For example, TIP Recommendation Service is a recommendation system which provides dynamic and personalized travel recommendations to the tourists based on analyzing the other tourists' histories and comments [7]. TIP Greenstone Service queries and extracts useful information from the Greenstone [8] digital library and displays the information in the TIP system. (Greenstone is digital library software for building and distributing digital library collections.) More TIP services are described in [1].

1.2 Limitations of TIP

As mentioned in the above section, the TIP system is a location-based tourist information system. It is able to provide sight-related information according to the travellers' interest and position. Although the TIP system is able to deliver many different kinds of information to tourists, as a tourist information system it still has some limitations:

- All information provided to users is in the form of text-based webpage. There are two types of information that would appear in a webpage provide by the TIP system, which are geographic information and non-geographic information. Geographic information refers to any information that identifies a geographic location on a map. For example, physical address and latitude/longitude coordinates. Non-geographic information, on the other hand, refers to any kind of information defining some non-geographical aspects of an object. For example, a name of a sightseeing place and history of a city. There is no problem with displaying non-geographic information on a webpage. However, for geographic information, it would be more appropriate to presents the information visually on a map. For example, for the nearby sights recommendation, the system could show the nearby sights on a map in reference to a user's current position. This is especially helpful to those users who are unfamiliar with the place where they are visiting.

- The system does not support tourists to locate their own positions in unknown areas, to find the position of their destinations, and to get route from where they are to where they want to go. When tourists visit a new place, they often need to orient themselves in that place. Therefore, it would be very helpful if the system could provide tourists with an electronic map of the surround area.

- The system only provide information about an actual trip, which means that TIP users can only obtain the information that is related to their current locations from the TIP database. For example, if a user plans to visit Tokyo, he/she cannot get travel-related information about Tokyo from the TIP system until he/she arrives there. For this reason, the system cannot be used as a travel planning tool for tourists to make actual travel plans.

To overcome the above three limitations of the TIP system, we proposed and implementation a travel planning application as an extension to the TIP system.

1.3 Travel Planning (TP) Application

To overcome the limitations of the current TIP system described in section 1.2, the system need to have an alternative way for delivering travel-related information to TIP users, by providing users with a graphical view of the information stored in the TIP server-side database. Moreover, the system should provide information both for the trip planning phase and the actual trip.

With this goal in mind, we developed a GIS-based application called the Travel Planning (TP) Application as a new service for the TIP system. It offers the possibility of delivering travel-related information (e.g. sights, map, and popular travel routes) to tourists based on their actual moves or planed routes.

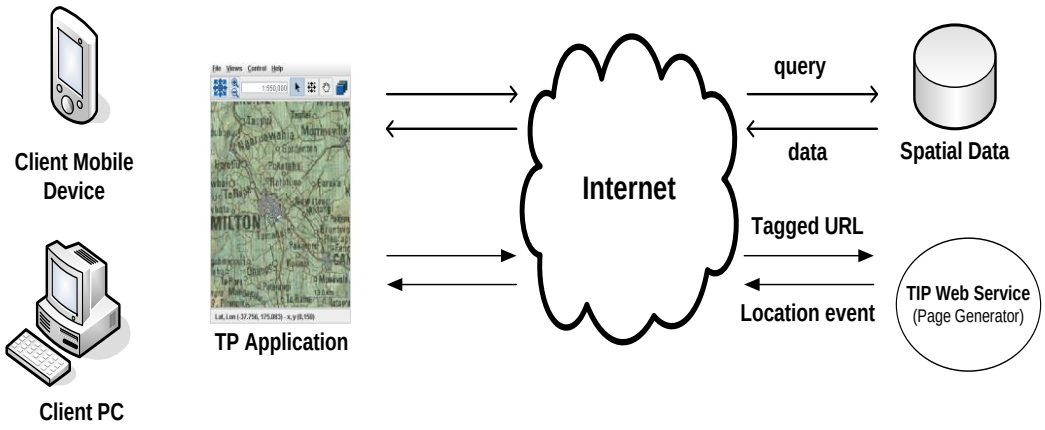


Figure 2: TP application system architecture

The above figure illustrates the system architecture of the TP application. Different from other TIP services, the TP application runs on client-side machines (desktop PC and mobile device). It directly queries the spatial data

stored on the TIP remote database, concerning on geographical locations, and displays the result set to users on an electronic map. Furthermore, the TP application has the ability to cooperate with other services on the TIP system. For example, if a user clicks on a displayed sight on the electronic map, the system will navigate to the recommendation webpage of the corresponding sight. On the other hand, the application also listens to event messages sent from the other services and brings up the map of the specific location specified in the event message.

1.4 Structure of this Report

The rest of the paper is organised into seven chapters. A brief description of each chapter is summarised below:

Chapter 2: **Background**

This chapter presents an overview of the development history of TIP and introduces the system architecture and the existing information services.

Chapter 3: **Requirement Analysis**

This chapter includes an analysis of tourists' travel behaviour, which is followed by a discussion upon the system requirements.

Chapter 4: Related Work

This chapter presents an overview of related work in the area of mobile Geographical Information System and compares them to our proposed system.

Chapter 5: System Design and Implementation

This chapter presents describes the implementation details of our TP application.

Chapter 6: System Evaluation

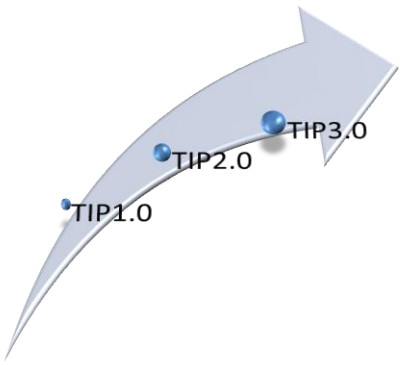
This chapter presents the evaluation of the system and our findings.

Chapter 7 Conclusion & Future Work

This chapter provides general discussion, conclusion and possible extension to the current work.

Chapter 2 Background

The Tourist Information Provider (TIP) system is an advanced mobile tourist information system. It consists of an event notification service (ENS) and a location-based service (LBS). The main purpose of the TIP system is to deliver various kinds of travel-related information to the user according to their location and personal preferences. The TIP system has been through three major versions which include: TIP 1.0, 2.0 and 3.0. TIP 1.0 is the first system prototype. TIP 2.0 is the current stable version. TIP 3.0 is still under development at the time of writing this paper.



This chapter is focused on the implementation detail of TIP. We firstly give an overview of the development history of the TIP system in Section 2.1. Then, we describe in detail the core architecture of TIP in Section 2.2. Finally, a Summary of this chapter is given in Section Summary 2.3.

2.1 Development History of TIP

The concept of the TIP system was first proposed in [3] and implemented as a first prototype at the University of Berlin in [4]. It was designed to deliver sight-

related information to mobile devices such as mobile phones or Personal Digital Assistants (PDA), based on their locations, personal interests, and/or travel routes. Though the system, users can acquire information such as places of interest and recommended sights. We refer to the first prototype of TIP as TIP 1.0.

TIP 1.0 was implemented using client-server architecture, with both the client and server written in Java. A PostgreSQL[1] object-related database was used as a central database accessed via PostgreSQL JDBC interface. The database was extended by PostGIS[2] spatial library in order to store and retrieve geo-spatial data. The core of the system is a filter engine cooperating with a location engine. The filter engine filters information (that is potentially interesting to the user) according the user's profile (which includes personal interests) and the location engine provides the user's location.

Furthermore, in the TIP database, sight objects are stored in different semantic groups, such as museum, historical buildings, art and so on. Each sight belongs to one or more semantic groups. The TIP 1.0 system contains a simple recommendation mechanism which recommends other sights in a semantic group if a user has visited at least two of the sights in the same group [7]. For example, if a user has visited two museums in a certain area, the system will inform the user about other museums in the same area.

The first system prototype was extended by Ottlinger [6]. He restructured the system architecture based on J2EE (Java 2 enterprise edition), using Apache's Struts[3] framework. We refer to this release as TIP 2.0. The new system provides a web-based interface which is accessible from any desktop computer or hand-held device that is internet-connected and has a web browser.

[1] http://www.postgresql.org/
[2] http://postgis.refractions.net/
[3] http://struts.apache.org/

Since the completion of TIP 2.0, TIP has been continually enhanced. A number of new services have been or are being implemented and integrated into TIP 2.0. A TIP service is an add-on application for TIP which adds new features and extras that allow TIP to deliver various types of travel-related information to users. E.g. Advanced recommendation service [7] extends the existing simple recommendation function in TIP 1.0. The service enables TIP to provide recommendations to a user based on other users' information (e.g., preferences, feedback). TIP Greenstone service [8] enables TIP to extract information that is related to a user's travel from Greenstone[4] digital libraries. We refer TIP 2.0 plus the add-on services as TIP 2.5. Basically, TIP 2.5 can be seen as an enhanced multiple-purpose version of TIP 2.0. It is the current stable version of TIP. In this dissertation, the TP application is implemented as one of the add-on services for TIP 2.5. More detailed information about TIP services and TIP 2.0 system architecture are described in next section.

At the time of writing this paper, TIP 3.0 prototype is being developed by Y. Michel [5]. The new architecture of TIP 3.0 is designed to achieve two goals: firstly to optimise the network performance using pre-fetching and caching mechanisms and secondly to extend the service availability which allows the system to connect to multiple information providers as well as peer-to-peer communication. Since our TP application is designed and implemented under TIP 2.0 framework, we will not go into the details of TIP 3.0 in this paper.

[4] http://www.greenstone.org/

2.2 System Architecture of TIP 2.0

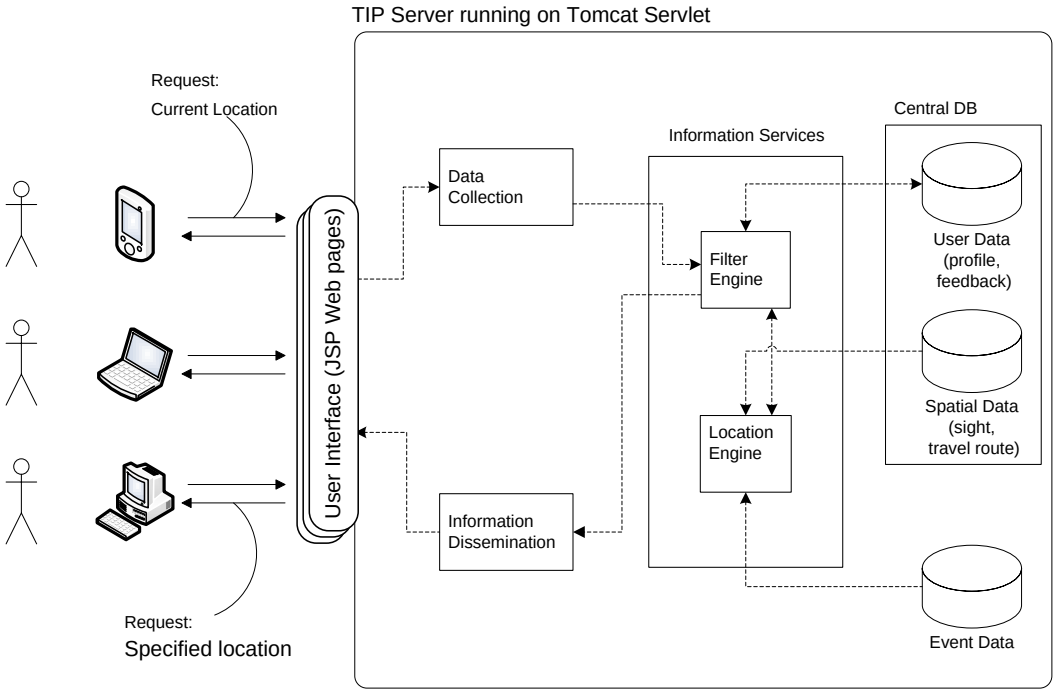


Figure 3: TIP Core Architecture (Adopted and modified from [5])

Figure 3 above illustrates the system architecture of TIP 2.0. The client side can be a desktop computer, or any other type of computer or hand-held device. Internet access and a web browser are required in order to view the JSP web pages generated by the TIP server. A client sends a request which contains location information to the TIP server asking for information about his/her current position. The location can be either the current position as detected by the GPS receiver connecting to the user's device or a virtual location specified by the user when using a desktop computer. The TIP server first parses the request, then it looks for the available information from the database according to the given location and the user's profile, finally, the server generates a dynamic webpage (JSP) containing the located information and sends it back to the client.

The TIP server side is a J2EE web application that runs on a Tomcat Servlet[5] container. It contains the following components:

- **Data Storage:** TIP 2.0 is implemented using a central database approach. The system uses a PostgreSQL object-relational database (version 7.3.4) with a postgis extension (version 0.75) as its central database. There are, in general, three types of data stored in the TIP central database: user data, spatial data, and event data. The user data includes user profiles, feedback, travel history, and user history. The spatial data refers to data items which associated with location information. i.e. sights and travel routes. Additionally, external events from other information providers can be provided by the TIP system. These events are handled similarly to the way that the system handles location events, e.g. weather alerts. These events are stored separately.

- **Location Engine:** The location engine is used to detect the current location of the user and provide the corresponding information from spatial and event data storage to the filter engine or other server-side components. The location engine can be triggered in two ways: by user or by the system. In the first way, the location engine is triggered by user interaction with the system interface. E.g. the user specifies a location (coordinates or name of the location) and presses a button to retrieve information about the location. This is mainly used when a user is accessing the system on a desktop computer. Secondly the location engine can be trigger automatically by the system when certain conditions are met, such as a time interval expires or a user moves more than a certain distance. This requires the client's device to automatically send updates of the current location to the server.

[5] http://tomcat.apache.org/

- **Filter Engine:** The filter engine receives an information requirement message from the client request. In responding to the request, it firstly requests the available location-based information (e.g., places of interest within a certain distance of the client's current position) from the location engine. Then, the retrieved information is filtered by the filter engine according to the client's profile and history before forwarding to the client.

- **Information Service:** In TIP 2.0, an information service is a functional extension to the system. Every service uses data from the TIP central database depending on the purpose of the service. For example, the trust-based recommendation service [9] uses the user's trust ranking level and feedback data to judge if a sight should be recommended to the user. Therefore, a service can be seen as a special filter engine which filters information according to its own purpose. All services may use basic functions provided by the filter engine and location engine. A client can access an information service through its interface.

2.3 Summary

In this chapter, we have introduced the TIP development history and explained the basic principles of the TIP 2.0 system architecture. In the next chapter, we firstly give an analysis of people's travel behaviour (Section 2.1). Based on the result, we propose the system requirements for our TP Application (Section 2.23.2).

Chapter 3 Requirement Analysis

In this section, we will firstly analyse tourists travel behaviours (Section 3.1) and then present the requirements that must be fulfilled in order to satisfy the traveller's information need (Section 3.2).

3.1 Travel Behaviour Analysis

We leave aside the technological issues and focus on the tourists' needs. In general, there can be two types of tourists: individuals and groups. Since our service is designed to be used by single users who equipped with mobile devices supporting GPS, we are only interested in the individuals. To gather information we carried out a small survey asking "how do you travel". By analysing the answers, we concluded that tourists can be divided into two groups. One group of people try to plan everything in detail (e.g. where to go, hotel, and transportation) before they start travelling. The other group of people do not plan their trips ahead and prefer to plan their travel activities on the move.

Obviously, the people of the second group are the potential (main) users of our service. Therefore we focused on analyzing the behaviours of people belong to the second group. Of course, it depends on the person, but all the answers can be generalized to the following scenario: When a traveller arrives at a place, he or she would usually go to the "must see" sights of that place. For example, if

one travels to Paris, he or she would firstly visit the Eiffel Tower, the Notre Dame, the Arch of Triumph, and the Louvre.

Besides the must see sights, if the traveller still has time, he or she would visit some other places of interest. There are two factors that decide if the traveller would visit a place of interest: **distance** (while visiting these "must see" sights, the traveller might visit some places of interest that are close by the travel routes of the must see sights) and **personal preferences** (tourist's personal interests e.g., history, arts or friends' recommendations). Since the core of our service is location-aware, we focus on dealing with information from the 'distance' aspect. The other aspect is related to the content-based recommendation and it has already been implemented for the TIP system ([7], [9]).

3.2 System Requirements

Based on the tourists' travel scenario described above, we identified the following requirements that should be met in order to fulfil the system's purpose.

3.2.1 Functional Requirements

1. The system should provide a map-based GUI for presentation geographical information stored at the TIP server-side database. In other words, the system should be able to display variety of graphic items on the map. Such as, sightseeing places, tourist's current position, text, travel routes, and so on. The system should allow users to customize how each item should be displayed (e.g. colour, size, and visibility). All

the user's customized settings should be stored automatically on exit and loaded on start up.

2. The system should provide map navigation functions, including zooming and panning on the map. By panning, we mean that users may scroll the map horizontally and vertically to view the place they wish to see. By zooming, we mean that users may change the view of the map to a closer or wider perspective. Moreover, the system should also support map navigation bookmark. By map navigation bookmark, we mean that users can save or restore a view of the map which they have navigated.

3. The system should operate in two modes: actual trip and virtual travel. The actual travel mode, as its name suggests, is to be used when users are already at their destinations. In the actual travel mode, the system should dynamically change the centre location of the map to be consistent with the users' current location. In other words, the system should update the map display along with the user's movement and provide relevant information about the location. The virtual travel mode is to be used when users plan their trips at home. In the virtual travel mode, the user can manually navigate the map to look to find out the information about their destinations.

4. The system should display all the "must see" sights of a place on the map as fixed information. Other sights should be displayed based upon the current location of the user. Furthermore, by clicking at a point on a sight (or another kind of operation), the system should open a window which displays the detailed information related to this sight, such as sight description, other users' ratings and comments are displayed.

5. The system should allow users to create/define virtual travel routes on the map. A virtual travel route can be defined by specifying a series of

waypoints on the map. The system should be able to locate and display sights which are within a certain distance (specified by the users) of a virtual travel route. This function would be very useful for users to plan their trips. Additionally, the system should allow users to store the virtual travel routes to their local disks or upload them to the TIP server-side database for further use. The virtual travel routes stored in the database can be accessible to the public or viewable only to the owner. This allows users to plan their trips based on other travellers' travel routes.

6. The system should allow users to print the map view currently displaying in the system including map image, graphic for sightseeing places, and planed travel routes.

7. As an extension service to the TIP system, the system should be able to interact with other TIP services. Another TIP service which runs on the service-side can remotely control system's map view. On the other hand, users to execute information services running at the TIP server from the client-side application.

3.2.2 Non-functional Requirements

- The system should operate on both PC based machines (desktops and laptops) and portables that have Java2-compliant virtual machine installed (version 1.4 or higher).

- The system architecture should be optimized to reduce internet traffic and minimum the memory requirement. This requirement is to ensure that the system can run on portable devices with low memory and slow internet bandwidth.

3.3 Summary

In this chapter we analysed the people's travel behaviour and defined the potential users of our system. We then proposed a number of system requirements according to results of the analysis. In the following chapter, we will take a look at some of the similar applications and examine if these applications fulfil the system requirements we proposed in this chapter.

Chapter 4 Related Work

A number of GIS mapping applications provide similar functionalities to our proposed Travel Planning application. In this chapter, we first introduce some of the most commonly used GIS mapping applications which are related to our work and then examine these applications to see if they fulfil the requirements discussed in Section 3.2. The purpose of this related work is to find a system which can be used as a basis to implement our proposed TP application.

4.1 Related GIS Mapping Applications

GIS applications can be generally divided into two categories: traditional GIS applications or web-based GIS applications. In this section, we explain the concepts of the two types of applications and introduce six GIS applications which include three traditional GIS applications and three web-based GIS applications.

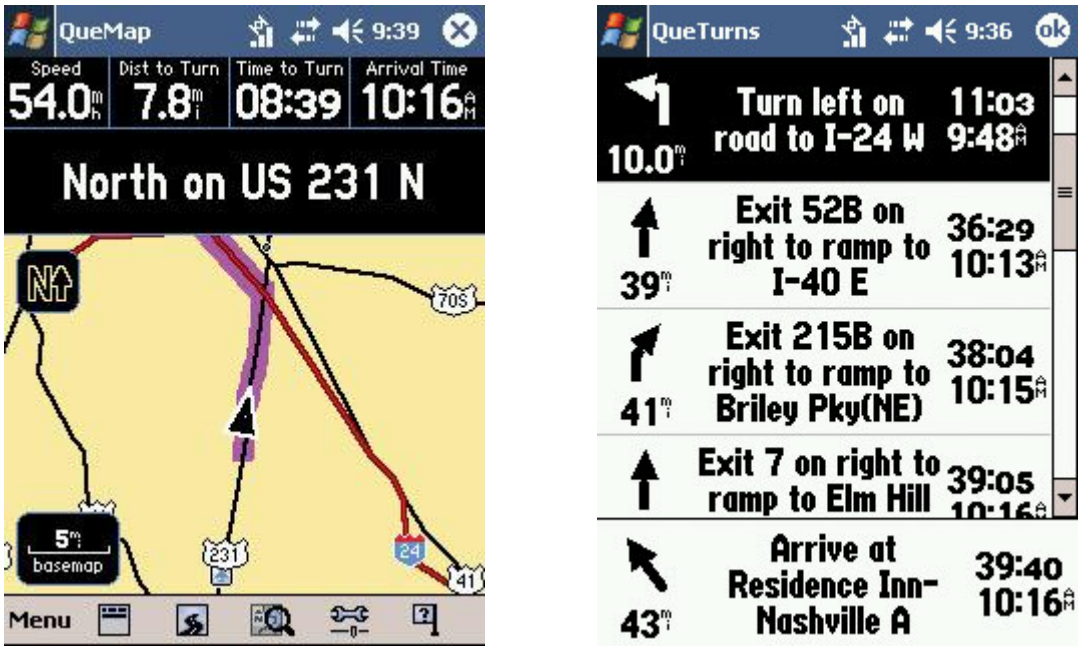
4.1.1 Traditional GIS Applications

Traditional GIS applications are installed and run on a user's computer (e.g. laptops) or mobile devices. They are designed to be used by a single user. Typically, the installation of the traditional GIS applications also includes a set of geographical data which is needed for GIS. For example, traditional GIS applications normally come with a set of map image data. The advantage of the

traditional GIS applications is that they can directly listen to the GPS signals received by the hardware device on which they are running, and provide information according to the real locations of users.

- **GARMIN's Que**[6]

GARMIN is the world leader in GPS technology. It has a diverse product line that includes GPS receivers, GIS software and a number of different GPS navigators (e.g. car navigators). GARMIN's Que is a GIS mapping application. It can be installed on PDAs running the Pocket PC and Windows Mobile for Pocket PC operating systems. A GARMIN-made GPS receiver (which uses GARMIN's GPS protocol) is required by the system to analyse the location information.



(a) Map view (b) Turning instructions

Figure 4: GARMIN's Que System Screenshots [13]

GARMIN's Que system can be used on a PDA. It enables users to see where they are, location a street address, point of interest, or even to know where the next

[6] http:// www.garmin.com/

turn is via visual and voice guidance. Figure 4 shows two screenshots of the Que system. The left screenshot shows the map navigation view. The right screenshot shows a combination of turning instructions. As can be seen from the screenshot on the left, the map view provided by GARMIN's Que is a vector image created by computer program based on the geographical data of the displayed area. The system does not use raster graphic images scanned from maps, photographs, or satellite data. This mechanism is designed to reduce the source device storage and memory requirements (which are the shortcomings of PDAs).

- **OziExplorer**[7]

OziExplorer is GIS Mapping software produced by Newman. It is very similar to GARMIN's Que software. The main difference between OziExplorer and GARMIN's Que is that OziExplorer uses raster map images (scanned images) to display map, whereas Que uses vector map images (geographical land-base such as AutoCAD DWG, DXF and ESRI SHP, etc).

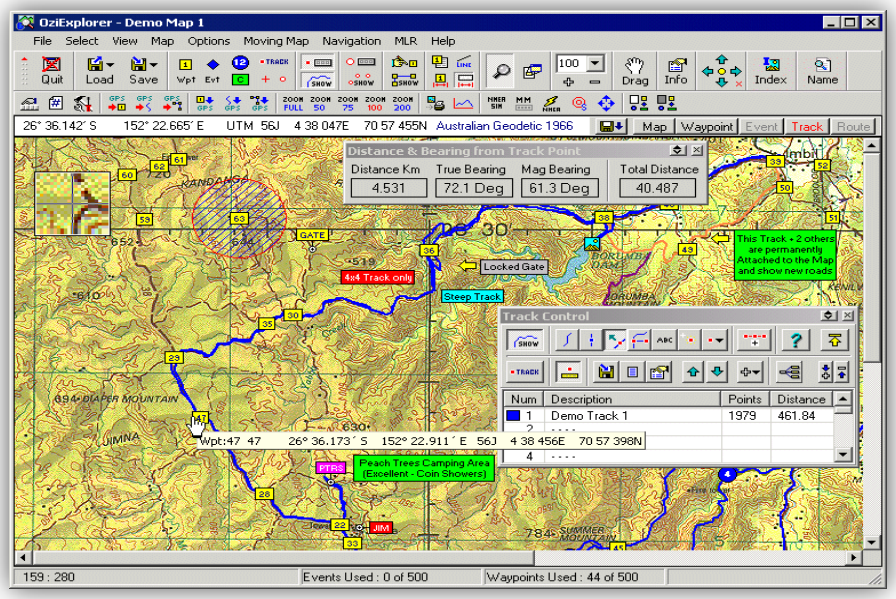


Figure 5: Screenshot of OziExplorer [14]

[7] http://www.oziexplorer.com/

OziExplorer is available in three different versions: OziExplorer, OziExplorerCE and OziExplorer3D. OziExplorer is the full version which runs on PC or laptop with Windows OS later than 95. (Figure 5 is a screenshot of the PC version of OziExplorer.) It allows users to use digital maps they purchased form internet or scanned themselves. By using these maps, users can plan their trips on the map by defining waypoints, routes and tracks and upload these to their GPS. OziExplorer can be running in moving map mode when using a laptop connecting to GPS- receiver. In moving map mode, the system automatically updates map according to the geographical location of a user.

OziExplorerCE is a GIS mapping software which runs on a PocketPC and Window CE PDA's. It allows users to track their position based on a GPS signal then display their position on a map. However OziExplorerCE cannot work on its own. It requires the full PC version of OziExplorer to calibrate maps, plan trips by adding waypoints etc. Therefore it can be seen as an add-on to the full PC version of OziExplorer software.

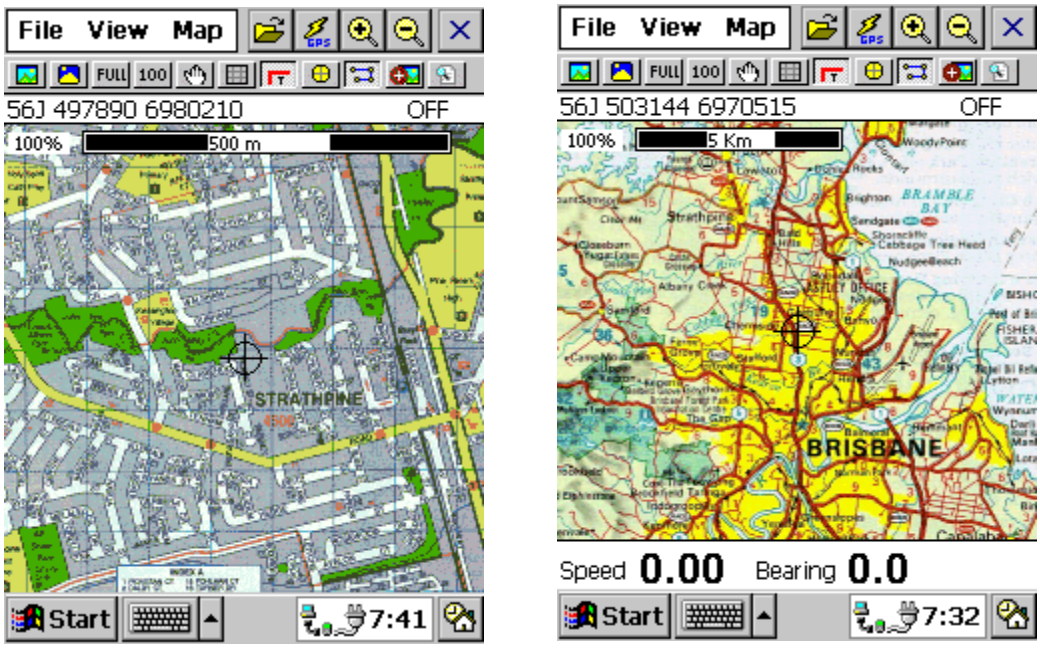


Figure 6: Screenshot of OziExplorer CE [14]

OziExplorer3D is 3D map viewer software. It allows map images to be viewed in 3D mode with the ability to rotate in all directions and zoom in and out of the view. We will not discuss it further as it is not related to our project.

- **OpenMap™ Package**[8]

BBN Technologies' OpenMap™ package is a free JavaBeans based programming toolkit for building GIS mapping applications that allow users to view and manipulate geographical spatial data. Moreover, OpenMap is primarily for data viewing and offers very little in the way of analysis functionality. Earth surface distance measurement is the only supported analysis function in OpenMap.

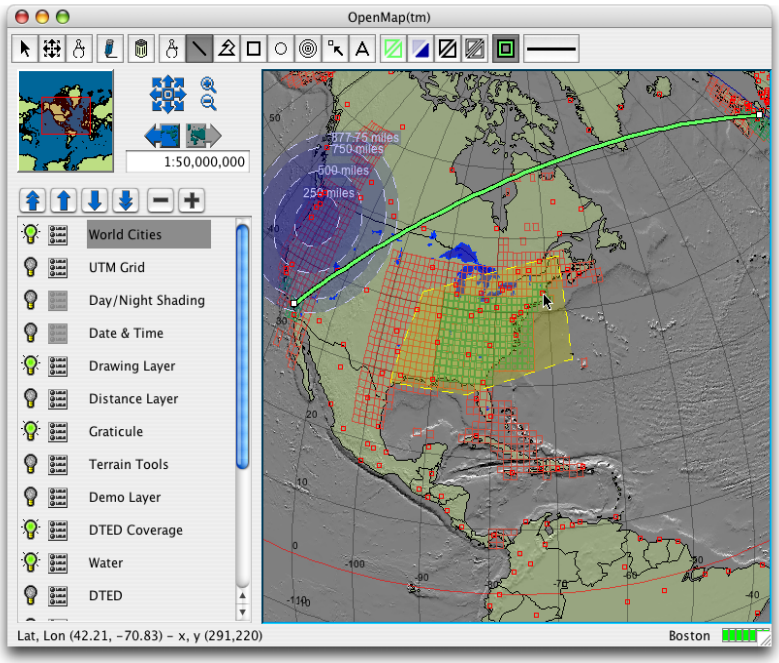


Figure 7: Screenshot of OpenMap Software [2]

The OpenMap™ distribution comes with a sample application that can be used to view and manipulate geographical spatial data. Figure 7 is a screenshot of the OpenMap sample application. Programmers can use the sample application as a basis for their own applications.

[8] http://openmap.bbn.com/

The core of OpenMap package is a set of Swing components that understand geographic coordinates. Using these components, programmers can write applications and applets that access data from legacy databases and applications. More detail information about the OpenMap architecture is described in Section 5.1. By the time this paper is written, there will already exist many published applications that are developed using OpenMap. A list of these applications can be found at: *http://openmap.bbn.com/whoelse.html*.

4.1.2 Web-based GIS Applications

Web-based GIS applications, as the name suggests, are designed to run inside a web browser. They enables users to view or query geospatial data provided by a specific information provider. Recently, there has been a new trend in developing web-based GIS applications. Unlike the traditional GIS applications, web-based GIS applications are platform independent as they do not require users to install software. The only requirement is a web browser and an internet connection. With the increasing speed of the internet connections, web-based GIS applications are becoming more and more prevalent. In the following section, we will introduce three web-based GIS mapping applications.

- **Google Maps Beta**[9]

Google Maps is a web-based mapping service, which provides Internet browser-based, door-to-door directions as well as maps of a particular location. It is one of Google's latest products. Google Maps allows users to search a map for a particular location by specifying a location name or a zip number. The map image used in Google Maps can be displayed in two modes: either hybrid mode (Figure 8), which displays vector images constructed by a computer using spatial

[9] http://maps.google.com

data, or satellite mode (Figure 9), which display raster map images acquired from satellite radar. Both modes can display very detailed maps. Furthermore, Google Maps gives users full control over map navigation, including panning, zooming and centring by zip code, city and state.

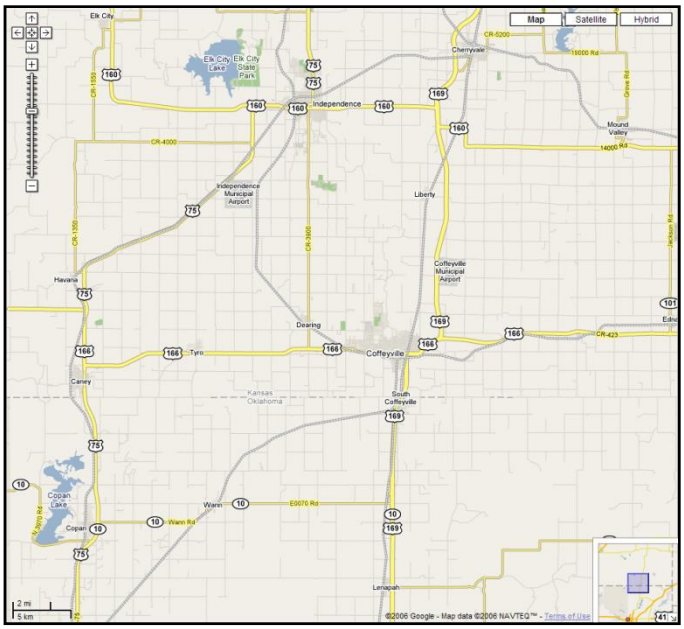


Figure 8: Google Maps – Normal Mode [15]

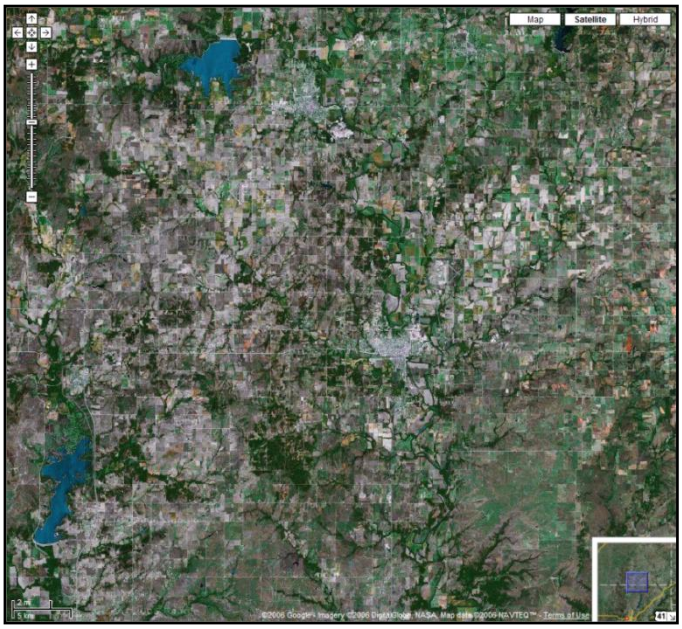


Figure 9: Google Maps – Satellite Mode [15]

Besides the Goole Maps application, Google also introduced an API that enables developers to create their own applications that interact directly with the Google Maps service. For example, you can create a picture view program that shows where your photos were taken and displays them on Google Maps. However, the Google Maps API is still in beta phase and it is only available for implementation on website using JavaScript. When we first started this project, the Google Maps API did not support geocoding (which is the process of assigning geographic identifiers to map features and other data records, such as street addresses) and routing capability (which allows users to create custom routes on the map). However, at the time we are writing this report, both features are being added to Google Maps.

- **Yahoo Local Maps Beta**[10]

Yahoo Local Maps Beta (also known as Yahoo Maps) is a web-based mapping service provided by Yahoo as a competitor to Google Maps. It not only provides all the functions available in Google Maps (e.g. search, satellite view, navigation), but also many new features, including:

1) Flash interface, Yahoo Local Maps is coded in Flash, so the interface is much more visual and attractive when compared to Google Maps.
2) Browse-based search, which allows user to search without typing anything. For example, users can drag an address from the map into the search forms.
3) Live-Traffic, which displays traffic trouble spots on the map
4) Multi-point routing, which allows people to plot driving directions for multiple locations.
5) Local directory service, which helps users find a location and things to do in a certain place.

[10] http://maps.yahoo.com/

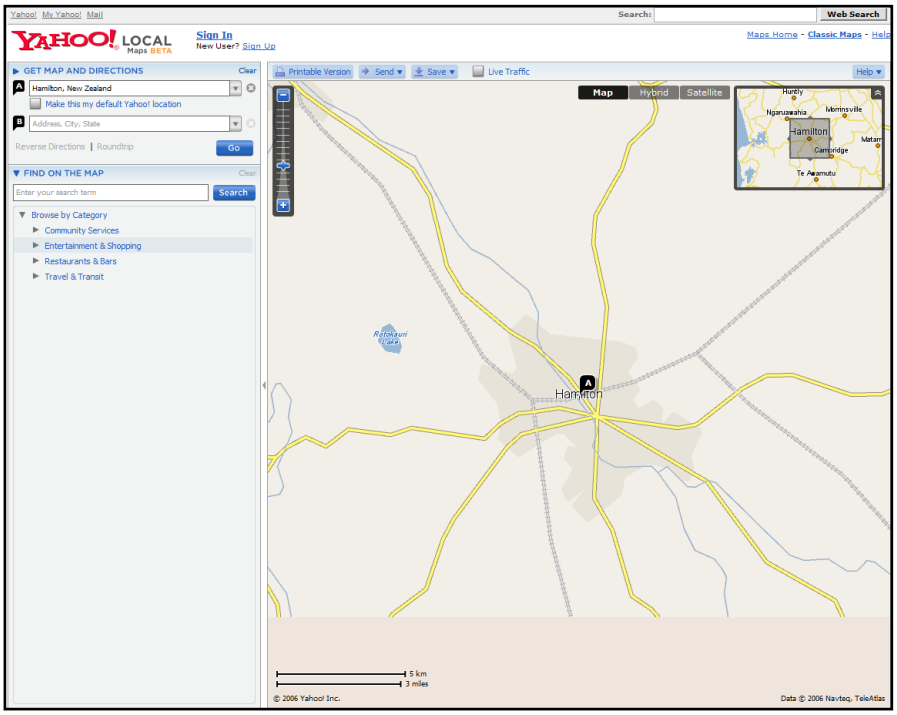


Figure 10: Yahoo Local Maps Beta [17]

Like Google Maps, Yahoo Local Maps also introduced an API which allows developers to pull a map onto their own web site and overlay their own content on the map. The Yahoo Maps API is more advanced than the Current Google Maps API. It not only supports geocoding, but also includes downloadable source code that allows developers to add interesting hacks to maps, such as distance measuring tools, a freehand drawing tool or other interesting widgets.

As the Yahoo Local Maps beta is unfinished, it still has some shortcomings. For example, Yahoo's satellite view mode does not zoom in as closely as those on Google Maps and current maps only cover the US and Canada. Moreover, Yahoo Local Maps API does not offer as much in the way of tech support as the Google Maps API does.

- **Windows Live Local Beta**[11]

To ensure not being left behind by Google and Yahoo, Microsoft recently realised its own online web service called Windows Live Local (WLL). WLL is built on Microsoft Virtual Earth SDK which is free to the public and supported in Microsoft Developers Network (MSDN) website.

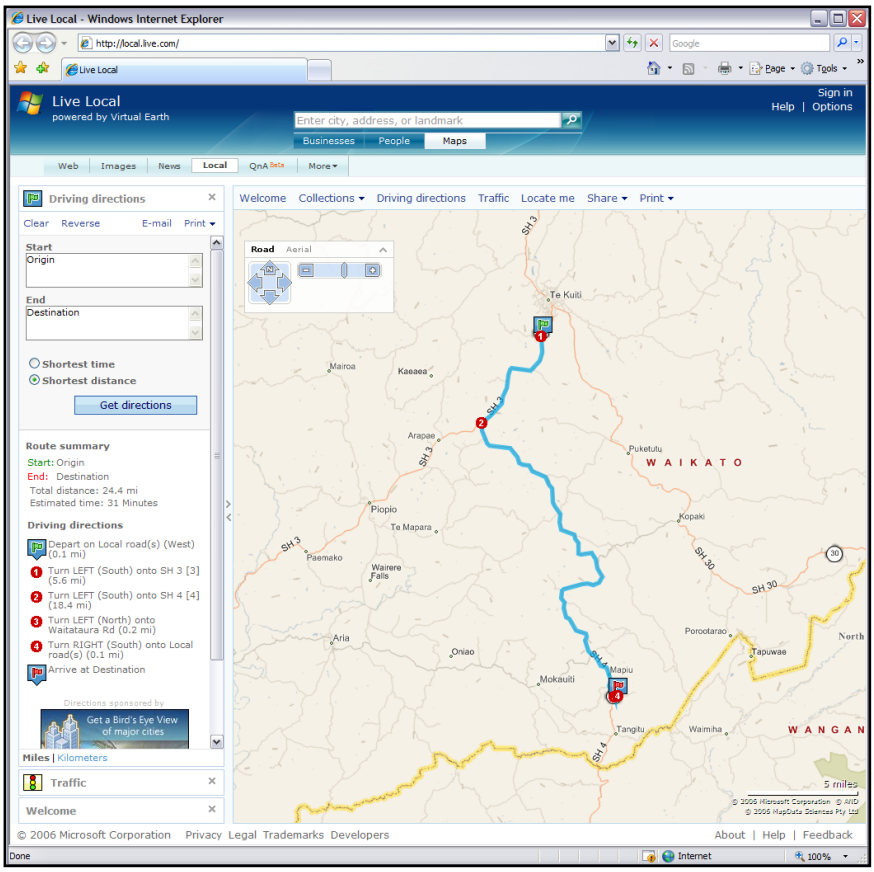


Figure 11: Windows Live Local [16]

Like other online mapping services, Windows Live Local also allows text-based map search, map navigation and map routing. However the map routing function is much more advanced than routing functions on Google Maps and Yahoo maps. It does not have the concept of end point, which means users can define an arbitrary number of waypoints on the map. For example, users can define a route from point A to point B, from point B to points C and beyond.

[11] http://maps.live.com

Based on a set of way points, Windows Live Local is able to come out with a best driving direction (shortest distance) and displays the route on the map. In Figure 11 above, the three red dots are user defined waypoints. The blue path is the recommended driving direction given by Windows Live Local.

Besides the common features, Windows Live Local provides a new feature called “Local me”. It enables users to locate their positions on a map based on one of two parameters:

1) Local IP address. In this case, Windows Live Local will try to calculate a location by using a computer IP address.
2) Wi-Fi hotspots. Windows Live Local will try to determine a location from nearby Wi-Fi networks.

Another advantage of Windows Live Local is that it can be integrated with other Microsoft Live services. For example, the integration of Live Messenger and Live Local enables users to find the position of the people which they are chatting with on a map.

4.2 Comparison

In this section, we compare the six related GIS mapping applications previously described. We compare these applications on terms of the functional requirements proposed in Section 3.2.1 .

Table 1: Requirements VS Related Works

	Garmin Que	Ozi Explorer	OpenMap	Google Maps	Yahoo Maps Beta	Windows Live Local
Requirement 1 Display geographical information stored in TIP	✗	✗	•	•	•	•
Requirement 2 Map navigation	✓	✓	✓	✓	✓	✓
Requirement 3 GPS Location analysis	✓	✓	•	•	•	•
Requirement 4 Sight recommendation	✗	✗	•	•	•	•
Requirement 5 Routing / Travel Planning	✓	✓	•	✓	✓	✓
Requirement 6 Printing	✓	✓	•	✓	✓	✓
Requirement 7 Connect to TIP	✗	✗	•	•	•	•

✓ : The function is supported on the system.

✗ : The function is not supported on the system.

• : The function is not supported, but it can be implemented when needed.

Form the above table we can see that Garmin's Que and Ozi Explorer meet most requirements (4 out of 7) proposed in section 3.2.1, such as display map according to GPS signals, routing and so on. However, both of the two systems are commercial software and don't allow third-party development. Therefore, we have no opportunity to implement extensions in order to fulfil the remaining requirements.

BBN Technologies' OpenMap appear to be the worst among the six systems. It only supports basic map navigation functions and its maps are simply the maps that only show the outlines of continents. However, OpenMap is not a ready to use system, but a framework which allows developers to quickly build GIS applications according to their own needs. As can be seen from Table 1, even though six out of seven requirements are not met in OpenMap, it is possible to implement them within the OpenMap framework. For this project, we chose to use OpenMap framework to implement our Travel Planning application. More information about the OpenMap framework will be covered in later sections.

The three competitors of web-based mapping services have the same score. They all met requirement 2, 5 and 6. For the unsatisfied requirements, it is possible to implement them using any of the three APIs. In fact, it is a better solution to implement our proposed system using one of the three mapping services than using OpenMap. That is because all the three web services have built-in map image support which can greatly reduce programming time. Beside, our Travel Planning system was implemented as an extension of the TIP system which is a web service accessed through web browsers. It would be easier for users to use our Travel Planning system within the same browsers than starting a standalone application. However, when we first started this project, only Google Maps had been published and its early versions do not support geocoding and custom routing. Therefore, OpenMap was the only choice for us at that time.

4.3 Summary

In this chapter, we firstly explained the concept of Traditional and Web-based GIS Application, and then we reviewed six GIS Mapping applications. In section 4.2 we compared the six applications to the system requirements we proposed in Section 3.2. According to the comparison result, we decided to use the OpenMap framework to implement out Travel Planning application. In the next chapter, we will present our system design and implementation.

Chapter 5 System Design and Implementation

In this chapter, we will present the system design details and implementation issues of the map application. We firstly provide an overview of the OpenMap™ GIS system based on which our Travel Planning application is implemented and define some terms used in OpenMap that will be mentioned in later sections (section 5.1). Secondly, we present the system architecture of the TP application (section 5.2). Thirdly, we will provide and explain the structural module (section 5.3), followed by a representation of the main classes composing the Travel Planning system (section 5.4). We will describe in detail the designs and functions of the three major components contained in the application (section 5.5), which are the Sight layer, the Route Layer and the Map Layer. Lastly, we will present the presentation of TP Application and TIP communication (section 5.6).

5.1 OpenMap Architecture

The OpenMap package is an Open Source JavaBeans based programming toolkit by BBN Technologies. It is for building applications and applets needing geographic information [2]. The core of OpenMap is a set of Swing components that handle geographic coordinates. Programmers can use these Swing components to display map data and handle user input events to manipulate the data.

In our Travel Planning application, we used Openmap for geographical mapping and display capabilities so that we can avoid reproducing existing basic GIS mapping functions. There are two ways for programmers to use the Openmap package. One is to develop applications as extensions to the OpenMap Viewer application which is included in the package. The other is to incorporate the OpenMap components in their own applications, which is what we did in developing our TP application. The reason we chose the second way is that we can reduce the memory requirement by only importing the necessary components into the application.

Among the components of OpenMap, MapBean and LayerBean play central roles. MapBean is a drawing canvas that derives from the Swing JComponent class. It manages a hierarchy of Layer Beans which can paint themselves to the canvas [2].

MapBean is associated with a Projection object that controls the view of the MapBean canvas (e.g. panning, zooming, and resizing). The Projection object specifies the view of the MapBean canvas from the following aspects:

- Geographical coordinate (latitude/longitude) of the centre point of the MapBean canvas.
- Scale of the MapBean canvas.
- Height and width of the MapBean canvas.
- Projection type (Mercator, Orthographic, etc). In our Travel Planning Application, the projection type is fixed at CADRG, which is an Equal Arc Projection, meaning that the variations in latitude and longitude are constant.

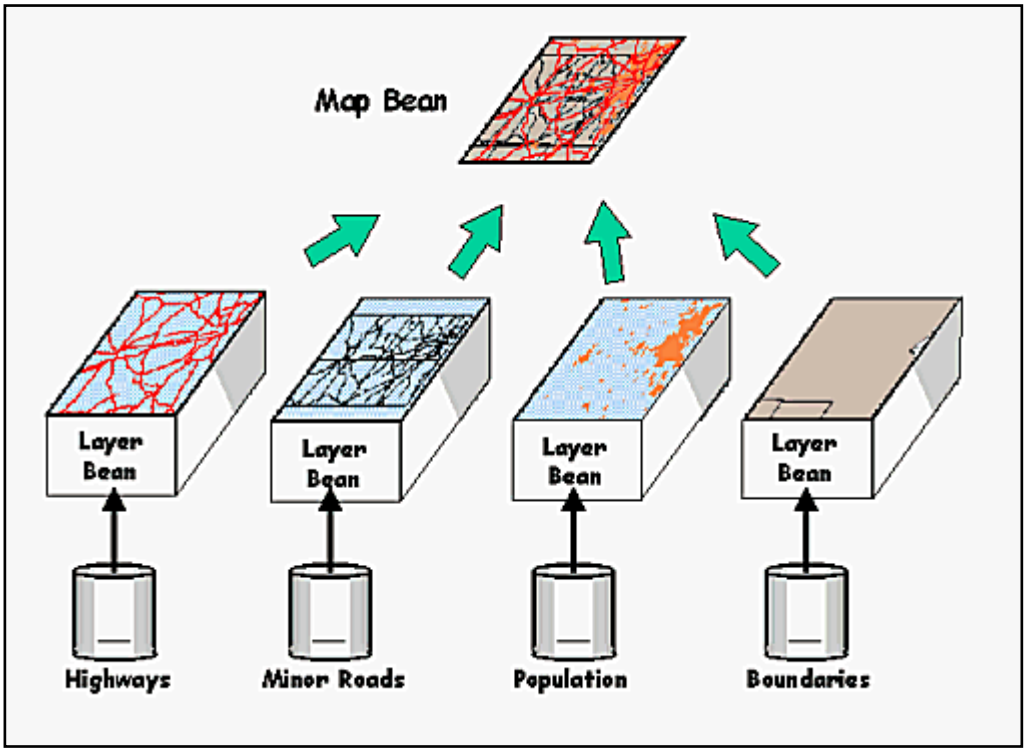


Figure 12: OpenMap Architecture (Adopted from [2])

LayerBean is the only component in the Openmap package that can be added to MapBean. LayerBean is responsible for acquiring, creating, managing and rendering its own spatial data on the MapBean. Figure 12 above illustrates the relationship between MapBean and LayerBean. The MapBean shown in the top part of the figure manages four LayerBean components. Every LayerBean component is in charge of displaying a type of spatial data from the database. (e.g., the first Layer displays highways on the MapBean. The second layer displays minor roads information.) LayerBean components are added to MapBean in a hierarchical stacking order, which means that first added LayerBean is located at bottommost position. The MapBean is displayed by painting the graphics of each LayerBean component starting with the bottommost one and proceeding up the hierarchy. Successive LayerBean component render their graphics on top of the graphics of lower ones.

When a LayerBean component is added to MapBean, it is automatically registered to receive the projection change event from the MapBean (This is

one of JavaBeans features). A projection event is automatically generated by MapBean when its projection has been changed. It contains information of the new states of the MapBean projection. The LayerBean is an abstract class. Every class that extends the LayerBean class must implement a function called "projectionChange". This function is invoked in responding to a projection change event. Inside the function, a series of statements will be executed to update the graphic displays on the layer based on the changing of the projection. For example, redrawing the map or remapping graphical objects according to the new project area displayed on the MapBean canvas.

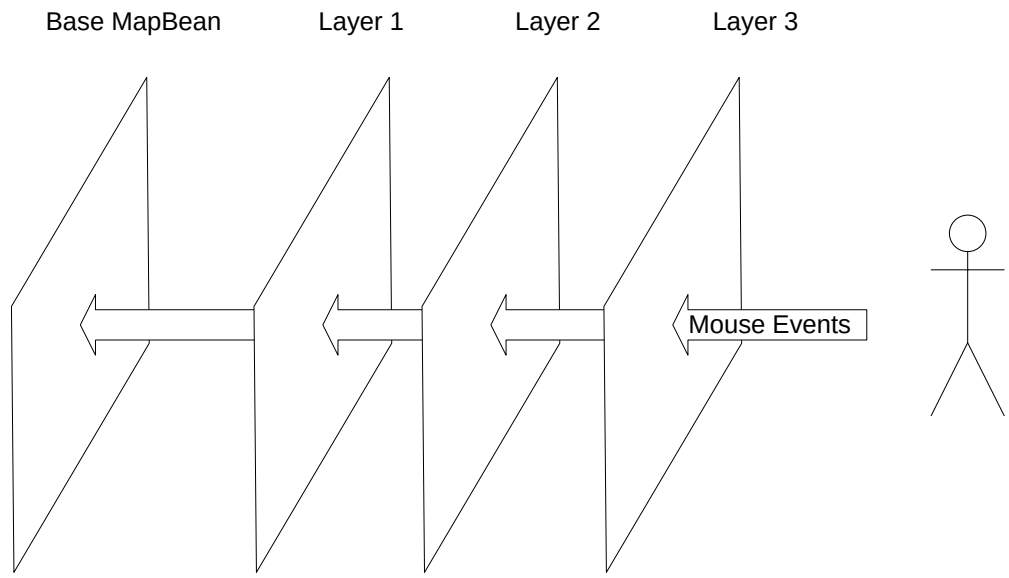


Figure 13: Mouse events in Openmap

In addition to the Projection Change events, Layer Beans can also be interactive, by registering for mouse events the Map Bean. Figure 13 above illustrates the principle of how mouse events are handled in the Map Bean. Different from a projection change event, a mouse event is sent to the layers in the stack order. When MapBean sends out a mouse event, the layer on top of the stack will first receive the event. It can either consume the mouse event or pass it onto the layer beneath it. Furthermore, every LayerBean can construct its own GUI

widget controls which allow users to easily customize its appearance. (e.g., drop down lists for choosing background or foreground colour.) Once the GUI controls of a LayerBean component are constructed, they will be automatically added to the associated MapBean GUI control panel.

In this section, we have explained the basic architecture of the Openmap framework. In the following section, we will propose the system architecture of the TIP application and explain how OpenMap components are used in our system.

5.2 System Architecture

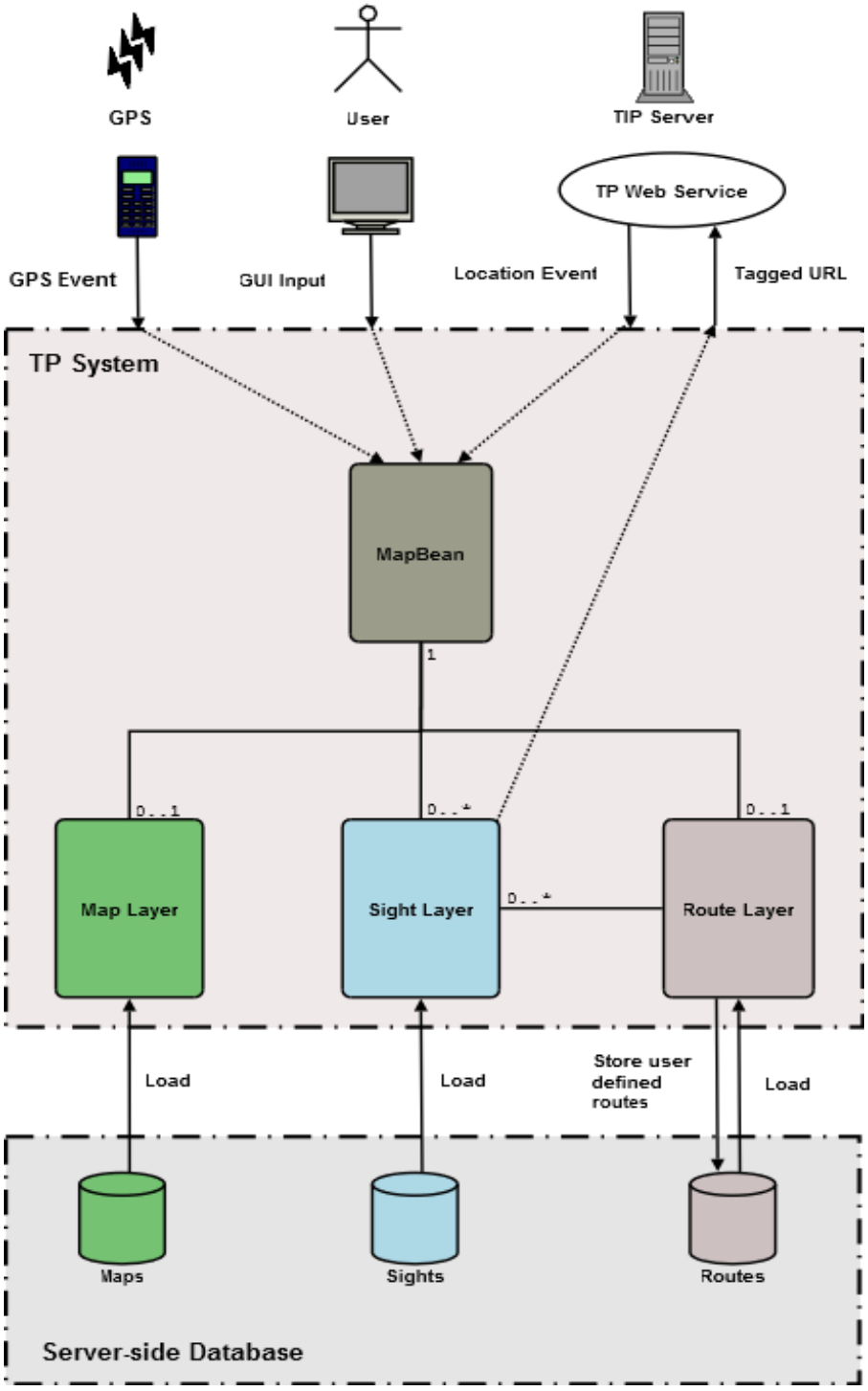


Figure 14: TP System Architecture

Our system was implemented using J2SDK (Java 2 Standard Development Kit) 1.4.2 and the Openmap library. As mentioned in the previous section, there are two ways the way to use the Openmap software package, and we chose to incorporate the OpenMap components into our own application. The system architecture of our Travel Planning system (shown in figure 14) is similar to the structure shown in Figure 12. The MapBean component shown in the center of the figure is the core of our system. It is shared by three LayerBean components, which are SightLayer, RouteLayer and MapLayer. Each layer bean is responsible for rendering a type of spatial data on the MapBean canvas.

MapLayer constructs and renders map images according to the location area displayed in the MapBean canvas. Since it is the background layer, it must always be placed at the bottom of layer stack in order to be painted first. SightLayer is designed to locate and display sights from the TIP central database on the MapBean canvas. Furthermore, it also provides a function which allows a user to navigate from a sight displayed on the MapBean canvas to the corresponding web page on the TIP website. Finally, RouteLayer loads and displays users' travel routes. It can also highlight sights from all of the existing SightLayer instances according to a given travel route. More detailed information about the functions and implementations of the three Layer Beans will be described in section 5.4 .

The Travel Planning system provides three ways to control its MapBean (as shown on top part of the figure):

1. The Map Bean's projection can be automatically updated by GPS signals. When the TP system receives a GPS signal from user's mobile device, it will update the center variable of the MapBean projection. This projection will match the location specified in the GPS signal.

2. User input from GUI. The system provides the user with a number of GUI components to control the MapBean projection, with options such as move panel, zoom buttons, and scale field.

3. The Map Bean can also be changed by a location event sent from the TIP server. Besides the Travel Planning system, we also implemented a TIP server-side application (called TIP Map Service) which can send a location event from the TIP server to a Travel Planning application running on a client device.

5.3 Structural Model

This is the structural mode of our system. The package diagram illustrates the relationship between all of the packages used in our system.

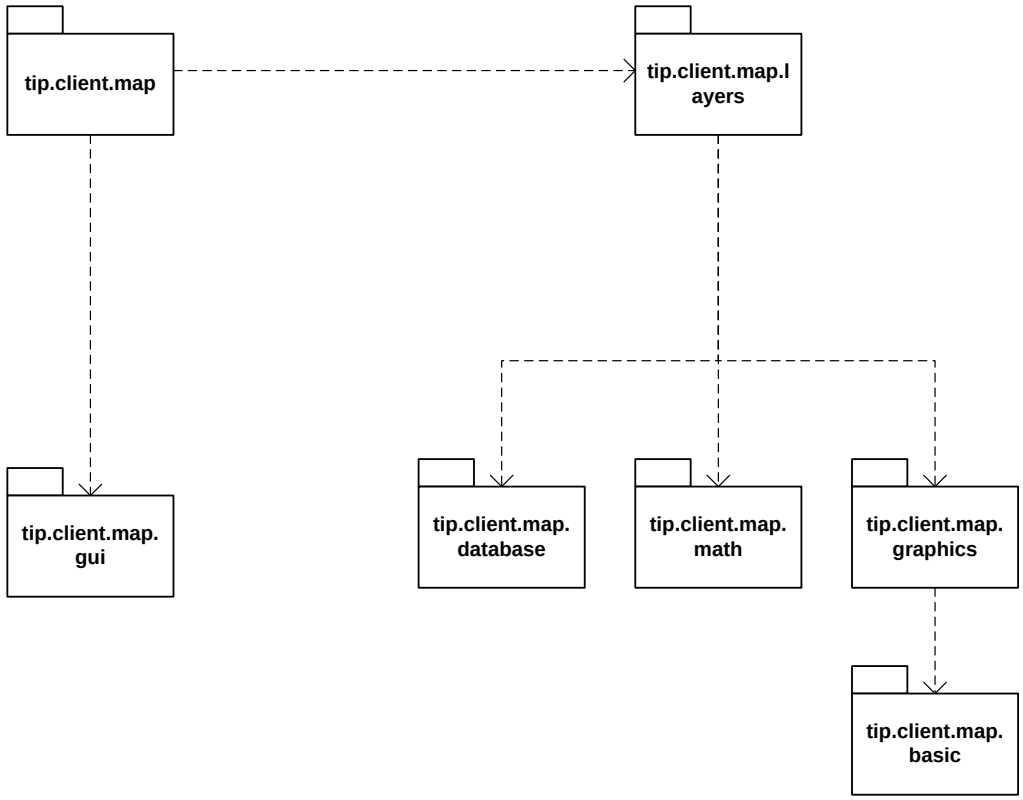


Figure 15: Package diagram

Package tip.client.map

This is the main package for the implementations of our Travel Planning Application. It contains the executable class for the application. All other packages are sub-packages of this package.

Package tip.client.map.gui

This package contains the GUI components which are used to control the Map Bean. These components have been written following the conventions of the OpenMap framework so that they can be connected automatically to the Map Bean after being initialised.

Package tip.client.map.layers

This package contains implementations of the three OpenMap Layer Beans which are MapLayer, SightLayer, and RouteLayer. Each Layer Bean reads and displays a unique set of spatial data. The RouteLayer can also create its own data for display.

Package tip.client.map.layers.graphics

This package provides a number of geographically based graphics classes which can be displayed on the OpenMap Layer Bean. These classes are referred as OMGraphics (OpenMap Graphics).

Package tip.client.map.layers.basic

This package defines the basic foundation classes for OMGraphics. These foundation classes only describe the geometry of OMGraphics without any reference to rendering attributes.

Package tip.client.map.database

This package contains several classes that are used to connect to the TIP central database, execute queries and statements, and manage query results.

Package tip.client.map.math

This package encapsulates mathematical functions and methods for operating on spatial data. e.g., ‘calculate surface distance between two sights or from a route to a sight’.

5.4 Class Diagrams & Internal Relationship between Classes

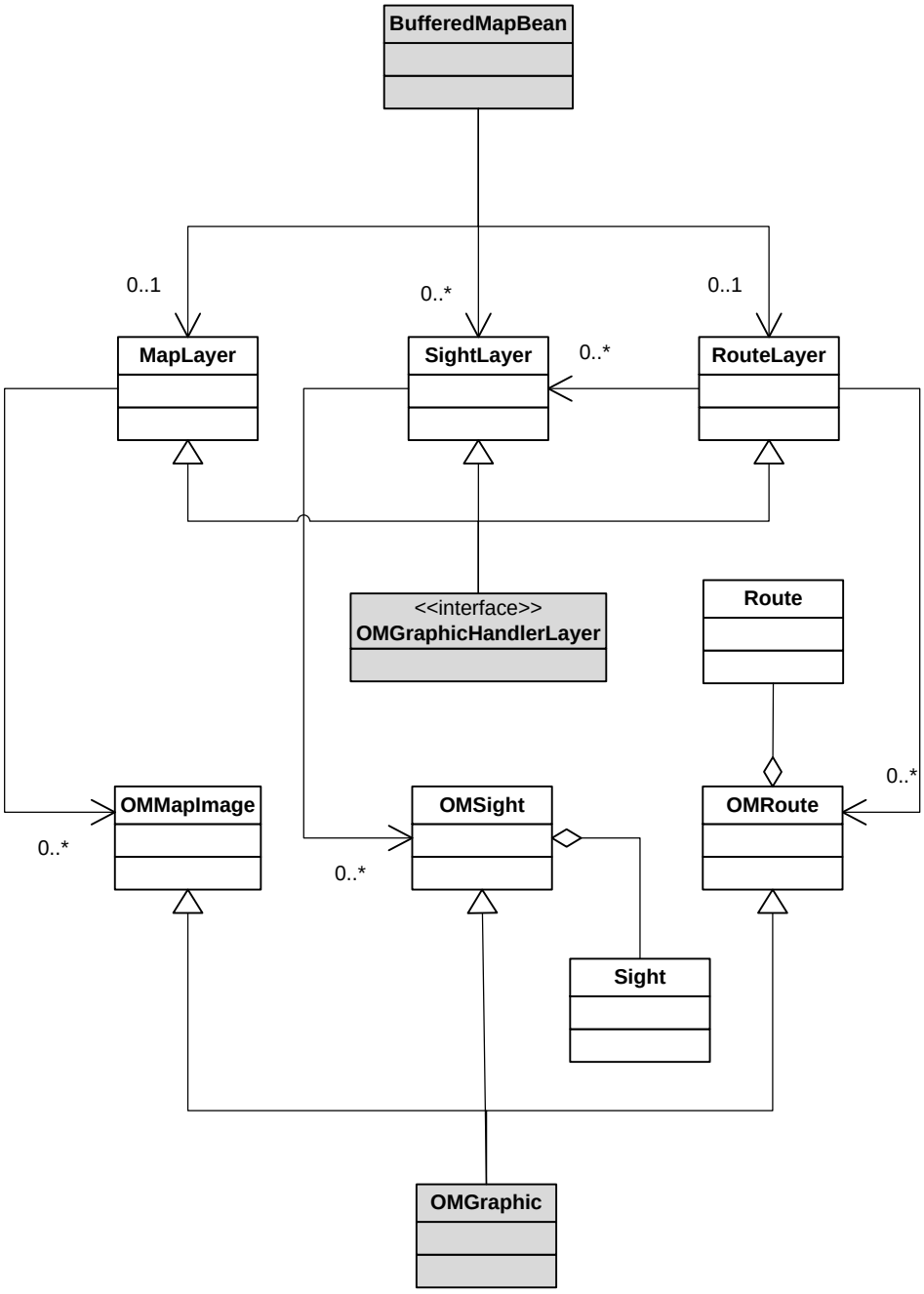


Figure 16: Class Diagram

Figure 16 illustrates the representation of the main classes composing the Travel Planning system. The classes BufferedMapBean, OMGraphic and the interface OMGraphicHandler (shown with a darker colour) are imported from the OpenMap library. The BufferedMapBean class is an extension of the MapBean class. It uses the clipping mechanism to solve the flicker problem caused by frequently repainting. The BufferedMapBean is associated with three layer classes: MapLayer, SightLayer and Route Layer. Each of the layer classes extends the OMGraphicHandler interface which describes a layer object that manages OMGraphics. The interface provides a mechanism to filter which OMGraphics should be displayed based on certain criteria (which in our case is the projection area of the MapBean).

Among the three layer classes, the SightLayer is slightly different from other two layers as there can be more than one SightLayer instance associated with a MapBean. This mechanism allows the application to represent various sight objects differently on the MapBean. For example, display parks as red squares and buildings as blue triangles. More detailed information of this mechanism will be provided in later section.

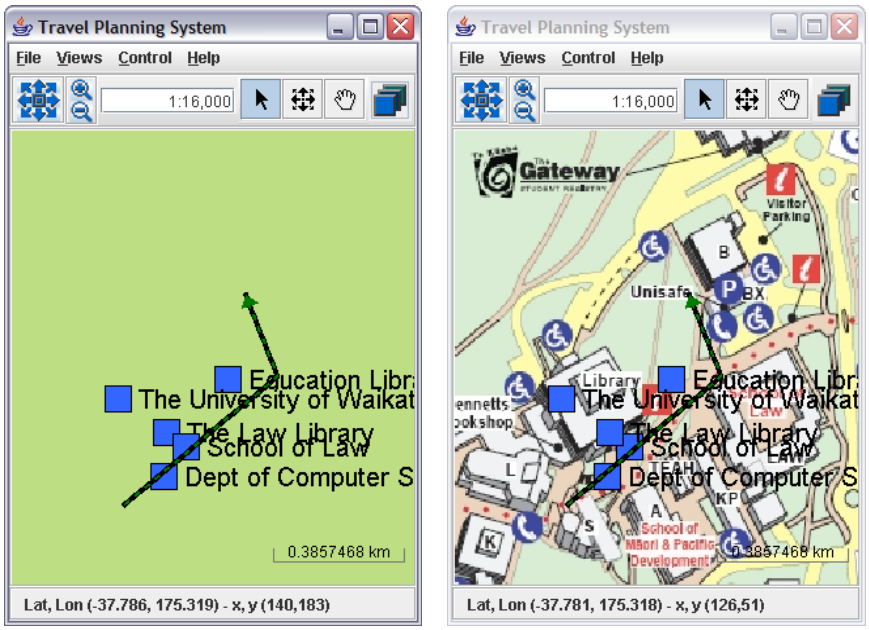
Each of the layer classes is capable of constructing and displaying a type of graphical object. In our system, we implemented three types of graphical object: OMMapImage, OMSight, and OMRoute. As can be seen from the class diagram, these three graphical classes inherit the same abstract class OMGraphic, which defines a vector graphic object and contains information about how the object should be drawn.

5.5 Design and Implementation of the Three Layer Beans

The Map layer, Sight layer, and route layer are the three major components of our Travel Planning applications. In this section, we will explain in detail the design and functions of the three layers.

5.5.1 Map Layer

Map Layer, as its name suggests, is designed to display map images on the MapBean according to its projection. The Map Image Layer can be seen as a background layer as it remains at the bottom of the layer stack and always gets rendered first. Figure 17 shows the effect when a Map Image Layer is added to the system. Instead of just using a plain color (screenshot 1) for the background, the corresponding map image is displayed in conjunction with other graphic objects (screenshot 2).



Screenshot 1: no Map Layer

Screenshot 2: with Map Layer

Figure 17: Effect of using Map Image Layer

There are two major issues in the design of the Map Image layer: storage size of the map image and multi-scale representation. We will address these issues and then explain the design of the Map Image Layer in the following sections.

Storage size of map image data

The storage size of map image data is substantial. For example, the small scale (1: 1,000,000) roster map image of New Zealand (provided at Land Information New Zealand) requires over 10 MB to store in compressed form. The storage size gets much larger as the scale of the map increases. The storage size can cause serious performance problems as larger size means longer transmission time and more memory requirements. This problem is compounded because the TP application was designed to run on portable devices with low memory and slow internet bandwidth. Therefore, one of our main goals for the Map Layer design is to minimize the storage size of map image data.

The storage size of a map image depends on the following three attributes:

1. Map scale: “Map scale means the ratio of units of linear measurement on a map to units of measurement on the earth.” [11] It is normally expressed as a ratio. (e.g. 1: 100,000) As map scale decreases, map image become less detailed and the storage size of the map image decreases as well.

The scale of a map image is fixed when it is printed on paper since the resolution of the map image remains the same. This map scale is what we refer to as the original map scale. However in a GIS, the scale of a map image does not need to be fixed, since the map image can be shrunk or enlarged to any size to fit the scale of the screen display (as shown in Figure 18). We refer to the scale of a map in GIS as ‘display scale’. There are three possibilities for display:

1) the original map scale is equal to the display scale;
2) the original scale is smaller than the display scale;
3) the original scale is larger than the display scale;

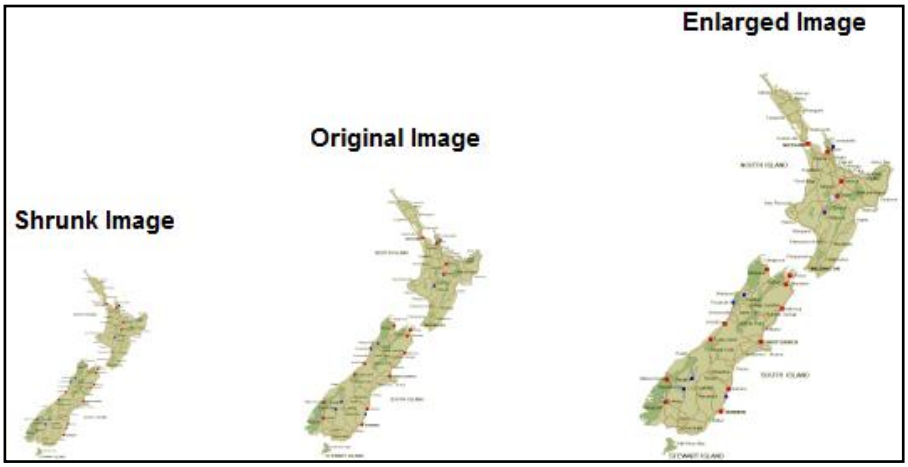


Figure 18: Same map image in different scales

When (a) applies, this map image looks exactly right and the storage size of the map image will be considered as the standard size. If (b) applies, the storage size of the map image is smaller than the standard size, but the detail of the map will be lost.

Figure 19 below shows the effect when displaying a map image at a much higher scale in the MapBean. The original scale of the map image (picture on the top) is 1:25,000 and its display scale in the MapBean is 1:8,000. As you can see, the details of the map are totally total lost.

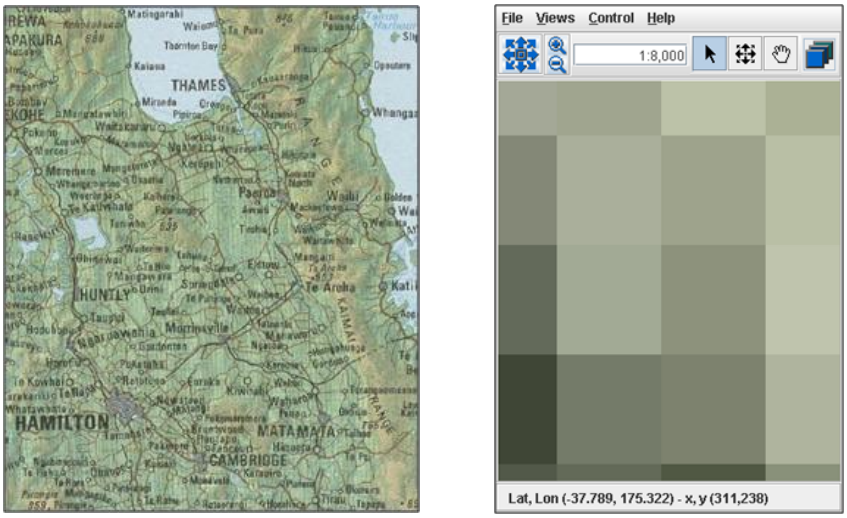


Figure 19: Effect of displaying a map image at a much higher scale

On the other hand, when (c) applies, the map image is merged into a small area and the map will become overwhelmed with detail. The storage size of the map image is then larger than the ideal size. This requires more memory requirement and longer transmission time.

Considering the above three conditions, we know that, using the map image with the same scale value as the MapBean projection will gives the best result for the map layer. However this is almost impossible as the scale of the MapBean projection can be changed arbitrarily by the users according to their need. Therefore the chance that the user preference will perfectly match the original map scale of the map image is very small. A compromise solution would be to make the map image scale as close as possible to the scale of the MapBean projection. More detailed information will be given later.

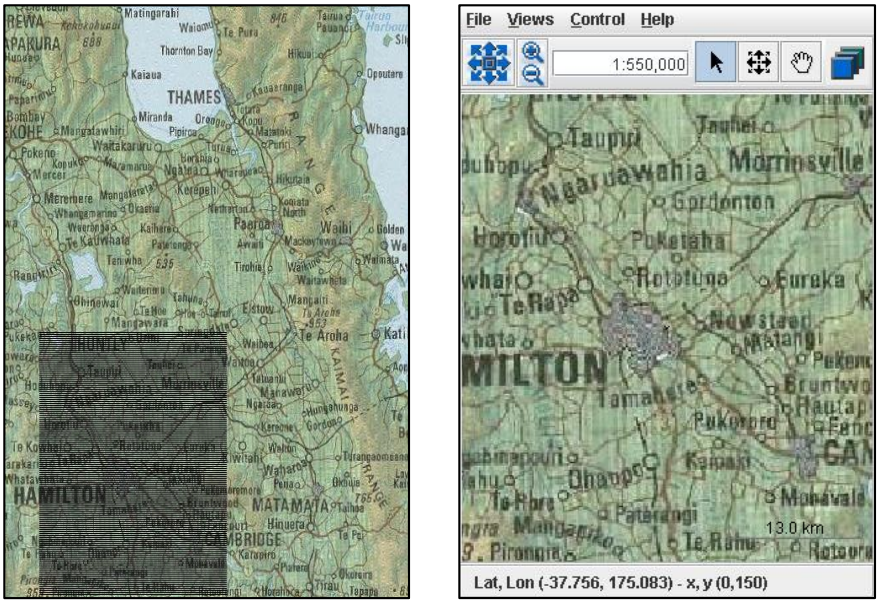


Figure 20: Displaying a map image at the same scale

2. Map projection: a map projection is a portion of the earth surface in the shape of rectangle. For a map image, if the map scale if fixed, the larger the map projection is, the more earth surface is shown in the image, and as a result, the storage size of the image increases.

If, when rendering a map image in the MapBean, the projection of the map image is larger than the projection of the MapBean, then only part of the image will be displayed on the screen. For example, in Figure 20 above, only the part of the map image within the dark rectangle is displayed in the MapBean. However in order to perform this display operation, the whole map image has to be loaded into the application, resulting in extra data transmission and memory requirement. On the other hand, when the projection of the map image is smaller than the projection of the MapBean, it will leave a blank area (an area not covered with the map image) on the MapBean. Again, the probability of a map image matching the MapBean projection is small. The closer the two projections are in size, the less data the system needs to deal with.

3. The storage size of map images also depends on the file format in which the images are saved. There are some approaches to minimizing storage size of map images based on image compression. For example, see “Compression of map images for real-time application” proposed by Pasi Franti in paper [10]. But this is our concern in this project. Compression image format JPEG is the only file format supported by the Map Image Layer.

Multi-scale representation

As mentioned above, in reality, the original scale and display scale of a map image are always different. The incompatible map scales causes the map image to lose detail and clarity. Other than that, maps of different scales represent the same area differently. Figure 21 below illustrates the difference in display of a map showing Nanjing (a city in China) at different scales. As can been seen from the four map images, the same area are represented extremely different. Therefore, it would not make sense to users if the original map image scale was far different from the scale of the MapBean’s projection.

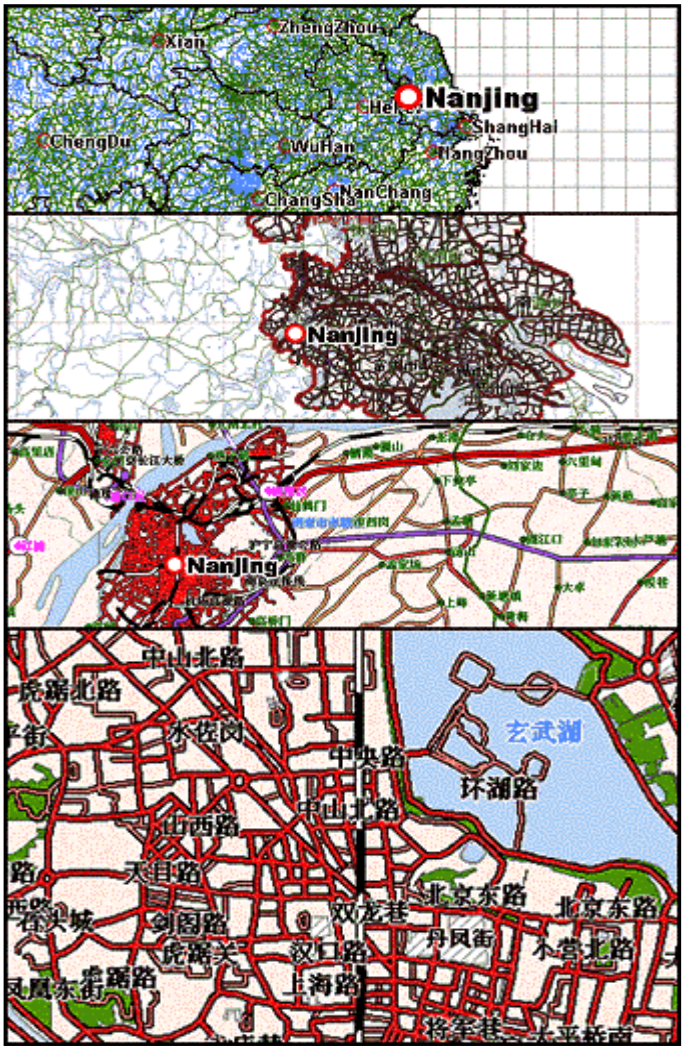
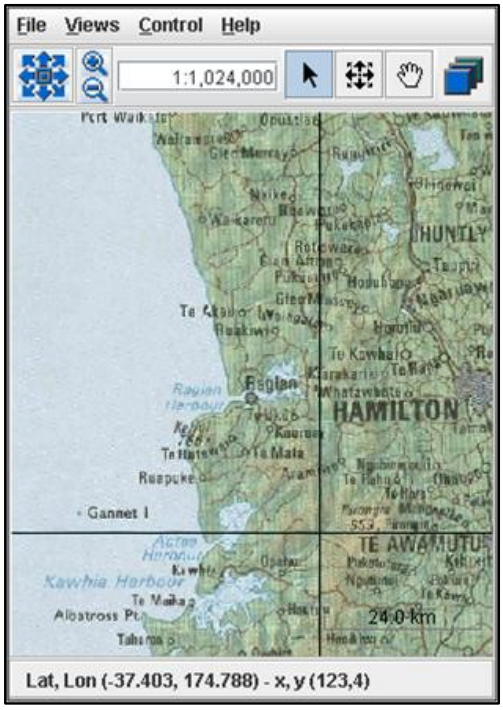


Figure 21: Example of a map shown in four different map scales in ascendant order

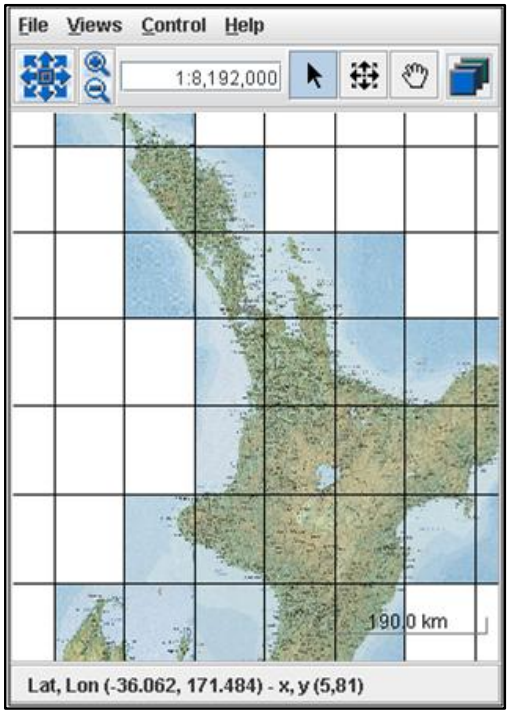
When a user uses the Travel Planning application, he/she may require maps at different scales based on different tasks. For example, the user may require a map at a relatively small scale for planning trips between cities or countries. The same user, when visiting a place and browsing the nearby places of interest will need a map which presents a larger scale. Therefore the Map Layer must have the ability to store maps at different scales and switch display among the different maps according to the scale specified by MapBean.

Multi-Brick Display

To solve the issues of storage size map image data and multi-scale representation, we implemented an algorithm solution for storing, retrieving and displaying map images, called, 'Multi-Brick Display'. This solution is widely used by many internet map providers (e.g. Google Maps, Yahoo Maps). The basic principles are that a big map image is divided into many image fragments where each image fragment is small in size. The map images fragments are catalogued and stored into different groups according to the scale of the original image. To obtain a map of an area at a certain scale, three steps have to be taken: 1) locating the image group whose scale is the closest in value to the specified scale; 2) Selecting all the image fragments which are necessary to cover the required area for from the located group; 3) reconstructing the map image for the area from the selected the image fragments.



An image composed of 4 image fragments



An image composed of 56 image fragments

Figure 22: The use of Image Fragments in TP application

The two screenshots in Figure 22 above show how image fragments are composed and displayed on in the Travel Planning application. The map image shown in the screenshot on the left is composed of four image fragments. The one on the right is made up of 56 images fragments. Note, that in the right hand screenshot some blocks are not being rendered (while areas) with map images. This is because the corresponding map image fragments are not available. In the following part of this section, we will provide implementation details and describe in detail how the algorithm is implemented in the Image Layer.

Image Group

In our system, map images (image fragments) are classified and stored into different image groups. Map images that belong to the same image group have the following attributes in common:

- Scale: The scale of the map image is determined by the amount of real-world area covered by the map image. Map Layer is responsible for determining the best-fit map image from a set of image groups according to a given map scale. The term "best-fit" means to find the value of the scale that minimizes the margin of the given map scale. The benefit of using best-fit images is that zooming (which reduces the display quality of the map images) can be minimized. For example, there are two map image groups A and B. The scale of A is 200,000:1 and the scale of B is 150,000:1. Suppose a user started the TP application. The scale of the map application (referred to as C) is currently set to 100,000:1. In this case, Map Layer will choose to use map image fragments from image group B to render maps on the application screen because the margin between B and C (50,000) is smaller than the margin between A and C (100,000). Furthermore, there cannot be two image groups of the same scale at the same time, as the scale is the only key to finding the best-fit image group.

- Span: This attribute specifies the distance coverage of a map image for both latitude and longitude in decimal degrees. For example, a span of value 1 means the latitudinal and longitudinal distances from the top left point to bottom right point are both 1 decimal degree. All the map images belonging to the same image group have an identical span value. However the resolution of each map image can be different horizontally

and vertically since the Map Layer will automatically stretch the map image for the best display.

- Decimal Places (d.p.): This attribute refers to the number of decimal places displayed in the latitudinal and longitudinal coordinates of the map. For example, if the d.p. value is “2” the coordinates (-37.788242, 175.31836) will be rounded to (-37.79, 175.32). This attribute is the key for image storing and retrieving; which will be described next.

Image Storing

Map images data used in the Map Layer can be stored at two places: either TIP server-side database or clients' local storage. Map Layer allows users to specify where map images should be loaded from. The advantage of loading images from TIP server is that it ensures the map images used are the most up to date. The disadvantaged is that but clients using low speed internet will suffer from slow loading time. To solve this problem, we also allow users to download an image package and place it in the resource folder of the Travel Planning application. In either case, a map image is stored based on its scale and the coordinates of the upper left corner.

In the TIP server, map images are stored in the central database, in table 'map_scale' and table 'maps'. Table 2 below shows the structure relationship of the two tables.

Table 2: Tables for storing map images

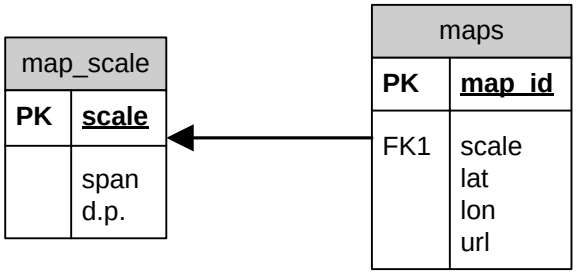


Table 'map_scale' stores all instances of image groups. It contains attributes scale, span, and d.p. (scale is its primary key). Table 'maps' stores information of all map image fragments. Its attribute scale is a foreign key referencing table 'map_scale'. The attributes lat and lon hold the latitude and longitude coordinates displayed at the upper left corner of each map image. The attribute url holds the URL to which each image is stored at the server.

When map images are stored at the client side, all the existing image groups must be firstly defined in the configuration file of the Travel Planning application. The configuration file is loaded and parsed by Map Layer upon execution of the system. Each map image group is encoded in the configuration file in the following format:

```
<ScaleToMap#> = 'Map Scale' ; 'Map Span' ; 'Map Precision'
```

<ScaleToMap#> is a unique key for this record. The term 'ScaleToMap' is a preserved word for Map Image Layer. It tells parser, the string that is about to follow which represents a source image group. '#' is the order in which the Map Image Layer's parse searches this record when looking for source image groups. It does not matter what order an image group record is dealt, but each record must have a unique number which starts from 1 and increment by 1 each time. The section after the equal sign specifies the values of the three attributes of the image group. Each value is followed by a semicolon to separate the clauses. The following example defines three map image groups for Map Layer.

```
# Declare Source Map Image Group Starts
image.scaleToMap1=1000000;1;1
image.scaleToMap2=30000;0.03;2
image.scaleToMap3=3000000;2;0
# Declare Source Map Image Group Ends
```

Map images stored in the client side source folder follow the following naming rule:

```
file_name = lat + "x" + lon + "." + image_suffix
```

Lat: rounded latitude coordinate of upper left corner

Lon: rounded longitude coordinate of upper left corner

Figure 23: Image naming example

In the above example map, the image (JPEG format) belongs to an image group whose d.p. value is 0. According to the image naming rule, the name of the image will be: (-37x175.jpg).

Image Retrieving

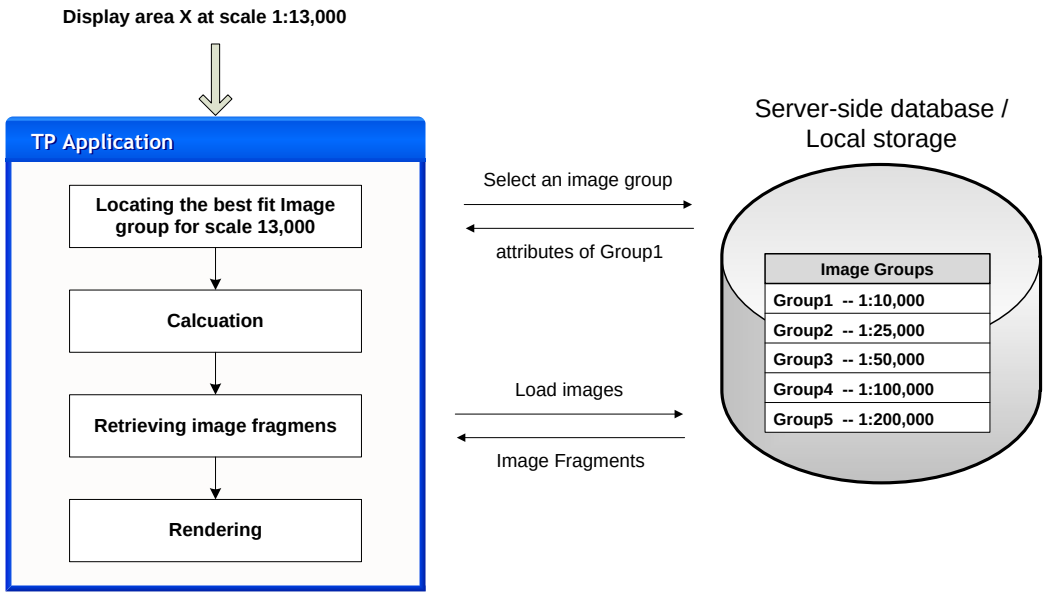


Figure 24: Processes for rendering a map image

Map Layer displays the map image according to the associated MapBean projection. When the projection changes, it updates and repaints the map image on the MapBean. Figure 24 shows the basic steps of retrieving and displaying map images inside Map Layer.

1) Locating the best-fit map image group

This step is simple as Map Layer only needs to traverse though all the existing map image groups (either defined in the database table or in the configuration file) and select the one whose scale is the closest to the scale of the MapBean projection. The formula below is used to calculate the minimum margin (α) between a given scale (s) and a set of scale values ($x_{1\to n}$).

$$\alpha = \min_{1\to n}(x_n - s)$$

2) Estimating the number of image fragments required to cover the MapBean projection

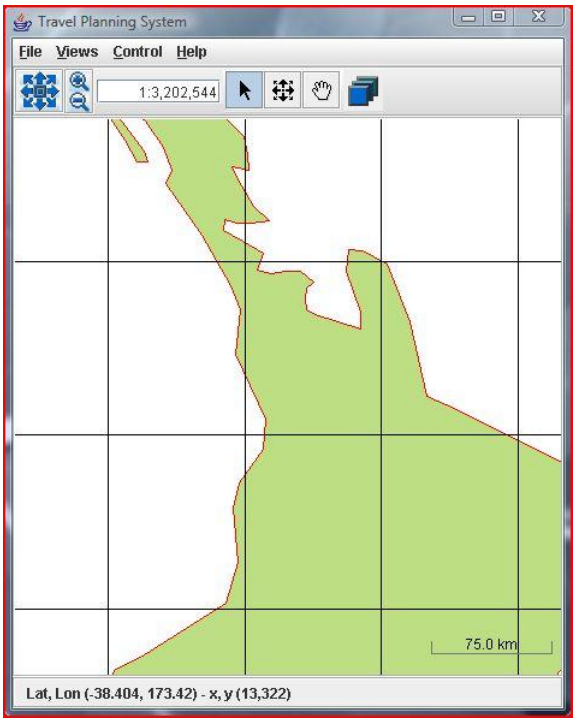


Figure 25: Estimating the number of image fragments

After the best-fit map image group is being confirmed, the next step is to calculate the total number of image fragments required to fully cover the MapBean projection area (horizontally and vertically). In the example shown Figure 25, in total 20 image fragments are required to render map image on the MapBean projection. The total number of image fragments can be calculated using the following formulas:

$$N_v = \left\| \left[\frac{lat2 - lat1}{span} \right] \right\| + 1$$

$$N_h = \left\| \left[\frac{lon2 - lon1}{span} \right] \right\| + 1$$

$$N_{total} = N_v \times N_h$$

N_v: The number of images required vertically.

N_h: The number of images required horizontally.

N_{total} : The total number of images required.

lat1, lon1: The coordinates (latitude and longitude) of the upper left corner.

lat2, lon2: The coordinates of the bottom right corner.

Span: The map span value of the target image group.

$[n]$: This means the the largest integer contained in the number n. For example, $[10.4] = 10, [112.9] = 112$.

3) Retrieving map images

After the number of image fragments has been estimated, the system starts retrieving images from the server or local source folder. In our system implementation, a map image fragment is retrieved based on its latitude and longitude coordinates of the upper left corner.

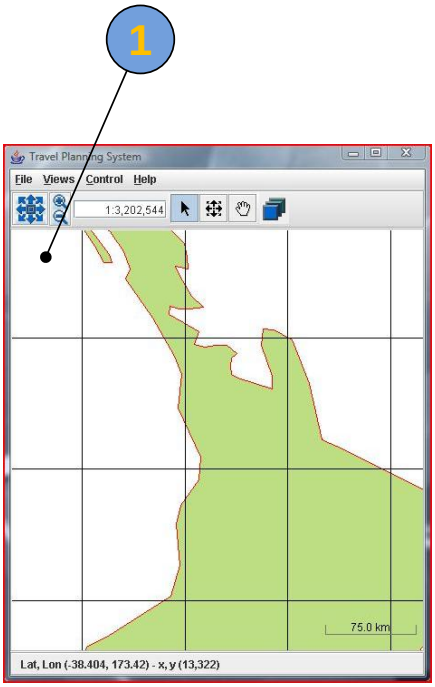


Figure 26: Retrieving map images

The coordinates of the upper left corner of the first image fragment (see Figure 26) can be calculated using the following formula:

$$Latitude_1 = Round\left(\left[\frac{lat}{span}\right] \times span\right) d.p.$$

$$Longitude_1 = Round\left(\left[\frac{lon}{span}\right] \times span\right) d.p.$$

lat, lon: The coordinates of the upper left corner of the MapBean projection.
Span: The map span value of the target image group.
d. p.: The decimal place value of the target image group.

After the coordinates of the first image fragment has been calculated, the rest can be easily calculated as follows:

$$Latitude_n = Latitude_1 + span \times n$$

$$Longitude_n = Longitude_1 + span \times n$$

The following (simplified) SQL statements are used to query the database for the corresponding URL address of the map image according its map scale (scale) and upper left corner coordinates (lat, lon):

```
SELECT url FROM maps m
WHERE  (
m.scale = scale AND
m.lat = lat AND
m.lon = lon
)
```

In addition, If Map Layer is set to use images stored in the resource fold of the TP application, the local address of the same map image described in the above case would be: `app/src/lat + "x" + lon + "." + suffix.`

4) Rendering map images

Finally, Map Layer adjusts (cuts and stretches) and paints the retrieved map images at the corresponding positions on the MapBean projection. Figure 27 shows the final result after the corresponding map image fragments have been painted on the MapBean projection.

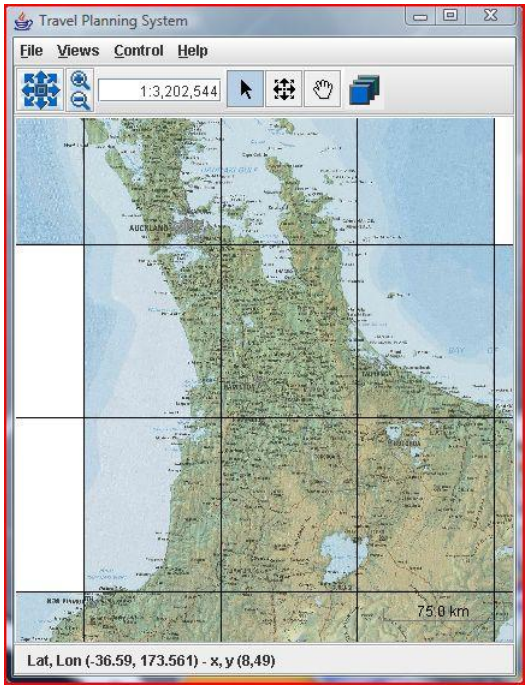


Figure 27: Final result after images been painted

Properties of Map Layer

Map Layer has three properties: load image from net, maximum images and draw edges. Figure 28 below is a screenshot of the Map Layer property window which allows users to modify the three properties at run time.

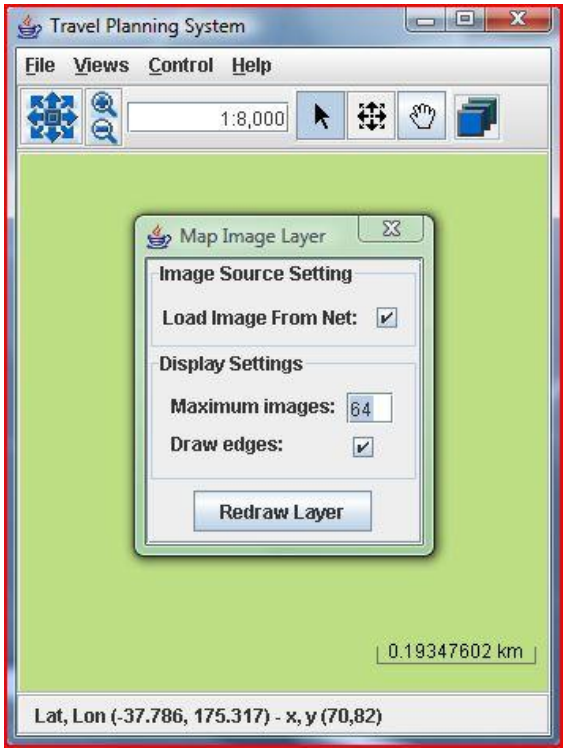


Figure 28: Properties window of Map Layer

The first property 'load from net' is a flag which tells the Map Layer where map images should be loaded from. Map Layer can use images either from the TIP server or from the resource folder of the TP application.

Property 'maximum images' sets a limit on the maximum number of image fragments that can be used when rendering a map image on the MapBean. The more image fragments are used, the longer the time required for processing image fragments and the slower the system becomes. The Map Image Layer will

not function when the number of image fragments needed exceeds the threshold value.

Figure 29: Outlines the Map Image Fragments

The third property ‘draw edges’ is a flag that controls whether the edges of the image fragments are hidden or visible. As shown in Figure 29 above, each map fragment is outlined with black lines to indicate how many image fragments are used in the map.

5.5.2 Sight Layer

The main purpose of Sight Layer is to provide user with a graphical view of sight data stored at TIP server-side database according to projection of the MapBean. Furthermore, it has the ability to act as a gateway of sight data for other layers.

Sight Information stored in Database

In the TIP server-side database, each sight record contains the following fields:

- **S_ID**: the unique id that identifies the sight.
- **Name**: the name of the sight.
- **Description**: the description of the sight.
- **Address**: the physical address of the sight.
- **Location**: the geographical location of the sight which is defined as the region within the sight area. It is stored in database in the form of polygon which is one of the basic geometry data type. A polygon is expressed as a sequence of points. e.g. 37.1, 175 ; 37.4, 175.1 ; 37.3, 175.2)
- **Sight Rating**: In TIP system, a sight can be rated by users who have visited it. The rating value is from 1~10, 10 being the highest. One sight can have many ratings from different users.
- **Sightgroup**: specifies a type of sight. (e.g. art, building, education, nature) A sight can belong to several sightgroups. For example, a library belongs to both building and education.

Note that there is other information associated with sights in the TIP database (e.g. picture_url, creator). They are not listed because they are used in Sight layer.

Sight Class

A Sight Class represents a sight (or a place of interest) stored in the TIP central database. Figure 30 shows the structure of the Sight class. The class has eight attributes which are directly associated with the data fields of a sight record, except attributes, which are CenterLocation and AVGRating.

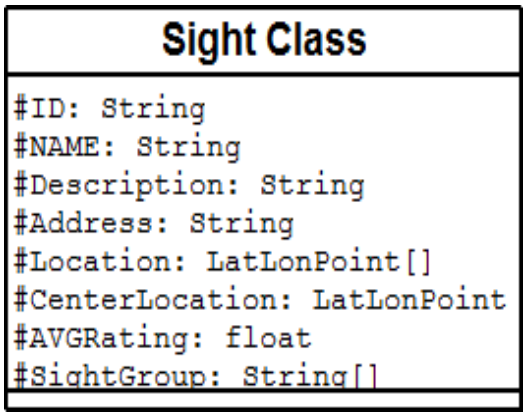


Figure 30: Structure of Sight Class

Attribute "CenterLocation" is calculated automatically by program according to the geometry data (polygon) defined in the location data field. It represents the geometrical center of the sight. Sights are display on the MapBean based on their center locations.

The displaying result for sight objects depends on two things: (1) characteristic of the sight objects, which means that different sight objects will be represented differently on the map; (2) user's preference, which means a user can control the displaying result based on his/her personal interest. Furthermore, the Sight Layer is also designed to be a gateway of sight data for other components, since it is the only component which can directly access to the sight and sight related data stored at the server-side database. All others components gain sight data through Sight Layer.

Multi-layer Representation

The relationship between the Sight Layer and a map application is one-to-many relationship. In other words, there can be more than one Sight Layer exist in a Map Application. Each of them can be set to only display one type of sight objects through the modifiable parameters of the Sight Layer. This architecture allows the application to separate different types of sight objects. So that each type of sight object can be represented on the screen in a different way. E.g. Different sizes, colors and shapes

The Sight Layer has a number of properties which are used to control the default operations and affect its performance. For the most part, these properties deal with sight selection. Users can explicitly define how a Sight Layer should work by modifying these parameters. The following list shows the most important parameters of the Sight Layer:

- Displaying parameters: The parameters that are used to specify the how sights should be represented on the layer.
- Sight Group parameter: This parameter defines a range of default sight groups. Sights belong one of the groups will be shown on the layer, otherwise they will be ignored.
- Sight rating parameter: This parameter defines the minimum rating threshold. Only sights whose average sight rating is higher than the threshold are visible on the layer.
- Performance parameter: The performance parameter decides the priority of memory allowance and application running speed. It can be set to either ‘memory efficiency first’ or ‘speed first’, depending on the number of records stored at database and the size of the main memory installed on the client system.

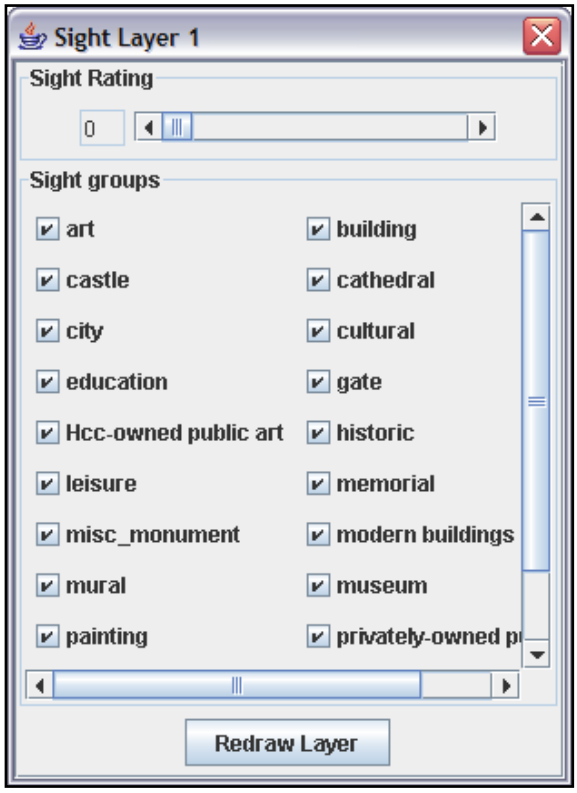


Figure 31: Sight Layer Property Window

The settings for these parameters are recorded in system property file and loaded at system execute time. It is possible to modify the values for sight rating and the sight group parameters at run time through the sight layer property window (as shown in above screenshot). Others are unable to change after the system has started.

The following example illustrates how these parameters can be used to improve the visibility of a Map Application. There are 20 sights objects within a fixed map projection. Half of them belong to sight group ‘build’ and another half of the sight objects belong to sight group ‘art’.

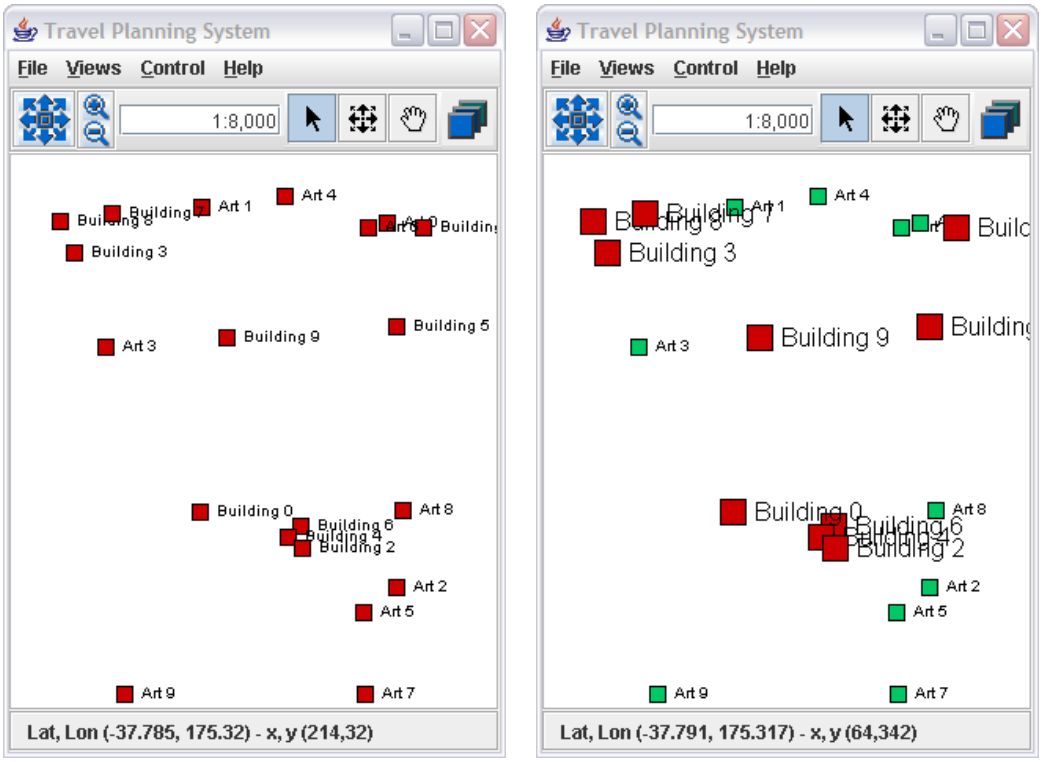


Figure 32: Example of Sight Layer

The first screenshot was taken from a map application which only contains one sight layer and the sight layer is set to display all kinds of sight objects. As a result, all sights are displayed using the same color and shape regardless the different sight groups they belong to. This makes it difficult for the users to distinguish different sights. On the other hand, under the same condition, we set up another map application which uses to sight layers A & B. Sight layer A is limited to only cope with sights which belong to sight group 'Building' and sight layer B only cope with sight group 'Art'. Each layer uses a unique color and size to represent sights. As can be seen in the second screenshot, the two types of sight objects now can be easily distinguished from their appearances. Compared to the first application, the second one is more straightforward for the uses. This example is just a simple case of using sight layers since only sight group parameter was used. It can be more complicated when other parameters are involved. For example, you can set three layers L1 L2 L3. L1 to display sights which have rating level higher than 5 and sight group belong to 'building' and

‘art’, L2 to display sight group ‘education’ and L3 to display sights which have rating level rating lower than 5.

The Performance parameter is very important to Sight Layer. It controls the way Sight Layer retrieves and stores sight data from database. As mentioned early, it can be set to either ‘memory efficiency first’ or ‘speed first’. In memory efficiency mode, a sight layer will establish a connection to database and retrieve sight data for those who are within the area of the current map projection and instance sight objects from the retrieved data, every time the map projection associated to it changes,. The advantage of this mode is that only those sight information stored at database which is displayable on the map projection is read into main memory. This is extremely important when the database contains a great number of sight records and the system running the map application does not have enough extra memory. However, frequently accessing to the remote database will be resulting in more time being consumed.

In the ‘speed first’ mode, a sight layer only access to database upon the first projection change. Instead of partially retrieving, all sight records will be read into the system main memory to be used for further projection change. Since all the necessary data are stored inside main memory, the program can be much faster in reacting to projection change event, except the first time projection change. The disadvantage is that more memory consumed to stored sight data. The difference of system performance in practical when applying two models will be examined later in this document.

5.5.3 Route Layer

Route Layer is the key layer of the Travel Planning Application. It is designed for use in planning travel routes and also gives recommendations of the nearby places of interest according to the geographical position. It provides many useful functions which are related to the concept of travel planning. For example, it has a function for locating points of interest which are within a certain distance of the user's physical position or a virtual route planed by the user.

Unlike the sight layer, there can be only one route layer existing inside a Travel Planning application at any one time. Route Layer also has access to TIP database to store or retrieve route data, however it does not retrieve sight data directly from the database, but through sight layers. Any sight layer created for an application will be automatically registered to the route layer (If there is one).

Similar to the other two layers, the route layer also has a number of properties that control its behavior. Figure 33 below is a screenshot of the Route Layer property dialog. It contains controls for the follow parameters:

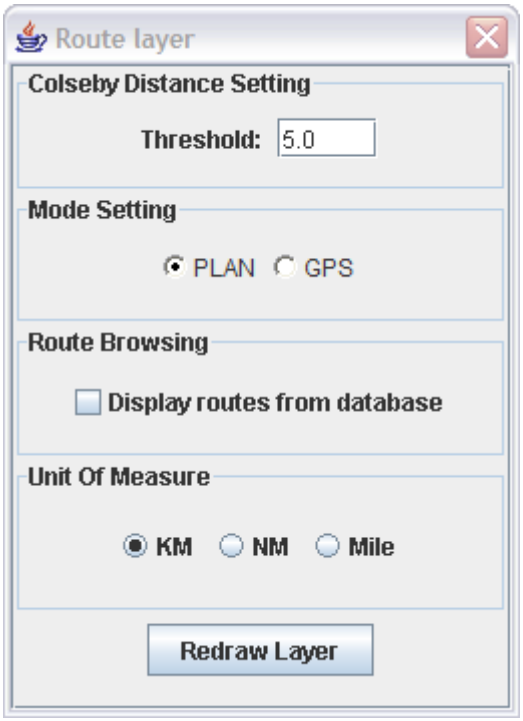


Figure 33: Route Layer Property Window

- **Closeby Distance threshold**: This threshold value is used to judge if a sight is a place of interest nearby. If the distance from the sight to a given location is smaller than the threshold, it will be considered as a closeby sight. Users can adjust the value of this parameter to meet their need.

- **Mode parameter**: The Route layer can run at two modes, 'PLAN' or 'GPS'. It is important to understand the difference between these two modes. By default, it runs at 'PLAN' mode which allows users to plan their trips on the layer. The 'GPS' mode should be used when a user is traveling in a certain area. When running in 'GPS' mode, the route layer listens to location change events from a GPS device attached to the same system. Based on the received location events it searches for sights which are located within the closeby distance threshold of the current position and displays them on the map.

- **Route Browsing:** If this option is enabled, the route layer will display routes stored at the TIP database which are plottable on the current map

projection. A route is composed of a series of route sections which are defined by two way-points. The route layer considers a route plottable if at least one of its route sections is within the projection. The figure below shows a route that contains three route sections: S1, S2, and S3. The route is plottable for Projection A, but not for Projection B as no route sections is completely inside the rectangle area of Projection B.

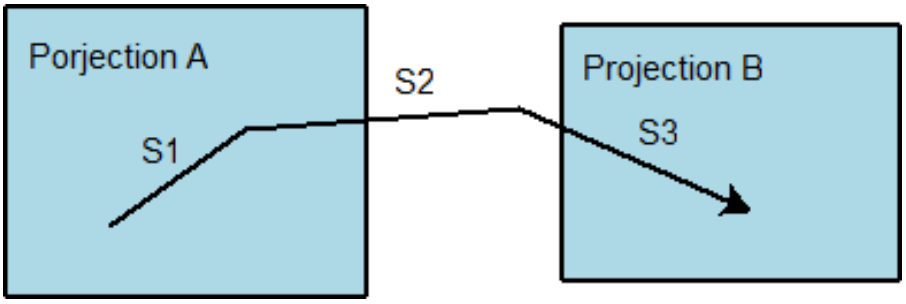


Figure 34: plottable route

This route browsing option should be only enabled when the map application is running in an environment that has an internet connection, otherwise the route layer will become extremely slow as it attempts to connect to the database for 10 seconds every time the map projection changed.

- **Unit of measure:** A Unit of measure is the actual unit in which the distance values are measured. It can be selected from one of three options: kilometer 'KM', nanometer 'NM' and mile.

Functions of Route layer

As mentioned at the beginning, the route layer provides many useful functions for users planning their trips. Next, we will give details of the most important functions of the route layer.

- ***Creating Travel Routes***

This is the fundamental function upon which all other functions of the route layer are based. It allows users to create virtual travel routes on the map projection by clicking on the position corresponding to their travel plan.

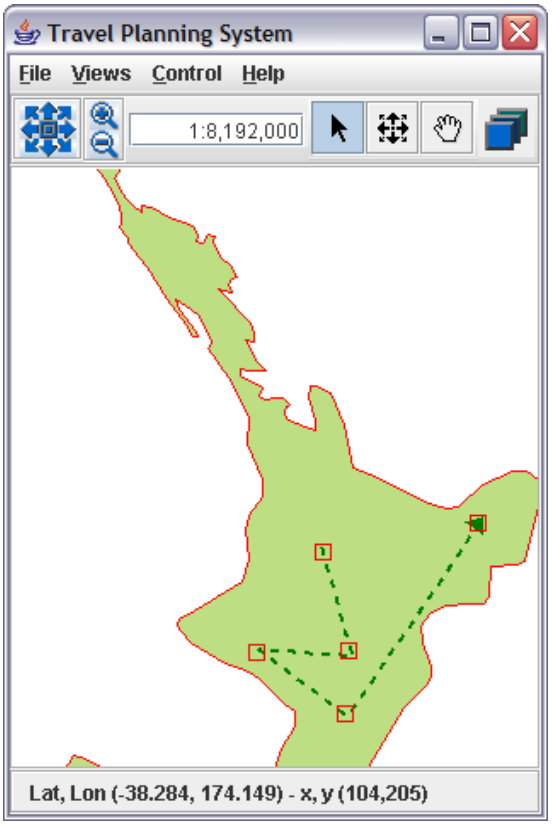


Figure 35: Creating a virtual route

As shown in the figure above, a virtual travel route consists of a series of waypoints (marked as red rectangles), it is displayed as a dashed polygon line.

An arrow is place at the last route point to indicate the direction. Similar to the sight layer, a set of parameters can be used to control the displaying of routes on the map. E.g. Colors, Dashed, Width.

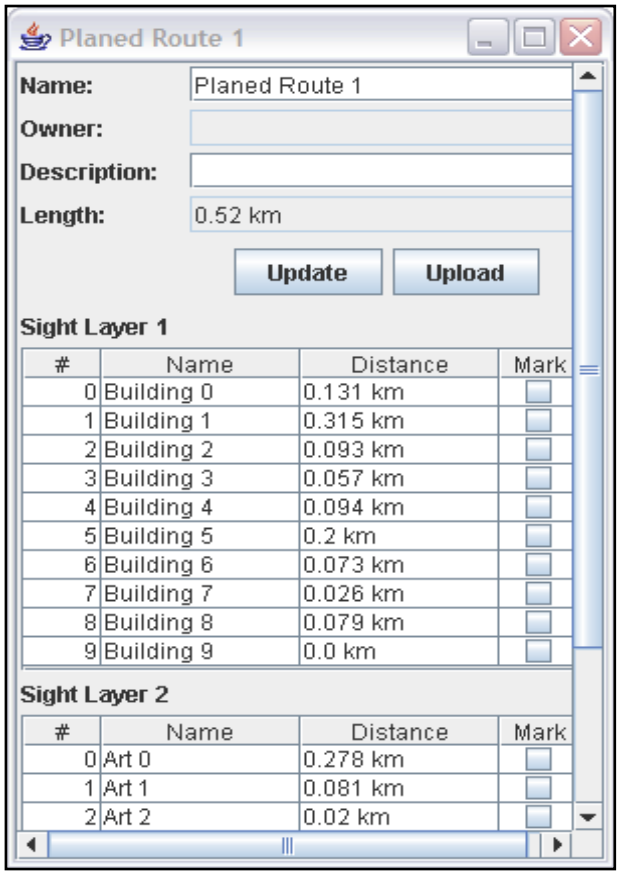


Figure 36: Property window of a virtual route

After a virtual route is created on the application map palette, users can click on it to open its property window. Figure 36 shows an example of a route property window. The top part of the window shows the detailed information of the route and two buttons. Button 'Update' is used to confirm the changes and button 'Upload' will start the procedure of uploading the route to the TIP database. There will be tables listing all the information about sights which are currently displayed on the map, including sight name and distance to the route. The number of tables depends on the number of sight layers registered at the route layer, one for each sight layer. Users can simply tick the 'Mark' column of a particular row to highlight the associated sight object on the map.

- ***Storing Routes***

Routes can be stored to local hard drive for the further use or uploaded to the TIP database so that they can be viewed by other clients. Uploading routes to database requires a valid username and password.

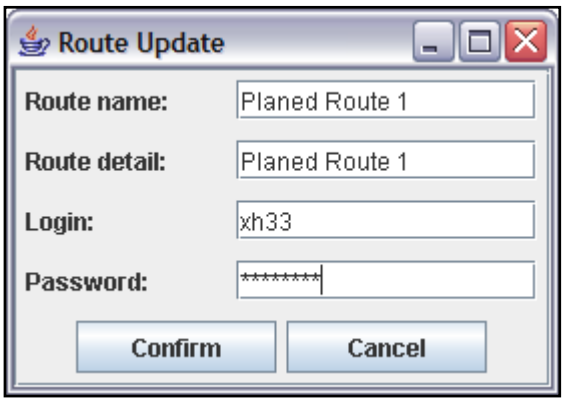


Figure 37: Upload route window

- ***Closeby Sight Recommendation***

In the Route Layer, sight recommendation means to recommend sights to clients based on their geographical locations. This is one of the main goals for developing this map application. The route layer provides two types of sight recommendation functions.

Recommendations based on a virtual route

The first recommendation function is used to give sight recommendation to the users according to the distance from a virtual route. Figure 38 below shows a simple example to illustrate how this works. We firstly set the sight rating parameter of the sight layer to 6, this means only the sights which have sight rating level greater than 6 will be directly shown on the application screen. As shown in the first screenshot, there are only three sights (Art0, Art4 and Art7)

that match the sight rating condition. These are shown on the screen. We then defined a virtual travel route which starts at sight 'Art 4' and ends at sight 'Art7'. The closeby threshold of the route layer is 2 km, which means that any sight with a distance of less than 2 km from the route is considered as a "closeby sight". Based on this setting, we applied the recommendation function to the route we created before.

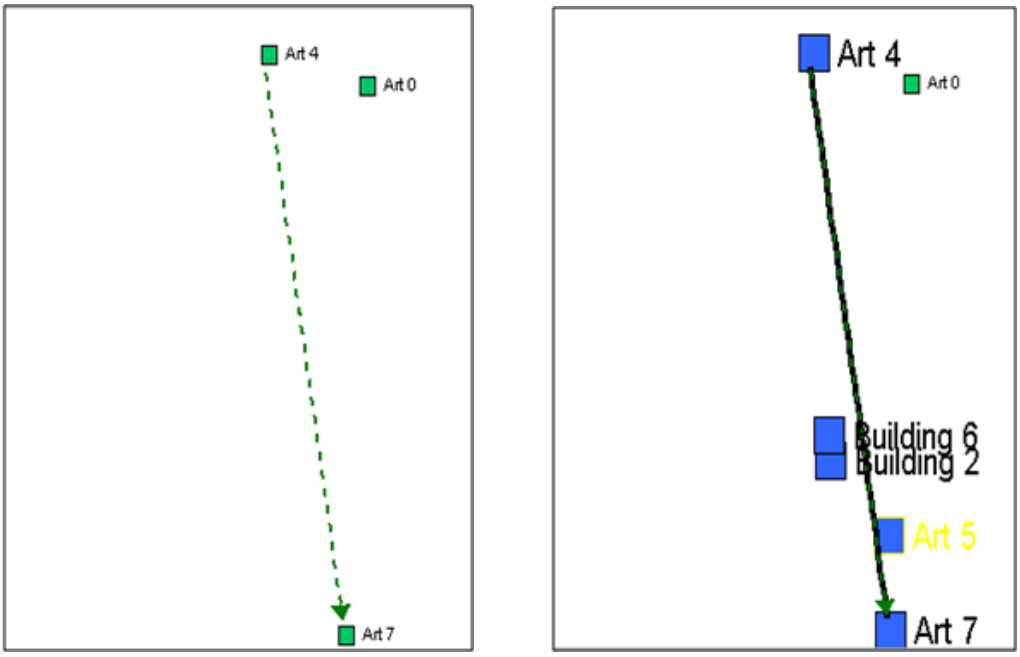


Figure 38: Sight Recommendation for Route

The results are shown in the second screenshot. There are two noticeable changes from the first screenshot. The first change is that some sights are displayed at a larger scale using a different color. These are the places of interest considered to the defined route. Sight 'Art 0' does not meet the closeby threshold condition, so it is displayed the same as in the first screenshot. The second difference is that three more sights have appeared on the screenshot. This is based on the rule of closeby sight recommendation defined in the route layer which is: ***All closeby sights should be displayed to users regardless what sight rating levels they got.***

The main calculation involved in recommendations for a virtual route is to find the distance between a sight and the route. The above case is simple because the route was one path segment (a straight line), so the distance from the route to the sight is the distance from the center point of sight to the perpendicular extension with the route (illustrated in Figure 39). When a route contains more than one path segment, the distance from a sight to the route is measured as the distance from the sight to the closest path segment. Therefore the more path segments are involved in a route, the more calculations are required to find the distance between the sight and the route.

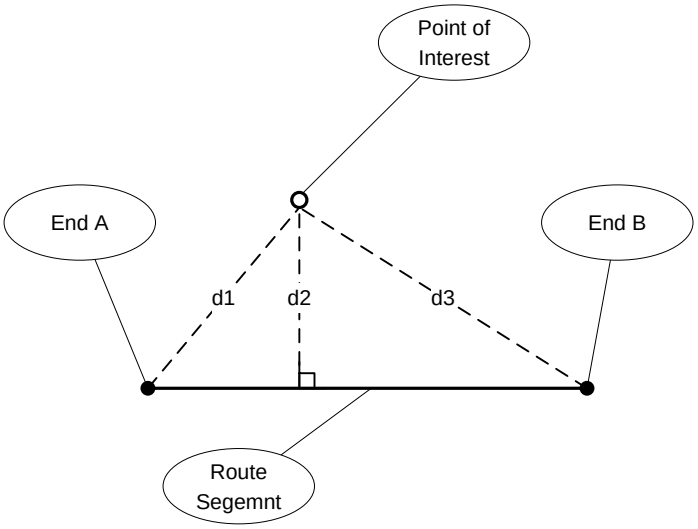


Figure 39: Distance from a sight to a route

Recommendations based on the GPS location

The second type of recommendation function is designed to give recommendations based on a user's real location, so that he or she can be aware of the nearby sights in relation to current position. When the route layer is running under GPS mode, the user's location will be displayed at the center of the screen. Based on this location, the route layer considers sights within the radius determined by the closeby threshold value property as closeby sights.

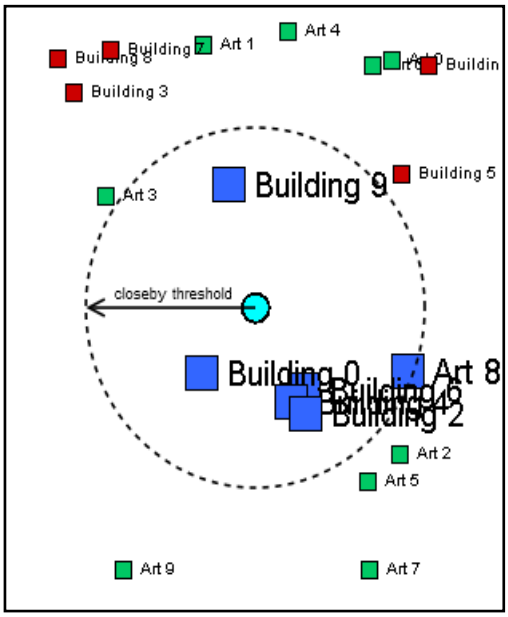


Figure 40: Recommendation for real locations

Figure 40 above is a screenshot taken from the route layer running under GPS mode, we added some marks to illustrate the principle of sight recommendation under GPS mode. A filled circle is positioned at the center of the screen which showing the user's current location. The dashed empty circle which center is the user's location and radius equals to the closeby threshold value is a marker we added to indicate the valid closeby area. All sights within the circle area are considered as closeby sights to the user's current location. As can be seen from the screenshot, sights that are within the dashed circle are labeled as closeby sights.

5.6 Interact with TIP

As mentioned in section 1.3, the TP application is proposed as a client-side service of the TIP system. In functional requirement 7, we analyzed that the TP application should be able to interact with other TIP services. In this section, we will describe how this is achieved in our implementation.

5.6.1 Form TP Application to TIP

In our current implementation of TP application, users can click on a sightseeing place mark to open up its detail information window. In the following example (see Figure 41), we first clicked at the 'Education Library' and opened up its information window. As can be seen from the image on the right, the information displayed in the window is limited.

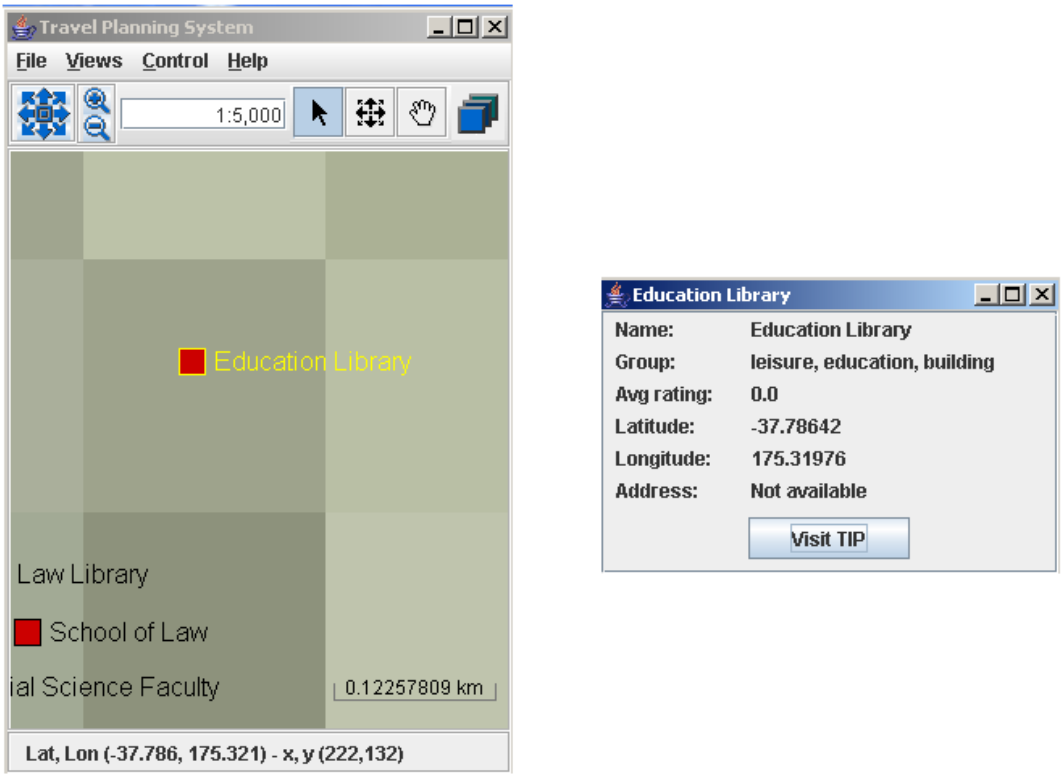


Figure 41: Link from TP Application to TIP Screenshot 1

If a user wants to see more information about this sight, he/she can click on the 'Visit TIP' button located at the bottom of the information window. After the button is clicked, TP application will fire up a browser and navigate to the corresponding TIP webpage of the sightseeing place. The following screenshot is the result we got from the above example. On the webpage, users can acquire

more detailed information on the sightseeing place, or access to other TIP server-side services. E.g. recommendation service.

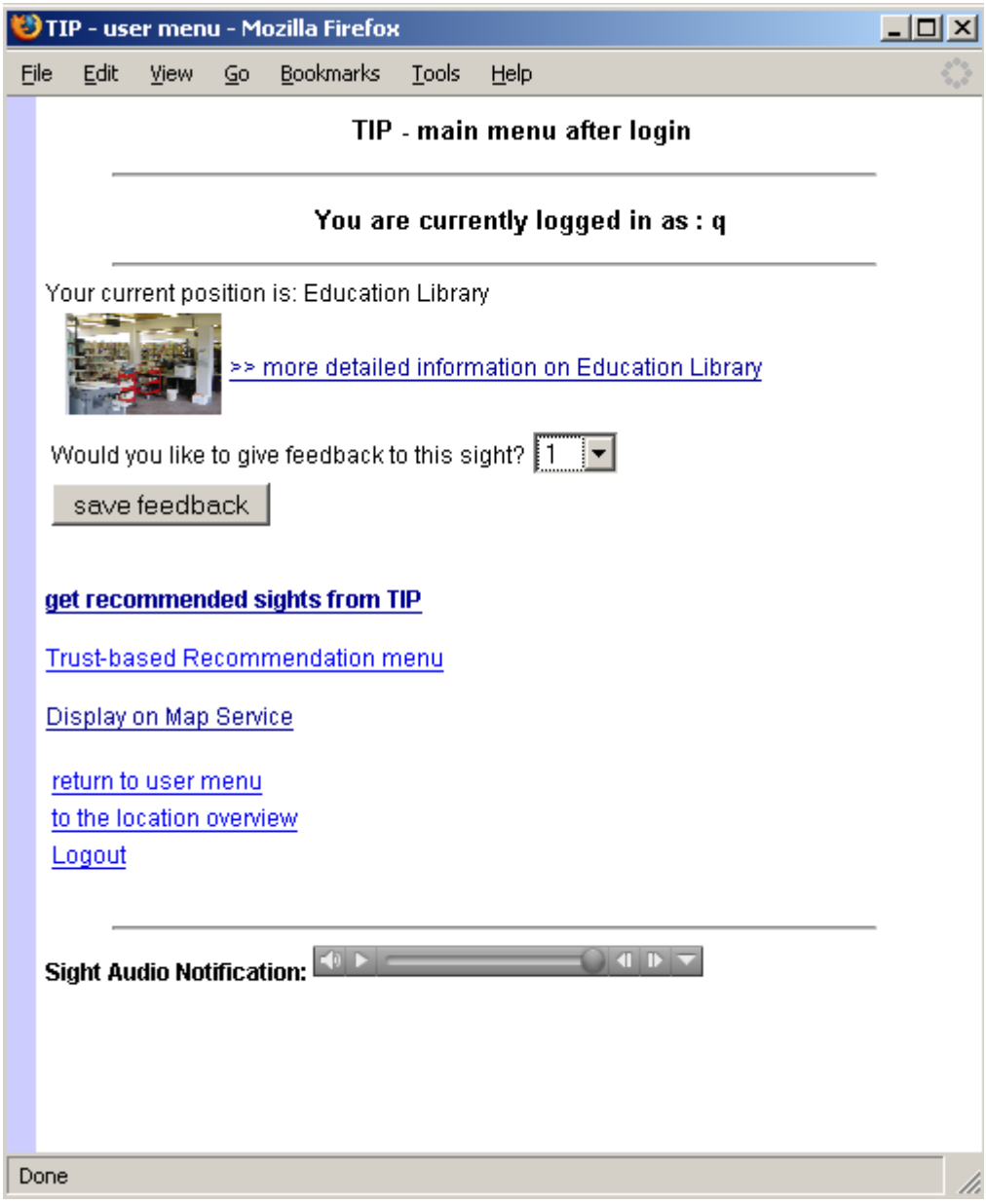


Figure 42: Link from TP Application to TIP Screenshot 2

Linking from TP application to TIP website is not complicated. As described in section 5.2, the base sight class contains a private attribute filed called 'ID' which represents the unique id associated with the corresponding sight record in TIP database. In TIP, each sight record has a unique URL which contains its ID. When required to link the corresponding webpage of a sight class instance, the

TP application uses the ID attribute to build the URL and send it to TIP server though a browser. Once the TIP server receives the request, it will parse the URL and display the related information on the sight in the browser.

5.6.2 From TIP to TP Application

Linking from TIP to TP application is more complicated. Our TP application contains implementations of network communication. When TP application executed, it create a ServerSocket bound to port 6666 on the local IP address of the client machine. After a client log on the TIP website (see Figure 43), his local IP address is stored in HttpSession. We can program to get the IP address from the HttpSession.

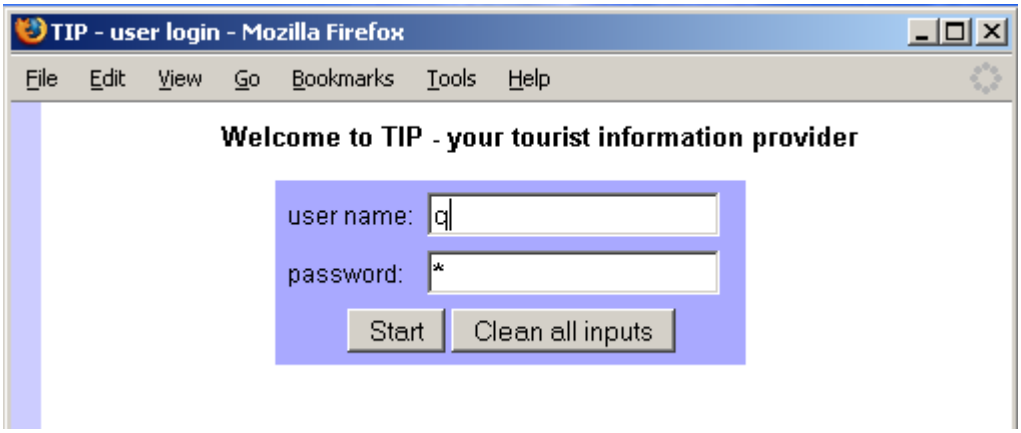


Figure 43: Link from TIP to TP Application Screenshot 1

As can been seen from the screenshot shown in Figure 44, there is a link 'Display on Map Service' on the webpage of every sight. When a user click on the link, TIP server will create a message which specifies the centre location (written in latitude and longitude coordinates) of the sight and send it to the client location IP address through port 6666. Once the TP application receives the message from TIP server, it parses the message and re-projects the map projection of its MapBean according to the parsed location.

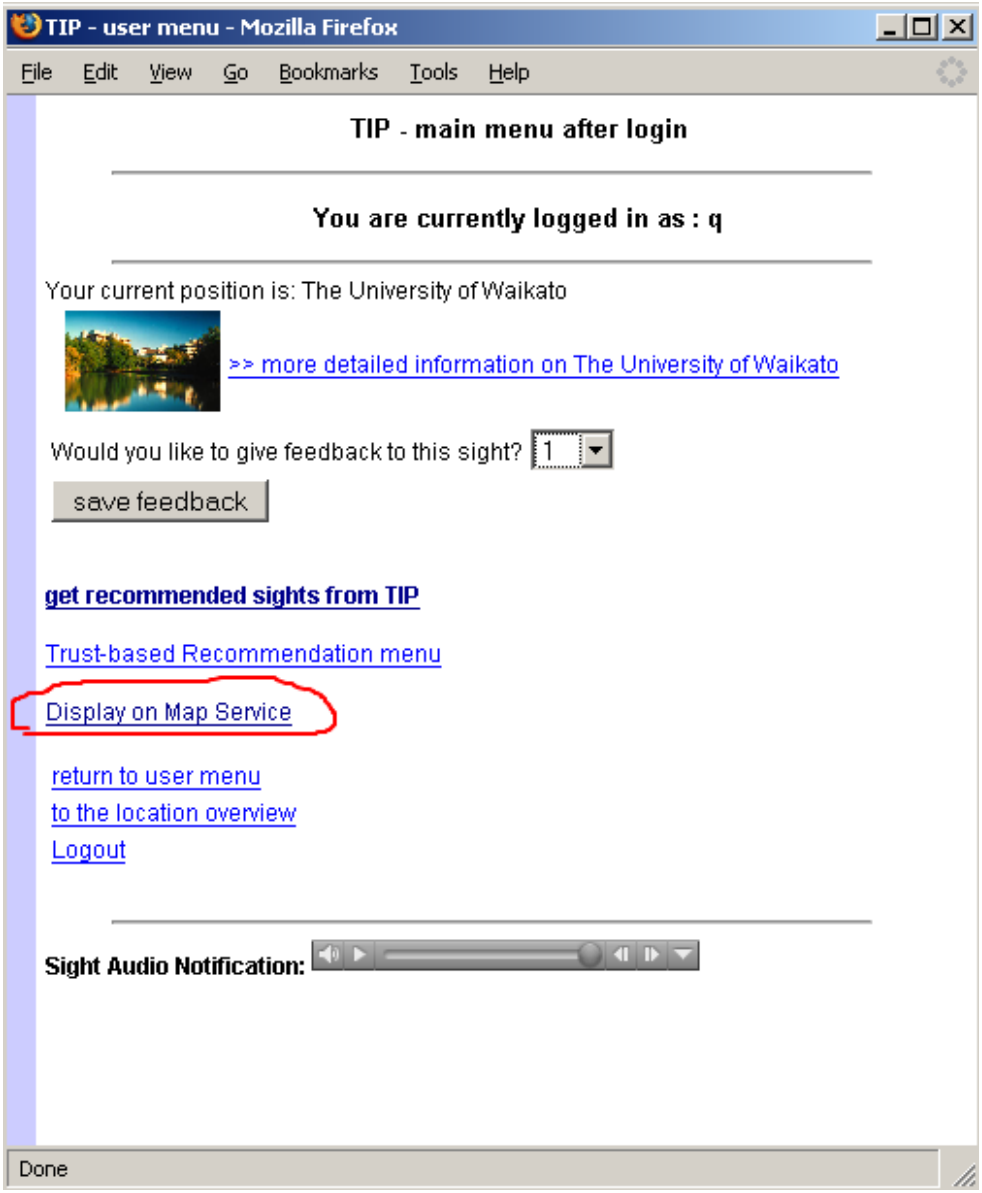


Figure 44: Link from TIP to TP Application Screenshot 2

The screen shot below shows the result after clicking on the link 'Display on Map Service'. The corresponding sight is displayed at the centre location of the TP application.

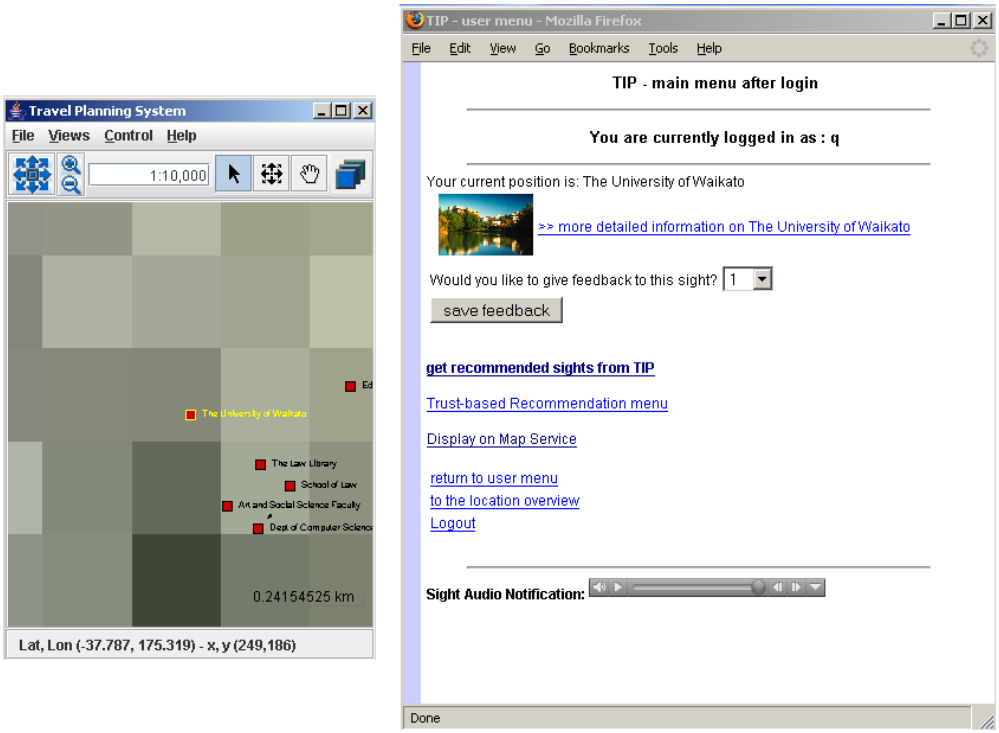


Figure 45: Link from TIP to TP Application Screenshot 3

Chapter 6 System Evaluation

In this chapter, we present the evaluation of the three layers implemented in the TP system and our finds.

After the implementation of the system, a series experiments had been conducted to evaluate the system performance. According to the layer architecture of the OpenMap framework, our system can be seen as a combination of a MapBean component and three customized layer components (sight layer, route layer and map layer). Each of the three layers is responsible for mapping and rendering one type of Geospatial data on the MapBean component. The performance of our TP system depends mainly on how fast the three customized layer components map and paint Geospatial data on the MapBean component. Therefore, we measure the system performance in the following three aspects:

1. How does the number of display graphics which represent sight-seeing places affect the system performance? (Section 6.1)
2. How does the route graphics displayed on route layer affect the system performance? (Section 6.2)
3. How does the number of map image fragments which are used cover the projection area of the MapBean instance affect the system performance? (Section 6.3)

In our performance tests, we do not consider the network impact such as network delay and the delays with which clients attend to connect to server-side database are factors that influence in the overall system performance. All the performance tests were performed the follow system environment:

- Processor Speed - Intel Pentium 1.66GHz
- Memory(RAM) - 512 Mbytes
- Hard Disk - 60G (30G free)
- Graphic Card - Intel GMA 950
- Operation System: Windows XP Professional SP2
- Java Virtual Machine (JVM) 1.5
- OpenMap Library 4.6.2

6.1 Performance Test 1: Sight Layer

Performance Test 1 illustrates the effect of increasing number of sight graphics (N_S) on the rendering performance of the MapBean component.

Hypothesis 1: We predict a linear increase in the time the system takes to rendering the projection of the MapBean upon a projection change event, along with the increasing number of displayed sight graphics. This is because that when Sight Layer receives a projection change event from its associated MapBean component, it firstly loads all the projectable from text-based source data and then constructs a list of OMSight instances (see Section 5.4) where each one represents a sight-seeing place. After the list is created, Sight layer maps and paints the OMSight instances from the list one by one on the MapBean's projection area. Since Sight Layer does not constructs and paints OMSight instances in parallel, but one after another and the time required to perform construction and painting operations for every sight instance should be

constant, therefore the total rendering time should increase in linear with the number of sight instances.

To prove hypothesis 1, we preformed the following test. First, we created a group of random data sets. Each data set contains a different number of sight instances, from 100 to 1,000. We let the system run with these data sets in turn and recorded the total time it took the system to render each data set. We run each test three times and took the average as the final result. The following graph shows our test results. The horizontal axis represents the number of displayed sight instances, and the system rendering time is show on the vertical axis. Line 'series1' represents our test results. The dashed line swings through line 'series' is the trend line derived from hypothesis 1.

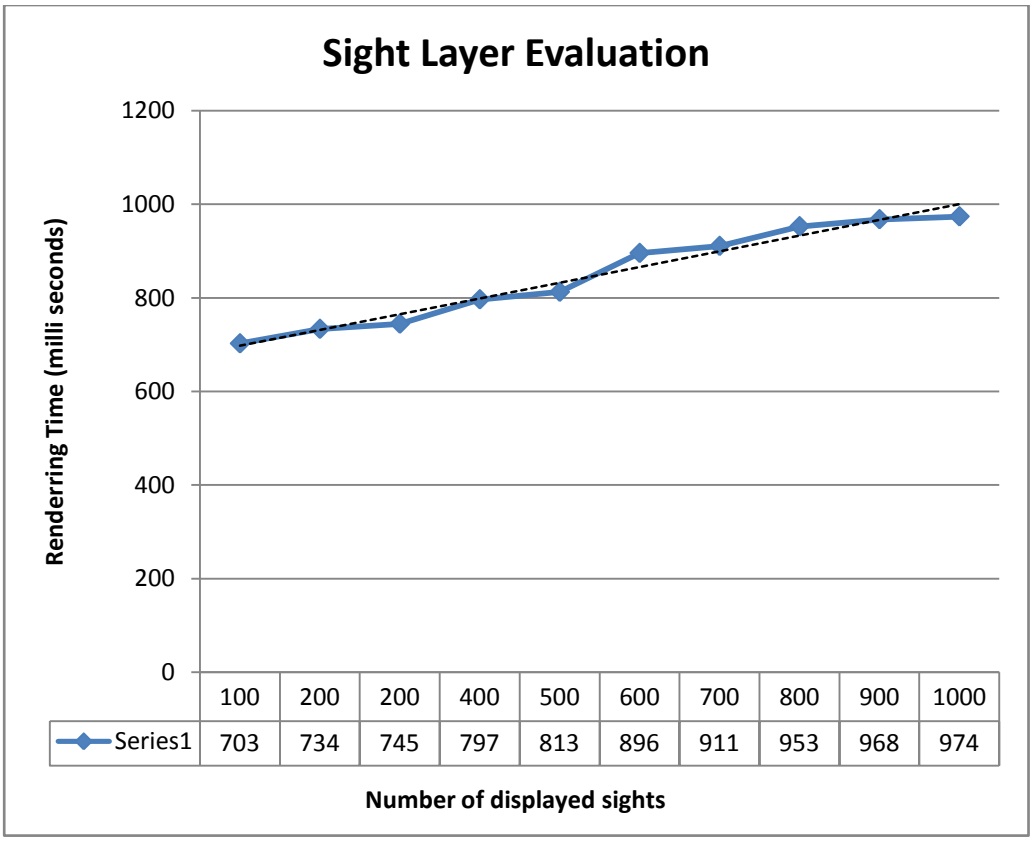


Figure 46: Performance Test 1 - Sight Layer

As you can see, our results indicate that the hypothesis 1 is basically correct. The growth rate of the rendering time for MapBean is close to linear. One

noticeable observation regarding the above figure is that the result line is somewhat fluctuated. It could be caused by measurement error. In java, the minimum time unit is one millisecond, which means the time below one second is ignored. The ignored time would have an influence the test results.

6.2 Performance Test 2: Route Layer

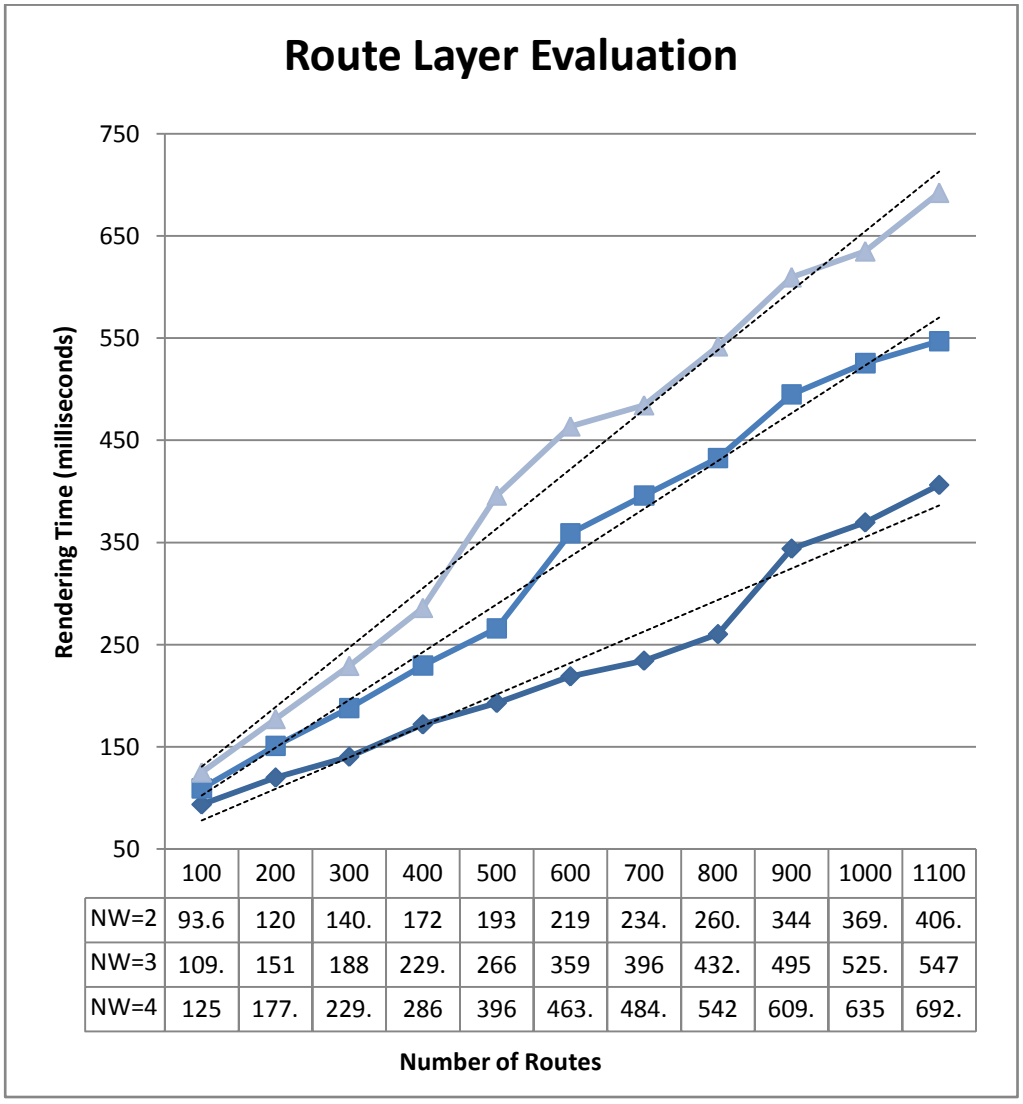
In Performance Test 2, we are going to show how Route Layer affects the system performance. Route Layer is more complex than Sight Layer as a route is defined by a series of waypoints. When rendering routes data on the MapBean's projection, not only the number of displayed routes (N_T), but also the number of waypoints contained in the routes (N_W) will have influences on the system performance.

Hypothesis 2: if N_W is fixed, we expect a linear increase in the overall rendering time as N_T increases. The structure of Route Layer is very similar to Sight Layer. In responding to a ProjectionChange event, it first builds a list of graphics from source data, and then paints them one by one on the MapBean's projection. The only difference is that the graphics built by Route Layer are OMRoute (see section 5.5.3).

Hypothesis 3: We predict that the lager Nw of a route, the longer the systems takes to construct OMRoute and paint it on the MapBean's projection. As described in section 5.5.3, in our system a route is defined by no less than 2 waypoints. It is drawn as a rubber-band from the first waypoint to the last waypoint. The more waypoints contained in a route, the more line needs to be drawn, and the more time it takes the system to render the route.

In performance test 2, we firstly create three groups of random data sets. The three groups are divided by Nw, which means all routes belong to a group has

the same Nw value. Nw of group one equals to 2, Nw of group two equals to 3, Nw of group three equals to 4. Each data set contains a different number of route instances, from 100 to 1,100. Similar to what we did in performance test 1, we let the system run with the different data sets in three times and take took the average as the final result.



	100	200	300	400	500	600	700	800	900	1000	1100
NW=2	93.6	120	140.	172	193	219	234.	260.	344	369.	406.
NW=3	109.	151	188	229.	266	359	396	432.	495	525.	547
NW=4	125	177.	229.	286	396	463.	484.	542	609.	635	692.

Figure 47: Performance Test 2 - Route Layer

Figure 47 illustrates the test results from performance test 2. The horizontal axis represents N_T, and the system rendering time is show on the vertical axis. Three different kinds of lines represent our test results from the three different groups. As can be seen, all these result lines are close to linear with only minor

fluctuations. It proves that that our hypothesis 2 is correct. Furthermore, when dealing the same amount of routes, it took more time to render the group which has a higher Nw value. This result matches our hypothesis 3. Also note that the higher the N_W value is, the faster the linear growth rate becomes.

6.3 Performance Test 3: Map Image Layer

In the following performance test, we will exam how Map Image Layer affects the system performance. According to the rendering mechanism of Map Image Layer, maps are painted using map image fragments. Thus, in this test, we will focus on testing the effect of the increasing number of map image fragments (N_F) on the rendering performance of the MapBean component.

Hypothesis 4: We expect that the time needed for rendering map image is depending on the number of the map image fragments used to compose the map. The more map image fragments are used, the longer time becomes. The plot of rendering time versus N_F should be a linear line. As mentioned in section 5.5.1 , it takes three steps to render map image on a given MapBean's projection: 1) Choose a best map image group which fits the scale of the projection; 2) Calculate the total number of image fragments required to cover the whole projection area and the start location of the upper left corner image fragment. 3) Load and paint the image fragments at the corresponding position on the projection one by one. The time it takes to perform the first and second step should be constant. However, the time spend on the third step is depending on the value of N_F. If all image fragments have the same size, then the time consumed for rendering each image fragment is the same. Thus the total rendering time should increase linearly with N_F.

To prove our hypothesis 4, we preformed the following test. We let the system run with 11 different scales. Each scale requires a different number of map

image fragments. We run each test three times to calculate and record the average rendering rime. During the test, the screen resolution of the MapBean's projection is fixed at 320x400. To eliminate effect of different image size, we chose to use the same image fragment for all images fragments. The result of the test is shown in the figure blow. The horizontal axis represents N_F, and the system rendering time is show on the vertical axis. Line 'series1' is plotted by N_F versus rendering time.

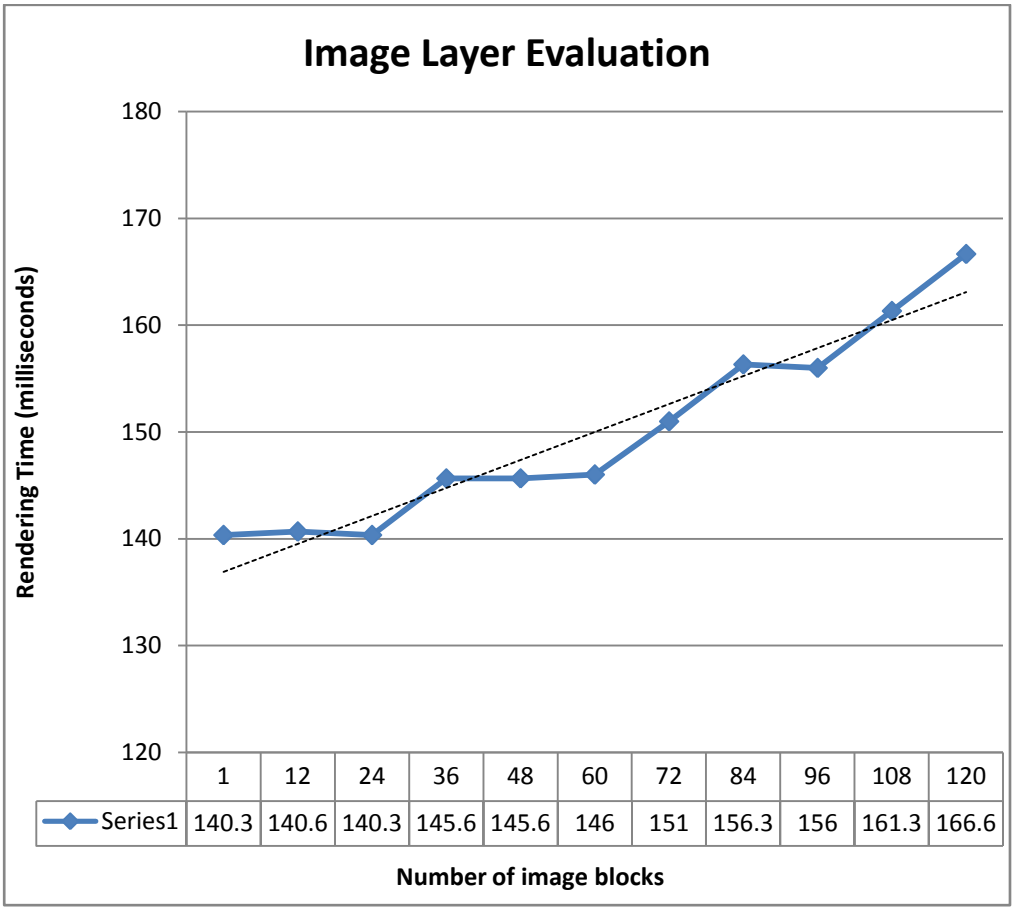


Figure 48: Performance Test 3 - Map Image Layer

As can been seen from result line, the growth trend is nearly linear. It proves that that our hypothesis 4 is correct. One again, the result line is fluctuated due to the measurement error described in performance test 1.

6.4 Summary

In the evaluation, we have performed three performance tests. In the first and second test, we have examined the impact on the system performance with different amount of sights and router displayed on the MapBean's projection. In the last performance, we have compared the map rendering speed of Image Layer when different numbers of map image fragments are used. In general, our TP system has done a good job of mapping and rendering the three different types geospatial data on the MapBean component. All the three layers have fast rendering speed.

Chapter 7 Conclusion & Future Work

7.1 Conclusion

This thesis presents an implementation of a travel planning system using GIS technology. The system was implemented based on the Openmap GIS framework and Java runtime environment (JRE) 5.0. It will run on any platforms which java 5.0 or later. Our system was implemented as a client-side service of the Tourist Information Provider (TIP) system (see section 1.2). It allows TIP users to edit, view, and upload geographical data to the TIP server-side database. Furthermore, the system also provides functions which allow users to plan their trips by defining a series of waypoints on the map and give travel recommendations based the planed waypoints.

An evaluation of the systems is presented in Section 6. We individually tested the effect of the sight, route, and image layer to the system rendering performance. In general, all the three layers have a linear increase of the rendering time with increase number of displayed graphical objects.

7.2 Future Work

The current implementation can be improved in several aspects. First, in the current system was implemented using Openmap GIS library as when we first

stated implementing the system, Google Maps API was just published and has a number of limitations. e.g. No geocoding support. But there had been a lot of improvements in the functions of Google Maps API during the time we were implementing the system. Now Google Maps API seems to be a much better choice as: 1) it contains implementation similar to our map image layer and the map images used in Google Maps are way better than any of the map image resources that can be found on internet; 2) the web-based GIS architecture makes it possible to integrate map functions into TIP webpage; Secondly, the administrator functions are not implemented in the current system. For example, add or modify public data stored at the TIP server-side database. Lastly, the Openmap library requires Java 1.4 or later version. It is ok for desktop PC, but for mobile devices, the higher version available nowadays is Java 1.2. Therefore we could not run and test our system on any mobile devices. The purpose of implementation of the system is to prove that the travel planning model we proposed would work. We hope in the future, when java 1.4 is supported on mobiles devices, someone could run and test the system again on mobile devices.

Bibliography

1. **A. Hinze, P. Malik, R. Malik.** Interaction Design for a Mobile Context-Aware System Using Discrete Event Modelling. University of Waikato, Hamilton. 2006.
2. **BBN.** OpenMap Developer's Guide. OpenMap(tm). [Online] BBN Technologies, 2006. http://openmap.bbn.com/. Last Accessed (2006.8).
3. **A. Hinze, A. Voisard.** Combining Event Notification Services and Location-based Services in Tourism. Freie Universitat, Berlin. 2003. Technical Report tr-b-03-06.
4. **K. Loeffler.** User Adapted Information Delivery in Context-Aware Systems. Freie Universitat, Berlin. 2004. Master Thesis.
5. **Y. Michel.** Location-aware Caching in Mobile Environments. Department of Computer Science, Waikato University. 2006. Master Thesis.
6. **P. Ottlinger.** Design and Implementation of an extensible Software Architecture for Distributing Context-sensitive Information. 2004. Master Thesis.
7. **S. Junmanee.** Design, Implementation And Evaluation of Advanced Recommendation Models in The Mobile Tourist Information System TIP. Department of Computer Science, Waikato University. 2006. Master Thesis.
8. **X. Gao.** Design and Implemetation of a Greenstone Service in a Mobile Tourist Information System. Department of Computer Science, University of Waikato. 2005. PGDipCS Project Report.
9. **Q. Qiu.** Trust-based Recommendations in a Mobile tourist Information System. Department of Computer Science, University of Waikato. 2006. Master Thesis.

10. **P. Firanti, E. Ageenko, P. kopylov, S. Grohn, F. Berger.** Compression of map images for real-time application. Department of Computer Science, University of Joensuu. 2001. Report A-2001-1.
11. **M. Hussain.** Explore Hyderabad - An Interactive Web-based GIS application Prototype. Department of Computer and Information Science, Linkopings University. 2006. Master Thesis.
12. **C. Streng.** Mobile GMaps. [Online] 2006. http://www.mgmaps.com/. Last Accessed (2006.8).
13. **GARMIN.** GARMIN QUE. GARMIN ON THE ROAD. [Online] 2006. http:// www.garmin.com/. Last Accessed (2006.4).
14. **Newman.** Des Newman's OziExplorer. Des Newman(tm). [Online] 2006. http://www.oziexplorer.com/. Last Accessed (2006.4).
15. **Google.** Google Maps. [Online] 2006. http://maps.google.com/. Last Accessed (2006.8).
16. **Microsoft.** Windows Live Local. Live Search. [Online] 2006. http://maps.live.com. Last Accessed (2006.8).
17. **Yahoo.** Yahoo Maps Beta. [Online] 2006. http://maps.yahoo.com/. Last Accessed (2006.8).

Appendix A: Database Structure of TIP

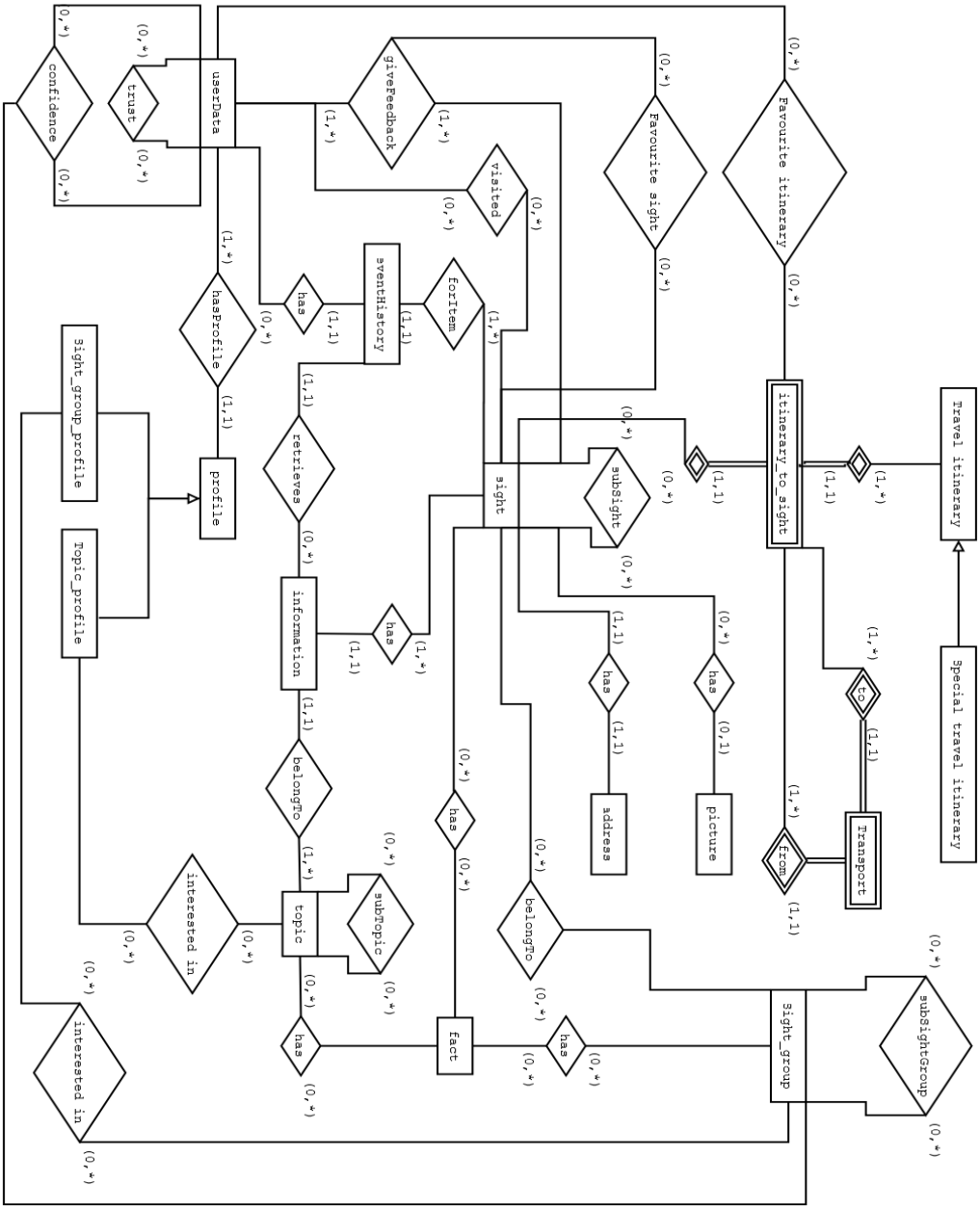


Figure 49: Entity-relationship-model of the TIP

Appendix B: TP Application Conf File Example

```
#---------------------------------------------------------------

# Start of properties file for TravelPlanning Application
#---------------------------------------------------------------

# MapBean settings

# Scale of the MapBean (1:scale)
scale=8000f
# Center latitude(float degree)
latitude=-37.788f
# Center longitude(float degree)
longitude=175.319f
# Background color
bgcolor=FFFFFF

# Layers to load and show on the map
layers=route sight1 sight2 scale image political

# Political layer settings
political.class=com.bbn.openmap.layer.shape.ShapeLayer
political.prettyName=Political Solid
political.visible=true
political.shapeFile=data/shapes/dcwpo-browse.shp
political.spatialIndex=data/shapes/dcwpo-browse.ssx
political.lineColor=ff0000
political.fillColor=BDDE83

# Scale layer settings
scale.class=com.bbn.openmap.layer.ScaleDisplayLayer
scale.prettyName=Scale Layer
scale.lineColor=ff777777
scale.textColor=ff000000
scale.unitOfMeasure=km
scale.locationXoffset=-10
scale.locationYoffset=-20
scale.width=100
scale.height=10
```

```
# Route layer settings
route.class=nz.ac.waikato.isdb.tip.client.map.layer.RouteLayer
route.prettyName=Route layer
# The unit of measurement.
# Possible values are km, nm, mile.
route.unitOfMeasure=km
route.closeByThreshold=5
route.localRouteColor=007A00
route.dbRouteColor=660066
route.tempRouteColor=FFFFFF
route.selectColor=FFFF00

# Sights layer settings

sight1.class=nz.ac.waikato.isdb.tip.client.map.layer.SightLayer
sight1.prettyName=SightLayer1
sight1.resourceFile=data/sights.dat
sight1.feedbackRating=0
sight1.basicScale=8000
sight1.basicRadius=5
sight1.maxRadius=10
sight1.loadFromFile=false
sight1.fastMode=false
sight1.fillColor=CC0000
sight1.lineColor=000000
sight1.selectColor=FFFF00
sight1.highlightColor=3366FF
#sight1.sightgroups=building
sight1.sightgroups=
sight1.closebyRadius=10

sight2.class=nz.ac.waikato.isdb.tip.client.map.layer.SightLayer
sight2.prettyName=SightLayer2
sight2.resourceFile=data/test_sights.dat
sight2.feedbackRating=0
sight2.basicScale=8000
sight2.basicRadius=5
sight2.maxRadius=10
sight2.loadFromFile=true
sight2.fastMode=false
sight2.fillColor=00C864
sight2.lineColor=000000
sight2.selectColor=FFFF00
sight2.highlightColor=3366FF
sight2.sightgroups=art
sight2.closebyRadius=10

# Map image layer settings
image.class=nz.ac.waikato.isdb.tip.client.map.layer.MapImageLayer
image.prettyName=Map Image Layer
image.resourceFolder=data/maps/
image.renderringThreshold=64
image.displayEdge=true
image.imagesFormNet=false
image.scaleToMap1=1000000;1;0;

#---------------------------------------------------------------
# End of properties file for TravelPlanning Application
#---------------------------------------------------------------
```