# LOCATION BASED INFORMATION SYSTEM FOR MOBILE DEVICES

*By*

| | |
|---|---|
| **Vishwendra Singh Atri** | **(081272)** |
| **Mukul Sirohi** | **(081278)** |
| **Shivam Prashar** | **(081279)** |
| **Vishal Krishna** | **(081407)** |

Under the Supervision of

**Mr Pradeep Kumar Gupta**

Submitted in partial fulfilment

Of the requirements for the degree of

**BACHELOR OF TECHNOLOGY (CSE / IT)**

**JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY WAKNAGHAT**

**SOLAN, HIMACHAL PRADESH INDIA**

**2012**

विद्या तत्त्व ज्योतिसम:

## JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY
## WAKNAGHAT
## SOLAN, HIMACHAL PRADESH

## CERTIFICATE

This is to certify that the work entitled **"LOCATION BASED INFORMATION SYSTEM"**, submitted by **Vishwendra Singh Atri, Mukul Sirohi, Shivam Prashar** and **Vishal Krishna** in partial fulfilment for award of degree of Bachelor of Technology (Computer Science & Engineering) **of Jaypee University of Information Technology** has been carried out in my supervision. This work has not been submitted partially or wholly to any other university or institution for award of this or any other degree programme.

**DATE:** 02/06/12

[illegible]

Project Supervisor

**Mr Pradeep Kumar Gupta**

**Sr. Lecturer (CSE & IT), JUIT**

# ACKNOWLEDGEMENT

We are highly grateful to **Brig. (Retd.) S.P. Ghrera**, Associate Professor and Head, Dept. of Computer Science & Engineering and IT, for providing us the opportunity to work on the project.

With great pleasure we express our gratefulness to our guide and mentor **Mr P.K.Gupta**, Senior Lecturer, Dept. of Computer Science & Engineering and IT, for his valuable and sustained guidance and careful supervision during the project.

We express our sincere thanks to all the faculty members of **JAYPEE UNIVERSITY OF INFORMATION AND TECHNOLOGY** who have helped us directly or indirectly. Without their help and guidance we would not have been able to successfully complete our project.

**Group Members**

| | |
|---|---|
| Vishwendra Singh Atri | [illegible] |
| Mukul Sirohi | MSirohi |
| Shivam Prashar | Shivam. |
| Vishal Krishna | Vishal Krishna |

# ABSTRACT

The availability of powerful mobile phones along with advances in software development platforms and databases have led to the creation of LBIS.LBIS combines all the technological advances to create a new breed of applications known as LOCATION BASED SERVICES (LBS). These services are mainly concerned with the location of the user. The application that we have built would be concerned with the location of the user. The application is able to fetch the user its current location on the map. The user can also get the direction from his source to destination path. The application will help the user to search for any place or location and the result will be shown on the map. Moreover, the user also has an option of getting to know about the nearby places such as restaurant, ATM, hospitals etc...

We have used android as our platform to build this application. The application will be using global positioning system (GPS) and general packet radio system (GPRS) in order to locate the position the of the user and perform other task of the application. The map will be downloaded using gprs using internet connection and the location will be shown by gps.

# Table of Contents

## LIST OF FIGURES

## LIST OF TABLES

Chapter 1

# Introduction

## 1.1 LOCATION BASED INFORMATION SYSTEM (LBIS)

The availability of powerful mobile phones along with advances in software development platforms and databases have led to the creation of LBIS.LBIS combines all the technological advances to create a new breed of applications known as LOCATION BASED SERVICES (LBS).Global demand for location-based services continues to skyrocket because of the availability of cellular phones, the new level of service these applications provide, and the important role of LBS will play in future software systems. Creating LBS is very challenging, as there are many players involved and many issues still unsolved. LBS have to deal with erroneous and variable information, as the accuracy of GPS fixes depends on the positioning system, user location, weather conditions, interferences, and others. Cellular communication networks also introduce challenges due to the nature of the wireless transmission where signals fade, transmissions disconnect, and errors in the data occur. LBIS need to be conscious about the limited resources available in the mobile devices. LBS, as a part of LBIS, interact with other systems and databases to retrieve context information and therefore, be able to provide better information to the user. Finally, cellular phone applications need to have a standard way to access position information and interact with the rest of the system, regardless of the cellular phone manufacturer and cellular system.

## 1.2 PROBLEM STATEMENT

To design a Location Based Information System that would provide user its current location. Depending upon the user location it should also provide information such as nearby Restaurants, ATMs, etc... It should provide user the directions from one place to another and provide user to explore the map of any location or places.

## 1.3 DEFINITION

LBS is an application that provides users with information based on the geographical position of the mobile device (user).This is one of the main differentiating features of LBS systems with regard to other systems, such as well-known enterprise systems, the system knows the physical location of the mobile device. Further, the location of the user is the major differentiator between first generation LBS applications and current ones, which provide more advanced and useful information. A typical first-generation LBS application example is the subscription-based service that provided traffic congestion notifications to mobile users. In those applications, users had to choose the roads they wanted to receive congestion notifications about using the service provider's Web page.

**"THE FUNDAMENTAL PIECE OF INFORMATION IS THE LOCATION OF THE USER".**

## 1.4 COMPLETE LBIS REAL-TIME TRACKING SYSTEM

5 major components

1) THE POSITIONING SYSTEM—GPS systems
2) THE CLIENT DEVICE
3) THE TRANSPORT NETWORK—the cellular network (GPRS)
4) THE SERVERS- Google maps

## 1.5 MOBILE DEVICE ARCHITECTURE

Cellular phones, as any other computer, contain the following three layers of software:

- APPLICATION-LEVEL software
- MIDDLEWARE software
- LOW-LEVEL software

APPLICATION-LEVEL software consists of all those programs running in the device that are meaningful to the user. One example of application level software is the tracking application.

MIDDLEWARE software provides the application developer with easy to use interfaces that automate commonly used tasks. It hides the developer from the details of providing the specific service, much in the same way drivers do, but at a higher level. Middleware has the double effect of shortening the application development time and reducing programming errors by offering well-designed and proved service interfaces.

LOW-LEVEL software consists of the device's operating system, the drivers, and virtual machines. The operating system is in charge of coordinating the activities and assigning the resources of the computer in a controlled manner to all running applications. As such, the operating system is directly attached to the hardware of the computer. Drivers are hardware- dependent-specific programs utilized by the operating system to communicate with a particular hardware device, such as a printer or a network interface card. Virtual machines are an additional software layer on top of the operating system that also executes computer programs. The importance of virtual machines is that they make the applications operating system independent allowing real portability of applications among computers. Virtual machines offer applications the same services and interfaces and hide the particularities of each operating system. Of course, there must be a specific virtual machine for each operating system. Since applications developed using the java programming language run on java virtual machines.

## 1.6 SOFTWARE ARCHITECHTURE

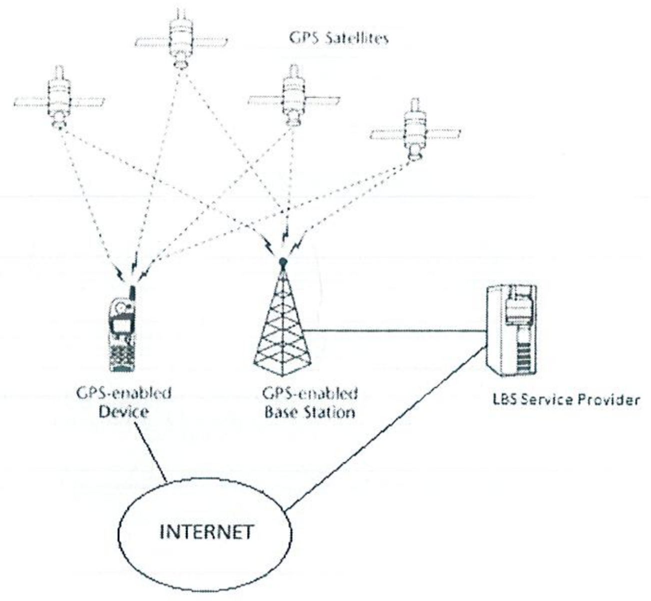


Figure 1.1: Software Architecture of the Application.

Here the above figure represents the software architecture of the application. The GPS satellites track the location of the user i.e. a GPS enabled device. The LBS Service provider with the help of network provider sends the information to the user through the internet.

### 1.6.1 LOCATION PROVIDER ARCHITECHTURE USED

MOBILE-BASED location provider architecture----Here, the LBS provider develops the application running in the user device, and the client device has the capability of obtaining its own information through the use of an embedded GPS, in collaboration with the cellular network provider. Once the client uses its location, it uses data connectivity through the cellular network via GPRS, to send it to the LBS provider server for storage and further updates and processing .The server receives location updates and queries from the user and then sends back the required information. It is worth mentioning that under this architecture, clients are not limited to cellular phones. Any GPS-enabled client with network connectivity can be part of the LBS. The main disadvantage of the mobile-based location provider architecture is that it has the potential to flood the network with unnecessary information, as different LBS providers implement their own applications and do not share the location information. On the plus side, mobile-based location provider architecture favours the rapid development of LBS applications, as it imposes neither major financial nor technical barriers.

## 1.6.2 CLIENT-SIDE SOFTWARE ARCHITECTURE

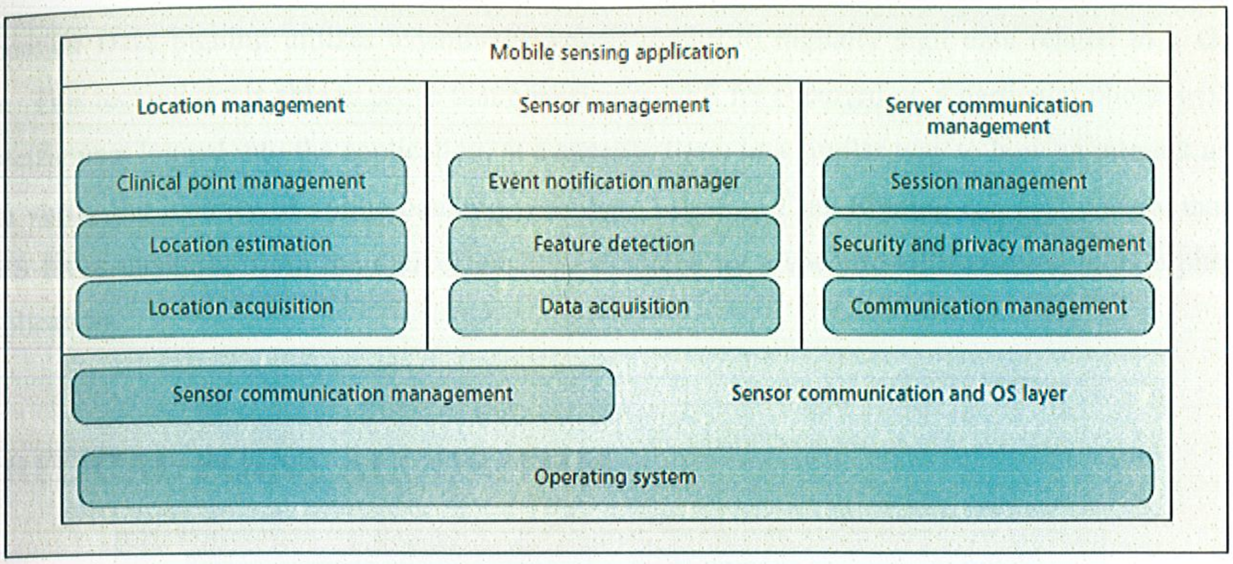

Mobile sensing application
Location management
Sensor management
Server communication management
Clinical point management
Location estimation
Location acquisition
Event notification manager
Feature detection
Data acquisition
Session management
Security and privacy management
Communication management
Sensor communication management
Sensor communication and OS layer
Operating system

Figure 1.2: Components of client side software architecture.

The client-side software architecture includes those software components that reside in the mobile device, the cellular phone. The architecture sits on top of the mobile device's operating system. The architecture has the platform at the bottom and the location-based application at the top, which is the application that runs in the mobile device and possibly interacts with the user. Between these two layers, the architecture presents several modules, each in charge of performing several functions. For example, it can be seen how the positioning data is obtained using the API included in the platform we use and how these GPS fixes are passed up to the application, either directly or through some other modules in charge of recalculating and estimating the position and making the position private. The GPS fixes can be sent from the application to the server directly using the UDP transport layer protocol, or they can be passed through some other modules in charge of saving energy not transmitting unnecessary fixes. Finally, the architecture also shows a Session management module, which is in charge of getting the log-in information from the user and establishing a new session with the server to store all the data from this particular device and user. This information is sent to the system's database using the TCP transport layer protocol for reliability purposes.

## LOCATION DATA SIGNING

GPS data are increasingly being used by businesses and governments to support key operations (e.g. verification of mileage and time for workers, confirmation of duration and location of car use for pay-as-you-drive insurance and taxes). However, these uses of GPS data have a key weakness: GPS data are falsifiable through tampering and cannot be independently verified.

Location Data Signing utilizes asymmetric cryptography to digitally sign data related to a GPS fix. This technology is able to prove that a particular GPS fix occurred on a particular phone with a specific user logged into the application, at a specific time, in a similar way to how an internet user can verify the identity of online banking websites. Location Data Signing can easily show that a GPS fix is unaltered from the data originally calculated by a specific GPS-enabled mobile phone application.

## CRITICAL POINT ALGORITHM

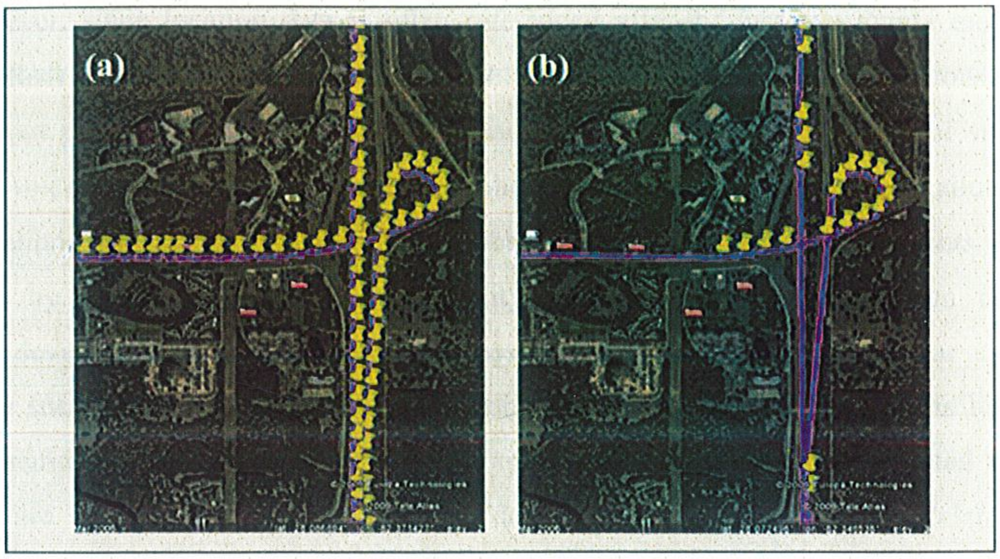


Figure 1.3 Critical Point Algorithm. (a)Tracking positions at all the points. (b) Tracking positions at only critical points.

Since GPS generates a large amount of location data, this data must be carefully managed to avoid wasting resources (e.g. battery energy) by transferring fixes to a server that may not contain useful information (e.g. repeated GPS fixes when the user is standing still, fixes lying upon the same vector when traveling in a straight line). Traditional LBS transfer fixes to the server regularly based on a fixed time period, which can result in transferring some of these irrelevant fixes, as well as accidentally discarding fixes that actually contain useful information.

The path of the user can be accurately represented by using only small portions of GPS data generated by mobile phones. The Critical Point Algorithm (CPA) uses the change in direction between sequential points as well as the user's speed to filter non-critical points from a set of GPS data, therefore, only critical points representing the path of the user remain (Figure 4). Only these critical points are transferred to a server for storage and analysis. By pre-filtering GPS data before it leaves the device, CPA saves battery energy, reduce data transfer costs, and saves network bandwidth, while still accurately representing the user's path. Table 1 shows the storage and financial savings per trip for transferring only critical points to the server. Assuming that each user takes around 4 trips per day, the savings quickly compound for thousands of users across long periods of time.

## ADAPTIVE LOCATION DATA BUFFERING

Since UDP is used to send location data to the server very efficiently, LAISYC 2-layer protocol by itself does not confirm that every location data fix successfully arrives at the server. In real-time tracking, the loss of occasional location fixes is acceptable since another location update will soon follow. However, since location data is often referenced after-the-fact to provide metrics (e.g. distance-travelled) and reconstruct users' paths, the loss of large numbers of contiguous fixes can cause significant problems for applications. Extended gaps can result from lack of support for simultaneous voice and data services on mobile phones or "dead zones" with no cellular signal. Adaptive Location Data Buffering increases the probability that most location data points will arrive at the server by performing occasional reliability checks. Before each location data UDP transmission, device-side APIs are checked to assess the current level of cellular signal and determine if a successful UDP transmission is probable. If not, the location data is buffered to either main memory or persistent storage on the mobile device. Once it is detected that UDP transmissions are likely to succeed, the buffered data is then sent via UDP and deleted on the device. Similarly, the device also occasionally checks with the server via the TCP protocol to ensure that it still has an open line of successful communication. If TCP fails, location data is buffered on the device until another future TCP check succeeds. Adaptive Location Data Buffering therefore increases the chance that most location data will successfully arrive at the server, while preserving the efficiency of using UDP to send most location data.

## LOCATION DATA ENCRYPTION

The interception of location data during transfer over the Internet is a significant security threat to LBS. While secure TCP connections are implemented within Java through Secure Socket Layer (SSL), the implementation of secure UDP is left to the application developer. We currently use the

Advanced Encryption Standard (AES) to encrypt location data as it is transferred using UDP to ensure a secure LBS.

## POSITION RECALCULATION MANAGEMENT

Dynamically varies the interval of time between location updates depending on the real-time needs of the system, and an estimation of whether the user is currently moving or is standing still. Numerous states, and gradual state transitions, are used to reduce the impact of GPS outliers. Figure 3 shows the battery life benefits of this technology, while still preserving the ability to quickly sample a user's location while they are moving. The current core technology estimates the correct state (e.g., moving or stationary) around 89% of the time, and recent experiments with modified Kalman Filters extend this to 93%.

## SESSION MANAGEMENT

In human-computer interaction, session management is the process of tracking the activity of the user across sessions of interaction with the computer system.

Typical session management tasks in a desktop environment includes the user's activity and tracking of open applications and documents so that whenever the user logs in sometime later, the same state can be restored. In case the session has expired, session management requires the user to login again. Various information is stored on the server-side between HTTP requests as well.

## DESKTOP SESSION MANAGEMENT

Saving and restoring of desktop sessions can be done by desktop session manager. A desktop session means all the currently running windows and its content. Session manager on Linux-based systems is provided by X session manager. On Microsoft Windows systems, no session manager is included in the system. There are numerous third-party applications which provide session management. For example- twins play.

## BROWSER SESSION MANAGEMENT

Session management is particularly useful in a web browser when it comes to saving of all the open pages and settings and their restoration at a later stage. In order to recover from a system or application crash, restoration of pages and settings can be done on next run. The browsers which support session management are Google Chrome, Omni Web and Opera. Other browsers such as Mozilla Firefox support session management through third-party plug- ins or extensions.

## WEB SERVER SESSION MANAGEMENT

Hypertext Transfer Protocol (HTTP) is stateless: a client computer running a web browser must establish a new Transmission Control Protocol (TCP) network connection to the web server with each new HTTP GET or POST request. The web server, therefore, cannot rely on an established TCP network connection for longer than a single HTTP GET or POST operation. Session management is the technique used by the web developer to make the stateless HTTP protocol support session state. For example, once a user has authenticated oneself to the web server, his/her next HTTP request (GET or POST) should not cause the web server to ask him/her for him/her account and password again. For a discussion of the methods used to accomplish this please see HTTP cookie.

The session information is stored on the web server using the session identifier (session ID) generated as a result of the first (sometimes the first authenticated) request from the end user running a web browser. The "storage" of session IDs and the associated session data (user name, account number, etc.) on the web server is accomplished using a variety of techniques including, but not limited to: local memory, flat files, and databases.

In situations where multiple web servers must share knowledge of session state (as is typical in a cluster environment—see computer cluster) session information must be shared between the cluster nodes that are running web server software. Methods for sharing session state between nodes in a cluster include: multicasting session information to member nodes.

## 1.7 TYPE of SERVICES

- Finding current location of the user.
- Obtaining directions to a place from current location.

- Finding nearby places e.g. ATMs, Restaurants, etc.
- Search map of any place e.g. Delhi, Chandigarh, New York, etc.

These services are of the request in which user queries the system, including the current location and the system responds with the specific information after searching in other systems and databases.

## 1.8 SCOPE

- Emergency Services and Disaster Management.
- Fleet and Logistics Management
- Navigation
- Vehicle Tracking

## 1.9 RESOURCES REQUIREMENTS

There are two types of requirements in this project:

- *Software Requirements*
- *Hardware Requirements*

### 1.9.1 SOFTWARE REQUIREMENT

- We need **Eclipse Platform** with **AndroidSDK** plugin to develop an application which will be hosted on client mobile device.
- **AVD Emulator Plugin**

### 1.9.2 HARDWARE REQUIREMENT

Computer system with

- 512 MBRAM
- Min 1 GB of free hard disk space.

**Mobile device** which is enabled with GPS and works on Android platform.

## 1.10 LIMITATIONS

There are different parties involved in the complete application.

- GPS enabled Handset.
- Network Provider.
- LBS Provider.
- GPS Satellites.
- Internet.

Therefore the performance of the application depends on a lot of things.

1. Application will not run on every handset. It should be GPS enables, GPRS enabled and Android based.
2. Application will not perform efficiently in low signals, or will not perform at all in absence of network signals.
3. Application will not perform in case there is some problem with LBS service provider.

Chapter 2

# The GPS

## 2.1 THE GLOBAL POSITIONING SYSTEM

The global positioning system is perhaps the most widely used and ubiquitous system to obtain users' positions. GPS is a complex and expensive system consisting of three main segments namely, the user segment, the space segment, and the control segment. These three segments give an opportunity to the GPS receivers to determine their speed, location, time and direction.

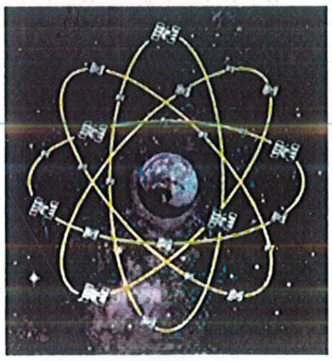
The GPS program is used to provide critical capabilities to military and commercial users across the globe. Moreover, GPS has been the driving force behind the modernizing the global air traffic system.

In order to overcome the earlier navigation system's limitations, the GPS project was developed in 1973. Integrating ideas from several predecessors, including a number of classified engineering design studies from the 1960s. GPS was created and realized by the **U.S.** D o D and was run with the help of 24 satellites, originally. In the year 1994, GPS became fully operational

It consists of three segments:

- Space Segment
- Control Segment
- User Segment

### 2.1.1 SPACE SEGMENT



❖ 24+ satellites
- ❑6 planes with 55° inclination
- ❑Each plane has 4-5 satellites
- ❑Broadcasting position and time info on 2 frequencies
- ❑Constellation has spares

Figure 2.1: Space segment of GPS.

The space segment consists of the orbiting GPS satellites. A total of 24 satellites, four in six orbital planes centred on the earth, are needed so that at least six satellites can be detected by a GPS receiver from almost anywhere on earth. Currently, the constellation of GPS satellites consists of 31 satellites; seven more have been added to provide redundant signals and improve the precision of GPS receivers and the reliability and availability of the system.

## 2.1.2 CONTROL SEGMENT

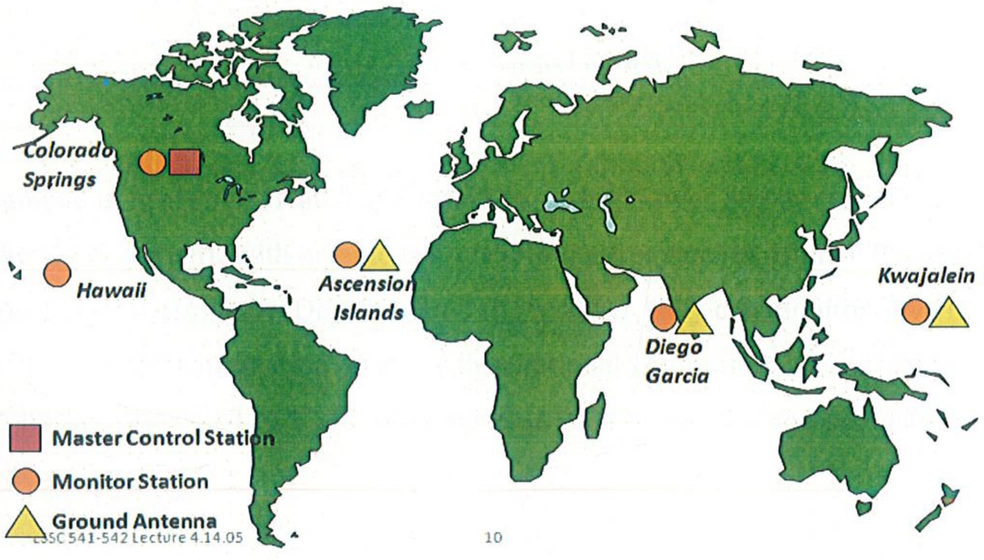


Figure 2.2: Control segment of GPS.

This segment comprises several ground stations that track and monitor the space segment and a main control station. The main control station is in charge of updating the on-board atomic clocks of the satellites and their ephemerides, or a table with the exact position of the satellites in the sky.

## 2.1.3 USER SEGMENT

•Dual Use System Since 1985
(civil & military)

•Civilian community was quick to take advantage of the system

- Hundreds of receivers on the market
- 3 billion in sales, double in 2 years
- 95% of current users

Figure 2.3: User segment of GPS.

The user segment is made up of all GPS receivers, a high-level architecture of a GPS receiver consists of the receiving part, with an antenna tuned to the frequencies of the GPS satellites, a main processor, and a crystal oscillator. Depending on the receiver, they can monitor anywhere from 4 to 20 channels .The calculated position along with additional information derived from the satellite may then be further processed to build other systems such as stand-alone navigational systems or tracking applications.

## 2.2 COORDINATION OF THREE SEGMENTS OF GPS

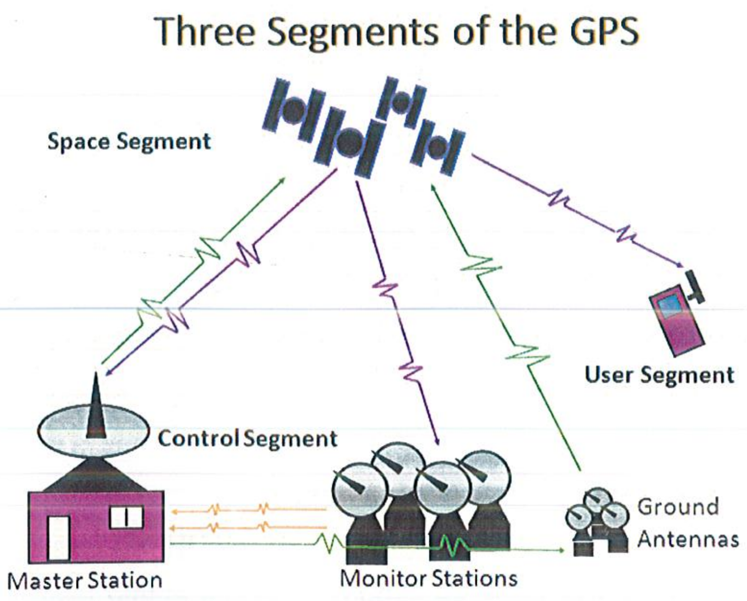


Figure 2.4: How the three segments of the GPS coordinates with each other.

Chapter 3

# Android Application Development

## 3.1 WHAT IS ANDROID – A GPHONE?

The weeks and months before Google released the Android SDK there had been a lot of rumours about a so called GPhone. It was said to be a mobile device manufactured by Google providing free communication by showing context-sensitive advertisements to the user on the device itself. Render of a potential GPhone But on November 5th 2007 Andy Rubin2 announced:
- [The] Android [Platform] – is more significant and ambitious than a single phone.

Google within the Open Handset Alliance (OHA) delivers a complete set of software for mobile devices: an operating system, middleware and key mobile applications. What was released a week later was not a final product, but a -First Look SDK‖ what many did not realize. Major news sites grabbed the discomforts of some developers who said that Android is full of bugs and heavily lacks of documentation. But the majority says that Android is not buggier than any other software at this stage.

Andy Rubin – Google Director of Mobile Platforms andbook - Android Programming Let's take a look at what the OHA emphasizes on its Android Platform:

### OPENNESS

Android was built from the ground-up to enable developers to create compelling mobile applications that take full advantage of all a handset has to offer. It is built to be truly open. For example, an application could call upon any of the phone's core functionality such as making calls, sending text messages, or using the camera, allowing developers to create richer and more cohesive experiences for users.‖

This is true, as a developer you can do everything, from sending short messages with just 2 lines of code, up to replacing even the HOME-Screen of your device. One could easily create a fully customized operating system within weeks, providing no more of Google's default application to

the user.
–Android is built on the open Linux Kernel. Furthermore, it utilizes a custom virtual machine that has been designed to optimize memory and hardware resources in a mobile environment. Android will be open source; it can be liberally extended to incorporate new cutting edge technologies as they emerge. The platform will continue to evolve as the developer community works together to build innovative mobile applications.‖

Here Google is talking of the so called Dalvik virtual machine (DalvikVM), which is a register based virtual machine, designed and written by Dan Bornstein and some other Google engineers, to be an important part of the Android platform. In the words –register based‖ we find the first difference to normal Java virtual machines (JVM) which are stack based.

## ALL APPLICATIONS ARE CREATED EQUALLY

Android does not differentiate between the phone's core applications and third-party applications. They can all be built to have equal access to a phone's capabilities providing users with a broad spectrum of applications and services. With devices built on the Android Platform, users will be able to fully tailor the phone to their interests. They can swap out the phone's home screen, the style of the dialler, or any of the applications. They can even instruct their phones to use their favourite photo viewing application to handle the viewing of all photos.‖

Once again this is all true. Developers can 100% customize their Android-Device. The Android System Communication is based on so called Intents, which are more or less just a String (with some data attached) which defines an action that needs to be handled. An example for this is:

‖android.provider.Telephony.SMS_RECEIVED‖ One can simply listen on that Intent by writing about 5 lines of definitions. The system would then recognize that there is more than one application that wants to handle that Intent and ask the user to choose which one he or she would like to handle the Intent.

## BREAKING DOWN APPLICATION BOUNDARIES

Android breaks down the barriers to building new and innovative applications. For example, a developer can combine information from the web with data on an individual's mobile phone - such as the user's contacts, calendar, or geographic location - to provide a more relevant user experience.

With Android, a developer could build an application that enables users to view the location of their friends and be alerted when they are in the vicinity giving them a chance to connect.‖ Fast & easy application development -Android provides access to a wide range of useful libraries and tools that can be used to build rich applications.

For example, Android Programming enables developers to obtain the location of the device, and allows devices to communicate with one another enabling rich peer-to-peer social applications. In addition, Android includes a full set of tools that have been built from the ground up alongside the platform providing developers with high productivity and deep insight into their applications.‖ Since the Web 2.0 revolution, making content rich applications within minutes is no more illusion. Android has brought developing to unknown speeds.

## 3.2 ANDROID DEVLOPEMENT

Android applications are written in the Java programming language. The compiled Java code along with any data and resource files required by the application is bundled by the apt tool into an Android package, an archive file marked by an .apk suffix. This file is the vehicle for distributing the application and installing it on mobile devices; it's the file users download to their devices. All the code in a single .apk file is considered to be one application. In many ways, each Android application lives in its own world:

- By default, every application runs in its own Linux process. Android starts the process when any of the applications code needs to be executed, and shuts down the process when it's no longer needed and system resources are required by other applications.
- Each process has its own Java virtual machine (VM), so application code runs in isolation from the code of all other applications.
- By default, each application is assigned a unique Linux user ID. Permissions are set so that the application's files are visible only that user, only to the application itself although they can be exported to other applications also.

It's possible to arrange for two applications to share the same user ID, in which case they will be able to see each other's files. To conserve system resources, applications with the same ID can also arrange to run in the same Linux process, sharing the same VM.

## 3.2.1 APPLICATION COMPONENTS

A central feature of Android is that one application can make use of elements of other applications (provided those applications permit it). For example, if your application needs to display a scrolling list of images and another application has developed a suitable scroller and made it available to others, you can call upon that scroller to do the work, rather than develop your own. Your application doesn't incorporate the code of the other application or link to it. Rather, it simply starts up that piece of the other application when the need arises.

For this to work, the system must be able to start an application process when any part of it is needed, and instantiate the Java objects for that part. Therefore, unlike applications on most other systems, Android applications don't have a single entry point for everything in the application (no main () function, for example). Rather, they have essential components that the system can instantiate and run as needed.

### 3.2.1.1 ACTIVITIES

An activity presents a visual user interface for one focused endeavour the user can undertake. For example, an activity might present a list of menu items users can choose from or it might display photographs along with their captions. A text messaging application might have one activity that shows a list of contacts to send messages to, a second activity to write the message to the chosen contact, and other activities to review old messages or change settings. Though they work together to form a cohesive user interface, each activity is independent of the others. Each one is implemented as a subclass of the Activity base class.

An application can comprise only one activity or, it may contain several like the text messaging application. What the activities are, and how many there are depends, of course, on the application and its design. Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched. Moving from one activity to another is accomplished by having the current activity start the next one.
Each activity is given a default window to draw in. Typically, the window fills the screen, but it might be smaller than the screen and float on top of other windows. An activity can also make use of additional windows for example, a pop-up dialog that calls for a user response in the midst of the

Activity or a window that presents users with vital information when they select a particular item on- screen.

The visual content of the window is provided by a hierarchy of views — objects derived from the base View class. Each view controls a particular rectangular space within the window. Parent views contain and organize the layout of their children. Leaf views (those at the bottom of the hierarchy) draw in the rectangles they respond to user actions which are directed at that space.

Thus, views are where the activity's interaction with the user takes place. For example, a view might display a small image and initiate an action when the user taps that image. Android has a number of ready-made views that you can use — including buttons, text fields, scroll bars, menu items, check boxes, and more.

A view hierarchy is placed within an activity's window by the Activity.setContentView() method. The content view is the View object at the root of the hierarchy. (See the separate User Interface document for more information on views and the hierarchy.)

### 3.2.1.2 SERVICES

A service doesn't have a visual user interface, but rather runs in the background for an indefinite period of time. For example, a service might play background music as the user attends to other matters, or it might fetch data over the network or calculate something and provide the result to activities that require it. Each and every service also extends the Service base class.

A prime example is a media player playing songs from a play list. The player application would probably have one or more activities that allow the user to choose songs and start playing them.

However, the music playback itself would not be handled by an activity because users will expect the music to keep playing even after they leave the player and begin something different. To keep the music going, the media player activity could start a service to run in the background. The system would then keep the music playback service running even after the activity that started it leaves the screen. It's possible to connect to (bind to) an on-going service (and start the service if it's not already running). While connected, you can communicate with the service through an interface that the service exposes. For the music service, this interface might allow users to pause, rewind, stop, and restart the playback.

Like activities and the other components, services run in the main thread of the application process. So that they won't block other components or the user interface, they often spawn another thread for time-consuming tasks (like music playback).

## 3.2.1.3 BROADCAST RECEIVRS

A broadcast receiver is a component that is responsible for the receiving and giving reactions to broadcast announcements. Many broadcasts originate in system code — for example, announcements that the time zone has changed, that the battery is low, that a picture has been taken, or that the user changed a language preference. Applications can also be used for initiating broadcasts such as, letting the other applications know that some data has been downloaded to the device and is available for them to use. An application can have any number of broadcast receivers to respond to any announcements it considers important. All receivers extend the BroadcastReceiverbase class. Broadcast receivers do not display a user interface. However, they may start an activity in response to the information they receive, or they may use the Notification Manager to alert the user. Notifications can get the user's attention in various ways — flashing the backlight, vibrating the device, playing a sound, and so on. They typically place a persistent icon in the status bar, which users can open to get the message.

## 3.2.1.4 CONTENT PROVIDERS

A content provider makes a specific set of the application's data available to other applications. The data can be stored in the file system, in a SQLite database, or in any other manner that makes sense. The content provider extends the Content Provider base class to implement a standard set of methods that enable other applications to retrieve and store data of the type it controls. However, applications do not call these methods directly. Rather they use a Content Resolver object and call its methods instead. A Content Resolver can talk to any content provider; it cooperates with the provider to manage any interposes communication that's involved.

See the separate Content Providers document for more information on using content providers. Whenever there's a request that should be handled by a particular component, Android has the duty of making sure that the component's application process is running, if it is not then starting it, and that an appropriate instance of the component is available, creating the instance if necessary.

Activating components: intents Content providers are activated when they're targeted by a request from a Content Resolver. The other three components activities, services, and broadcast receivers are activated by asynchronous messages called intents. Intent is an Intent object that holds the content of the message. For activities and services, it names the action being requested and specifies the URI of the data to act on, among other things. For example, it might convey a request for an activity to present an image to the user or let the user edit some text. For broadcast receivers, the Intent object names the action being announced. For example, it might announce to interested parties that the camera button has been pressed.

There are separate methods for activating each type of component:

- An activity is launched (or given something new to do) by passing an Intent object to Context.startActivity() orActivity.startActivityForResult(). The responding activity can look at the initial intent that caused it to be launched by calling its getIntent() method. Android calls the activity's onNewIntent() method to pass it any subsequent intents. One activity often starts the next one. In case it is expecting the activity it's starting to give back the result, it calls startActivityForResult() instead ofstartActivity(). For example, if it starts an activity that lets the user pick a photo, it might expect to be returned the chosen photo. The result is returned in an Intent object that's passed to the calling activity's onActivityResult() method.
- A service is started (or new instructions are given to an on-going service) by passing an Intent object to Context.startService(). Android calls the service'sonStart() method and passes it the Intent object. Similarly, an intent can be passed to Context.bindService() to establish an ongoing connection between the calling component and a target service. The service receives the Intent object in an onBind() call. (If the service is not already running, bindService() can optionally start it.) For example, an activity might establish a connection with the music playback service mentioned earlier so that it can provide the user with the means (a user interface) for controlling the playback. The activity would call bindService() to set up that connection, and then call methods defined by the service to affect the playback. A later section, Remote procedure calls, has more details

about binding to a service.

- An application can be used for initiating a broadcast by just passing an Intent object to methods like Context.sendBroadcast(), Context.sendOrderedBroadcast(), Context.sendStickyBroadcast() in any of their variations. Android has the duty of delivering the intent to all interested broadcast receivers by calling their onReceive()methods. For more on intent messages, see the separate article, Intents and Intent Filters.

### 3.2.1.5 <u>SHUTTING DOWN COMPONENTS</u>

A content provider is active only while it's responding to a request from a Content Resolver. And a broadcast receiver is active only while it's responding to a broadcast message. So there's no need to explicitly shut down these components. Activities, on the other hand, provide the user interface. They're in a long-running conversation with the user and may remain active, even when idle, as long as the conversation continues. Similarly, services may also remain running for a long time. So Android also comprises the methods which could shut down activities and services in an orderly way:

- An activity can be shut down by calling its finish () method. One activity can shut down another activity (one it started with startActivityForResult()) by calling finishActivity().
- A service can be stopped by calling its stopSelf() method, or by calling Context.stopService(). Components might also be shut down by the system when they are no longer being used or when Android must reclaim memory for more active components. A later section, Component Lifecycles, discusses this possibility and its ramifications in more detail.

### 3.2.1.6 <u>THE MANIFEST FILE</u>

Before Android can start an application component, it must learn that the component exists. Therefore, applications declare their components in a manifest file that's bundled into the Android package, the .apk file that also holds the application's code, files, and resources. The manifest is a structured XML file and is always named AndroidManifest.xml for all applications. It does a number of things in addition to declaring the application's components, such as naming any libraries the application needs to be linked against (besides the default Android library) and identifying any permission the application expects to be granted. But the principal task of the manifest is to inform Android about the application's components. For example, an activity might be *declared as follows:*

```
<?xml                  version="1.0"
encoding="utf-8"?>

<manifest .
. . >

  <application . . . >

    <activity
         android:name="com.example.project.FreneticAc
         tivity" android:icon="@drawable/small_pic.png"
         android:label="@string/freneticLabel"
         . . . >

    </activity>
    . .
  </application>

</mani
fest>
```

The name attribute of the <activity> element names the Activity subclass that implements the activity. The icon and label attributes point to resource files containing an icon and label that can be displayed to users to represent the activity. The other components are declared in a similar way —
<service> elements for services, <receiver> elements for broadcast receivers, and <provider> elements for content providers. Activities, services, and content providers that are not declared in the manifest are not visible to the system and are consequently never run. The broadcast receivers can either be declared in the manifest, or they can also be created in the code dynamically (as BroadcastReceiver objects) and registered with the system by calling Context.registerReceiver().

### 3.2.1.7 INTENT FILTERS

An Intent object can explicitly name a target component. If it does, Android finds that component (based on the declarations in the manifest file) and activates it. But if a target is not explicitly named, Android must locate the best component to respond to the intent. It does so by comparing the Intent object to the intent filters of potential targets. A component's intent filters inform Android about the various kinds of intents that the component can be used to handle. Like other essential information about the component, they're declared in the manifest file. Here's an extension of the previous example that adds two intent filters to the activity:

```
<?xml                    version="1.0"
encoding="utf-8"?>

<manifest .
. . >

 <application . . . >

   <activity
        android:name="com.example.project.FreneticAc
        tivity" android:icon="@drawable/small_pic.png"
        android:label="@string/freneticLabel"
        . .
        .
        >

     <intent-filter . . . >

       <action android:name="android.intent.action.MAIN" />

       <category android:name="android.intent.category.LAUNCHER"
/>

     </intent-filter>

     <intent-filter . . . >

       <action android:name="com.example.project.BOUNCE" />

       <data android:mimeType="image/jpeg" />

       <category android:name="android.intent.category.DEFAULT" />

     </intent-filter>

   </activity>

   . . .

   </application>
```

```
</manifest>
```

The first filter in the example - - the combination of the action "android.intent.action.MAIN" and the category "android.intent.category.LAUNCHER" — is a common one. It marks the activity as one that should be represented in the application launcher, the screen listing applications users can launch on the device. In other words, the activity is the entry point for the application; the initial one user would see when they choose the application in the launcher.

The second filter declares an action that the activity can perform on a particular type of data. A component can have any number of intent filters, each one declaring a different set of capabilities. If it doesn't have any filters, it can be activated only by intents that explicitly name the component as the target. For a broadcast receiver that's created and registered in code, the intent filter is instantiated directly as IntentFilter object. All other filters are set up in the manifest.

### 3.2.1.8 AVD EMULATOR

The Android SDK includes a mobile device emulator — a virtual mobile device that runs on your computer. The emulator lets you develop and test Android applications without using a physical device.

The moment at which the emulator is running, one can do interaction with the emulated mobile device just as you would an actual mobile device, except that you use your mouse pointer to "touch" the touchscreen and can use some keyboard keys to invoke certain keys on the device.

Chapter 4

# Project Development

## 4.1.UML DIAGRAMS

The Unified Modelling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software.

### Goals of UML

The primary goals in the design of the UML were:

1. Provide users with ready-to-use, expressive visual modelling languages so they can develop and exchange meaningful models.
2. Provide extensibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development processes.
4. Provide a formal basis for understanding the modelling language.
5. Encourage the growth of the OO tools market.
6. Support higher-level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

## 4.1.1 USE CASE DIAGRAM

Use case diagrams are important for visualizing, specifying, and documenting the behaviour of an element.

- A *use case diagram is a diagram that shows a set of use cases and actors and their* relationships.
- Use case diagrams commonly contain

  – Use cases

  – Actors

  – Dependency, generalization, and association relationships

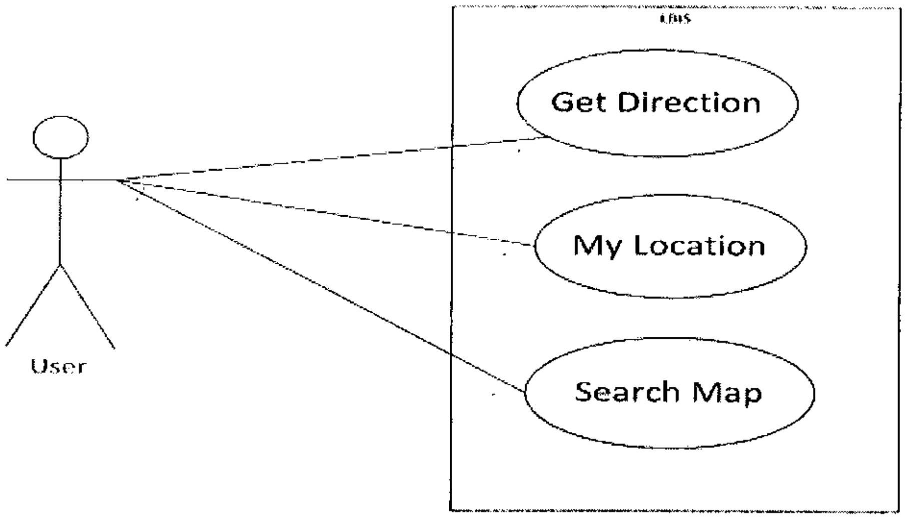


Figure 4.1: The use case diagram

### Use case diagram description

- User – The person who will be using the application.
- Get Direction – It is a function that will allow the user to get the direction of any place from any starting place.
- My Location – It is a function that will fetch the user his current location.
- Search Map – It will allow the user to search for any place on the map.

## 4.1.2 FUNCTIONALITY DECOMPOSITION DIAGRAM

It is a top-down representation of a function or process. FDD's also are called *structure charts.*

FDD's are used by system analysist model business functions and show how they are organized into lower-level processes. Those processes translate into program modules during application development.

Creating an FDD is similar to drawing an organizational chart–you start at the top and work your way down.

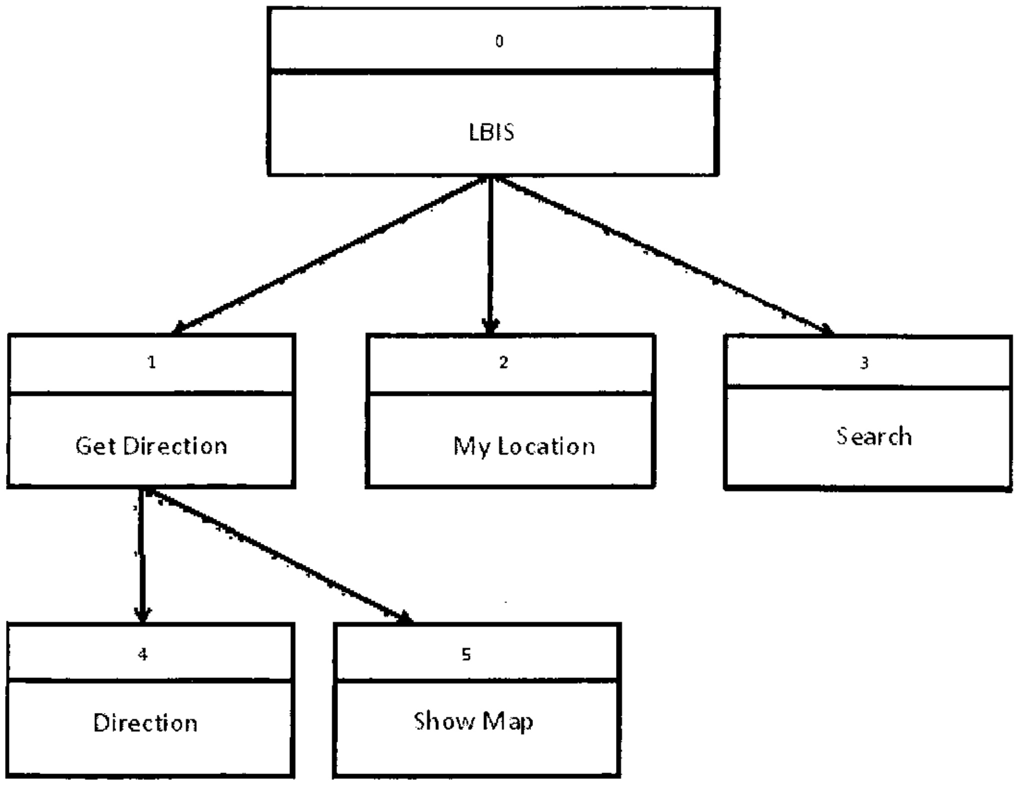


Figure 4.2: Functional Decomposition Diagram

## 4.1.3 CLASS DIAGRAM

The class diagram is the main building block in object oriented modelling. It is used both for general conceptual modelling of the systematic of the application, and for detailed modelling translating the models into programming code. The classes in a class diagram represent both the main objects and or interactions in the application and the objects to be programmed. In the class diagram these classes are represented with boxes which contain three parts.

A class with three sections:

- The upper part holds the name of the class.
- The middle part contains the attributes of the class.
- The bottom part gives the methods or operations the class can take or undertake.

In the system design of a system, a number of classes are identified and grouped together in a class diagram which helps to determine the static relations between those objects. With detailed modelling,

The classes of the conceptual design are often split into a number of subclasses.

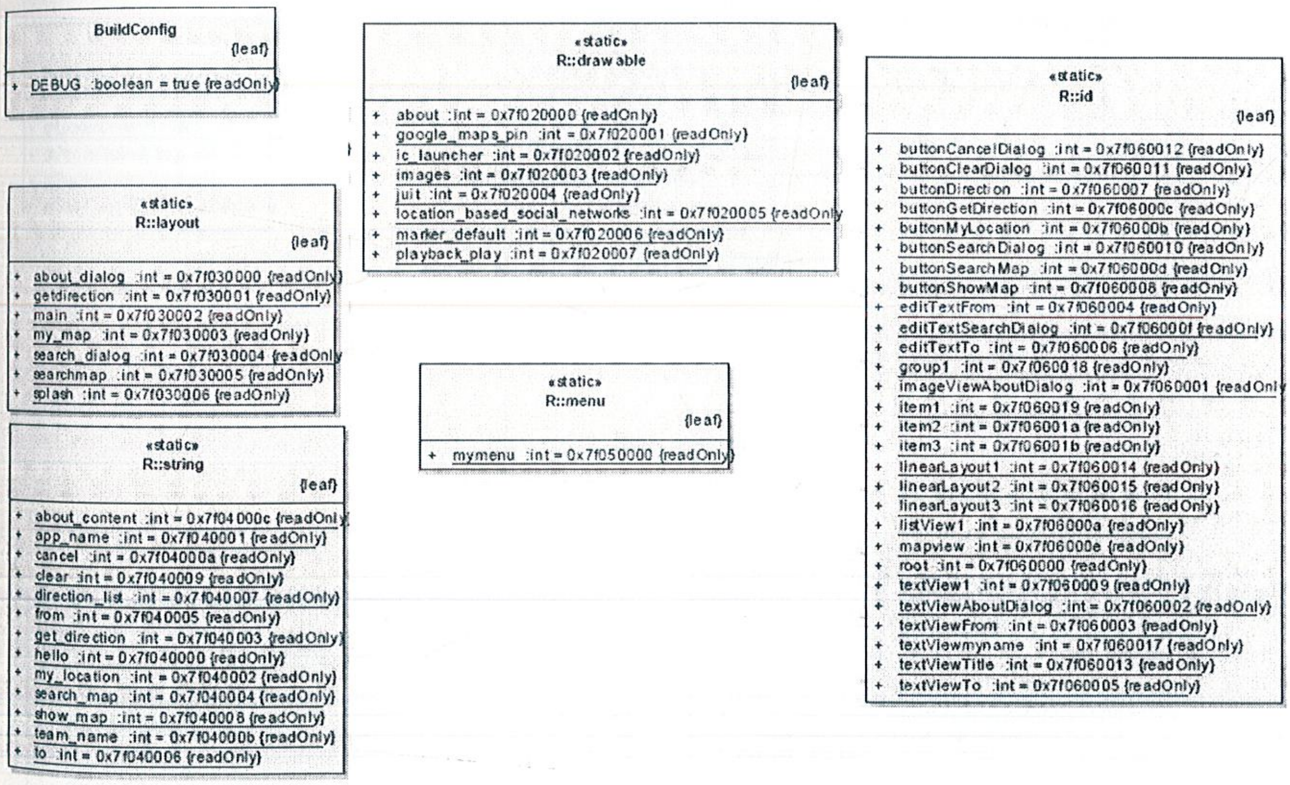

BuildConfig
{leaf}
+ DEBUG :boolean = true {readOnly}
«static»
R::layout
{leaf}
+ about_dialog :int = 0x7f030000 {readOnly}
+ getdirection :int = 0x7f030001 {readOnly}
+ main :int = 0x7f030002 {readOnly}
+ my_map :int = 0x7f030003 {readOnly}
+ search_dialog :int = 0x7f030004 {readOnly
+ searchmap :int = 0x7f030005 {readOnly}
+ splash :int = 0x7f030006 {readOnly}
«static»
R::string
{leaf}
+ about_content :int = 0x7f04000c {readOnly
+ app_name :int = 0x7f040001 {readOnly}
+ cancel :int = 0x7f04000a {readOnly}
+ clear :int = 0x7f040009 {readOnly}
+ direction_list :int = 0x7f040007 {readOnly}
+ from :int = 0x7f040005 {readOnly}
+ get_direction :int = 0x7f040003 {readOnly}
+ hello :int = 0x7f040000 {readOnly}
+ my_location :int = 0x7f040002 {readOnly}
+ search_map :int = 0x7f040004 {readOnly}
+ show_map :int = 0x7f040008 {readOnly}
+ team_name :int = 0x7f04000b {readOnly}
+ to :int = 0x7f040006 {readOnly}
«static»
R::drawable
{leaf}
+ about :int = 0x7f020000 {readOnly}
+ google_maps_pin :int = 0x7f020001 {readOnly}
+ ic_launcher :int = 0x7f020002 {readOnly}
+ images :int = 0x7f020003 {readOnly}
+ juit :int = 0x7f020004 {readOnly}
+ location_based_social_networks :int = 0x7f020005 {readOnly
+ marker_default :int = 0x7f020006 {readOnly}
+ playback_play :int = 0x7f020007 {readOnly}
«static»
R::menu
{leaf}
+ mymenu :int = 0x7f050000 {readOnly}
«static»
R::id
{leaf}
+ buttonCancelDialog :int = 0x7f060012 {readOnly}
+ buttonClearDialog :int = 0x7f060011 {readOnly}
+ buttonDirection :int = 0x7f060007 {readOnly}
+ buttonGetDirection :int = 0x7f06000c {readOnly}
+ buttonMyLocation :int = 0x7f06000b {readOnly}
+ buttonSearchDialog :int = 0x7f060010 {readOnly}
+ buttonSearchMap :int = 0x7f06000d {readOnly}
+ buttonShowMap :int = 0x7f060008 {readOnly}
+ editTextFrom :int = 0x7f060004 {readOnly}
+ editTextSearchDialog :int = 0x7f06000f {readOnly}
+ editTextTo :int = 0x7f060006 {readOnly}
+ group1 :int = 0x7f060018 {readOnly}
+ imageViewAboutDialog :int = 0x7f060001 {readOnly
+ item1 :int = 0x7f060019 {readOnly}
+ item2 :int = 0x7f06001a {readOnly}
+ item3 :int = 0x7f06001b {readOnly}
+ linearLayout1 :int = 0x7f060014 {readOnly}
+ linearLayout2 :int = 0x7f060015 {readOnly}
+ linearLayout3 :int = 0x7f060016 {readOnly}
+ listView1 :int = 0x7f06000a {readOnly}
+ mapview :int = 0x7f06000e {readOnly}
+ root :int = 0x7f060000 {readOnly}
+ textView1 :int = 0x7f060009 {readOnly}
+ textViewAboutDialog :int = 0x7f060002 {readOnly}
+ textViewFrom :int = 0x7f060003 {readOnly}
+ textViewmyname :int = 0x7f060017 {readOnly}
+ textViewTitle :int = 0x7f060013 {readOnly}
+ textViewTo :int = 0x7f060005 {readOnly}

Figure 4.3 Class diagram 1

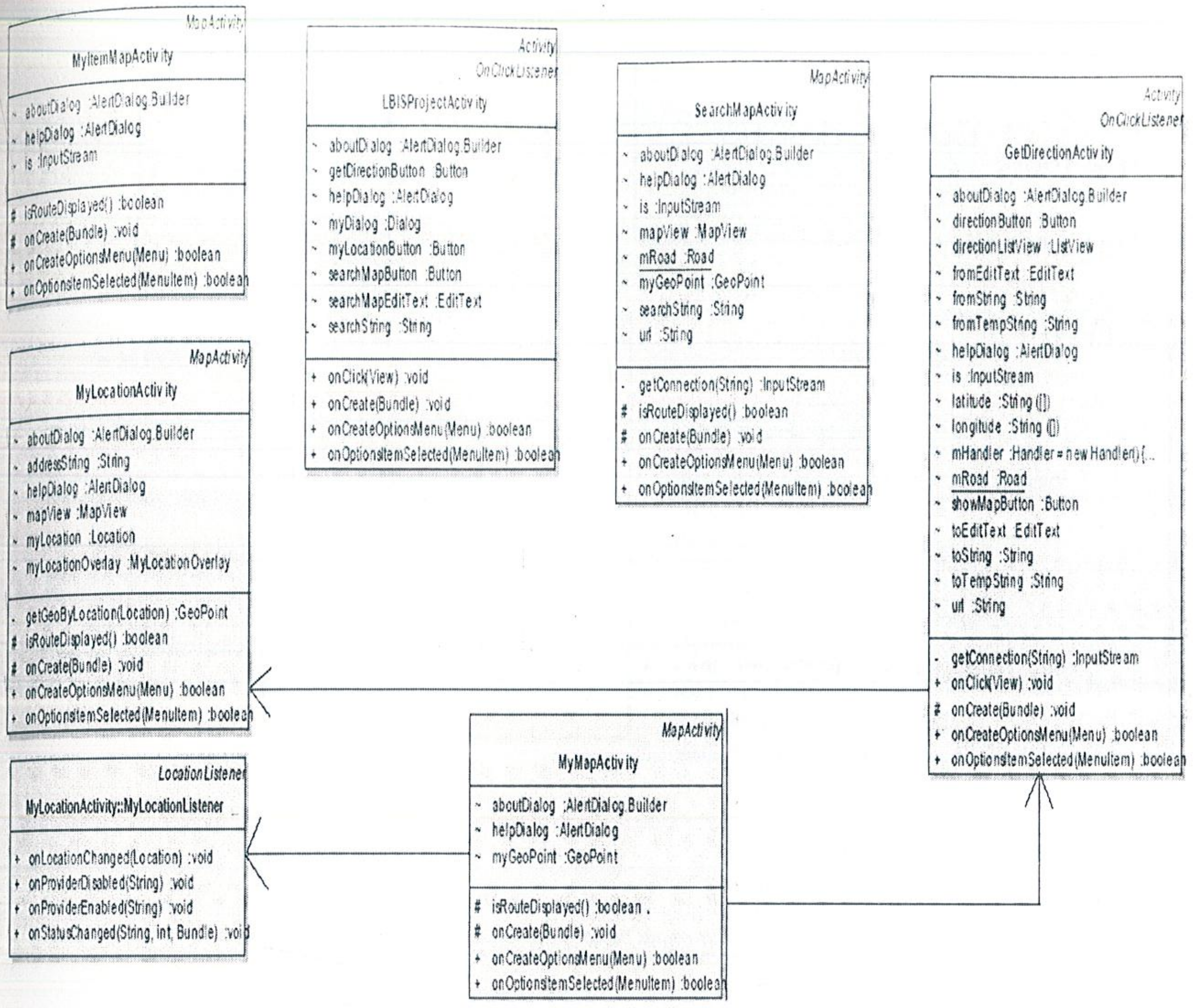

MapActivity
MyItemMapActivity
~ aboutDialog :AlertDialog.Builder
~ helpDialog :AlertDialog
~ is :InputStream
# isRouteDisplayed() :boolean
# onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea
Activity
OnClickListener
LBISProjectActivity
~ aboutDialog :AlertDialog.Builder
~ getDirectionButton :Button
~ helpDialog :AlertDialog
~ myDialog :Dialog
~ myLocationButton :Button
~ searchMapButton :Button
~ searchMapEditText :EditText
~ searchString :String
+ onClick(View) :void
+ onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea
MapActivity
SearchMapActivity
~ aboutDialog :AlertDialog.Builder
~ helpDialog :AlertDialog
~ is :InputStream
~ mapView :MapView
~ mRoad :Road
~ myGeoPoint :GeoPoint
~ searchString :String
~ url :String
- getConnection(String) :InputStream
# isRouteDisplayed() :boolean
# onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea
Activity
OnClickListener
GetDirectionActivity
~ aboutDialog :AlertDialog.Builder
~ directionButton :Button
~ directionListView :ListView
~ fromEditText :EditText
~ fromString :String
~ fromTempString :String
~ helpDialog :AlertDialog
~ is :InputStream
~ latitude :String ([])
~ longitude :String ([])
~ mHandler :Handler = new Handler(){...
~ mRoad :Road
~ showMapButton :Button
~ toEditText :EditText
~ toString :String
~ toTempString :String
~ url :String
- getConnection(String) :InputStream
+ onClick(View) :void
# onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea
MapActivity
MyLocationActivity
~ aboutDialog :AlertDialog.Builder
~ addressString :String
~ helpDialog :AlertDialog
~ mapView :MapView
~ myLocation :Location
~ myLocationOverlay :MyLocationOverlay
- getGeoByLocation(Location) :GeoPoint
# isRouteDisplayed() :boolean
# onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea
LocationListener
MyLocationActivity::MyLocationListener
+ onLocationChanged(Location) :void
+ onProviderDisabled(String) :void
+ onProviderEnabled(String) :void
+ onStatusChanged(String, int, Bundle) :void
MapActivity
MyMapActivity
~ aboutDialog :AlertDialog.Builder
~ helpDialog :AlertDialog
~ myGeoPoint :GeoPoint
# isRouteDisplayed() :boolean
# onCreate(Bundle) :void
+ onCreateOptionsMenu(Menu) :boolean
+ onOptionsItemSelected(MenuItem) :boolea

Figure 4.4 Class Diagram 2

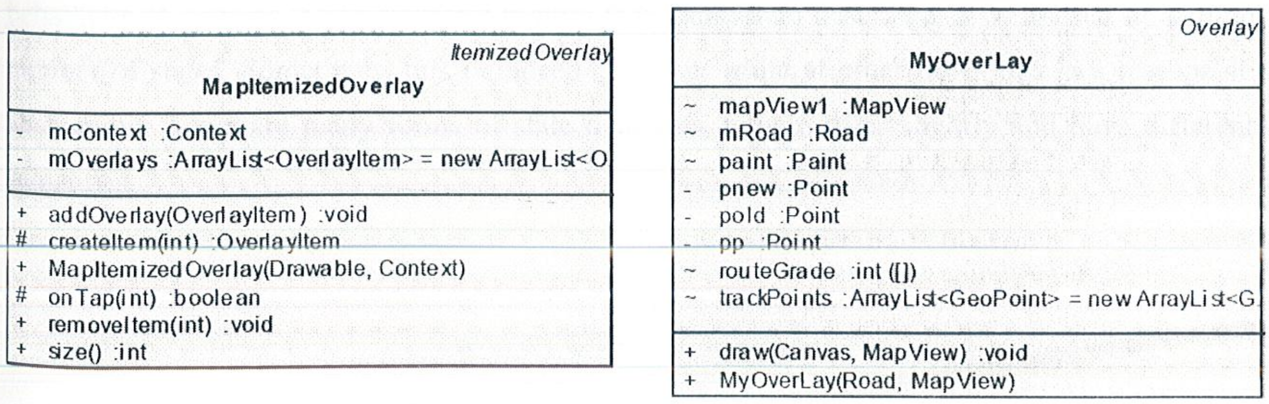

ItemizedOverlay
MapItemizedOverlay
- mContext :Context
- mOverlays :ArrayList<OverlayItem> = new ArrayList<O
+ addOverlay(OverlayItem) :void
# createItem(int) :OverlayItem
+ MapItemizedOverlay(Drawable, Context)
# onTap(int) :boolean
+ removeItem(int) :void
+ size() :int
Overlay
MyOverLay
~ mapView1 :MapView
~ mRoad :Road
~ paint :Paint
- pnew :Point
- pold :Point
- pp :Point
~ routeGrade :int ([])
~ trackPoints :ArrayList<GeoPoint> = new ArrayList<G
+ draw(Canvas, MapView) :void
+ MyOverLay(Road, MapView)

Figure 4.5 Class Diagram 3

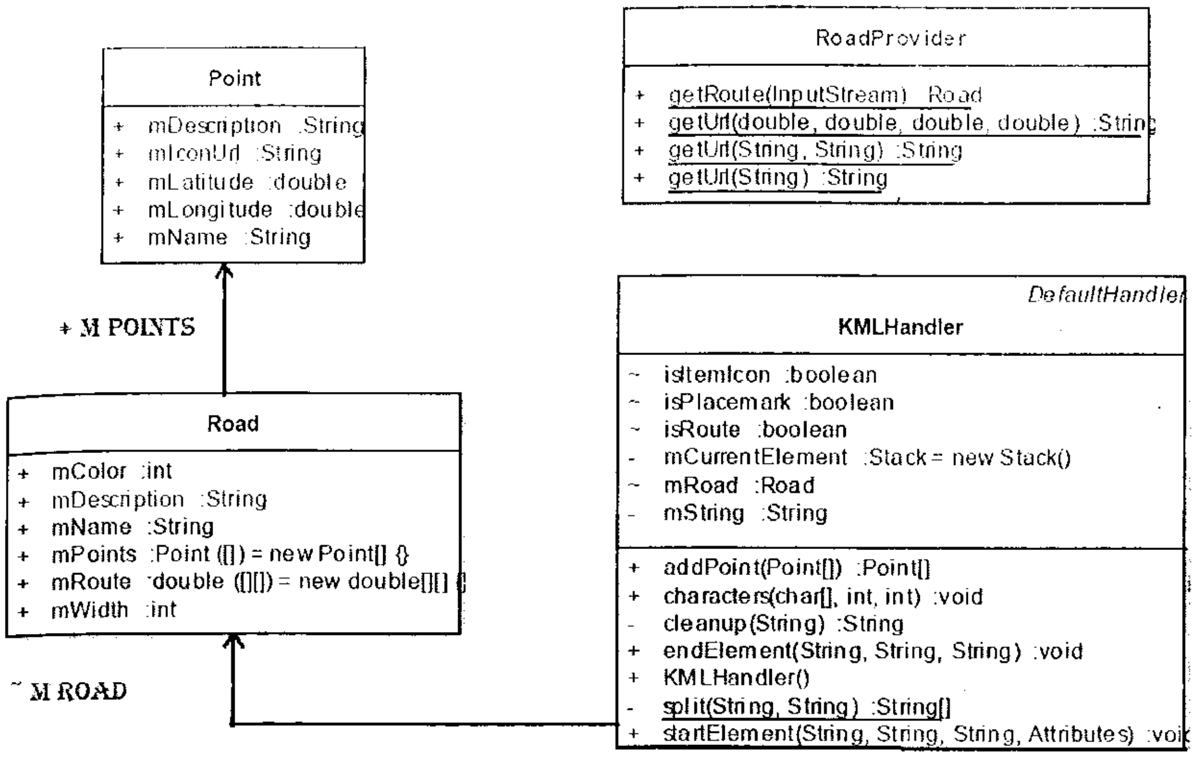

Point
+ mDescription :String
+ mIconUrl :String
+ mLatitude :double
+ mLongitude :double
+ mName :String
+ M POINTS
Road
+ mColor :int
+ mDescription :String
+ mName :String
+ mPoints :Point ([]) = new Point[] {}
+ mRoute :double ([][]) = new double[][]
+ mWidth :int
~ M ROAD
RoadProvider
+ getRoute(InputStream) :Road
+ getUrl(double, double, double, double) :String
+ getUrl(String, String) :String
+ getUrl(String) :String
DefaultHandler
KMLHandler
~ isItemIcon :boolean
~ isPlacemark :boolean
~ isRoute :boolean
- mCurrentElement :Stack = new Stack()
~ mRoad :Road
- mString :String
+ addPoint(Point[]) :Point[]
+ characters(char[], int, int) :void
- cleanup(String) :String
+ endElement(String, String, String) :void
+ KMLHandler()
- split(String, String) :String[]
+ startElement(String, String, String, Attributes) :void

Figure 4.6 Class Diagram 4

## 4.1.4 STATE DIAGRAM

A **state diagram** is a type of diagram used in computer science and related fields to describe the behaviour of systems. State diagrams require that the system described is composed of a finite number of states. Sometimes, this is indeed the case, while at other times this is a reasonable abstraction. There are many forms of state diagrams, which differ slightly and have different semantics.

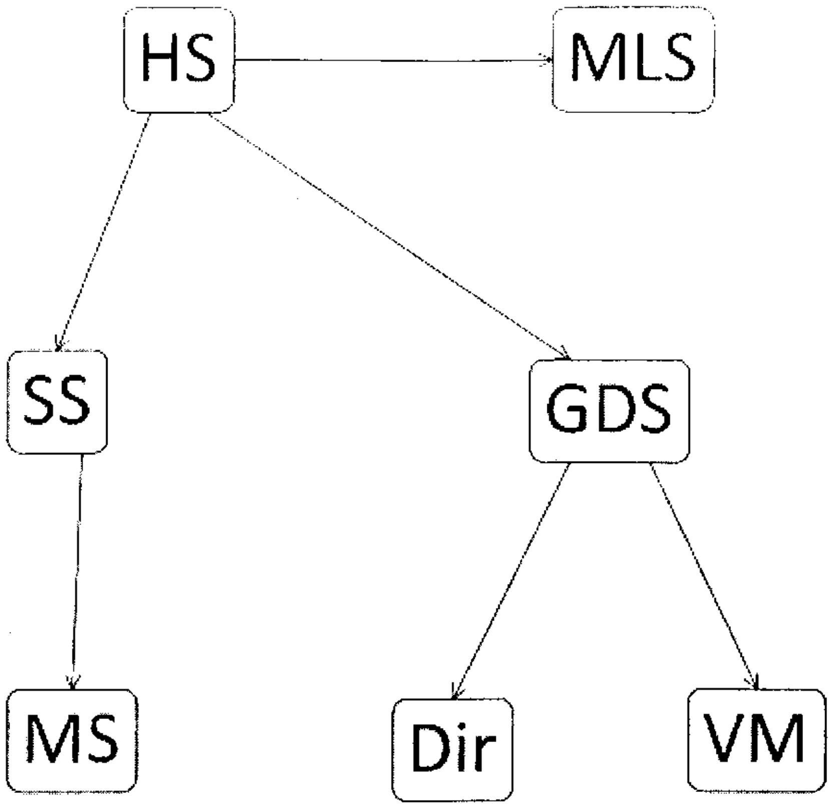
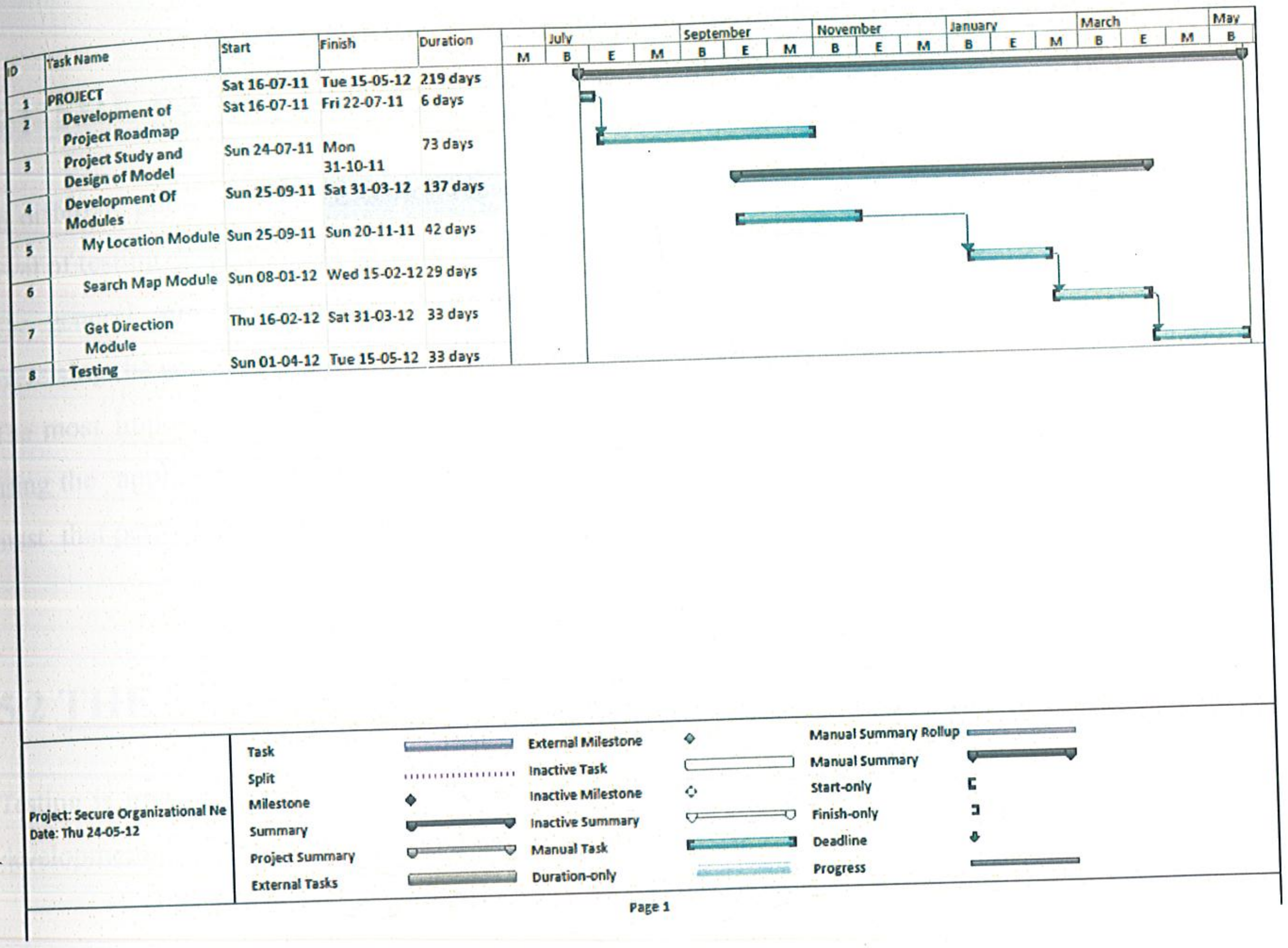


Figure 4.7: State Diagram of application

| S. No. | Short Name | Full Name |
|---|---|---|
| 1 | HS | Home Screen |
| 2 | MLS | My Location Screen |
| 3 | SS | Search Screen |
| 4 | GDS | Get Direction Screen |
| 5 | MS | Map Screen |
| 6 | Dir | Direction |
| 7 | VM | View Map |

Table 4.1: Components of state diagram

## 4.2 GANTT CHART



| ID | Task Name | Start | Finish | Duration |
|---|---|---|---|---|
| 1 | PROJECT | Sat 16-07-11 | Tue 15-05-12 | 219 days |
| 2 | Development of Project Roadmap | Sat 16-07-11 | Fri 22-07-11 | 6 days |
| 3 | Project Study and Design of Model | Sun 24-07-11 | Mon 31-10-11 | 73 days |
| 4 | Development Of Modules | Sun 25-09-11 | Sat 31-03-12 | 137 days |
| 5 | My Location Module | Sun 25-09-11 | Sun 20-11-11 | 42 days |
| 6 | Search Map Module | Sun 08-01-12 | Wed 15-02-12 | 29 days |
| 7 | Get Direction Module | Thu 16-02-12 | Sat 31-03-12 | 33 days |
| 8 | Testing | Sun 01-04-12 | Tue 15-05-12 | 33 days |

Project: Secure Organizational Ne
Date: Thu 24-05-12

Page 1

Figure 4.8: Gantt chart

Chapter 5

# Application Testing

## 5.1 THE GOAL OF TESTING

In different publications, the definition of testing varies according to the purpose, process, and level of testing described. Miller gives a good description of testing.

The general aim of testing is to affirm the quality of software systems by systematically exercising the software in carefully controlled circumstances.

The most important will be performance testing as when so many users will be simultaneously using the application and the database will be accessed as such frequent high rates it is must that performance should not deteriorate.

## 5.2 THE TESTING SPECTRUM

Testing is involved in every stage of our software, but the testing done at each level of software development is different in nature and has different objectives.

## 5.3 UNIT TESTING

This is done at the lowest level. It tests the basic unit of software, which is the smallest testable piece of software, and is often called –unit‖, –module‖, or –component‖ interchangeably.

## 5.4 INTEGRATION TESTING

This is performed when two or more tested units are combined into a larger structure. The test is often done on both the interfaces between the components and the larger structure being constructed, if its quality property cannot be assessed from its components.

## 5.5 SYSTEM TESTING

This type of testing is required to confirm the end-to-end quality of the entire system. It is based on the functional specification of the system. It is also useful when it comes to the checking of non-functional quality attributes, such as security, reliability and maintainability.

## 5.6 ACCEPTANCE TESTING

It is done when the completed system is handed over from the developers to the customers or users. The purpose of acceptance testing is rather to give confidence that the system is working than to find errors.

All the aforementioned techniques will be implemented by us in our next semester on the software that we aspire to make.

## 5.7 PERFORMANCE TESTING

Special tools are available which enable us to performance test any application with n number of users. Here it is very important.

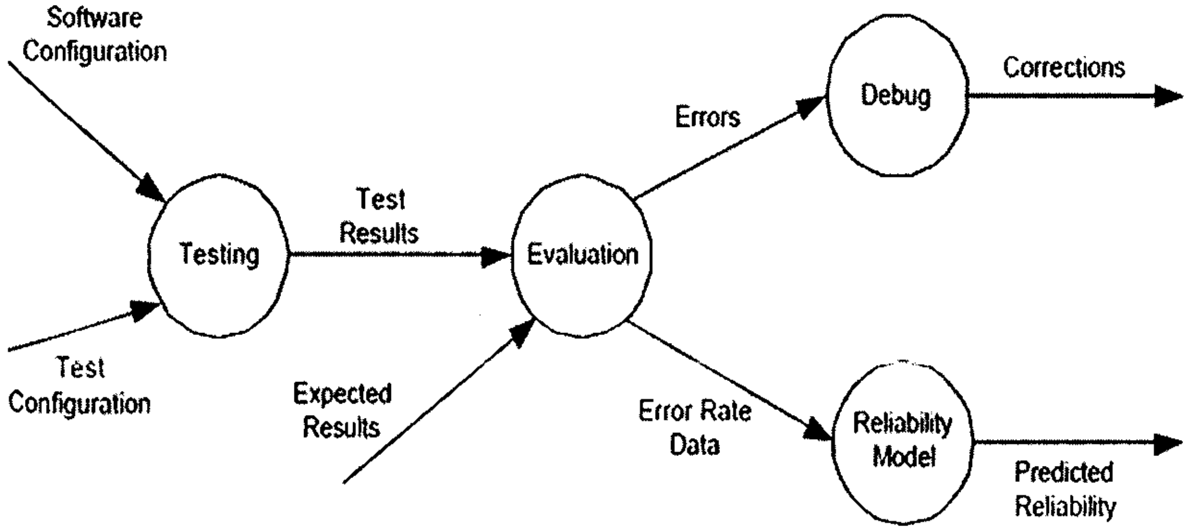


Fig 5.1: Testing Information Flow

UNIT TEST CASES :

| Sl. No. | Test case name | Test case description | Expected output | inference |
|---|---|---|---|---|
| 1 | My location | Test the location of the user using phone. | The function was giving the correct result | Pass |
| 2 | Get Direction | When the name of the places are entered it should show the route on map | We should get the direction on the map | Pass |
| 3 | Search Map | When the name of any place is entered it show on map | If wrong place is entered then null should be shown on screen else a map view. | Pass |

Table 5.2: Test Cases

## Testing Get Direction Function

| S.NO. | INPUT 1 | INPUT 2 | OUTPUT |
|---|---|---|---|
| 1 | Shimla | Chandigarh | Pass |
| 2 | Shimla | xyz | Null |
| 3 | 123 | 123 | Null |
| 4 | Xyz | Delhi | Null |

Table 5.3: Get Direction Test Cases

## Testing Search Map Function

| S.NO. | INPUT | OUTPUT |
|---|---|---|
| 1 | Delhi | Pass |
| 2 | 123 | Null |
| 3 | abc . | Null |

Table 5.4: Search Map Test Cases

# Chapter 6
# Conclusion

## 6.1 RESULTS AND CONCLUSION

The application gives the desired result in the form of directions from a given place to another. The destination is to be mentioned by the user and he gets the direction at each step from the source. The application also keeps a track of the places visited while reaching to the destination. Moreover, the user also has an option of getting to know about the nearby places such as restaurant, ATM, hospitals etc.

### 6.1.1 MY LOCATION

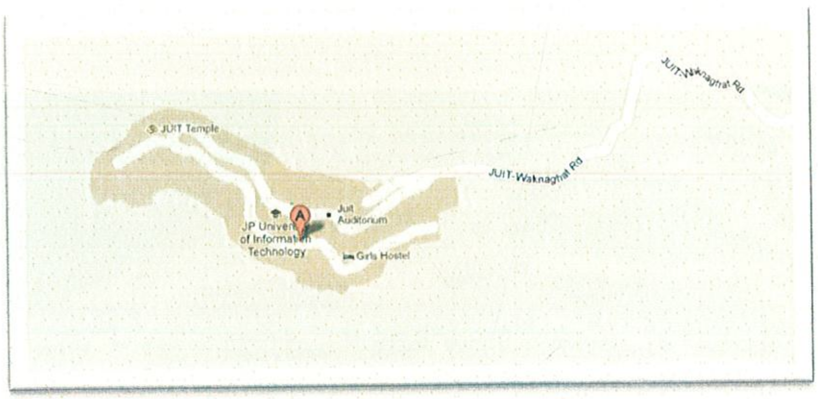
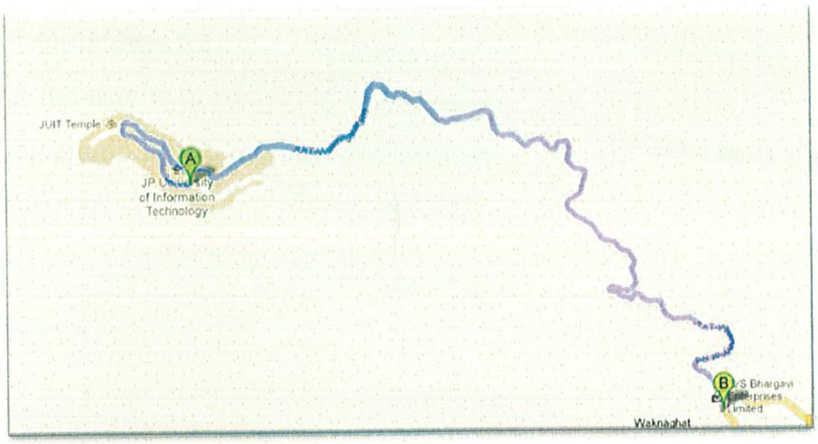


Figure 6.1: Show the current location of the user i.e. Jaypee University of Information Technology.

## 6.1.2 GET DIRECTION



JUIT Temple
JP University of Information Technology
S Bhargavi Enterprises Limited
Waknaghat

Figure 6.2 Showing Directions from JUIT to Waknaghat

## 6.1.3 SHOW MAP

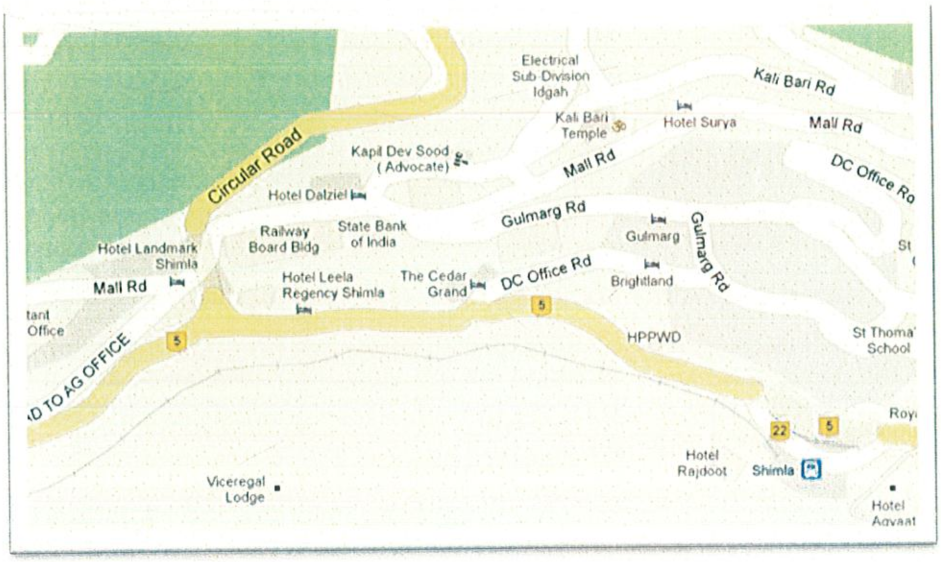


Figure 6.3: Showing the map of Mall Road Shimla. Also showing nearby places, hotels etc.

## 6.2 CONTRIBUTION OF THE PROJECT

There are many applications that exist in the market which provide the user with current location. But the application that we have built uses minimum resources when compared to other application that exist. Further this application can be extended to tag any particular place and write some memo about it. So, whenever the user is in particular radius of that place user would receive a notification about that place and the memo would remind user about the work that have to be done.

# References

1. **LBIS (Developing Real Time Tracking Application)** - M. A. Labrador, Alfredo J., Pedro Wightman.
2. **A Scalable Architecture for Global Sensing and Monitoring**, IEEE Networks Magazine- A. J. Perez, M. A. Labrador, S. Barbeau.
3. **Co-operative Location Update Algorithm for Mobiles In Next Generation Cellular Networks**-Samir K. Shah, Sirin Tekinay, Cem Saraydar.
4. **Foundation of Location Based Services-** Stefan Steiniger, Moritz Neun and Alistair Edwardes.
5. **What is Android?** Android Developers.
6. **Philosophy and Goals".** Android Open Source Project.
7. **Professional Android Application Development** - Reto Meier
8. **Hello, Android**- Ed Burnette
9. **Android Wireless Application Development**

# Appendix

## GET DIRECTION ACTIVITY

```
package thakur.lbis.controller;

import java.io.IOException;

import java.io.InputStream;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.URLConnection;

import java.util.ArrayList;

import java.util.Arrays;

import thakur.lbis.route.Road;

import thakur.lbis.route.RoadProvider;

import android.app.Activity;

import android.app.AlertDialog;

import android.app.AlertDialog.Builder;

import android.content.DialogInterface;

import android.content.Intent;

import android.net.Uri;

import android.os.Bundle;

import android.os.Handler;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.view.ViewGroup;

import android.view.View.OnClickListener;

import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;
public class GetDirectionActivity extends Activity implements OnClickListener
{
        EditText fromEditText,toEditText;
        Button directionButton,showMapButton;
        static Road mRoad;
        InputStream is;
        String url,fromString,toString,fromTempString,toTempString;
        String[] longitude,latitude;
        AlertDialog.Builder aboutDialog;
        AlertDialog helpDialog;
        ListView directionListView;
        @Override
        protected void onCreate(Bundle savedInstanceState)
{
                // TODO Auto-generated method stub
                super.onCreate(savedInstanceState);
                setContentView(R.layout.getdirection);
                directionButton=(Button) findViewById(R.id.buttonDirection);
                showMapButton=(Button) findViewById(R.id.buttonShowMap);
                directionButton.setOnClickListener(this);
                showMapButton.setOnClickListener(this);
        }
        public void onClick(View arg0) {
                // TODO Auto-generated method stub
                switch (arg0.getId()) {
```

```
case R.id.buttonDirection:
                fromEditText=(EditText) findViewById(R.id.editTextFrom);
                toEditText=(EditText) findViewById(R.id.editTextTo);
                fromTempString= fromEditText.getText().toString();
                toTempString=toEditText.getText().toString();
                fromString= fromTempString.replace(" ", "%20");
                toString= toTempString.replace(" ", "%20");
                url=RoadProvider.getUrl(fromString, toString);
                is = getConnection(url);
                mRoad = RoadProvider.getRoute(is);
                mHandler.sendEmptyMessage(0);
                directionListView= (ListView) findViewById(R.id.listView1);
                // String[] directionArrayList=Arrays.asList(mRoad.mPoints).toArray(new
String[mRoad.mPoints.length]);
                //String[] diStrings = null;
                //          String string = null;
                ArrayList<String> directionArrayList = new ArrayList<String>();
                for (int i=0;i<(mRoad.mPoints.length-2);i++)
{

        directionArrayList.add(i,(mRoad.mPoints[i].mName.toString()+","+mRoad.mPoints[i].mDescription.toString(
)));
                        //+","+ mRoad.mPoints[i].mDescription.toString()));
                }

ArrayAdapter<String> directionAdapter=new ArrayAdapter<String>(GetDirectionActivity.this,
android.R.layout.simple_list_item_1,directionArrayList);
directionListView.setAdapter(directionAdapter);
directionListView.setOnItemClickListener(newAdapterView.OnItemClickListener()
{

            public void onItemClick(AdapterView<?> arg0, View arg1,int arg2, long arg3)
```

```
// TODO Auto-generated method stub

        Toast.makeText(GetDirectionActivity.this, "position is "+arg2, Toast.LENGTH_LONG).show();    Intent
directionItemIntent= new Intent(GetDirectionActivity.this, MyItemMapActivity.class);

        directionItemIntent.putExtra("pName", mRoad.mPoints[arg2].mName.toString());

        directionItemIntent.putExtra("pDescription", mRoad.mPoints[arg2].mDescription.toString());

        directionItemIntent.putExtra("pIconUrl", mRoad.mPoints[arg2].mIconUrl.toString());

        directionItemIntent.putExtra("pLatitude", String.valueOf((int)((mRoad.mPoints[arg2].mLatitude) * 1e6)));

directionItemIntent.putExtra("pLongitude", String.valueOf((int)((mRoad.mPoints[arg2].mLongitude)* 1e6)));

startActivity(directionItemIntent);

                        }

                    });


                    break;

            case R.id.buttonShowMap:

                    /*Intent mapIntent= new Intent(this,MyMapActivity.class);

                    startActivity(mapIntent);*/

                    fromEditText=(EditText) findViewById(R.id.editTextFrom);

                    toEditText=(EditText) findViewById(R.id.editTextTo);

                    fromTempString= fromEditText.getText().toString();

                    toTempString=toEditText.getText().toString();

                    fromString= fromTempString.replace(" ", "%20");

                    toString= toTempString.replace(" ", "%20");

                    Intent intent = new Intent(android.content.Intent.ACTION_VIEW,
        Uri.parse("http://maps.google.com/maps?saddr="+fromString+"&daddr="+toString));

                    startActivity(intent);

                    break;

            }

    }


    private InputStream getConnection(String url) {

            //InputStream is = null;

            try {
                URLConnection conn = new URL(url).openConnection();
                is = conn.getInputStream();
            } catch (MalformedURLException e)
{
    e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }
            return is;
    }
    Handler mHandler = new Handler() {
            public void handleMessage(android.os.Message msg) {
                TextView textView = (TextView) findViewById(R.id.textView1);
                textView.setText(mRoad.mName + " " + mRoad.mDescription);
                /* MapOverlay mapOverlay = new MapOverlay(mRoad, mapView);
      List<Overlay> listOfOverlays = mapView.getOverlays();
      listOfOverlays.clear();
      listOfOverlays.add(mapOverlay);
      mapView.invalidate();
                */                      };
    };

    public boolean onCreateOptionsMenu(Menu menu) {
            MenuInflater myInflater= getMenuInflater();
            myInflater.inflate(R.menu.mymenu, menu);
            // TODO Auto-generated method stub
            return super.onCreateOptionsMenu(menu);
    }
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
            // TODO Auto-generated method stub
```

```
switch (item.getItemId()) {
case R.id.item1:
        helpDialog= new AlertDialog.Builder(GetDirectionActivity.this).create();
        helpDialog.setTitle("Help");
        helpDialog.setMessage(getResources().getString(R.string.about_content));
        helpDialog.setButton("Ok", new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int which) {
                        // TODO Auto-generated method stub
                        helpDialog.dismiss();
                }
        });

        helpDialog.show();
        break;
case R.id.item2:
        aboutDialog= new AlertDialog.Builder(GetDirectionActivity.this);
        aboutDialog.setTitle("About Us");
        LayoutInflater inflater =
(LayoutInflater)this.getSystemService(LAYOUT_INFLATER_SERVICE);

          View layout = inflater.inflate(R.layout.about_dialog,
(ViewGroup)findViewById(R.id.root));
        /*aboutDialog.setMessage("");
        aboutDialog.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);
        //aboutDialog.setContentView(getLayoutInflater().inflate(R.layout.image_layout, null));

        aboutDialog.setContentView(R.layout.image_layout);
        aboutDialog.setButton("Ok", new DialogInterface.OnClickListener() {

                public void onClick(DialogInterface dialog, int which) {
```

```
                                        // TODO Auto-generated method stub

                                        aboutDialog.dismiss();

                                }

                        });*/

                        aboutDialog.setView(layout);

                        aboutDialog.show();

                        break;

                case R.id.item3:

                finish();

                        break;

                }

                return super.onOptionsItemSelected(item);

        }

}
```

## MY LOCATION ACTIVITY

```
package thakur.lbis.controller;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import thakur.lbis.overlay.MapItemizedOverlay;
import thakur.lbis.overlay.MyOverLay;

import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.content.Context;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;
import android.location.Address;
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;
import android.widget.Toast;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;

public class MyLocationActivity extends MapActivity {


        String addressString;
        Location myLocation;
        MyLocationOverlay myLocationOverlay;
        MapView mapView;
        AlertDialog.Builder aboutDialog;
        AlertDialog helpDialog;

        @Override
        protected boolean isRouteDisplayed() {
                // TODO Auto-generated method stub
                return false;
        }

        @Override
        protected void onCreate(Bundle icicle) {
                // TODO Auto-generated method stub
                super.onCreate(icicle);
                setContentView(R.layout.my_map);
                mapView = (MapView) findViewById(R.id.mapview);
                mapView.setBuiltInZoomControls(true);

                List<Overlay> mapOverlays = mapView.getOverlays();
                Drawable drawable = this.getResources().getDrawable(R.drawable.google_maps_pin);
                MapItemizedOverlay itemizedoverlay = new MapItemizedOverlay(drawable, this);


                LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
                //List<String> providers = locationManager.getProviders(true);
/*              LocationListener mlocListener = new MyLocationListener();
```

```
        locationManager.requestLocationUpdates( LocationManager.GPS_PROVIDER, 0, 0,
mlocListener); */
        String locationProvider = locationManager.GPS_PROVIDER;
        myLocation=locationManager.getLastKnownLocation(locationProvider);
        int latitude = (int) (myLocation.getLatitude() * 1e6);
        int longitude = (int) (myLocation.getLongitude() * 1e6);
        MapController mapController = mapView.getController();
        GeoPoint point1 = new GeoPoint(latitude, longitude);

        int maxZoom = mapView.getMaxZoomLevel();
int initZoom = maxZoom-2;
mapController.setZoom(initZoom);

        mapController.animateTo(point1);
        OverlayItem overlayitem = new OverlayItem(point1, "My Location", addressString);
        itemizedoverlay.addOverlay(overlayitem);
        mapOverlays.add(itemizedoverlay);

        mapController.animateTo(point1);

}
public class MyLocationListener implements LocationListener

{

        public void onLocationChanged(Location loc)

        {

                loc.getLatitude();

                loc.getLongitude();

                String Text = "My current location is: " + "Latitude = " + loc.getLatitude() + "Longitud = "
+ loc.getLongitude();

                mapView.getController().animateTo(getGeoByLocation(loc));

                Toast.makeText( getApplicationContext(),Text,Toast.LENGTH_SHORT).show();

        }


        public void onProviderDisabled(String provider)

        {

                Toast.makeText( getApplicationContext(), "Gps Disabled",Toast.LENGTH_SHORT
).show();

        }


        public void onProviderEnabled(String provider)
        {
                Toast.makeText( getApplicationContext(),"Gps Enabled", Toast.LENGTH_SHORT).show();
        }

        public void onStatusChanged(String provider, int status, Bundle extras)

        {

        }

}
private GeoPoint getGeoByLocation(Location location)
{
        GeoPoint gp = null;
        try
        {
                if (location != null)
                {
                        double geoLatitude = location.getLatitude()*1E6;
                        double geoLongitude = location.getLongitude()*1E6;
                        gp = new GeoPoint((int) geoLatitude, (int) geoLongitude);
                }
        }
        catch(Exception e)
        {
                e.printStackTrace();
        }
        return gp;
}

public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater myInflater= getMenuInflater();
        myInflater.inflate(R.menu.mymenu, menu);
        // TODO Auto-generated method stub
        return super.onCreateOptionsMenu(menu);
}
@Override
public boolean onOptionsItemSelected(MenuItem item) {
        // TODO Auto-generated method stub

        switch (item.getItemId()) {
        case R.id.item1:
                helpDialog= new AlertDialog.Builder(MyLocationActivity.this).create();
                helpDialog.setTitle("Help");
                helpDialog.setMessage(getResources().getString(R.string.about_content));
                helpDialog.setButton("Ok", new DialogInterface.OnClickListener() {

                        public void onClick(DialogInterface dialog, int which) {
                                // TODO Auto-generated method stub
                                helpDialog.dismiss();
                        }
                });

                helpDialog.show();
                break;
                case R.id.item2:
                        aboutDialog= new AlertDialog.Builder(MyLocationActivity.this);
                        aboutDialog.setTitle("About Us");
                        LayoutInflater inflater =
(LayoutInflater)this.getSystemService(LAYOUT_INFLATER_SERVICE);

                        View layout = inflater.inflate(R.layout.about_dialog, (ViewGroup)findViewById(R.id.root));
                        /*aboutDialog.setMessage("");
                        aboutDialog.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);
                        //aboutDialog.setContentView(getLayoutInflater().inflate(R.layout.image_layout, null));

                        aboutDialog.setContentView(R.layout.image_layout);
                        aboutDialog.setButton("Ok", new DialogInterface.OnClickListener() {

                                public void onClick(DialogInterface dialog, int which) {
                                        // TODO Auto-generated method stub
                                        aboutDialog.dismiss();
                                }
                        });*/

                        aboutDialog.setView(layout);
                        aboutDialog.show();

                        break;
                case R.id.item3:
                        finish();
                        break;


                }
                return super.onOptionsItemSelected(item);
        }
}
```

## SEARCH MAP ACTIVITY

```
package thakur.lbis.controller;


import java.io.IOException;

import java.io.InputStream;

import java.net.MalformedURLException;

import java.net.URL;

import java.net.URLConnection;
import java.util.List;


import thakur.lbis.overlay.MapItemizedOverlay;

import thakur.lbis.route.Road;

import thakur.lbis.route.RoadProvider;

import android.app.Activity;

import android.app.AlertDialog;

import android.app.AlertDialog.Builder;

import android.content.DialogInterface;

import android.graphics.drawable.Drawable;

import android.os.Bundle;

import android.view.LayoutInflater;

import android.view.Menu;

import android.view.MenuInflater;

import android.view.MenuItem;

import android.view.View;

import android.view.ViewGroup;


import com.google.android.maps.GeoPoint;

import com.google.android.maps.MapActivity;

import com.google.android.maps.MapController;

import com.google.android.maps.MapView;

import com.google.android.maps.Overlay;

import com.google.android.maps.OverlayItem;


public class SearchMapActivity extends MapActivity {


        String searchString;

        GeoPoint myGeoPoint;

        MapView mapView;

        static Road mRoad;
```

```
    String url;

    InputStream is;

    AlertDialog.Builder aboutDialog;

    AlertDialog helpDialog;


    @Override

    protected void onCreate(Bundle savedInstanceState) {

            // TODO Auto-generated method stub

            super.onCreate(savedInstanceState);

            setContentView(R.layout.my_map);


            searchString= getIntent().getExtras().getString("search");

            mapView = (MapView) findViewById(R.id.mapview);

            mapView.setBuiltInZoomControls(true);


            List<Overlay> mapOverlays = mapView.getOverlays();

            Drawable drawable = this.getResources().getDrawable(R.drawable.google_maps_pin);

            MapItemizedOverlay itemizedoverlay = new MapItemizedOverlay(drawable, this);


            url= RoadProvider.getUrl(searchString);

            is = getConnection(url);

      mRoad = RoadProvider.getRoute(is);

      MapController mapController = mapView.getController();

      int latitude = (int) ((mRoad.mPoints[0].mLatitude) * 1e6);

            int longitude = (int) ((mRoad.mPoints[0].mLongitude) * 1e6);

      myGeoPoint= new GeoPoint(latitude,longitude);

      int maxZoom = mapView.getMaxZoomLevel();

int initZoom = maxZoom-2;

mapController.setZoom(initZoom);

            mapController.animateTo(myGeoPoint);

            OverlayItem overlayitem = new OverlayItem(myGeoPoint, "Location for ", searchString);
```

```
        itemizedoverlay.addOverlay(overlayitem);

        mapOverlays.add(itemizedoverlay);


}

private InputStream getConnection(String url) {

        //InputStream is = null;

         try {

          URLConnection conn = new URL(url).openConnection();

          is = conn.getInputStream();

         } catch (MalformedURLException e) {

          e.printStackTrace();

         } catch (IOException e) {

          e.printStackTrace();

         }

         return is;

}

@Override

protected boolean isRouteDisplayed() {

        // TODO Auto-generated method stub

        return false;

}

public boolean onCreateOptionsMenu(Menu menu) {

        MenuInflater myInflater= getMenuInflater();

        myInflater.inflate(R.menu.mymenu, menu);

        // TODO Auto-generated method stub

        return super.onCreateOptionsMenu(menu);

}

@Override

public boolean onOptionsItemSelected(MenuItem item) {

        // TODO Auto-generated method stub
```

```
switch (item.getItemId()) {

case R.id.item1:

        helpDialog= new AlertDialog.Builder(SearchMapActivity.this).create();

        helpDialog.setTitle("Help");

        helpDialog.setMessage(getResources().getString(R.string.about_content));

        helpDialog.setButton("Ok", new DialogInterface.OnClickListener() {


                public void onClick(DialogInterface dialog, int which) {

                        // TODO Auto-generated method stub

                        helpDialog.dismiss();

                }

        });


        helpDialog.show();

        break;

case R.id.item2:

        aboutDialog= new AlertDialog.Builder(SearchMapActivity.this);

        aboutDialog.setTitle("About Us");

        LayoutInflater inflater =
(LayoutInflater)this.getSystemService(LAYOUT_INFLATER_SERVICE);


          View layout = inflater.inflate(R.layout.about_dialog,
(ViewGroup)findViewById(R.id.root));

        /*aboutDialog.setMessage("");

        aboutDialog.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);

        //aboutDialog.setContentView(getLayoutInflater().inflate(R.layout.image_layout, null));


        aboutDialog.setContentView(R.layout.image_layout);

        aboutDialog.setButton("Ok", new DialogInterface.OnClickListener() {


                public void onClick(DialogInterface dialog, int which) {

                        // TODO Auto-generated method stub
```

```
                    aboutDialog.dismiss();
                }
            });*/

            aboutDialog.setView(layout);
            aboutDialog.show();

            break;
        case R.id.item3:
        finish();
            break;

        }
        return super.onOptionsItemSelected(item);
    }
}
```

## LBIS PROJECTACTIVITY

```
package thakur.lbis.controller;

import android.app.Activity;
import android.app.AlertDialog;
import android.app.Dialog;
import android.content.DialogInterface;
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.ViewGroup;
import android.view.Window;
import android.widget.Button;
import android.widget.EditText;

public class LBISProjectActivity extends Activity implements OnClickListener {
        /** Called when the activity is first created. */

        Button myLocationButton, getDirectionButton, searchMapButton;
        EditText searchMapEditText;
        String searchString;
        Dialog myDialog;
        AlertDialog.Builder aboutDialog;
        AlertDialog helpDialog;
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);

                myLocationButton= (Button) findViewById(R.id.buttonMyLocation);
                getDirectionButton= (Button) findViewById(R.id.buttonGetDirection);
                searchMapButton= (Button) findViewById(R.id.buttonSearchMap);
                myLocationButton.setOnClickListener(this);
                getDirectionButton.setOnClickListener(this);
                searchMapButton.setOnClickListener(this);

        }
        public void onClick(View arg0) {
```

```
        // TODO Auto-generated method stub

        switch (arg0.getId()) {
        case R.id.buttonMyLocation:
                Intent myLocationIntent=new Intent(this,MyLocationActivity.class);
                startActivity(myLocationIntent);


                break;
        case R.id.buttonGetDirection:
                Intent getDirectionIntent=new Intent(this,GetDirectionActivity.class);
                startActivity(getDirectionIntent);
                break;
        case R.id.buttonSearchMap:


                myDialog= new Dialog(this);
                myDialog.setContentView(R.layout.search_dialog);
                myDialog.setTitle("Search on Map");
                myDialog.setCancelable(true);
                /*EditText searchMapEditText=(EditText)
myDialog.findViewById(R.id.editTextSearchDialog);
                searchString= searchMapEditText.getText().toString();*/


                Button searchDialogButton= (Button) myDialog.findViewById(R.id.buttonSearchDialog);
                searchDialogButton.setOnClickListener(new OnClickListener() {


                        public void onClick(View v) {
                                // TODO Auto-generated method stub
                                searchMapEditText=(EditText)
myDialog.findViewById(R.id.editTextSearchDialog);
                                searchString= searchMapEditText.getText().toString();
                                String temp= searchString.replace(" ", "%20");
```

```
                                        Intent searchMapIntent=new
Intent(LBISProjectActivity.this,SearchMapActivity.class);
                                        searchMapIntent.putExtra("search", temp);
                                        startActivity(searchMapIntent);
                                }
                        });
                        Button clearDialogButton= (Button) myDialog.findViewById(R.id.buttonClearDialog);
                        clearDialogButton.setOnClickListener(new OnClickListener() {

                                public void onClick(View v) {
                                        // TODO Auto-generated method stub
                                        searchMapEditText.setText("");
                                }
                        });
                        Button cancelDialogButton= (Button) myDialog.findViewById(R.id.buttonCancelDialog);
                        cancelDialogButton.setOnClickListener(new OnClickListener() {

                                public void onClick(View v) {
                                        // TODO Auto-generated method stub
                                        myDialog.dismiss();
                                }
                        });
                        myDialog.show();
                        break;
                default:
                        break;
                }

        }
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
                MenuInflater myInflater= getMenuInflater();
                myInflater.inflate(R.menu.mymenu, menu);

                // TODO Auto-generated method stub

                return super.onCreateOptionsMenu(menu);

        }

        @Override

        public boolean onOptionsItemSelected(MenuItem item) {

                // TODO Auto-generated method stub


                switch (item.getItemId()) {

                case R.id.item1:

                        helpDialog= new AlertDialog.Builder(LBISProjectActivity.this).create();

                        helpDialog.setTitle("Help");

                        helpDialog.setMessage(getResources().getString(R.string.about_content));

                        helpDialog.setButton("Ok", new DialogInterface.OnClickListener() {


                                public void onClick(DialogInterface dialog, int which) {

                                        // TODO Auto-generated method stub

                                        helpDialog.dismiss();

                                }

                        });


                        helpDialog.show();

                        break;

                case R.id.item2:

                        aboutDialog= new AlertDialog.Builder(LBISProjectActivity.this);

                        aboutDialog.setTitle("About Us");

                        LayoutInflater inflater =
(LayoutInflater)this.getSystemService(LAYOUT_INFLATER_SERVICE);


                         View layout = inflater.inflate(R.layout.about_dialog,
(ViewGroup)findViewById(R.id.root));

                        /*aboutDialog.setMessage("");
```

```
                    aboutDialog.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);
                    //aboutDialog.setContentView(getLayoutInflater().inflate(R.layout.image_layout, null));


                    aboutDialog.setContentView(R.layout.image_layout);
                    aboutDialog.setButton("Ok", new DialogInterface.OnClickListener() {


                              public void onClick(DialogInterface dialog, int which) {
                                        // TODO Auto-generated method stub
                                        aboutDialog.dismiss();
                              }
                    });*/


                    aboutDialog.setView(layout);
                    aboutDialog.show();
                    break;
          case R.id.item3:
          finish();
                    break;
          }
          return super.onOptionsItemSelected(item);
     }

}
```

## MY ITEM MAP ACTIVITY

```
package thakur.lbis.controller;


import java.io.IOException;
import java.io.InputStream;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.URLConnection;
import java.util.List;


import thakur.lbis.overlay.MapItemizedOverlay;
import android.R.integer;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.content.DialogInterface;
import android.graphics.drawable.Drawable;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.ViewGroup;


import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.Overlay;
import com.google.android.maps.OverlayItem;
```

```
public class MyItemMapActivity extends MapActivity {

	AlertDialog.Builder aboutDialog;
	AlertDialog helpDialog;
	InputStream is;



	@Override
	protected boolean isRouteDisplayed() {
		// TODO Auto-generated method stub
		return false;
	}
	@Override
	protected void onCreate(Bundle icicle) {
		// TODO Auto-generated method stub
		super.onCreate(icicle);

		setContentView(R.layout.my_map);

		MapView mapView = (MapView) findViewById(R.id.mapview);
		mapView.setBuiltInZoomControls(true);
		MapController mapController = mapView.getController();
		String pName= getIntent().getExtras().getString("pName");
		String pDescription= getIntent().getExtras().getString("pDescription");
		String pIconUrl= getIntent().getExtras().getString("pIconUrl");
		int pLatitude= Integer.valueOf(getIntent().getExtras().getString("pLatitude"));
		int pLongitude= Integer.valueOf(getIntent().getExtras().getString("pLongitude"));

		List<Overlay> mapOverlays = mapView.getOverlays();
		/*URL url = null;
```

```
            try {

                    is = url.openStream();

            } catch (IOException e) {

                    // TODO Auto-generated catch block

                    e.printStackTrace();

            }*/


            Drawable drawable =this.getResources().getDrawable(R.drawable.playback_play);

            MapItemizedOverlay itemizedoverlay = new MapItemizedOverlay(drawable, this);

            GeoPoint myGeoPoint= new GeoPoint(pLatitude,pLongitude);


            int maxZoom = mapView.getMaxZoomLevel();

int initZoom = maxZoom-2;

mapController.setZoom(initZoom);

            mapController.animateTo(myGeoPoint);

            OverlayItem overlayitem = new OverlayItem(myGeoPoint, pName, pDescription);

            itemizedoverlay.addOverlay(overlayitem);

            mapOverlays.add(itemizedoverlay);

    }

    public boolean onCreateOptionsMenu(Menu menu) {

            MenuInflater myInflater= getMenuInflater();

            myInflater.inflate(R.menu.mymenu, menu);

            // TODO Auto-generated method stub

            return super.onCreateOptionsMenu(menu);

    }

    @Override

    public boolean onOptionsItemSelected(MenuItem item) {

            // TODO Auto-generated method stub


            switch (item.getItemId()) {

            case R.id.item1:
```

```
				helpDialog= new AlertDialog.Builder(MyItemMapActivity.this).create();
				helpDialog.setTitle("Help");
				helpDialog.setMessage(getResources().getString(R.string.about_content));
				helpDialog.setButton("Ok", new DialogInterface.OnClickListener() {

					public void onClick(DialogInterface dialog, int which) {
						// TODO Auto-generated method stub
						helpDialog.dismiss();
					}
				});

				helpDialog.show();
				break;
			case R.id.item2:
				aboutDialog= new AlertDialog.Builder(MyItemMapActivity.this);
				aboutDialog.setTitle("About Us");
				LayoutInflater inflater =
(LayoutInflater)this.getSystemService(LAYOUT_INFLATER_SERVICE);

				View layout = inflater.inflate(R.layout.about_dialog,
(ViewGroup)findViewById(R.id.root));
				/*aboutDialog.setMessage("");
				aboutDialog.getWindow().requestFeature(Window.FEATURE_CUSTOM_TITLE);
				//aboutDialog.setContentView(getLayoutInflater().inflate(R.layout.image_layout, null));
				aboutDialog.setContentView(R.layout.image_layout);
				aboutDialog.setButton("Ok", new DialogInterface.OnClickListener() {
					public void onClick(DialogInterface dialog, int which) {
						// TODO Auto-generated method stub
						aboutDialog.dismiss();
					}
				});*/
```

```
                aboutDialog.setView(layout);

                aboutDialog.show();


                break;

        case R.id.item3:

        finish();

                break;

        }

        return super.onOptionsItemSelected(item);

    }

}
```