GEOAWARE APPLICATION ON ANDROID

A Thesis

Presented to the

Faculty of

San Diego State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Neha Sindavala Karnam

Summer 2011

SAN DIEGO STATE UNIVERSITY

The Undersigned Faculty Committee Approves the

Thesis of Neha Sindavala Karnam

GeoAware Application on Android

Joseph Lewis

Joseph Lewis, Chair
Department of Computer Science

Kris Stewart

Kris Stewart
Department of Computer Science

[illegible]

Caroline A. Macera
Graduate School of Public Health

5/23/11

Approval Date

DEDICATION

This work is dedicated to my parents, teachers and my best friends.

ABSTRACT OF THE THESIS

GeoAware Application on Android
by
Neha Sindavala Karnam
Master of Science in Computer Science
San Diego State University, 2011

The increasing availability of GPS-enabled devices is changing the way people interact with the web and bringing a large number of GPS trajectories representing people locations.

GeoAware is a location-aware to-do list application; unlike typical calendar applications which reminds you when to do something, GeoAware also reminds you where to do it. Unlike other alarms, which issue time based alerts, GeoAware also issues location-based alerts. GeoAware which lets you create to-do lists on your Android phone and then have the phone tell you when you're going to be able to do them. In other words, you can enter that you need to pick up groceries; then when you're in the area of the grocery store your Android phone will notify you that it would be a good time to do that errand.

All of the errands are conveniently laid out on a map, this allows the user to compare the current location and to the location of things which he/she need to get. As the user travels, the scanning module, scans within the radius of the current location to the possible true value of the locations tagged on the list and notifies the user once found. The user can then decide to complete the errand or to-do later.

Challenges included developing a user interface for a small screen, power management, and reliability.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

1.	GPS	Global Positioning System
2.	NLP	Network Location Provider
3.	GNSS	Global Navigation Satellite System
4.	LBS	Location-based services
5.	OS	Operating System
6.	PC	Personal Computer
7.	RFID	Radio-Frequency Identification
8.	VM	Virtual Machine
9.	UI	User Interface
10.	GUI	Graphical User Interface
11.	AWT	Abstract Window Toolkit
12.	SWT	The Standard Widget Toolkit
13.	J2ME	Java Platform Micro Edition
14.	SDK	Software Development Kit
15.	SQL	Structured Query Language
16.	API	Application Programming Interface
17.	ID	Identifier

ACKNOWLEDGEMENTS

This research project would not have been possible without the support of many people. I wish to express my gratitude to my adviser, Prof Dr. Joseph Lewis who offered me invaluable assistance, guidance and support. My deepest gratitude to Prof Dr. Kris Stewart and Prof Dr. Caroline A. Macera without whose support and assistance this study would not have been successful.

Thanks to my parents for their moral support. Special thanks to my friends - Siddartha Reddy, Jyothi Harish and Kishore Reddy for providing invaluable assistance and advice.

CHAPTER 1

INTRODUCTION

In the recent times, we have noticed how the mobile technology has shown immense growth. It has made its way into every aspect of our lives. We have seen laptops and personal organizers evolve into all-in-one smart phones, cloud computing and virtual private networks allowing accessing any information you want, wherever you are. Landline phones are beginning to slip into memory, and mobile communications and technology are on the rise, expanding and growing rapidly all over the globe. With the introduction of intelligent and smart phones, the handset industry has changed from low budget handset devices to high-performance mobile devices. Today's phone is almost everything - it is fashionable, innovative, appealing, high-performing, durable, stylish and multi-tasking. It can serve many purposes such as email, blogging, browsing internet, games, multimedia, access to social networking sites like YouTube, Google search, Facebook and more.

The worldwide demand for smartphones has also fueled the business market [1]. Smartphones show higher application usage than feature phones even at the basic built-in application level. The percentage of people who use their phone for only voice communications has been dropping with more people interested in smart phone and what more it offers. The use of the built-in camera, video capability and Wi-Fi features also add to the popularity of the smart phones.

Mobile phone applications, so-called apps, have experienced dramatic growth globally with people today looking for more information on the move. This is one area of mobile phone technology enhancement that allows developers and programmers to offer users just what they seek under their preferred area of interest. Google's Android is one of the latest and unique innovations, which instantly has taken over the mobile market. It is an open source mobile platform which allows developers from around the world to develop applications for Android supported mobile devices. Android supports to develop a location-aware application utilizing Global Positioning System (GPS) and Android's Network Location Provider (NLP) to acquire the user location. Although GPS is most accurate, it only works outdoors; it quickly consumes battery power, and doesn't return the location as quickly as users want [2]. Android's Network Location Provider determines user location using cell tower and Wi-Fi signals, providing location information which works indoors and outdoors, responds faster, and uses less battery power. To obtain the user location in the application, both GPS and the Network Location Provider can be used or just one.

With the increasing focus on mobile applications and penetration of GPS based mobile phones, the focus is on Location-Based applications. Location-based apps are the hottest in the app market. Ten billion apps have been downloaded in the past three years. There are 17,000 location-based apps on the market, and 160 million app-compatible devices are owned worldwide – iPhones, Androids, BlackBerrys and tablet devices such as the iPad and Motorola Xoom [3].

In this report, I have explained my research which is based on development of a user-friendly Android application called GeoAware. This is a location-aware to-do application designed to be able to create a to-do list and prompt the user when and where to do it. GeoAware is a location-aware to-do list application; unlike typical calendar applications which reminds you when to do something, GeoAware also reminds you where to do it. Unlike other alarms, which issue time based alerts, GeoAware also issues location-based alerts. GeoAware which lets you create To Do lists on your Android phone and then have the phone tell you when you’re going to be able to do them. In other words, you can enter that you need to pick up groceries; then when you’re in the area of the grocery store your Android phone will notify you that it would be a good time to do that errand.

Chapter 2 deals with Literature describing the research and study of this thesis project and challenges which were faced. The application design and high-level architecture is discussed in Chapter 3 followed my implementation in Chapter 4. Chapter 5 is a user manual for developers and end-users on how to install the application and use it. Future enhancements and conclusion are explained in Chapter 6.

CHAPTER 2

LITERATURE

2.1 Mobile Platform

Smart phones today have made our life easier with millions of people around the world who use their phones to manage their lives, stay in touch with friends, and use the internet for a lot of things. Users of smart phones have come to take it for granted that they can surf the internet from almost any location and do everything they could do with a computer. When mobile phones were first introduced to the public, they were bulky, expensive, and some even required a base unit that had to be transported along with the phone. Good reception was a major problem and in general, early mobile phones could only be used in certain locations were the signal was particularly strong. As mobile phone technology advanced, the difficult in using them became less of a problem. Today, mobile phone reception has improved greatly due to the use of satellites and wireless services. As mobile phones evolved and became simple to use, the importance of them increased accordingly.

It was only a matter of time before Google started to dominate the smart phone arena, and they have, their Android which is now among the popular smartphone operating system on the market. The Android is a smart phone operating system created by Google. The reason for the popularity of the Android is all of the apps available. One of the great advantages of Android is its capacity to run several applications simultaneously. Android gives users the freedom to listen to music, receive notifications etc. without keeping application open. At the same times users can download their favorite apps from the Android market.

Global Positioning System, popular and commonly known as GPS is a space-based global navigation satellite system (GNSS) that provides location and time information in all weather, anywhere on or near the Earth, where there is an unobstructed line of sight to four or more GPS satellites. It is maintained by the United States government and is freely accessible by anyone with a GPS receiver. The U.S. military developed and implemented this satellite network as a military navigation system, but soon opened it up to everybody else.

Thanks to the development of GPS technology, an advanced navigation technology-real time GPS service becomes true. It’s been fast and wildly adopted in many fields and is becoming to play a crucial role in our modern life, especially in security and safety field.

GPS on smartphones is no longer an emerging trend. It's almost a must-have feature nowadays, and more and more handsets are offering it. With the embedded GPS receiver and a mapping service, one can get real-time position tracking, text- and voice-guided directions, and points of interest. The latest location-based services are also expected to boost demand in the GPS handsets industry. A collaborative effort among telecom carriers, mobile phone makers and GPS chip providers has expanded location-based services (LBS) applications and increased demand for models that support these functions.

2.2 RESEARCH AND STUDY

While Apple and Google seem to be splitting the smartphone market and looking at the market share overall, Android is beginning to open up a considerable lead when it comes to recent smartphone adopters (see Figure 2.1 [4]). The ability to customize many aspects of the Android operating system (OS), coupled with the fact that it is open source encourages the device manufacturers and the carriers to tweak the OS and make it exclusive to them unlike other OSs.

Android is steadily climbing the wall of the smartphone OS in the market. Since January, Android’s mobile ad consumption has increased almost 1000%, thanks to the slew of new handsets coming out from every nook and corner. As the months continue, and more handsets are released, along with the debut of Android tablets, we can expect this market share to only increase (see Figure 2.2) [4].

The Global Positioning System (GPS) has become an integral part of daily life for many individuals and businesses, as well as for the government and military. The **GPS** is actually a **constellation** of 27 Earth-orbiting satellites (24 in operation and three extras in case one fails). Each of these 3,000- to 4,000-pound solar-powered satellites circles the globe at about 12,000 miles (19,300 km), making two complete rotations every day. The orbits are arranged so that at anytime, anywhere on Earth, there are at least four satellites "visible" in the sky [5].

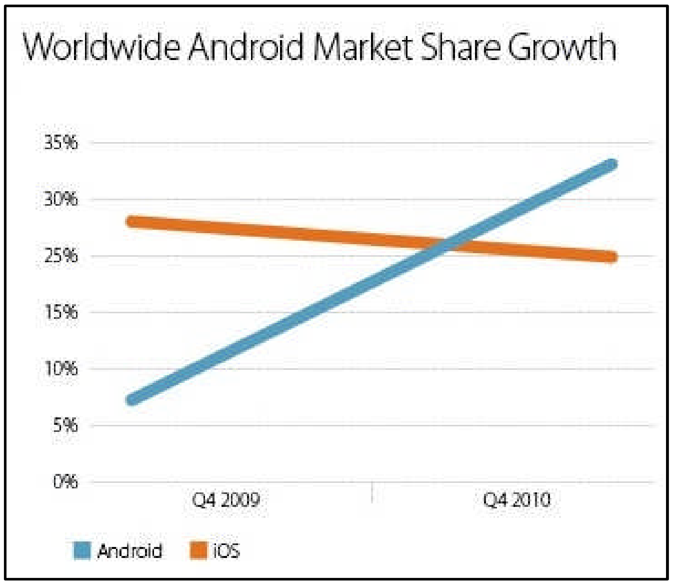


Figure 2.1. Android and iOS market share iOS data includes iPod Touch. Source: MILLENNIAL MEDIA, *Mobilemix: The media device index*. Millennial Media, http://www.millennialmedia.com/wp-content/images/mobilemix/MM-MobileMix-March2011.pdf, accessed April 2011, 2011.

A GPS receiver's job is to locate four or more of these satellites, figure out the distance to each, and use this information to deduce its own location. This operation is based on a simple mathematical principle called trilateration.

In the recent past, the wireless world has been buzzing about LBS, applications made primarily for wireless devices that use GPS, Wi-Fi or cellular radio signals (or a combination of all three) that enable consumers to get information and advertisers to reach them based on their location. Overall**, 1 in 3 smartphone owners currently uses a Location Based Service at least once a month. An additional 20% of smartphone users said that they would be interested in using LBS if they knew more** about what was available and how to use them [6]. Figure 2.3 [6] shows some specific types of LBS applications consumers are using now (shown in the dark blue bars) or would use if they had more information (light blue bars).

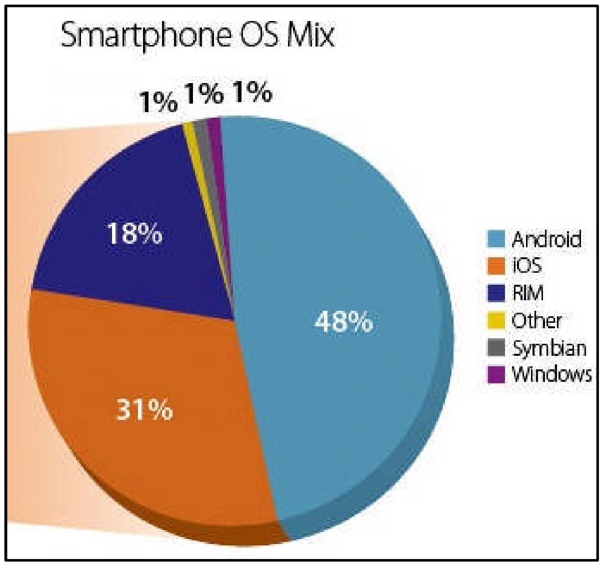


Figure 2.2. Smartphone OS in market. Source: MILLENNIAL MEDIA, *Mobilemix: The media device index*. Millennial Media, http://www. millennialmedia.com/wp-content/images/ mobilemix/MM-MobileMix-March2011.pdf, accessed April 2011, 2011.

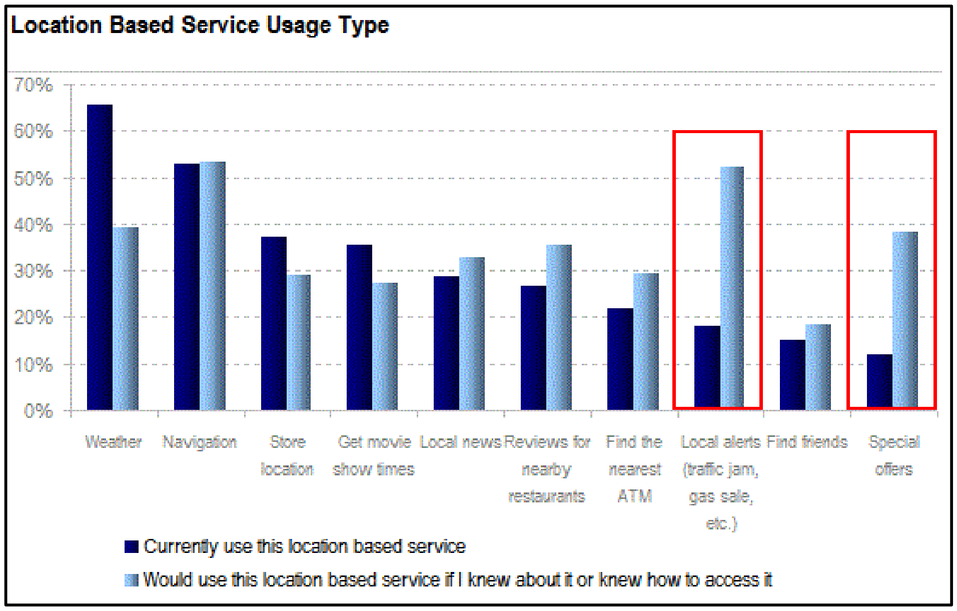


Figure 2.3. LBS usage statistics. Source: A. DEGARAVILLA, *Location based services: Why smartphone apps will pay off for advertisers, carriers, application providers.* Compete Pulse, http://blog.compete. com/2009/06/02/location-based-services-applications-carriers-advertisers/, accessed April 2011, 2009.

LBS receives the location coordinates from the ground based mobile station and sends it to the mobile phone user and the communication center that can be used for various location based services.

Location Based Services in next coming years will offer a myriad of new functionalities that are difficult to conceive today. It is anticipated that the next generation of mobile network infrastructure, or 4G, will have been pervasively rolled out in most developed countries allowing data transfer speeds over the air many times faster than those available today. More powerful mobile handsets will bring PC-standard processing power to the pocket that will allow devices to interact with their environment and their owner in brand new ways.

Location-aware computing refers to systems that can sense the current location of a user or device and change behavior based on this location. The best-known example is the GPS navigation device. Since a GPS device knows its current location, it can give directions to the GPS user for how to get to a new location, and it can update these directions continuously as the device moves. The reduction in price of location sensors, combined with the widespread availability of highly sophisticated portable devices (particularly smartphones), has resulted in the increased impact of location-aware systems in recent years. Most commonly, a location-aware system will determine its location through one of the following methods, ordered from least precise to most precise:

- *Mobile-phone triangulation:* determines its location by estimating its distance from multiple cell towers using the strength of each signal
- *Wi-Fi triangulation:* uses the same principle as mobile-phone triangulation but estimates the distance from Wi-Fi access points.
- *GPS:* determines a location based on signals from multiple satellites
- *Radio-Frequency Identification (RFID):* uses the signal from one or more RFIDs

Other location-aware devices have been based on TV signals, Bluetooth, infrared light, vision, ultra-wideband radio, and ultrasonic signals.

Once a location-aware device determines its location, it can then take action or update content based on that information. A range of applications—from "friend finders" to surveillance systems—can track the location of individuals with location-aware devices and report them. This does, however, raise important privacy issues. Smart-guide systems can give the user information as he/she passes through an environment, such as a museum or a college campus. Rich-media systems can offer sounds, sites, and data associated with specific locations to bring a historic event to life. Simulation games can change behavior based on the choices that users make to move through a space. Location information can be "mashed up" with systems such as Google Earth to create new applications relatively rapidly and inexpensively.

2.3 CHALLENGES IN MOBILE APPLICATION DEVELOPMENT

Making mass market mobile LBS apps that work effectively and universally (meaning independent of mobile operator or country or handset manufacturers) is challenging. There are there key challenges that one needs to take into account when it comes to fostering adoption of their LBS applications: Cost of access, handset manufacturers and operators and user experience.

More often than not, mobile application developers and start-ups in the field decide to go after a specific niche or segment -which may be platform-related say, developing exclusively for the Android, or geographic, limiting activity to North America, for example. Indeed, one of the key challenges still facing developers of mass market mobile applications is to overcome (often prohibitively high) charges for data usage and overcome the consumers' fear of mobile phone bill shock for using applications which need expensive data usage. While it is likely that flat rate or "all-you-can-eat" mobile data tariffs will soon be the norm in developed markets, the onus still remains on application developers to reduce data transfer in the case where the end user is being charged on a "per kb" transferred basis. Even where data roaming charges are not an issue, the cost to the consumer for accessing the application has to be set at the right level to encourage mass adoption.

When it comes to mobile handset manufacturers, it is important to bear in mind that different manufacturers have differing constraints associated with them -some offer more reliable GPS positioning than others, so if an application is being built where this is key, going for these specific manufacturers is essential.

Also various device models support different functionalities, such as touch screen, gravity censors, camera flash, etc. Hardware performances also vary between devices. Screen resolution is crucially important in application development. Ongoing trend is to have bigger screen resolutions for expanded multimedia support, data presentation, and browsing. However, device manufacturers tend to support multi-range of resolutions to address the needs of different user segments.

Some of the practical concerns related to location-based services are accuracy of location information, visualization, timeliness of the information, and transparency of location information. Privacy concerns must be a part of the design for location-based services. Users must be aware of when they can be uniquely identified, who has access to their location data, and how long this data may persist.

User experience is very essential for the success of the application. It justifies that the end-user is able to find the application useful, easy to use, interactive and fun to use. Different hardware show different functionality and thus a different experience. The challenge is to support as many devices as possible and to maintain a rich user experience. With some foresight and good planning, the best user experience can be created.

CHAPTER 3

DESIGN SPECIFICATION

3.1 OVERVIEW

GeoAware is an application built using Android framework for mobile platform. This application is inspired to build context-aware location-based services and follows a defined approach [7, 8]. The application is used to create a to-do list and tag each of the list items to a location. While on mobile, the application continuously scans for the locations in the list and once inside proximity of any one of them or more, it throws an alert to the user.

3.2 ANDROID PLATFORM

The Android platform is Google Inc.'s open and free software stack that includes an operating system, middleware and also key applications for use on mobile devices, including smartphones. Android is an Open Handset Alliance Project. 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. For example, an application can 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.

Figure 3.1 is inspired from the original architecture [9]. Android is built on the open Linux Kernel. Furthermore, it utilizes a custom virtual machine that was designed to optimize memory and hardware resources in a mobile environment. Android is 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.

Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik virtual machine (VM) executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint. The VM is register-based, and runs

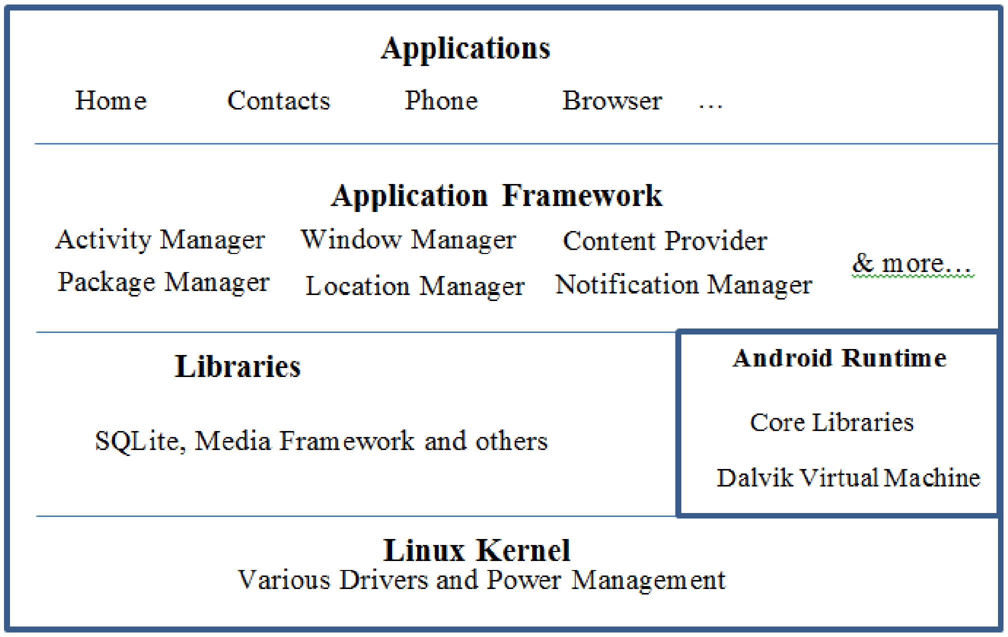


Figure 3.1. Android architecture.

classes compiled by a Java language compiler that have been transformed into the .dex format by the included "dx" tool.

The Dalvik VM relies on the Linux kernel for underlying functionality such as threading and low-level memory management [9].

3.3 Component Architecture

The five built-in components within the application namely activity, user interface, location manager, map display and database interact with each other to seamlessly produce a highly interactive system (see Figure 3.2).

3.3.1 Activity

An *Activity* represents the presentation layer of an Android application, e.g. a screen which the user sees. An Android application can have several activities and it can be switched between them during runtime of the application. Almost all activities interact with the user, so the *Activity* class takes care of creating a window in which the user interface (UI) can be placed with *setContentView (View)*. There are two methods almost all subclasses of *Activity* will implement:

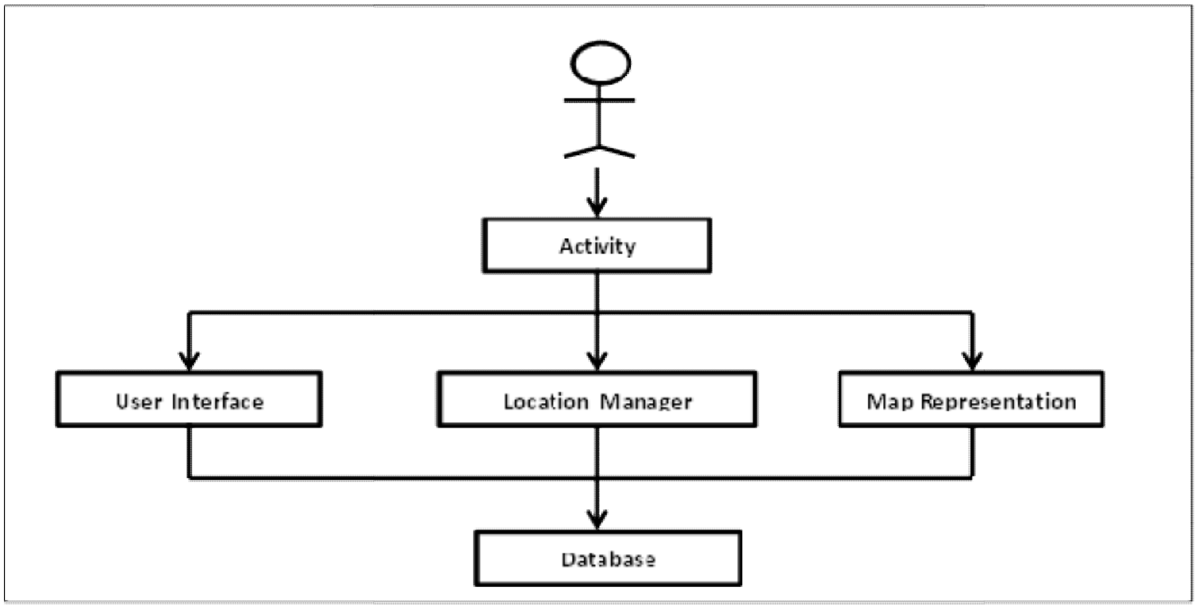


Figure 3.2. High-level component design.

1. *onCreate (Bundle)* is where the activity is initialized. It is here where the *setContentView (int)* method is called with a layout resource defining the UI, and using *findViewById (int)* to retrieve the widgets in the UI that is needed to interact with, programmatically.
2. *onPause ()* is for dealing with the user leaving the current activity. Most importantly, any changes made by the user should at this point be committed (usually to the *ContentProvider* holding the data).

To make use of *Context.startActivity (),* all activity classes have a corresponding *<activity>* declaration in their package's *AndroidManifest.xml* [10].

The *Activity* class is an important part of an application's overall lifecycle, and the way activities are launched and put together is a fundamental part of the platform's application model.

Activities in the system are managed as an *activity stack*. When a new activity is started, it is placed on the top of the stack and becomes the running activity -- the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.

Intents are communication channel between activities in Android. An `Intent` object is a bundle of information. It contains information of interest to the component that receives the intent (such as the action to be taken and the data to act on) plus information of interest to the Android system (such as the category of component that should handle the intent and instructions on how to launch a target activity). Thus, Intent is an abstract description of an operation to be performed. It can be used with *startActivity ()* to launch an Activity, *BroadcastIntent* [10] to send it to any interested *BroadcastReceiver* components, and *startService (Intent)* or *bindService(Intent, ServiceConnection, int)* to communicate with a background service.

3.3.2 User Interface

In the Android Application, the user interface is build using *View* and *ViewGroup* objects. There are many types of views and view groups, each of which is a descendant of the `View` class. View objects are the basic units of user interface expression on the Android platform.

The *View* class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons. The *ViewGroup* class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.

A *View* object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A *View* object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a *View* is also a point of interaction for the user and the receiver of the interaction events. The Android UI toolkit offers several layout managers that are easy to use to implement a user interface.

This application employs *Linear Layout*, *Frame Layout* and *Relative Layout*. *Linear Layout* wraps the *TextView, EditText, ListView* and *Buttons*; *MapView* exploits *Frame Layout* and *Relative Layout* is used in Map View Activity for displaying the various locations in the map.

The Android environment adds yet another Graphical User Interface (GUI) toolkit to the Java ecosphere, joining AWT, Swing, SWT, and J2ME (leaving aside the web UI toolkits). Android GUI is single-threaded, event-driven and built on a library of nested components. The Android UI framework is organized around the common Model-View-Controller pattern.

The Model: The model represents data or data container. It can be seen as a database of pictures on the device. Say, any user wants to hear an audio file, he clicks play button and it triggers an event in the app, now the app will get data from data store or database and as per input and creates data to be sent back to the user. This data can be referred as Model.
The View: The View is the portion of the application responsible for rendering the display, sending audio to speakers, generating tactile feedback, and so on.
Now as per above example, the view in a hypothetical audio player might contain a component that shows the album cover for the currently playing tune. User will always interact with this layer. User actions on this layer will trigger events that will go to the application functions.
The Controller: The Controller is the portion of an application that responds to external actions: a keystroke, a screen tap, an incoming call, etc. It is implemented as an event queue. On User’s action, the control is passed over to controller and this will take care of all logic that needs to be done and prepare Model that need to be sent to view layer.
UI component objects such as buttons and text boxes actually implement both View and Controller methods.

3.3.3 Location Manager

The most enticing of the Android features are the Location Based Services that give the application geographical context through Location Providers (GPS etc.). Most Android devices allow determining the current geo-location. This can be done via a GPS device, via cell tower triangulation or via Wi-Fi networks for which the geo-location is known.

Android provides the package "*android.location*" which provides the API to determine the current geo position. The class "*LocationManager*" provides access to the location service. To call the location manager, the context is set to *getSystemService ()* for location services. The Android device might have several providers available and a "*Criteria*" object can be used to select the “*best*” among them. Then a "*LocationListener*" is registered with the "*LocationManager*" which receives periodic updates about the geo position. The class "*LocationProvider*" is the superclass of the different location providers which deliver the information about the current location. It is important to register with the location manager to receive the location update which is done using the *onResume ()* method and can be unregistered from location notifications in the *onPause ()* method [9]. Tracking the location can be expensive on the battery and CPU of the device, so this is a good practice to stop doing the work while the application is paused would not require any updates.

The *LocationListener* which is implemented allows getting certain callbacks. Those callbacks include location, provider and status changes. Besides all this, appropriate permissions need to be added to the *AndroidManifest.xml* file to notify the Android system [10].

3.3.4 Map Representation

Google provides in the package "*com.google.android.maps*" [11] a library for using Google Maps in Android. Google Maps is not a part of the standard Open Source Platform Android and requires an additional key to use them. The class "*MapActivity*" extends the class "*Activity*" and makes the usage of maps simple. "*MapActivity*" includes per default a "*MapView*" which displays a map with data obtained from the Google Maps service. It also takes care of the network communication required for the map. The class "*MapController*" can be used to interact with the "*MapView*", e.g. by moving it.

When the *MapView* has focus, it will capture key presses and touch gestures to pan and zoom the map automatically; including handling network requests for additional maps tiles. It also provides all of the UI elements necessary for users to control the map. To use Google Maps in the application, a Maps API key had to be obtained to register with the service and Android system had to be notified that the application wishes to implement the add-on Google APIs which is external to the base APIs. This was done by using the *<uses-library>* element in the Android manifest file, informing Android that the application used classes from the *com.google.android.maps* package. The *com.google.android.maps* [11] package used in this application offered built-in downloading, rendering, and caching of Maps tiles, as well as a variety of display options and controls. The key class in the Maps package is *com.google.android.maps.MapView* [11], a subclass of *ViewGroup*.

A "*Geopoint*" is a position described via latitude and longitude and the class "*Overlay*" can be used to drawn on the map, for example position markers. This application also uses *Map Overlays* to display to the user the various list items drawn on the map. This is done by creating map markers and lay-overs. This has been possible by using *ItemizedOverlay* class to manage all the individual items placed on the map.

3.3.5 Databases

The storage system in Android is mainly divided into three types: Shared Preferences, Files and SQLite Databases [10]. GeoAware makes use of SQLite Databases to save the locations and their geo-points. The application as such is stored in Android system uses internal storage of the device. By default, the application once installed is stored in the internal storage of the Android system. This is private to the application and other applications cannot access it. When the user uninstalls the application, these files get removed.

By default all files, databases and preferences is restricted to the application that created them. If the data has to be shared with other applications then the Content Providers is used. Content Providers is not a storage mechanism but provide well defined interface for sharing private data.

Shared preferences store data in terms of key-value pairs and can be accessed by application components working in the same context.
Data can also be stored on local files directly using standard Java I/O classes and methods. These methods only support files in the current application folder; specifying path separators will cause an exception to be thrown.

Android also provides full support to SQLite databases. All databases that are created in the application are accessible by name to any class in the application but none outside. This is implemented by creating a sub-class to *SQLiteOpenHelper* and overriding the *onCreate()* method to execute the SQLite command to create the tables in the databases. The methods *getWritableDatabase()* [11] and *getReadableDatabase()* [11] are called for write to and read from the database which return *SQLiteDatabase* [11] object.

The Android SDK includes a sqlite3 database tool which is required to browse the table contents, run SQL commands and perform other SQL functions. Objects that are used to insert new rows in the database are called *contentValues* [11]. Each *contentValue* represent a new row. *Cursor is an* object which is used for extracting over query results. It acts as a pointer to the result set from a database query. *Cursor* provides several functions navigate through your query results like *moveToFirst*, *moveToNext*, *getCount*, *getColumnName* etc. All Android databases are stored in the *data/data/<package_name>/databases* folder on the device.

In this application, relational database is used to create three tables for storing the various categories of items, the list of items in each category and the tagged locations.

CHAPTER 4

IMPLEMENTATION

4.1 OVERVIEW

GeoAware is a to-list application with location tagging. The user can create categories of to-list items and tag each of the categories to a place of interest. While on the move, if the user enters a proximity region, the application throws an alert displaying the location. The application stores all the categories, their respective list of items and the tagged locations in the SQLite database supported by Android. Figure 4.1 illustrates the various java classes implemented in this application.

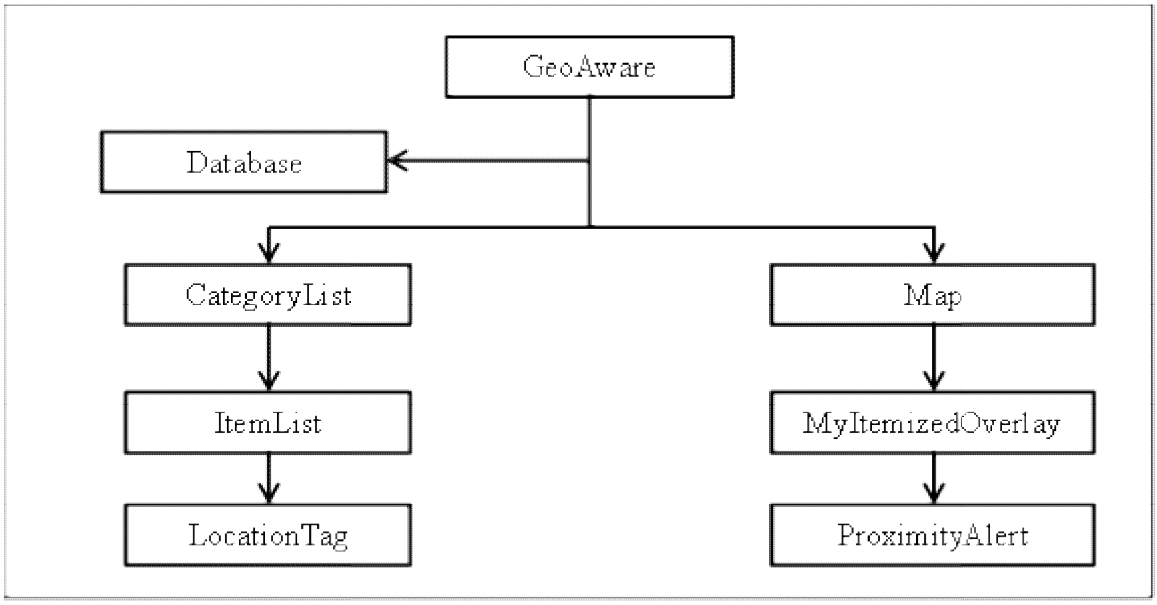


Figure 4.1. The workflow diagram showing the java classes.

These class files are the soul of the entire application. Each class file has a specific purpose in creating/calling the Activities of the application. Besides these, there is *AndroidManifest.xml* [10] which is present in the application's root directory and is mandatory for every application. The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code. It names the Java package for the application. It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities (for example, which `Intent` messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched. It determines which processes will host application components. It declares which permissions the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.

4.2 Application Modules

The Android application, GeoAware has five main modules: List, Map, Category, Items and Geotag. These modules are developed using many developer-defined Activities and .java classes. Each of the modules in this application is created as Activity and has specific function to perform. The Figure 4.2 is created to illustrate the modules of this application and will be explained in detail in this chapter.

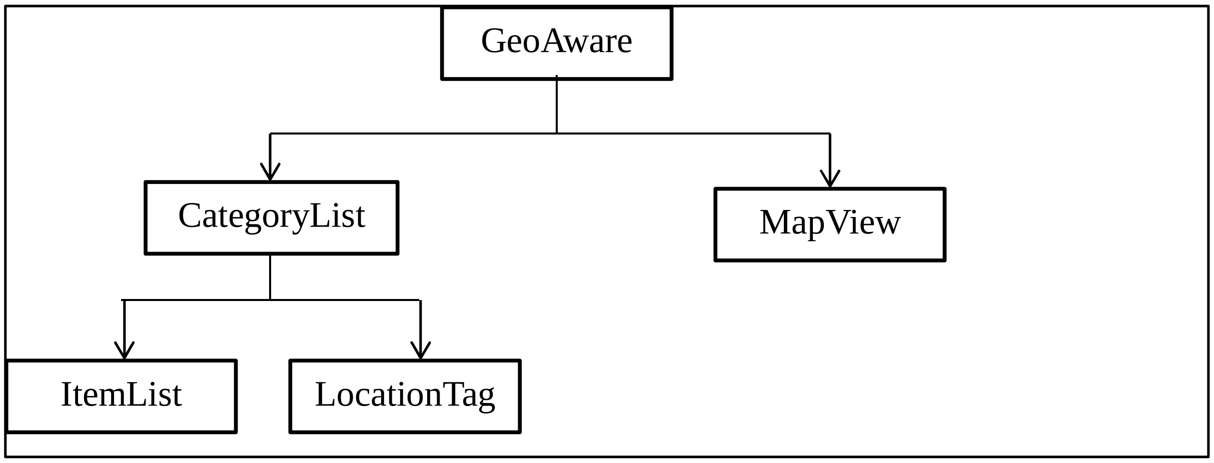


Figure 4.2. Implementation modules.

GeoAware is the first Activity to load when the application starts. It extends the Tab Activity to load the two tabs: Category List and Map (see Figure 4.3).

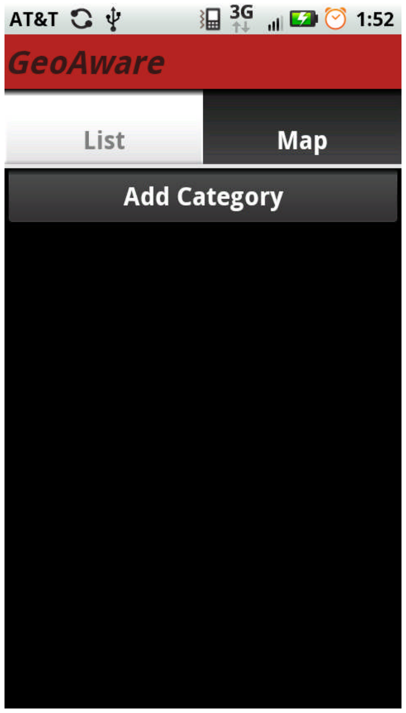


Figure 4.3. GeoAware activity.

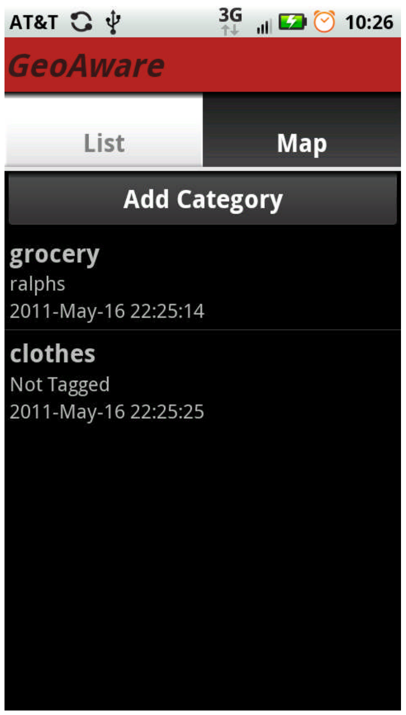
4.2.1 Category List

CategoryList is the host tab which gets loaded when *GeoAware* is started. It has a button named *AddCategory* which upon clicked lets the user to type a category name. Once the category name is inserted, the view loads back to the *CategoryList* and the user can view the category created.

The *CategoryList* displays all the categories saved by the user in the order of time it is created. It displays the category name, location name which it is tagged to (else displaying Not Tagged) and the date and time which it is created (see Figure 4.4). All the categories can be viewed and deleted from this Activity only.

CategoryList implements two listeners:

1. *setOnClickListener ()*: This listener is activated when the user clicks on the *AddCategory*. It opens up an alert box to create a new category. This also opens a new entry for the new category in the database using the method *addCategory ().*
2. *setOnItemClickListener ():* This listener is activated when the user clicks on any of the category name from the list and then calls the method *onItemClick ()*. This method opens an alert box to show options: *View*, *Cancel* and *Delete*.



grocery
ralphs
2011-May-16 22:25:14
clothes
Not Tagged
2011-May-16 22:25:25

Figure 4.4. CategoryList view.

To add the list items to the category, the category name can be clicked which loads an alert box giving options to *View*, *Cancel* or *Delete* (see Figure 4.5). If the user selects *View*, the application loads the *Items* Activity. If the user selects *Cancel*, it loads back to the initial Activity and if the user dislikes the category, he/she can simply go ahead and *Delete* the category.

There are also two default methods that are always implemented: *getCategoryList ()* and *showCategoryList ()*. *getCategoryList ()* retrieves already existing categories and calls the *showCategoryList ()* to display it during the launch of the Activity. *getCategoryList ()* also manages a Cursor to handle the records of the stored database.

A unique identifier (ID) is assigned to each category. When a new category is selected, an ID assigned to the trip is passed on to the Tab Activity and this ID holds active throughout the active state of the trip. The methods *getCategoryList ()* and *showCategoryList()* are used to retrieve the category names, locations tagged and the date and time from the database storage and display it in the list view.

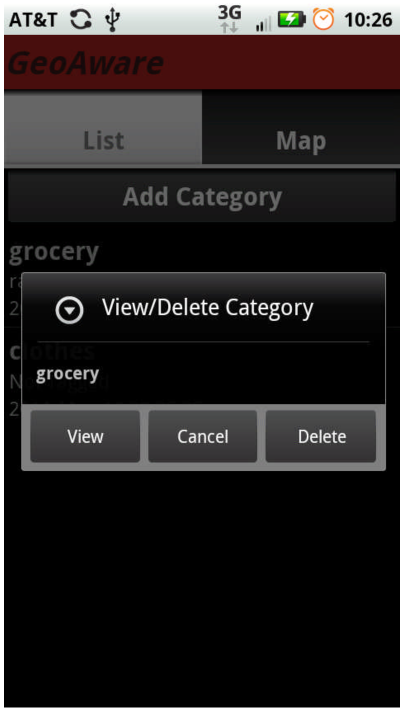


Figure 4.5. On clicking a category name.

4.2.2 Item List

ItemList is the activity which allows the users to add items to the category and delete them if they don't want it. There is also an option where the category can be tagged to the desired location. This can be done by clicking on *Tag* button (see Figure 4.6).

ItemList gets the ID from the intent which captures the category being clicked and loads the set of items in the category using the *getItemList ()* and *showItemList ()*.

ItemList implements a listener: *setOnClickListener ()*: This listener is activated when the user clicks on the *Add* button and *Tag* button. Upon clicking the Add button, the onClickListener calls the onClick () method. This method opens up an alert dialog which asks for the user input to add the item for that category (see Figure 4.7). Once the user inserts the item and presses *Save*, it gets stored into the database under that category ID.

The *onClickListener ()* is also set on the *Tag* button and when the user clicks on the button, it calls *the onClick ()* method which directs to a new Activity called *LocationTag*.

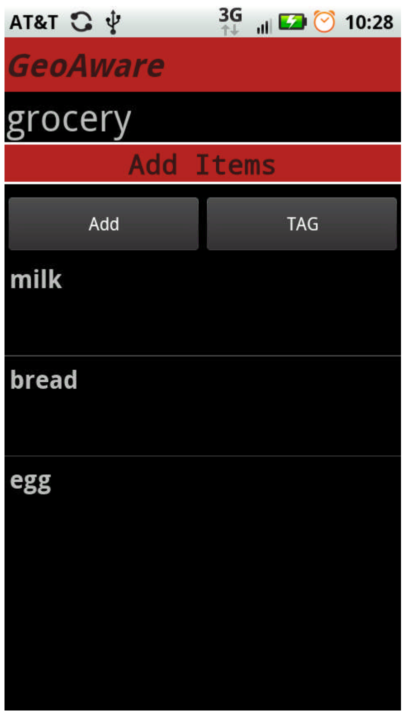


Figure 4.6. ItemList view.

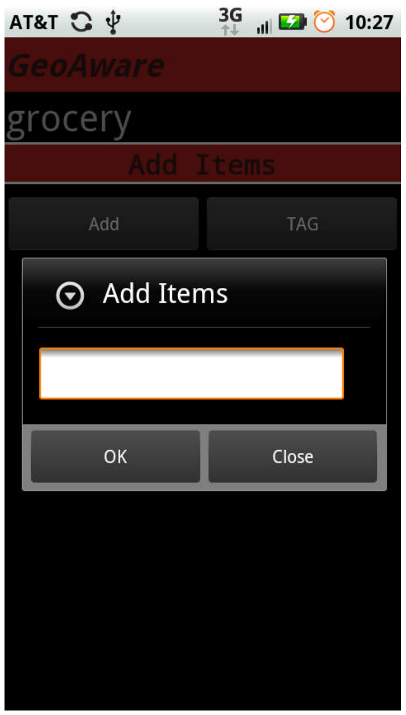


Figure 4.7. Adding items to the category.

4.2.3 Location Tag

LocationTag is the most interesting activity of the application. It is here where the user can type in the location he/she desires to tag the category to and the application saves it to the category name. The application also shows the recent tagged locations to enable the user to select if desired. This Activity calls *onTextChanged ()* method while the user types in the edit box. This method queries the string as the user begins to type and searches for any pre-defined locations stored in the database. If found, it prompts the user and the user can click on it to tag the location (see Figure 4.8). Once selected, the category gets tagged with the location along with the geo-points and it will be displayed on the map.

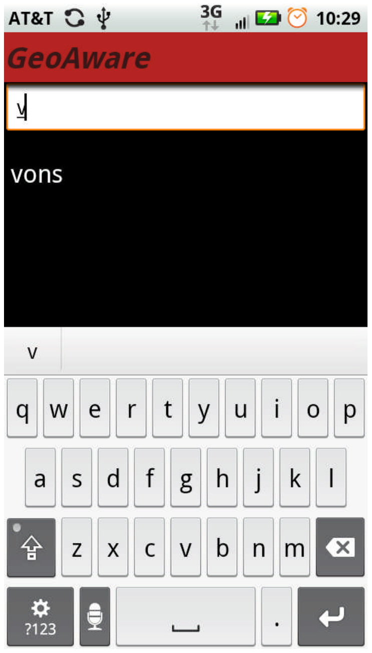


Figure 4.8. Location tagging.

4.2.4 Map View

Map View is opened through the click of another Tab. Map View uses the location services supported by the device through the classes in *android.location* package [12]. The Maps package *com.google.android.maps.MapView* [12], a subclass of *ViewGroup* is used to display a map with data obtained from the Google Maps service. Figure 4.9 shows the map view.

To use the Google maps data, Maps service had to be registered and MD5 fingerprint of the certificate which is required to sign the application had to be provided to obtain a

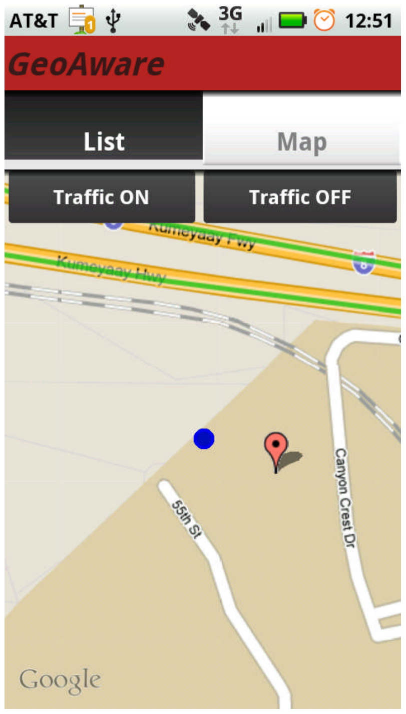


Figure 4.9. Map view.

Maps API key. This key had to be inserted in the .xml file of the Map View for the value of *android:apiKey* to enable the Google maps [12].

To use the Maps external library, <*uses-library*> element in the Android Manifest file is defined to *"com.google.android.maps."* This thus enables the build tools to link the application against the Maps external library. This is to also ensure that the Android system checks the required library is available to install the application on the device.

To use the Maps in the application, another step is required. *MapActivity* class had to be extended and a layout had to be created that included a *MapView* element. *MapActivity* is a special sub class of Activity which has important map capabilities.

Map View properties are need to be set. To enable zoom controls in the map, *setBuiltinZoomControls ()* is set to true.

MapView class provides a wrapper around the Google Maps API to manipulate the Google Maps data through class methods.

To create map markers and lay-overs, *ItemizedOverlay* class had to be implemented. All the locations that need to be tagged are added to *OverlayItem ArrayList*. The *populate ()*

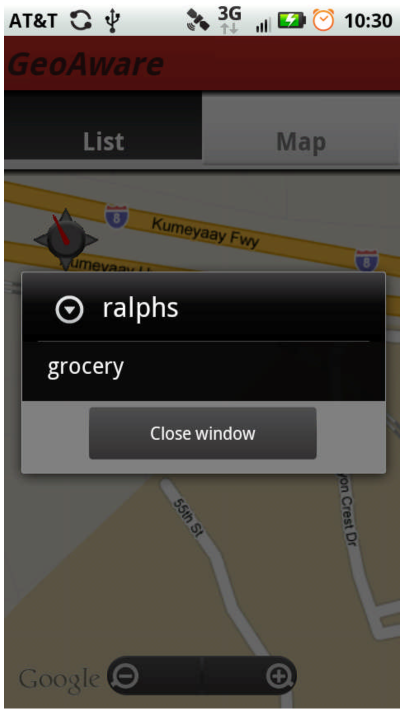
method is called each time a new *OverlayItem* is added to the list which reads each of the objects in the list and prepare them to be drawn.

To handle the touch events, *onTap ()* method is called. This method obtains the latitude and longitude of the point. This method uses the member *android.content.Context* to create a new *AlertDialog.Builder* and uses the tapped *OverlayItem'*s title and snippet for the dialog's title and message text.

GeoAware shows the Map View with the current position of the user with a moving dot along with all the category-tagged locations on the map. Once the user starts to move, the dot also keeps on updating the current location, when the user is in the proximity of a tagged location, it throws up a proximity alert with a sound to alert the user. The user can also enable the *Traffic* buttons provided to view the traffic while looking at the map to estimate the travel to a desired place.

This activity has a *Location Listener* which constantly captures the user's current position and compares it with the category-tagged locations. The *Location Listener* scans the radius of the current location which is set by some pre-defined conditions. If the user enters a proximity region of a tagged location, the listener throws up a message using intent. This message is captured by the *BroadcastReceiver* class which displays an alert to the user saying "You are now near to xxx", xxx being the location name. This message pops up with a sound to alert the user.

If the map markers are clicked, an alert box pops up showing the location name and the category which it is tagged to (see Figure 4.10).



ralphs

grocery

Figure 4.10. On clicking on the tagged location.

CHAPTER 5

USER MANUAL

5.1 OVERVIEW

This chapter educates the users and developers of the application, by providing information on getting started and installation of the application on their mobile devices.

5.2 SETTING UP THE DEVELOPMENT ENVIRONMENT

A typical developer needs to set up an environment for android development. The installation steps given below assume that the developer is aware of setting up the development environment for the android development which is given in android developer site. This application can be developed on any platform which runs Java/android. The installation steps for the development:

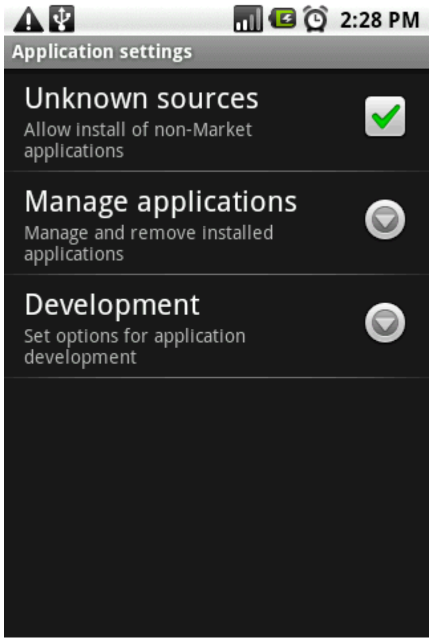
- Install Java JDK and JRE
- Install Eclipse IDE
- Install Android SDK
- Install ADT plug-in in Eclipse for Android Development

Once the setup of the machine is done, the next step is to download the source code and make changes on to it:

- Copy the zip files of the source code to your machine.
- Unzip the file to your workspace.
- Import the project into the Eclipse

5.3 INSTALLING APPLICATION ON DEVICE

The first step is to get the .apk files from the Unknown Source or the other way is to download from the Android Market. Installation from Android Market is direct and simple. If it has to be downloaded from other source (except Android Market) then the phone settings should be changed. Figure 5.1 shows the installation from unknown source.


2:28 PM

Application settings

Unknown sources
Allow install of non-Market applications ✓

Manage applications
Manage and remove installed applications

Development
Set options for application development

Figure 5.1. Installation from unknown source.

For the manual installation, the following instructions should be followed:

- Check if you are in HOME screen
- Click on Menu button
- Select Settings
- Click on applications
- Check the option for Unknown Sources

Now the application will be installed on to the device.

5.4 Using GeoAware on an Android Device

GeoAware is developed to give a fun, interactive experience to the end-user. This session will describe how an end-user can use the application on the Android device.

Adding categories of to-do items: A first time user needs to select the "*Add Category*" button to kick off the application and enter the first record in the database (see Figure 5.2). Initially, the database in empty and shows no categories.

Once the button is clicked, an alert box pops up prompting the user to type in a category name. The user can also wish to cancel if desired. Once the user saves the category, it gets displayed on the category list with the date and time it is created. The user can also see that the category is shown as "*Not Tagged*" (see Figure 5.3).

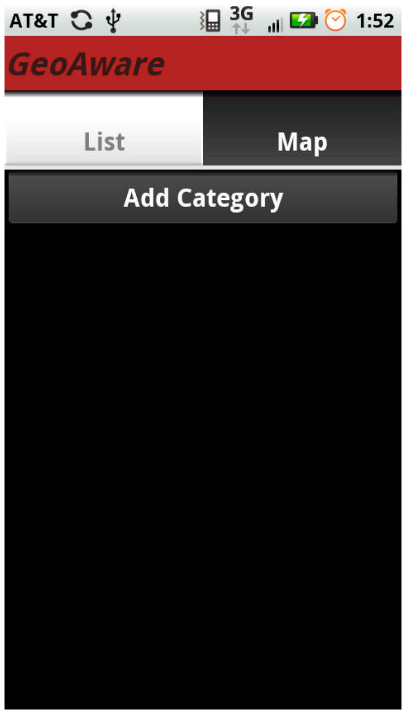
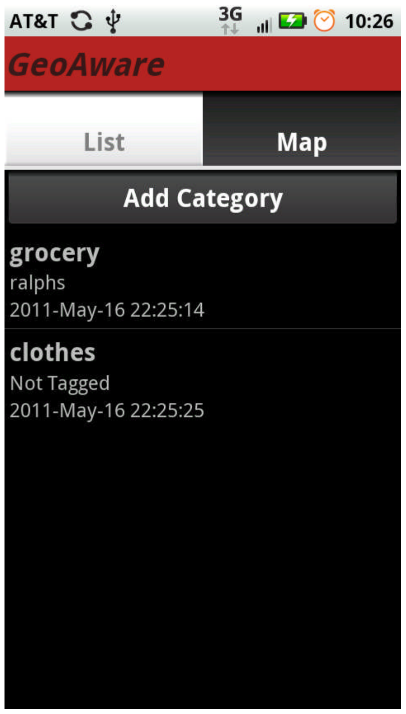


Figure 5.2. Step 1 first category to be added.



grocery
ralphs
2011-May-16 22:25:14
clothes
Not Tagged
2011-May-16 22:25:25

Figure 5.3. Step 2 category list.

Adding items to the category: To add items to the category, the user must click on the category name. The application then prompts the user with an alert box to *View*, *Cancel* or *Delete* (see Figure 5.4). Once the user chooses to *View*, the application loads to a different activity for the user to add items to the list (see Figure 5.5).

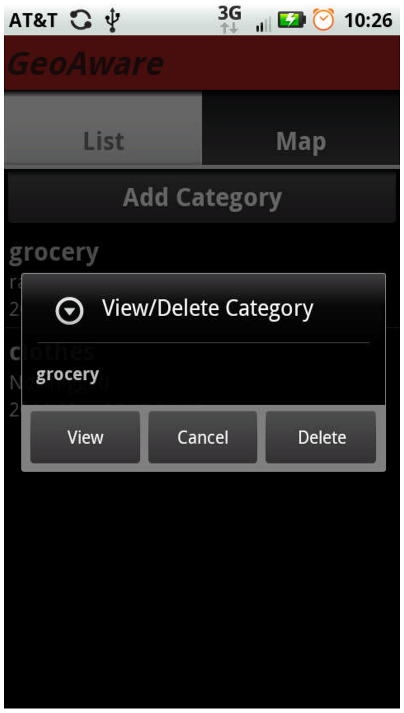


Figure 5.4. Step 3 to view and add or cancel or delete the category.

When the user clicks on the *Add* button, a new alert pops up with a text box. The user can type the item name in the text box and choose *Ok* to save it to the list or *Close* to cancel (see Figure 5.6).

Tagging the Category: Once the user finishes adding all the items, he/she can tag the category of items to a desired location. This can be done so by clicking the Tag button (see Figure 5.7). The user can type in the desired location or choose what the application prompts from the previous tagged locations. When the user selects a location, the application brings back to the category list and the user can see that the location is placed under the category.

Viewing on the Map: If the user selects the Map tab, he/she can visually note that all the places tagged are laid out on a map. The tagged places are illustrated with a small balloon

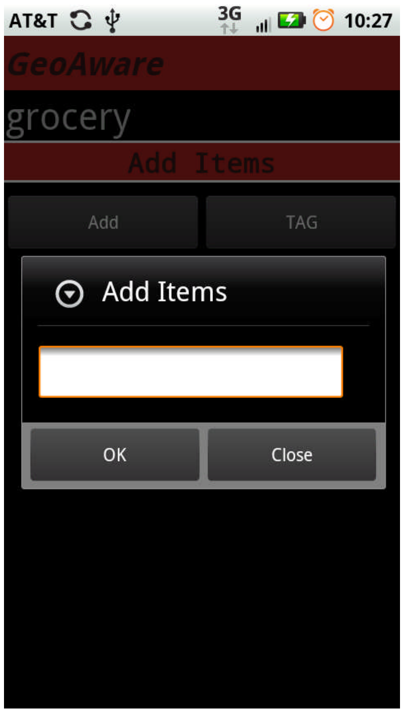


Figure 5.5. Step 4 adding items.

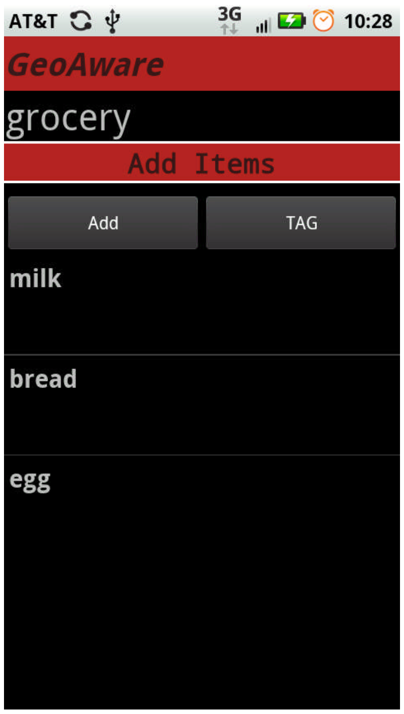


Figure 5.6. Step 5 the list of items in a category.

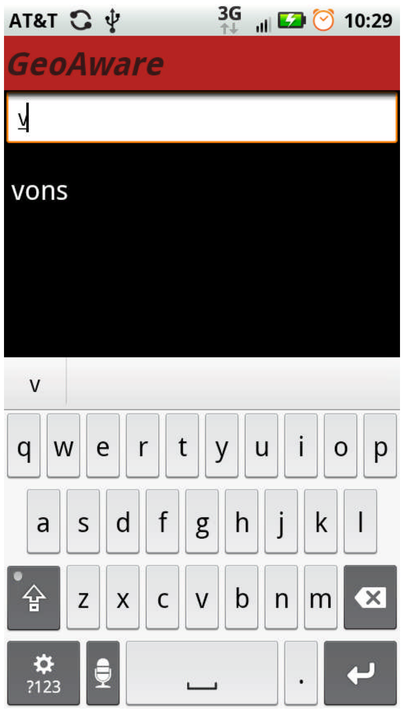


Figure 5.7. Step 6 tagging to a location.

and the current location is shown as a blue dot (see Figure 5.8). When the user selects the balloon, a small alert box pops up showing the name of the place and the tagged category (see Figure 5.9).

When the user is on the move, the blue dot constantly updates the current position of the user and if the user is in a proximity regionof a tagged location, a proximity alert message pops up with a sound. The message shows the category and the location name and this alarms the user to complete the tagged task.

The user also has an option to check the traffic by clicking on Traffic ON button and when not required clicking on Traffic OFF button. This is designed to help the users to estimate the current traffic situation and then commute to the tagged location.

Deleting the category: If the user wishes to delete the task completed, he/she can go to the Category activity and select the category to be deleted, choose the Delete option. This deletes the category from the list and also from the database.

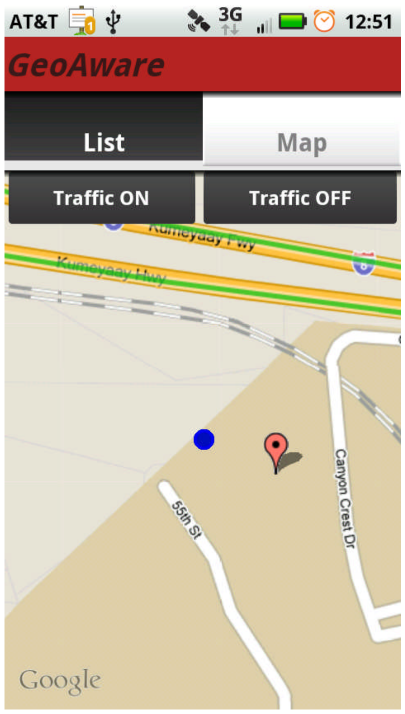
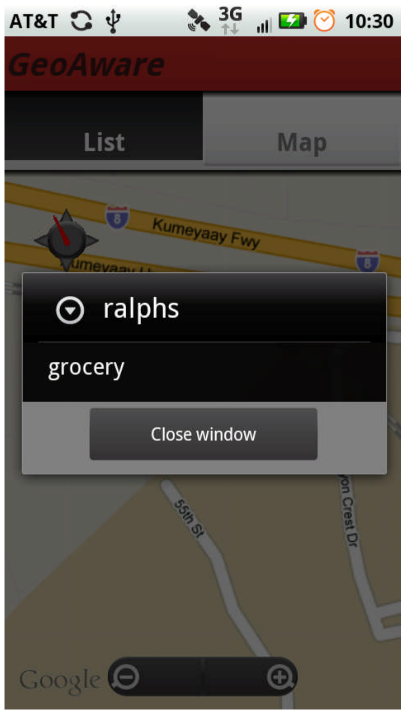


Figure 5.8. Step 7 viewing the places on the map.



ralphs

grocery

Figure 5.9. Step 8 on clicking the tagged location.

CHAPTER 6

DISCUSSION

6.1 FUTURE ENHANCEMENTS

This thesis project was developed as a beginer application for location-aware to-do list. There is always more to add-on and enhance this application to provide a better end-user experience. It can be integrated with Google's "Places" or created a new one of a similar kind to constantly search for the places typed in by the user. The application can also suggest the user for other locations.

The application can also be extended to provide an optimal path for running all errands which can help the user to complete all the tasks suggested b the application.

To provide more options to the proximity alert sounds, user-chosen sounds or ringtones can be integrated. This would alert the user about the proximity of the location.

There can navigation alerts also embedded for the user to be prompted to get to the location. This will help the user to get live navigation support.

The categories can be color-coded to set preference and timeline which they have to be completed. This would make an efficient to-do application. Items added to each category and category name should be made to edit and modify.

Moreover, the list can be made to share among friends or family. This can help in getting things done quicker.

6.2 CONCLUSION

The motivation of this thesis project is to build an Android based application for providing a simple location-aware to-do list. By using Android SDK and Eclipse IDE as the development environment the application is built keeping in mind about the design standards and maintainability of the code.

GeoAware is developed to help users do their day-to-day errands. The application helps to create to-do list of action items and tag them to a location so that when the user is on the move and enters a proximity region of the tagged location, the application simply reminds the user the set of action items that has to be completed.

This application being very user-friendly and interactive would be very helpful for the users to do their simple routines, keep a record of their action items and tag those to the locations where it has to be done.

GeoAware is tested on many Android devices to check the functionality and UI of the application. Motorola Bravo, Motorola Flip-Out, Samsung Captivate is few of the devices which were used to run the application.

REFERENCES

[1] N. BILTON, *Smartphone market expected to soar in 2011*. New York Times, http://bits.blogs.nytimes.com/2011/03/29/smartphone-market-expected-to-soar-in-2011/, accessed April 2011, 2011.

[2] F. CEJAS. *Obtaining user location*. Anroid 10, http://www.android10.org/index.php/articleslocationmaps/209-obtaining-user-location, accessed April 2011, 2010.

[3] F. KELLETT, *The best travel apps*. Telegraph, http://www.telegraph.co.uk/travel/travelaccessories/8330846/The-best-travel-apps.html, accessed March 2011, 2011.

[4] MILLENNIAL MEDIA, *Mobilemix: The media device index*. Millennial Media, http://www.millennialmedia.com/wp-content/images/mobilemix/MM-MobileMix-March2011.pdf, accessed April 2011, 2011.

[5] M. BRAIN AND T. HARRIS, *How GPS receivers work.* HowStuffWorks, http://electronics.howstuffworks.com/gadgets/travel/gps.htm, accessed April 2011, 2010.

[6] A. DEGARAVILLA, *Location based services: Why smartphone apps will pay off for advertisers, carriers, application providers.* Compete Pulse, http://blog.compete.com/2009/06/02/location-based-services-applications-carriers-advertisers/, accessed April 2011, 2009.

[7] T. ZHU, C. WANG, G. JIA, AND J. HUANG, eds., *Proceedings of the 2010 International Conference on Electronics and Information Engineering*, Kyoto, 2010, IEEE.

[8] K. A. LI, T. SOHN, AND W. G. GRISWOLD, *Evaluating Location-Based Reminders*. UCSD Computer Science and Engineering, San Diego, 2005.

[9] ANDROID DEVELOPERS, *The android architecture*. Android Developers, http://developer.android.com/, accessed March 2011, 2010.

[10] ANDROID DEVELOPERS, *The androidmanifest.xml file.* Android Developers, http://developer.android.com/guide/topics/manifest/manifest-intro.html, accessed June 2010, 2010.

[11] ANDROID DEVELOPERS, *What is Android.* Android Developers, http://developer.android.com/guide/basics/what-is-android.html, accessed May 2011, 2010.

[12] ANDROID DEVELOPERS, *Android location package*. Android Developers, http://developer.android.com/reference/android/location/package-summary.html, accessed August 2010, 2010.