VAASAN AMMATTIKORKEAKOULU
VASA YRKESHÖGSKOLA
UNIVERSITY OF APPLIED SCIENCES

Lawal Olufowobi

SMS BASED ANDROID ASSET TRACKING SYSTEM

Technology and Communication

2011

VAASAN AMMATTIKORKEAKOULU
Information Technology Degree Programme

TIIVISTELMÄ

Tekijä	Lawal Olufowobi
Opinnäytetyön nimi	SMS- ja Android-pohjainen seurantajärjestelmä
Vuosi	2011
Kieli	Englanti
Sivumäärä	50 liitettä
Ohjaaja	Johan Dams

Arvokkaiden esineiden kuten esimerkiksi autojen, moottoripyörien ja lemmikkien turvaaminen on yhä useammin yksityishenkilöiden ja pienien yritysten huolena turvallisuuspalveluyrityksien sijaan. Tämä luo tarpeen helppokäyttöiselle ja halvalle seurantaratkaisulle suuren yleisön käyttöön.

Tässä projektissa esitetään ratkaisu ongelmaan hyödyntäen kännykkää WRD Systems Ltd:n tarjoaman SMS-pohjaisen GPS jäljittimen seuraamiseen. SMS-pohjainen ratkaisu soveltuu erityisesti sijainteihin jossa GPRS-verkko ei välttämättä ole saatavilla.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Information Technology Degree Programme

ABSTRACT

Author	Lawal Olufowobi
Title	SMS Based Android Asset Tracking System
Year	2011
Language	English
Pages	50
Name of Supervisor	Johan Dams

As security increasingly becomes not just the concern of security institutions but also that of small companies and individuals, there is a need to offer a solution for everyday common problems of monitoring and tracking valuable assets such as cars, pets, motor-bikes and other valuables. This solution has to be cheap, convenient and inexpensive to be adopted by the general mass of people.

This project provides such a solution with the use of a mobile phone for monitoring an SMS-based GPS tracker provided by WRD Systems Ltd especially in locations where GPRS may not be available.

Keywords: Android, GPS, SMS, Tracking, Geographic Location, Maps.

ACKNOWLEDGEMENT

To my mother, she taught me life. To my father who made me know, I don't have the luxury of failing! To my grandfather, the only sincere Nigerian politician I know, who died during my study year and whose funeral I couldn't attend.

To Johan Dams, my supervisor, a mentor and teacher. To Dr Yang Liu, an instructor of admirable intelligence. And to all teachers in VAMK and elsewhere that have impacted my life.

ABBREVIATIONS

UI	User Interface
SDK	Software Development Kit
GPS	Global Positioning System.
SMS	Short Message Service
NMEA	National Marine Electronics Association
OS	Operating System
UTC	Coordinated Universal Time
SIM	Subscriber Identity Module
API	Application Programming Interface
GPX	GPS Exchange Format
KML	Keyhole Markup Language
IDE	Integrated Development Environment
SQL	Structured Query Language
AES	Advanced Encryption Standard

CONTENTS

1 INTRODUCTION

With global mobile phone subscribers estimated to 4.5 billion by 2012 [1], the mobile phone is by far the most adopted consumer electronic in the world. As their processing power increases, the ability to leverage their mobility and computing power to solve daily problems increases. From playing games, bar-code scanning, photography, to social networking, mobile phones have become integral part of our modern existence. Hence, applications that have been traditionally confined to desktop computing are steadily being adapted for mobile phones. One of these applications is that of monitoring and tracking. While desktop computing still offers more processing power, the mobile phone has the advantage of constant reachability and mobility that desktop computing lacks with comparable rich user experience.

1.1 Introduction of WRD Systems Ltd

WRD Systems Ltd is a technology company that provides, among other things, an active asset tracking management solution, called IAM. It can locate the position, speed and direction of an asset anywhere in the world with an accuracy of up-to 15 feet. Their product is currently in use in countries such as Algeria and the USA [2].

1.2 Existing Solutions and the merit for this project

The present solution provided by WRD Systems involves a tracking unit, a SIM-based gateway and a software that runs on a Laptop or Desktop PC. The gateway hardware is always connected to the laptop or PC, so when a location update is requested, the tracker sends the location to the gateway via SMS and the desktop application then reads the data off the gateway [2]. Figure 1 depicts the connection between the hardware in the existing system.

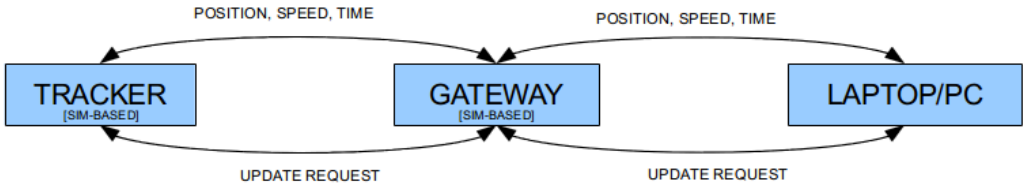

Figure 1: Existing System

This project eliminates the need for the gateway hardware, thus cutting cost in two folds. First, we have the fact that the cost of the gateway hardware is totally eradicated. Secondly, the maintenance cost of this SIM-based hardware is also eliminated as this project allows users to leverage an existing SIM-based hardware, the mobile phone. This project therefore cuts down on initial price, maintenance cost and increases customer satisfaction by making the application more reachable for users. Figure 2 illustrates the proposed system.

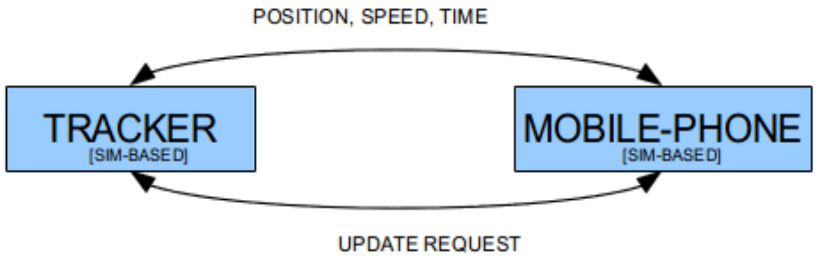

Figure 2: Proposed System

Also worth noting is that this solution is server-less and does not require a monthly fee. While other tracking solution vendor exist such as VisiRun [3], TrackPeers [4] and BizSpeed [5] and others, none of them offers this option. This is particularly of major concern to companies and individuals who do not want their location information to be held by a third-party. At the time of writing, we are unaware of any Android SMS-based Asset Tracking application.

1.3 Scope and Limitation

This project supports the Android OS platform only and makes communication with the tracker through SMS messages only. The architecture, security and

accuracy of the tracking unit itself are beyond the scope of this project.

2 MOBILE-PHONE APPLICATION DEVELOPMENT

Mobile phone application development comes with its own unique set of challenges. While advancements have been made in processing power, touch interface and internet connectivity; battery longevity is still a major drawback. Therefore, application developers must find a way to conserve power as much as possible when using battery-draining resources such as the GPS receiver. Responsiveness is also a paramount factor in a mobile phone application, a 100 to 200 milliseconds delay is noticeable by a user [6], and therefore the application must always find a way of engaging the user while doing resource intensive work. Also, due to the small screen size of mobile phones, mobile phone developers must find creative ways to provide easy navigations from one functionality to another.

Mobile phone development also lacks cross-platform tools like those that can be found in desktop software development in which a developer can write the source code once and run it on different operating system. Due to lack of such tool, time and other resource constraints, one of the mobile platforms has to be chosen for this project.

2.1 Why Android?

Android is a software stack for mobile devices that includes an operating system, middle-ware and key applications. Android includes the application framework, the Dalvik virtual machine, media support, integrated browser and optimized graphics support. It also includes support for GPS, Blue-tooth, Accelerometer, Camera, WIFI and 3G Networks amidst other things [6]. Figure 3 below shows the major components of the Android operating system. This project directly makes use of the Applications and the Application-framework layer and also makes use of the SQLite library.

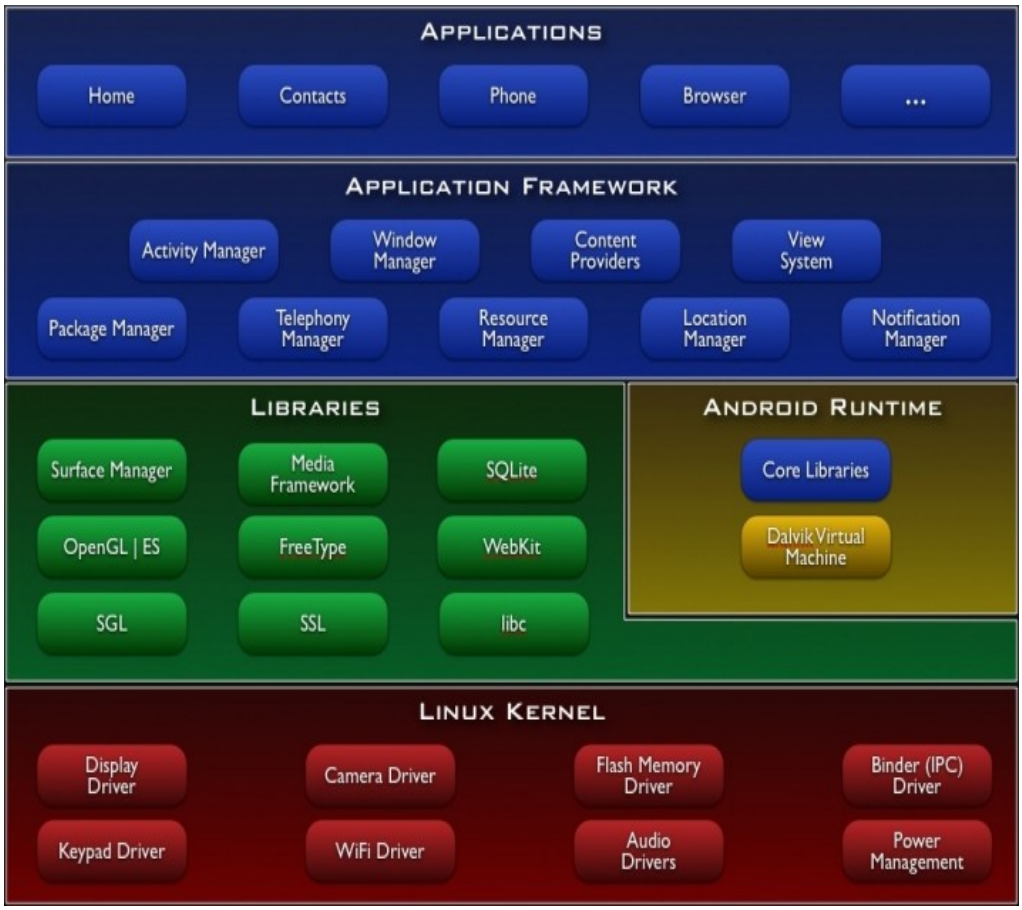

Figure 3: Android Architecture[6]

Presently, the dominant mobile phone Operating System are Apples' iOS, Google's Android, Nokia's Symbian, BlackBerry's OS and Microsoft's Windows Mobile. The criteria considered for choosing are, the market share, tablet support, the ease of application development, supported desktop platform and the license of the application. Table 1 compares these operating systems according to the license, programming language and the officially supported development platform.

	License	Programming Language	Development Os Platform
Android	Open Source	Mainly Java	Cross Platform
iOS	Proprietary	Objective-C	Mac Only
Windows Mobile	Proprietary	Dot NET languages (C#, C++, VB. Net)	Windows Only
Blackberry Os	Proprietary	Java	Windows
Symbian	Proprietary	C++	Cross Platform

Table 1: Mobile Operating System Comparison

Android was preferred because of its open source nature, ease of development, zero barrier to entry and pervasiveness. “It quickly gained attractions by developers because of its fully developed features to exploit the cloud-computing model offered by web resources and it also enhances that experience with local data stores on the handset itself” [7]. Presently, it is the market leader in smart phone[8]. Application written for the handset can also be easily deployed for tablet devices with little or no technical changes needed.

Android is released under two different open source licenses. The Linux kernel which it is based upon, is released under the GNU Public License (GPL) as is required for anyone licensing the Linux operating system kernel. The Android platform, excluding the kernel, is licensed under the Apache Software License (ASL) [9].

2.2 Android Development

What goes on in an Android application is mainly divided into two, including the visual part, which the user interacts with and the non-visual, which runs in the background. The visual part is termed an activity. An activity is usually a single screen that the user sees on the device at one time. An application typically has

multiple activities, and the user flips back and forth among them [10]. The previous activity maybe paused or destroyed and the new one maybe newly created or just resumed. Figure 4 below shows an activity life-cycle and the different states which it can exist in.

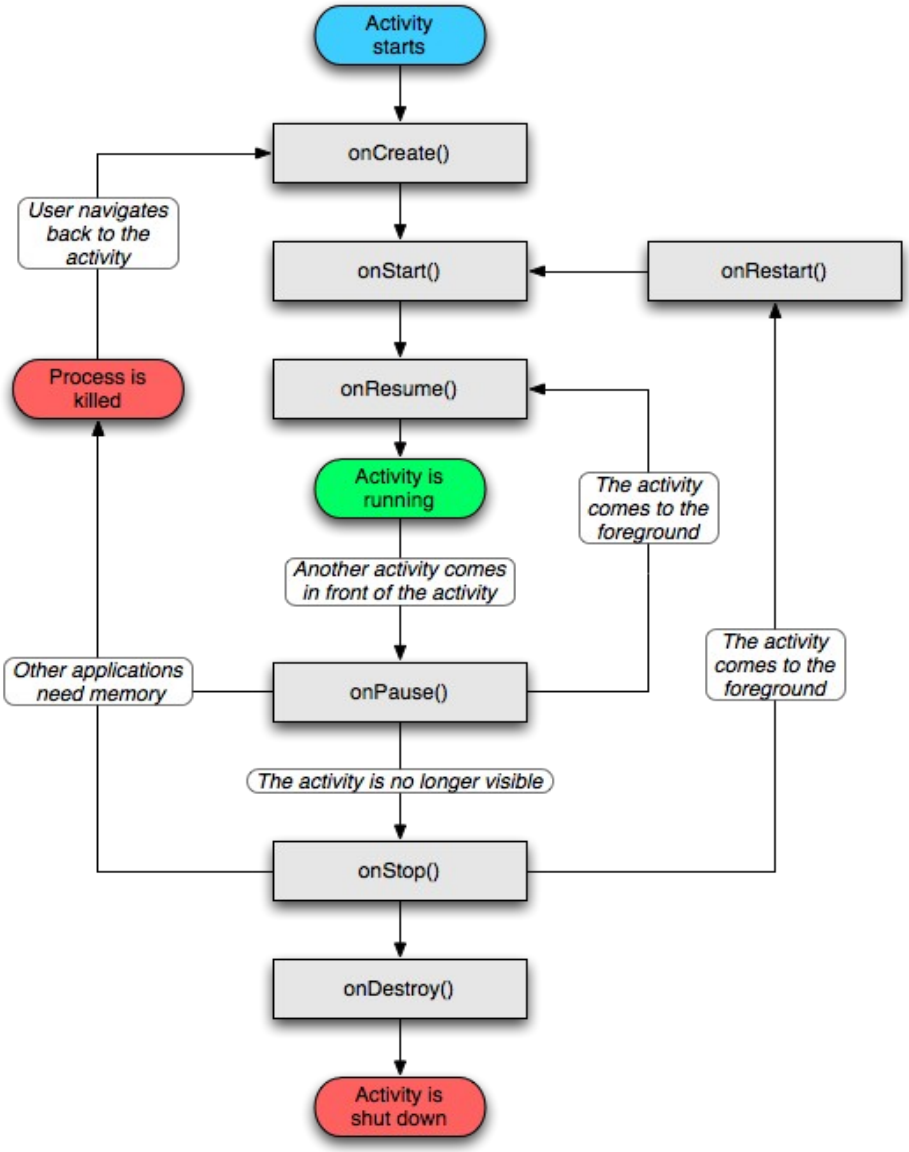

Figure 4: Activity Life cycle [6]

The non-visual parts are called Services. They run in the background and don't

have any user interface components. They perform actions such as polling data from a server on the Internet, playing music in the background etc. They are responsible for those actions that must continue while the user flips between activities of the same or different applications.

BroadcastReceiver is also an important component of Android development. It is the system's way of alerting applications of certain events in the systems when they occur. For example, the system broadcasts an SMS arrival event when a new SMS arrives on the mobile phone. Subsequently all application that are registered to receive this events will be notified and each of them will respond accordingly.

2.2.1 Android SDK and Eclipse

The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language. The SDK is available on Windows, Linux and Mac [6].

Eclipse is a multi-platform development environment that runs on all major operating system. Android provides a plug-in for Eclipse that allows easy use and control of the Android SDK facilities. This plug-in is called ADT Plugin (Android Development Tool). Once installed the ADT plug-in is made to point to the SDK location thereupon most of the SDK tools can then be accessed through Eclipse. While other IDEs exist for Android development, Eclipse was preferred because it's the officially supported environment [6].

2.2.2 Android Debug Bridge

Android Debug Bridge (adb) is a versatile tool that allows managing the state of an emulator instance or an Android-powered device [6]. It is part of the tool bundled with Android SDK. With it, one can issue commands to the emulator or device and inspect or modify its internal states. Table 2 shows the adb commands that was commonly used during the development of this project.

Command	Description
devices	Prints a list of all attached emulator/device instances
shell	Starts a remote shell in the target emulator/device instance.
logcat	Prints log data to the screen.
push <local> <remote>	Copies a specified file from your development computer to an emulator/device instance.
pull <remote> <local>	Copies a specified file from an emulator/device instance to your development computer.

Table 2: Commonly used adb commands [6]

Some of Linux command line functions are also accessible through the adb. First, the command *adb shell* has to be run to log in; afterwards, commands such as *ps, ls* and *top* can then be run. While the Eclipse plug-in provides a GUI front-end to most of the adb functionalities, the command line is well suited or sometimes necessary to access the device or emulator. Command-line invocation is also preferred, because sometimes the eclipse plug-in loses communication with the emulator server.

During the development of this project, the command-line has been mainly used in viewing the database, viewing the system logs and managing more than one emulator at a time. Viewing the database is done by evoking the sqlite3 command with the full path to the database location [6]. Figure 5 below shows the invocation of the shell and the logcat command.

```
lawal@lawal-ubuntu:~$ adb shell
# sqlite3 /data/data/com.tracker/databases/itracker
SQLite version 3.6.22
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .tables
android_metadata  assets       assettypes      data
sqlite>


lawal@lawal-ubuntu:~$ adb logcat | grep System.out
I/System.out(  324): DeleteActivity.onClick() null
I/System.out(  324): MainMapActivity.onActivityResult() 44 -1
Intent { (has extras) }
I/System.out(  324): MainMapActivity.onRestart()
I/System.out(  324): MainMapActivity.removeLastPosMarker()
I/System.out(  324): MainMapActivity.onResume()
I/System.out(  324): MainMapActivity.onPause()
I/System.out(  324): MainMapActivity.onDestroy()
```

Figure 5: adb command line

3 OVERVIEW AND SYSTEM COMPONENTS

The hardware involved in the project includes the tracker and the mobile phone to which the application is installed. The tracker includes an SMS module and a GPS receiver. The tracker obtains the location, direction and speed information from the GPS receiver and sends it via SMS to a configured mobile phone number at preconfigured intervals or when certain events occur. The application receives and interprets the SMS, and shows the asset location on a map and performs other associated logic that are described in 3.2.1. Figure 13 shows an overview of the system components.

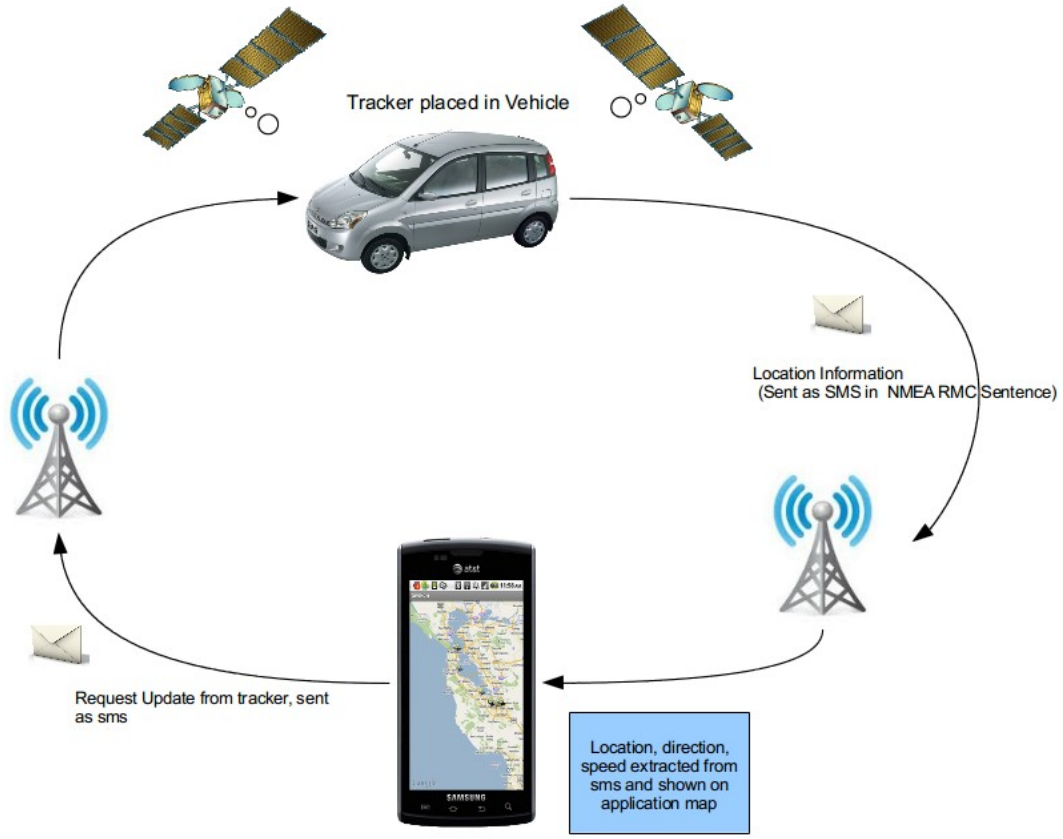

Figure 6: System Components Overview

3.1 Application Requirements

The application is targeted at any Android powered mobile device running at least Android 2.2, Froyo. It should be intuitive to use and be locale aware. Users would be offered feedback for important actions, to make them feel on track. Also when user error occurs, the user should get simple and easy to understand error message and what to do next.

3.1.1 Asset Management

An asset in the context of this project refers to a real world object to which the tracker is attached; for example, a vehicle or truck. The user should be able to register a new asset, edit and delete existing ones. Table 3 outlines the fields that

are to be provided when registering an asset. It also describes the type of data required for registration and from where the user will obtain them.

Field	Data Type	Source	Compulsory
Unique Identification (UID)	String, not more than 10 characters.	Provided from the tracker device.	Yes
Tracker Phone Number	Phone Number	Provided from the tracker device.	Yes
Password	Any combination of letters and numbers between 3 and 10	Provided from the tracker device.	Yes
Image	Image types	Provided by user independently.	No
Asset type	Description of the kind of Asset being tracked, whether vehicle, truck, digger etc.	Provided by user independently.	Yes

Table 3: Registration requirements

At each instance of the application an asset must be selected and monitored. The application should also allow user to select which asset is being actively monitored.

3.1.2 Location Management

At each instance of the application, the last known location of the selected asset must be visible on the map. The location of an asset, whether selected or not must always be updated whenever a message meant for it arrives. The user should also be able to view her location relative to the asset and the proximity of distance should be made known to the user. The markers used to show the asset's last location and those before it, should be different. Separate markers should also be used to show the user's location.

One key feature needed is the ability to create a “geofence”, a circular boundary, as shown in Figure 7 with a green circle. The asset is not expected to step out of this geofence, so when it is determined to be out of it, some form of notification

should be used to alert the user of the violation. The notification alert can be by sound or vibration or both, based on user preference.

Initially the tracker is configured to send location data only on request, however it also has the ability to send data at some regular intervals or when certain events occurs. User should have the ability to change this setting, and make the tracker send data at some intervals as described in Table 5.

Figure 7: Mock up of Application

3.1.3 Data Management

The users should also have complete control over the location data and should be given the option to delete them all or within some specified time range. It is desirable that the location data should be exportable to other format such as the GPX or KML format.

The tracker's password should not be visible or accessible to an external party. Some form of obfuscation or encryption should be provided such that if the user loses the phone, the trackers password is not easily obtainable from any of the data storing mechanism employed. At no point should any unrelated user specific data be stored or collected through the application to a web-server.

3.2 Tracker Communication

Communication between the application and the tracker is done only through SMS messages. Since both the tracker and mobile phone, are SIM based and battery-powered devices, loss of battery and lack of SMS Gateway may cause them not to receive a message. As for the tracker, when it regains connectivity or power, it processes the first SMS it receives and discards any new ones it receives while doing so. Therefore, the application should provide a way of ensuring its state and that of the tracker stays synchronized with regards to update interval described in 3.2.2. When the phone recovers from a loss of power or connectivity it should be able to cope with the high volume of messages it may subsequently receive.

From the tracker side, messages are only initiated when requested, or when demanded to be sent at some certain intervals or certain event as further explained in 3.2.2.

3.2.1 Receiving data

The message received from the tracker contains the GPS location information, speed and direction of the asset being tracked encoded in a modified NMEA RMC format. The RMC, Recommended Minimum Sentence C, is the NMEA version of

essential GPS pvt (position, velocity, time) data [11]. A complete data unit is referred to as a sentence. An NMEA sentence begins with a dollar sign and the letters GP, ($GP) and contains a number of fields, separated by commas. Each field in the sentence is important in determining the location, velocity, direction of the asset at a particular point in time except for the last field which is a checksum to ensure the data has not been tampered with. The fields and their description are shown in Table 4. Using the following NMEA sentence as a guide, the table describes the fields and their interpretation.

$ GP**RMC,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A**

FIELD	DESCRIPTION	EXAMPLE	INTERPRETATION
Sentence Type	Shows it is an RMC	RMC	Sentence is RMC
Time	Six-digit letters describing the time the data was taken in UTC	123519	12:35:19 UTC
Status	The value can either be A (Active) or V(Void)	A	Good fix
Latitude	Latitude value in degrees	4807.038	48° 7' 38.000"
Latitude direction	The direction of latitude. Can either be N or S (North or South)	N	North
Longitude	Longitude value in degrees	01131.000	11° 31' 000"
Longitude direction	The direction of latitude. Can either be E or W (East or West)	E	East
Speed	Speed over the ground in Knots	022.4	22.4 Knots
Track angle	Track angle in degrees	084.4	84° 4'
Date	Date when the data was taken in the format (ddmmyy)	230394	23rd of March 1994
Magnetic Variation	Magnetic Variation	003.1	3° 1'
Magnetic Variation Direction	Magnetic Variation Direction	W	West
Checksum	Checksum to check the validity of the data. Starts with *	*6A	Checksum value

Table 4: NMEA Fields

The NMEA sentence was modified by replacing the first 6 characters i.e., “$GPRMC” to '%$' followed by the four letters that uniquely identifies the asset. Therefore for an asset with id, WXYZ, a sample message will be **%$WXYZ**,123519,A,4807.038,N,01131.000,E,022.4,084.4,230394,003.1,W*6A.

3.2.2 Sending data

Messages sent to the tracker from the application are primarily to change how often or under what circumstances the tracker sends location updates. All messages sent should be made only with explicit consent of the user and the monetary consequence should be made known to the user.

Primarily, the tracker sends location data when it receives command requesting it to do so. However, it can also be configured either to work in a timing mode or in a tracking mode. In the timing mode, the tracker sends location information regardless of the state of the asset at specific time intervals. The interval between each location updates vary from ten minutes to five hours as shown Table 5. In the tracking mode however, location updates are only sent when variables such as the speed and direction of the asset changes. Changing time interval in timing mode or turning off updates can be done by sending commands to the tracker. The commands and the format in which they are sent are listed and explained in Table 5 below. For the purpose of the illustration, we assume the password for the tracker is BW1.

MESSAGE	FORMAT	EXAMPLE
Request immediate updates	Password+ 'S'	BW1S
Turn off updates	Password+'R0'	BW1R0
Change update interval to 10 minutes	Password+'R1	BW1R1
Change update interval to 30 minutes	Password+'R2	BW1R2
Change update interval to 1 hour	Password+'R3'	BW1R3
Change update interval to 2 hours	Password+'R4'	BW1R4
Change update interval to 5 hours	Password+'R5'	BW1R5
Change to tracking mode	Password+'RT'	BW1RT

Table 5: Message format to tracker

The characters in the SMS text message **must** be capitalized.

4 APPLICATION DESIGN AND IMPLEMENTATION

Based on the requirements described in 3.1 and the description of how the tracker communication works in 3.2, the application has to be designed using the right methodology, libraries, database design, programming design patterns whilst delivering the best user experience possible on a mobile phone. Future maintenance, security and testing must also be put in mind as the application is to be developed in such a way that guarantees its robustness.

4.1 System Development Methodology

An agile development strategy was adopted in the development of this project. Other strategy such as the Waterfall model will not be suitable for a couple of reasons. Firstly, obtaining a complete analysis of the way the tracker works at the beginning of the project was impractical. Only at certain stage of development will additional information be demanded and obtained, because no formal document detailing the internal workings of the tracker was obtainable from the owner company. Secondly, requirements change so often, mainly due to the constraints of mobile phone development, such as speed, size and usability. If Waterfall was adopted, this will require a complete redesign of the system every-time such changes occur. Lastly, considering that the project involves communication without another product, the source of bugs found in each stage has to be established before the project can move on. Had the Waterfall model been adopted, testing will be delayed till the end, which means bugs found may result to the rewrite the whole program.

4.2 Application Implementation

From the requirements obtained in 3.1, a use case can be made for the application. This is illustrated by diagram Error: Reference source not found.

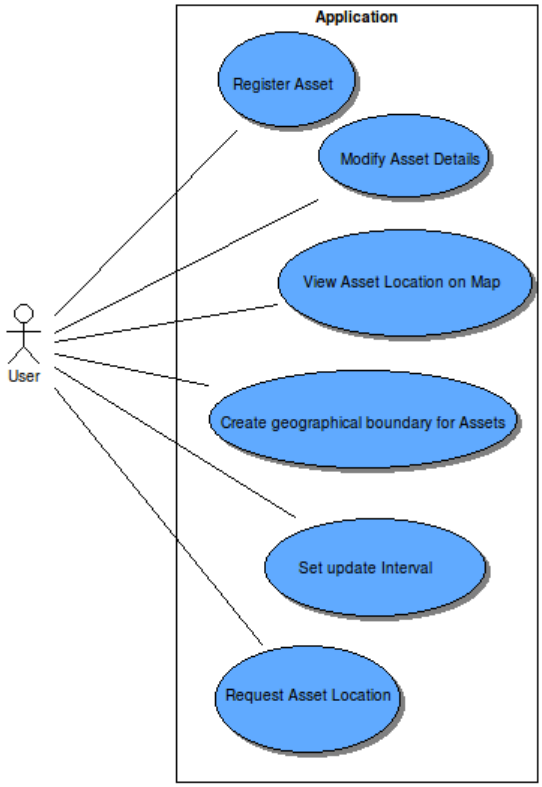

Figure 8: Use Case Diagram

Implementation of the use cases was done by using the right data structures and programming pattern. Since Android application is primarily written in Java, which is an object-oriented language, classes were written to separate logic and data structures. The classes in the application can be broadly divided into those for UI, background services, data-structure and utilities. A broad class diagram showing how the activity and services classes are connected is Figure 9.

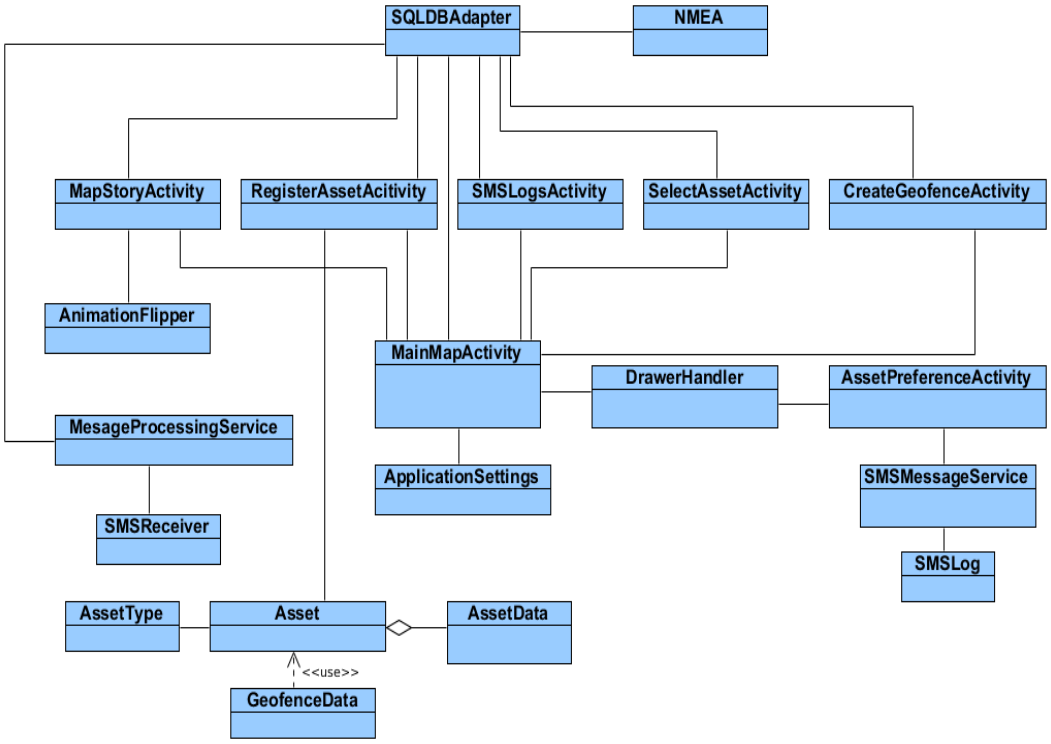

Figure 9: A broad class diagram

4.2.1 GUI Classes

Classes that display user interface in Android must extend the Activity class or a class that does so. The following classes are responsible for the user interface in the application. MainMapActivity, MapStoryActivity, SMSLogActivity, SetGeofenceActivity, RegisterAssetActivity, SelectAssetActivity, ViewAssetDetails, ApplicationSetting and AssetPreference.

The MainMapActivity class is the main entry point for the application. It displays a map and overlays the location markers on the map if they are available. It also houses the main menu of the application. From the main menu some of the other activities can be launched. It also houses a sliding layout by the right, with which visible location markers are toggled on and off on the map. The class diagram of the MainMapActivity is shown in Figure 10 below.

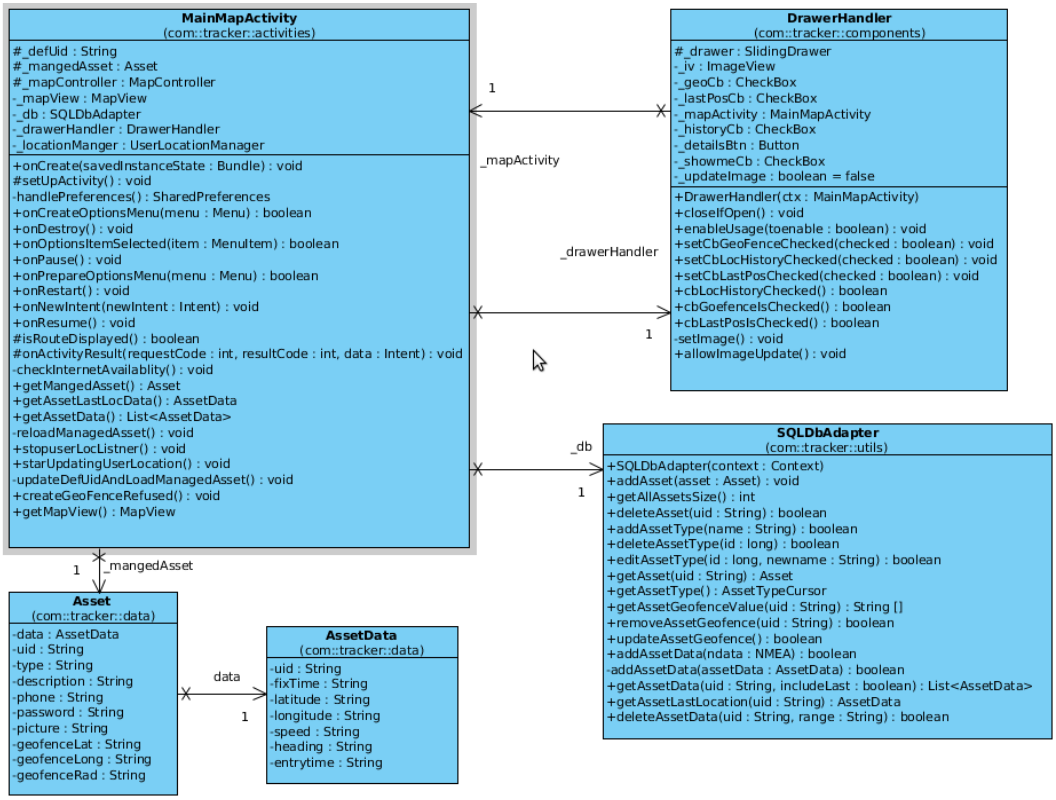

Figure 10: MainMapActivity

The DrawerHandler class is responsible for routing event from the components inside the slider view to the MainMapActivity class to process. It also handles the display of an asset image in the slider if available.

The RegisterAssetActivity class displays a form which is filled by the user to perform asset registration. It then handles the saving of the data entered into the database. It is also used when if the user needs to edit the details of an already existing asset. The class diagram of the RegisterAssetActivity class is shown in Figure 11 below.

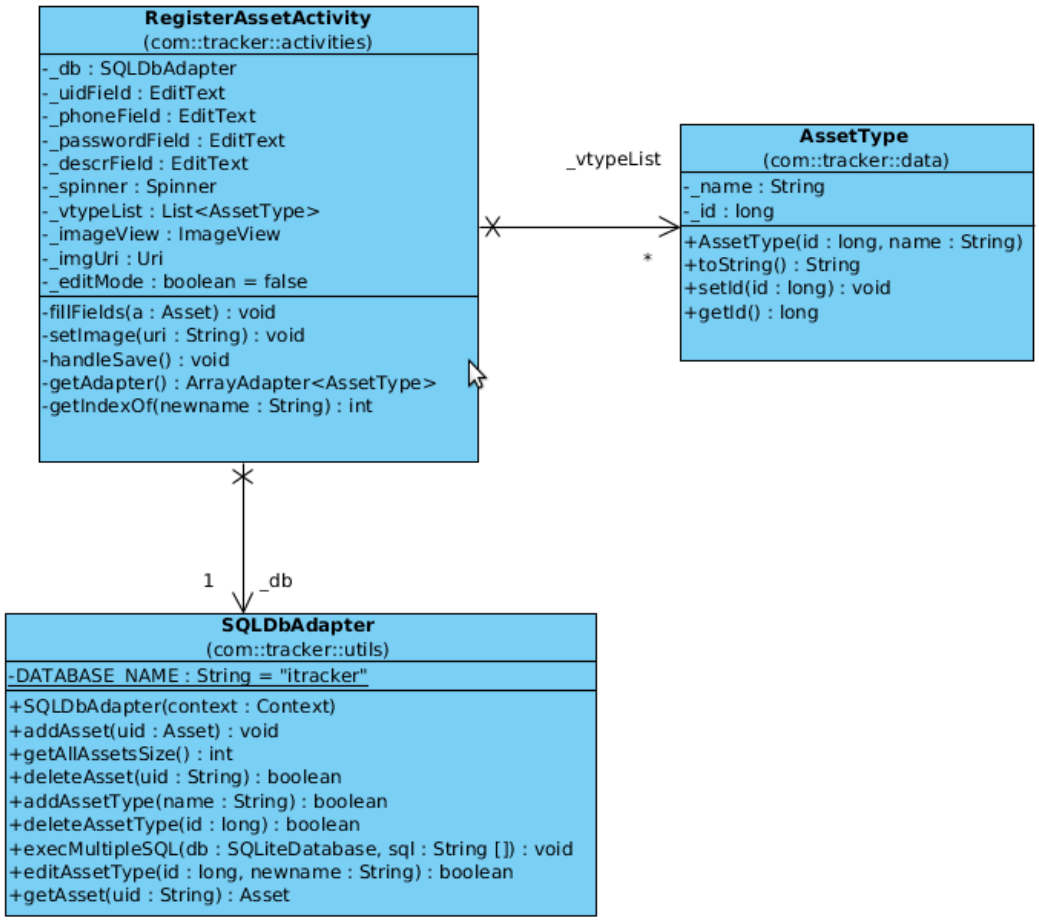

Figure 11: RegisterAssetActivity Class diagram

SetGeoFenceActivity class handles the UI and associated logic for creating the "Geofence". The class displays a map and detects the latitude and longitude of the position the user taps on the map. This point is set as the centre of the geofence. When the radius is changed, the class overlays on the map a proportionate transparent circle that shows the extent of the geofence. Figure 12 shows the UI that is generated from this class.

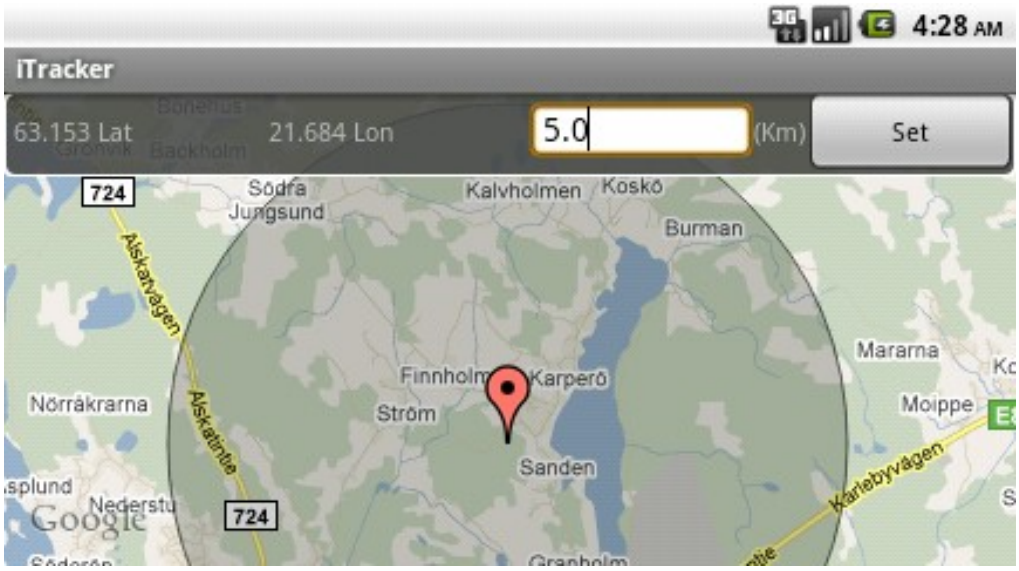

Figure 12: Geofence Creation UI

SMSLogActivity displays the list of SMS received or sent by the application. It provides a button with which user can delete the logged SMS.

The MapStoryActivity class is used to animate the display of asset location on the map, thereby showing sequentially how the map has been updated and time and speed difference between each data.

4.2.2 Data Classes

The data classes are classes used to hold logically related data. They include Asset, AssetType, AssetData, SMSlogs, GeofenceData and NMEA class. These classes have logical equivalence to tables in the database. They transform the data in the table into Java objects. The first three classes are equivalent to tables asset, asset_type, asset_data and sms_log table respectively as described in 4.3.2. The GeofenceData class is class that holds a subset of the Asset data, its geofence radius and center position. In the case of NMEA, it takes as input the raw NMEA Sentence described in 3.2.1 and parses and validates the contained data in it such as latitude, longitude, speed, time, which then becomes values for asset data. The class diagram of the NMEA is shown in Figure 13 below.

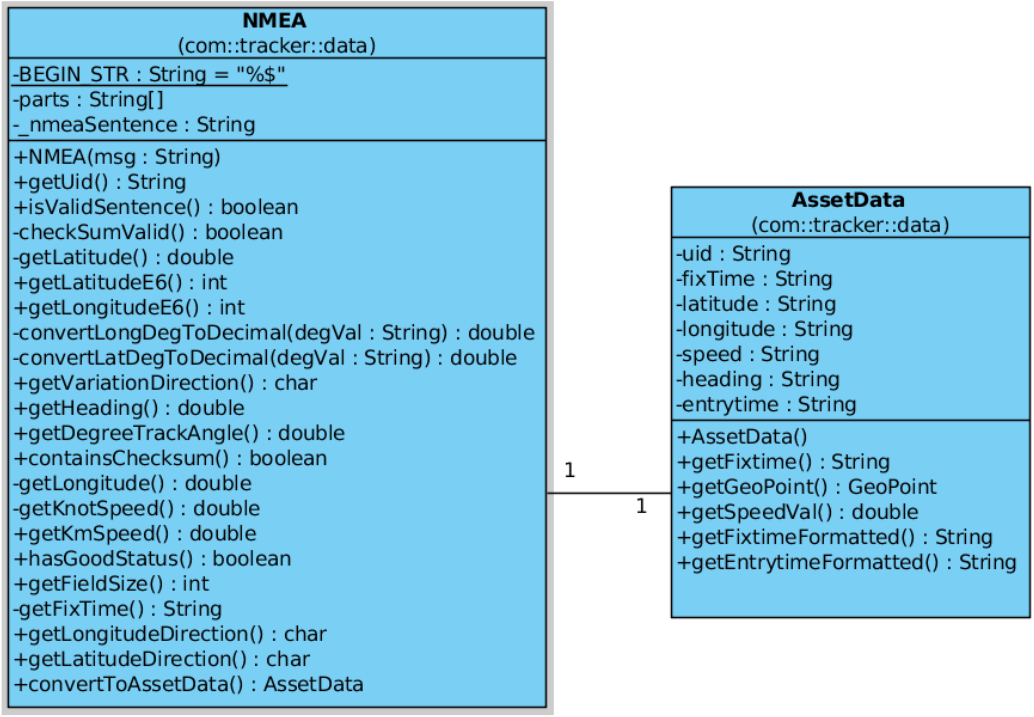

Figure 13: NMEA class diagram

4.2.3 Service Classes

Service classes are responsible for the non-visual part of the application. They include the MessageProcessingService, SmsMessageSender and SmsMessageReceiver class. The SmsMessageReceiver class is more accurately describe as a BroadcastReceiver. It is a service that is started by the system when a new SMS arrives on the phone. It determines if the message is meant for the application. If it is, it starts the MessageProcessingService and passes the SMS to it for further processing. It passes the SMS to the MessageProcessingService class because the receiver is only started for a few seconds before it is killed by the system. Figure 14 below shows the process that starts when a new SMS arrives on the mobile-phone for the application.

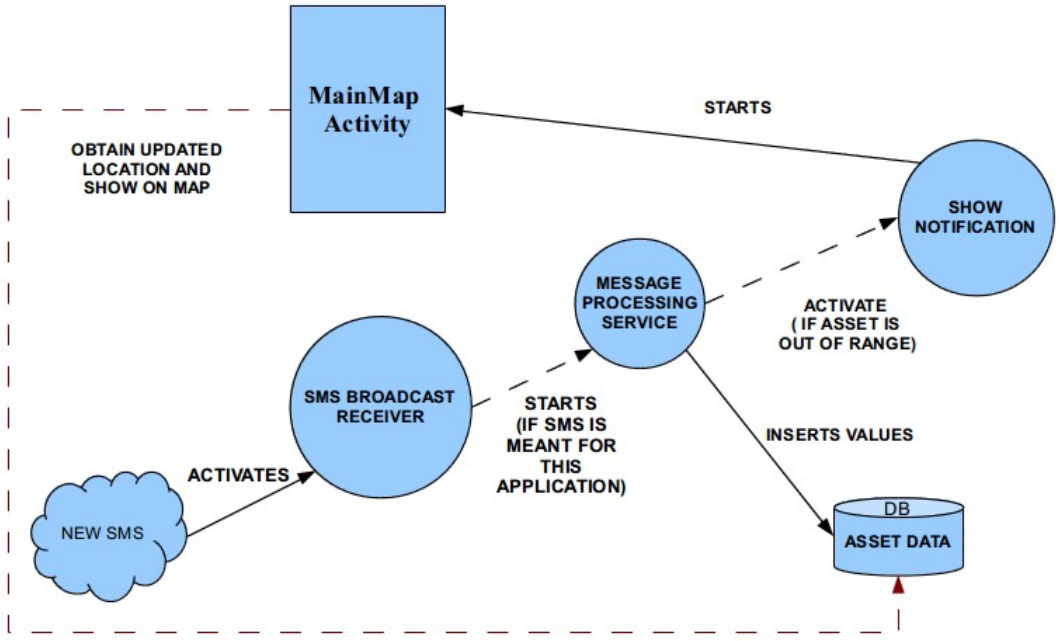

Figure 14: Events on new SMS

MessageProcessingService class is a service that processes the SMS messages it receives from the SMSMessageReceiver class and insert the relevant fields to the database. It also handles firing of notifications to signal a new SMS for the application has been received or show that the concerned asset is out of its geofence.

SmsMessageSender class is started whenever the application needs to be send an SMS to the tracker. In addition to sending the SMS, it also dynamically register for a BroadcastReciever for the Sent SMS to receive validation that it was truly sent and then logs the SMS. The class diagram of the services is shown in Figure 15 below.

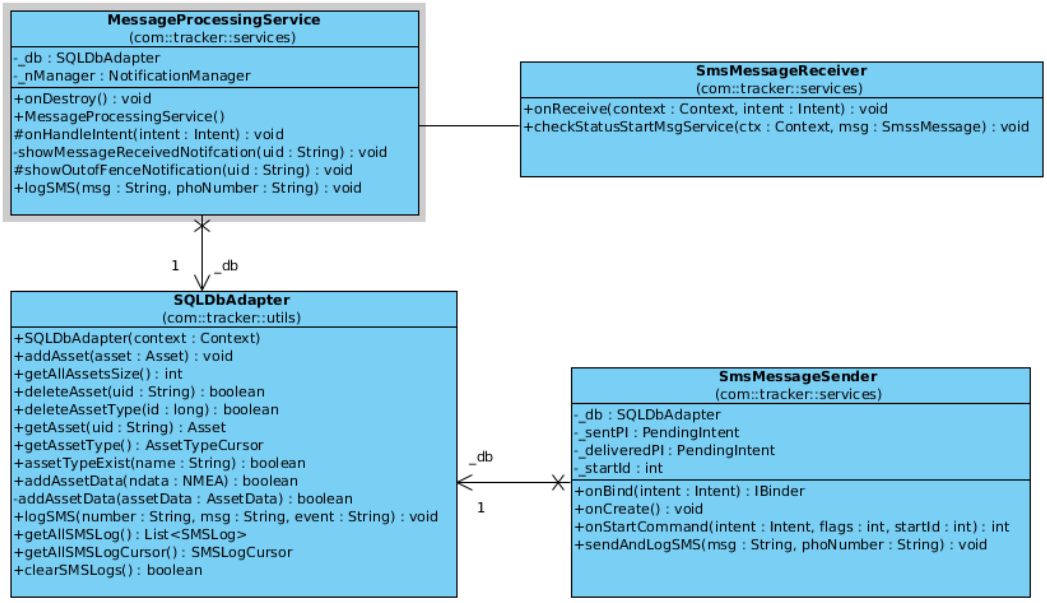

Figure 15: Services Class diagrams

4.2.4 Utility Classes

Some utility classes were written to group functions together according to context. This is to avoid repetition of the same functions in different classes, these functions are grouped together according to their context. They include SQLDBAdapter, StringUtils, MapUtils, Helper and ViewUtils class.

SQLDBAdapter class extends the SQLiteOpenHelper class and is used to manage database queries. It handles the opening of the database, executing the queries, converting the result of the queries into Java Object and closing the database afterwards. This class is needed because it separates database concerns away from other classes.

The StringUtils, MapUtils, ViewUtils classes are responsible for common manipulation done on strings, map data and UI components respectively. The Helper class is a general purpose class that aggregate functions that doesn't fit any of these three categories. The class diagram of these classes is shown in Figure 16 below.

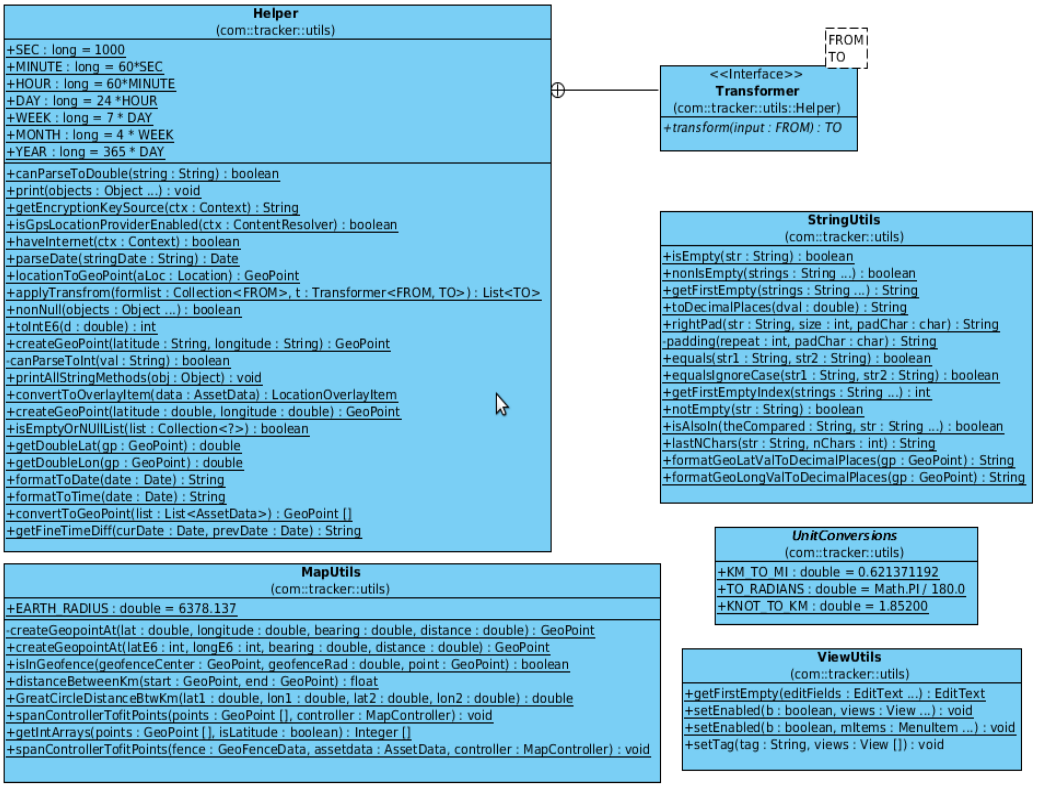

Figure 16: Utility Classes

4.3 Data Persistence

Persisting data in an Android application can be done by either by storing it on the device itself or transferring it through a network connection to a web server. The latter was not used because the application is designed to be server-less. To store the data on the device, one has the option of using the provided SQLite database or creating and writing the data to a self-managed file. Using the database is more suited to the project, as this will ensure data and referential integrity. It will also allow complex data manipulation or queries to be easily made when obtaining data.

4.3.1 SQLite

SQLite is an in-process library that implements a self-contained, server-less, zero-configuration, transactional SQL database engine [12]. Android provides an SQLite database with which any application can persist data. This database is

sand-boxed and can be accessed only from within the application that created it [6]. While SQLite syntax is similar to that of any other SQL database, some of its features such as Manifest typing and variable-length records are unorthodox. Manifest typing implies that the column data-type does not restrict the data-type that can be accepted [12]. So rather than the column type to be fixed as defined when creating the table, the column type is more of a suggestion and this is known as a type affinity [13]. Variable-length records means the length associated with a column is not enforced and the size taken up by a column varies for each row [12].

Despite these two unorthodox concepts, the application database design was still done in the traditional manner. This is because it serves as a sort of documentation and it expresses the intended logical structures. It poses no problem because SQLite has a series of rules in mapping the traditional column types to its most logical type affinity [13].

4.3.2 Entity Diagrams

Four tables were created, namely *assets*, *assetsdata*, *assetstypes* and *smslogs*. The first three are used in persisting data related to the location data. The smslogs table is used to log the SMS sent or received by the application. Figure 17 shows the entity-relationship diagram of these tables.

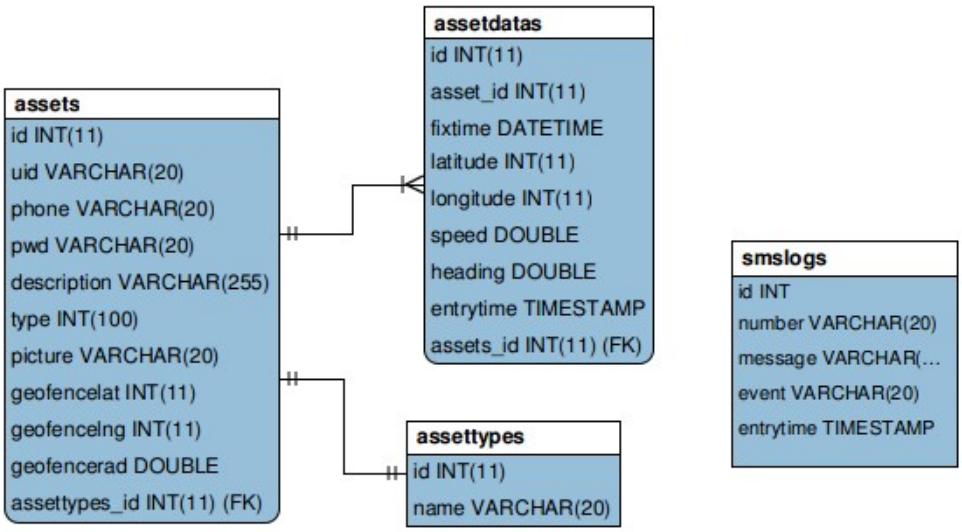

Figure 17: Entity Diagram

4.3.3 Data Security

Values stored in the database are visible for anyone that can get hold of the mobile phone database files. Therefore it is necessary to obfuscate the assets password from being in plain text in the database. AES encryption was used to obfuscate the password. The encryption could be done using the asset unique identifier or a unique identification on the device if available. The encryption is applied before the password is inserted into the database and decrypted when needed afterwards.

4.4 GUI

The layouts of UI controls in Android is primarily done using XML declarations and can also be done in code at runtime. Using XML allows the separation of logic from presentation. So changes can be easily made to the layouts without modifying the source code. It also allow easy creation of layout for different screen orientation which is then automatically handle by the framework when the device orientation changes. Figures 18 and 19 shows such scenario.

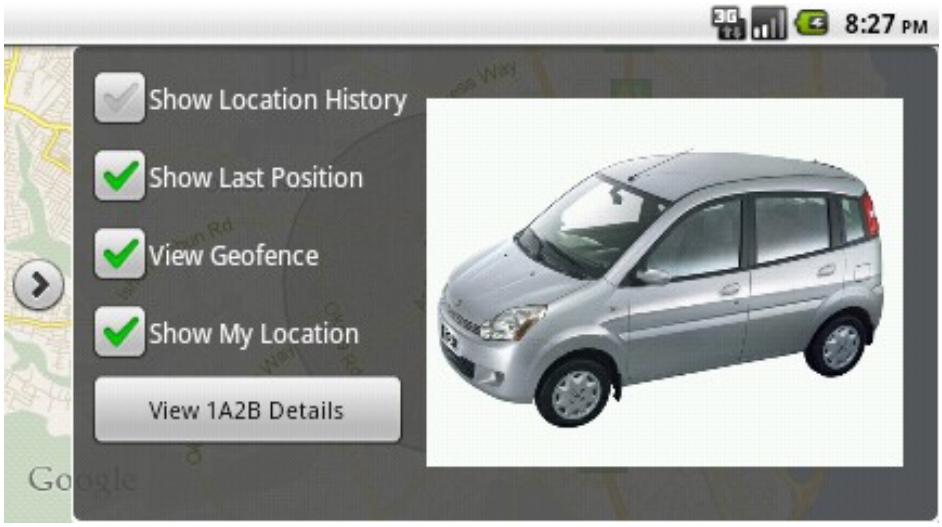

Figure 18: Sliding Drawer Landscape Layout

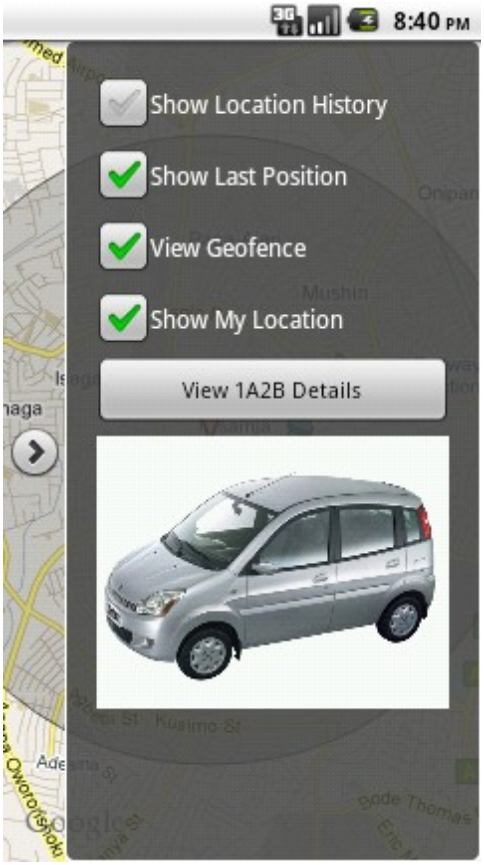

Figure 19: Sliding Drawer Portrait

Except in some cases, XML was used to create the UI for most of the activities' layouts.

4.5 Testing

The agile methodology used in the development of the project demands that testing is done at each cycle of development. For each of the functionalities implemented, the application is tested and shown to the company's representative. From the feedback obtained, bugs are fixed and some changes are made before proceeding to the next feature.

The testing that was done can be categorized into those within the application itself and that which involves communication with the tracker. The former is primarily done using the Android provided testing framework and the latter, by the use of a simulated tracker most often and the real tracker itself.

4.5.1 JUnit Testing

JUnit is an API that enables developers to easily create Java test cases. It provides a comprehensive assertion facility to verify expected versus actual results [14]. Android provides a testing framework which is based on JUnit, that could be used to test every aspect of the application at every level [6]. With the JUnit it is possible to programmatically input text into fields, make selections, click buttons and navigate the application., thereby making it possible to test the expected outcome of events. This was particularly useful in guaranteeing some of the applications logic still works when heavy changes were being made to the source code. Figure 20 shows the test written to assert that the RegisterAssetActivity class sets an error message if the user does not select an asset type.

```
@MediumTest
public void testRegistration() {

    // insert testcar in uid field
    sendKeys(KeyEvent.KEYCODE_T, KeyEvent.KEYCODE_E, KeyEvent.KEYCODE_S, KeyEvent.KEYCODE_T,
            KeyEvent.KEYCODE_C, KeyEvent.KEYCODE_A, KeyEvent.KEYCODE_R);

    // go down
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);

    // // phoneNumber field
    sendKeys(KeyEvent.KEYCODE_PLUS, KeyEvent.KEYCODE_3, KeyEvent.KEYCODE_6, KeyEvent.KEYCODE_8,
            KeyEvent.KEYCODE_4, KeyEvent.KEYCODE_4, KeyEvent.KEYCODE_0, KeyEvent.KEYCODE_4,
            KeyEvent.KEYCODE_2, KeyEvent.KEYCODE_8, KeyEvent.KEYCODE_6, KeyEvent.KEYCODE_3, KeyEvent.KEYCODE_1);

    // go down
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);

    // password field
    sendKeys(KeyEvent.KEYCODE_W, KeyEvent.KEYCODE_X, KeyEvent.KEYCODE_Y);

    // go down four fields
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);
    sendKeys(KeyEvent.KEYCODE_DPAD_DOWN);

    assertEquals(_uidField.getText().toString(), "testcar");
    assertEquals(_phoneField.getText().toString(), "+368440428631");
    assertEquals(_passwordField.getText().toString(), "wxy");

    // now save button has focused, press enter
    assertTrue("Save button should have the focus now", _savebtn.isFocused());
    sendKeys(KeyEvent.KEYCODE_DPAD_CENTER);

    // no asset type is selected so error message should be set to errorText now.
    assertEquals(_errorText.getText().toString(), _regActivity.getString(com.tracker.R.string.error_no_assetType));

}
```

Figure 20: RegisterActivity Junit Test case

It was also used in testing the application reaction to changes in system configuration like screen orientation, lack of SMS, slow network connection etc.

4.5.2 Tracker Communication Testing

A separate Android application was written to simulate the functionalities of the tracker. This was written at first when the tracker was unavailable and subsequently used to avoid incurring too much cost of sending real SMS if the actual mobile phone and tracker is used. The simulation was possible due to the fact that Android provides the ability to have two emulator instances which can communicate with each other via SMS. Also latitude and longitude values could also be sent to an emulator to simulate its position. The simulator application was written such that it sends it location information at some intervals as the tracker would have done using the same NMEA format. Figure 6 is a screen-shot of both the tracker simulator application and the real application side by side. Testing with the actual tracker was performed also to ensure that functionalities work as expected.

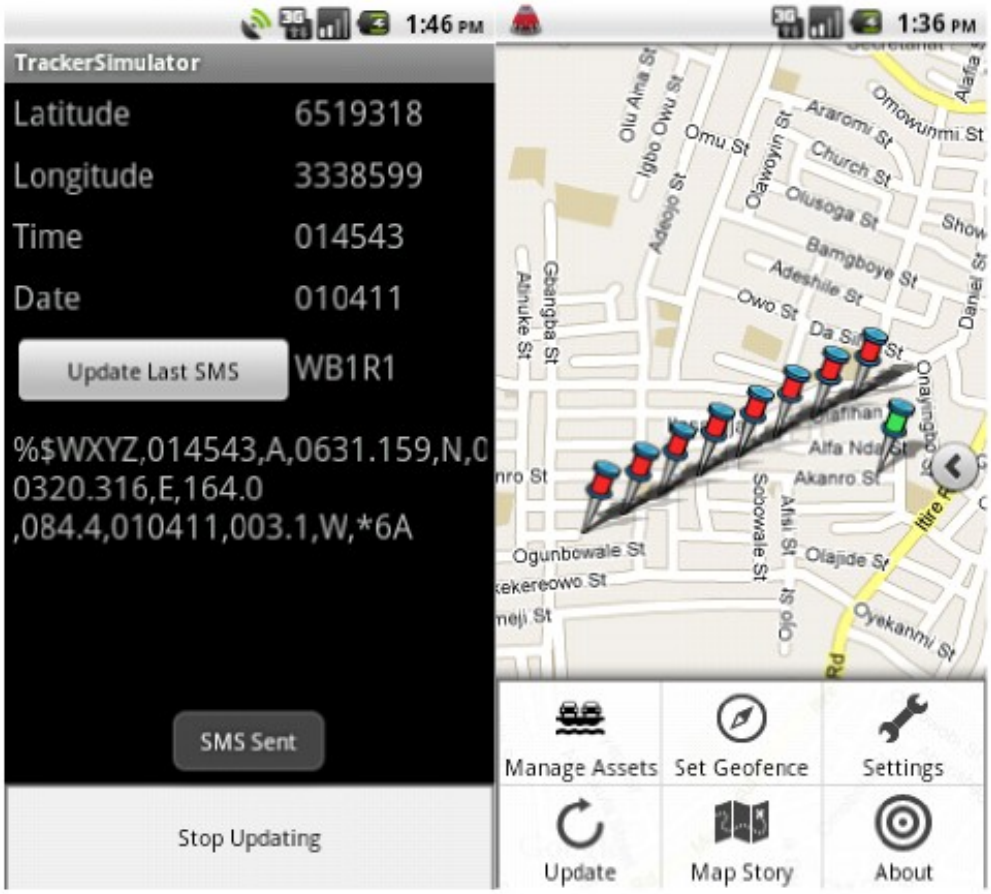

Figure 21: Simulator and Application

5 CHALLENGES

Every software projects comes with its own unique set of challenges and this project is not an exception. Many of them where due to insufficient knowledge of the Android API especially where it diverges from the standard Java way of doing things. Others were more generic to the implementation of this project and it is those that will be mentioned.

5.1 Calculation Distance between two locations.

The first challenge was to find a suitable and easy way to calculate the distance between two geographical location bases on their latitude and longitude. This problem was solved by using the Great Circle Formula, which assumes the earth to be sphere. If we take two geographical positions p and q as point on Spherical

Earth of radius R. It can then be proven that

If p, q represent two point on a sphere with radius R

$$|pq| = \frac{2\pi R}{360} \cdot t \qquad \ldots (1)$$

in radian

$$|pq| = R \cdot t \qquad \ldots (2)$$

We can represent these two points as vectors on a sphere from the center as $\vec{u}$ and $\vec{v}$ whose spherical coordinate are (r_1, θ_1, ϕ_1) and (r_2, θ_2, ϕ_2) respectively where r is the radius of the sphere

In rectangular cordinates

$$u = (r\cos\phi_1 \sin\theta_1, r\cos\phi_1 \cos\theta_1, r\sin\phi_1)$$

$$v = (r\cos\phi_2 \sin\theta_2, r\cos\phi_2 \cos\theta_2, r\sin\phi_2) \quad \ldots (3)$$

Using the great circle that passes through these two vectors,

$$\vec{u} \cdot \vec{v} = |u||v| \cdot \cos t$$

$$t = \arccos\left(\frac{u}{|u|} \cdot \frac{v}{|v|}\right) \quad \ldots (4)$$

$$|u| = (r\cos\phi_1 \sin\theta_1)^2 + (r\cos\phi_1 \cos\theta_1)^2 + (r\sin\phi_1)^2 = r$$

$$|v| = (r\cos\phi_2 \sin\theta_2)^2 + (r\cos\phi_2 \cos\theta_2)^2 + (r\sin\phi_2)^2 = r$$

Therefore normalizing u and v cancels out r, thus

$$t = \arccos[(\sin\theta_1 \cos\phi_1 \sin\theta_2 \cos\phi_2) + (\cos\theta_1 \cos\phi_1 \cos\theta_2 \cos\phi_2) + (\sin\phi_1 \sin\phi_2)]$$

On further reduction

$$t = \arccos[\cos\phi_1 \cos\phi_2 \cos(\theta_2 - \theta_1) + \sin\phi_1 \sin\phi_2] \quad \ldots (5)$$

Hence substituting in [2], we obtain the distance between p and q

$$|pq| = R \cdot \arccos[\cos\phi_1 \cos\phi_2 \cos(\theta_2 - \theta_1) + \sin\phi_1 \sin\phi_2]$$

This is then implemented in code as shown in Figure 22 below.

```
public static double distanceBtw(double lat1, double lon1, double lat2, double lon2) {
    /* Great Circle Distance Formula */
    /* r * acos[sin(lat1) * sin(lat2) + cos(lat1) * cos(lat2) * cos(lon2 - lon1)] */

    double EARTH_RADIUS = 6378.137; // in kilometres
    double lat1Rad = Math.toRadians(lat1);
    double lat2Rad = Math.toRadians(lat2);
    double diffLonRad = Math.toRadians(lon2 - lon1);

    double a = Math.sin(lat1Rad) * Math.sin(lat2Rad);
    double b = Math.cos(lat1Rad) * Math.cos(lat2Rad) * Math.cos(diffLonRad);
    double radDistance = Math.acos(a + b);
    double distance = radDistance * EARTH_RADIUS;
    return distance ;
}
```

Figure 22: Great circle implementation

5.2 Creating Boundaries

The creation of circular boundaries on the map presented another mathematical challenge when the user is about to create a “geofence”. The point the user selects on the map is considered the centre of the fence, the challenge was to correctly overlay the circle of the right radius on the map at any zoom level of the map. This was solved by recognizing that was really needed was the geographical coordinates of location at 0 or 180 degrees from the centre of the geofence. That means the longitude remains the same, but the offset on latitude is calculated and accounted for. Then with the Android provided projection function, these positions are then converted to pixel position on the screen, those are then used to draw the circle. The mathematical derivation of this is described thus.

If p, q represent two point on a sphere and the position of q is unknown
The length of the arc formed from p to q on the great circle joining them is,

$$|pq| = \frac{2\pi R}{360} \cdot t \qquad \ldots (1)$$

where t is the angle substended by the arc joining them
Hence,

$$t = \frac{360 \cdot |pq|}{2\pi R} \qquad \ldots (2)$$

Since t is the angle between them, the angle of q can be obtained by adding or subtracting t degree offset from that of p
As shown in the Figure 13 below, the angle formed by

$$x\hat{o}\,q = x\hat{o}\,p + \hat{t} \qquad \ldots (3)$$

where x is on the equator

$x\hat{o}\,p$ is the latitude of p

$x\hat{o}\,q$ is the latitude of q

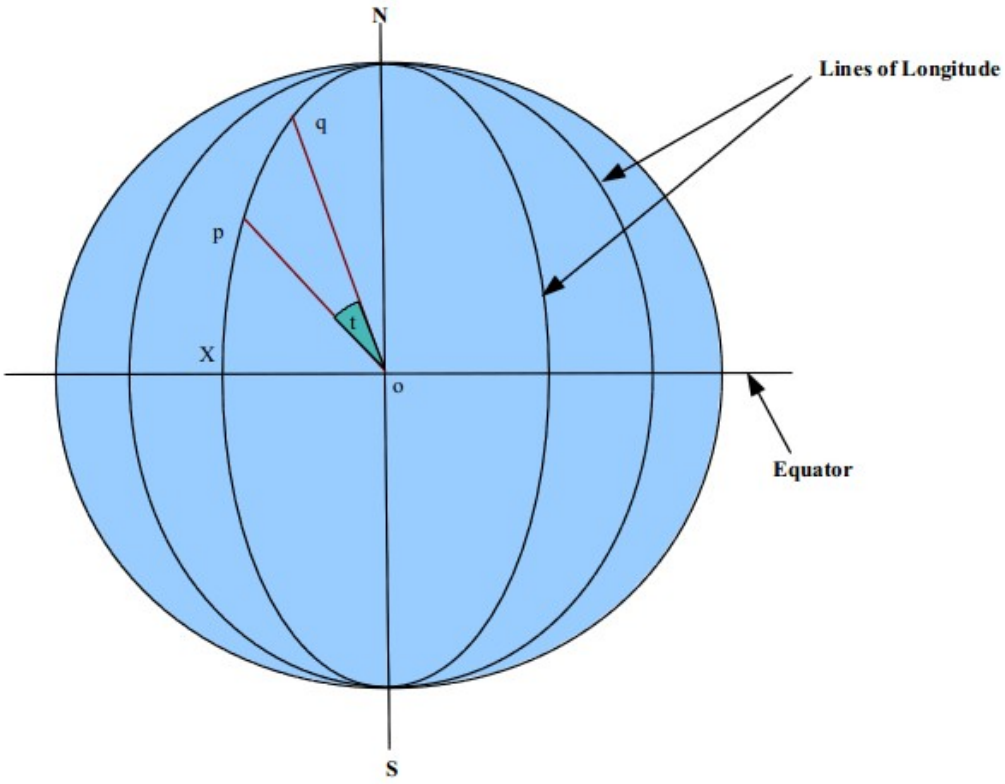

Figure 23: New Latitude point

It should be noted that this algorithm also assumes the earth to be a sphere.

The implementation in code of this derivation is presented in Figure 24 below.

```
private GeoPoint getOppositeGeoPoint(GeoPoint gp, double radius, boolean up) {
    GeoPoint newGeos = null;
    double radDistance = (360 * radius) / (2 * Math.PI * MapUtils.EARTH_RADIUS);
    double radLat = gp.getLatitudeE6() / 1E6;

    if (up) {
        int newlat = (int) ((radLat + radDistance) * 1E6);
        newGeos = new GeoPoint(newlat, gp.getLongitudeE6()); //longitude remains same

    } else {
        int newlat = (int) ((radLat - radDistance) * 1E6);
        newGeos = new GeoPoint(newlat, gp.getLongitudeE6());

    }

    return newGeos;
}
```

Figure 24: Obtaining new latitude on same longitude at some distance

5.3 Map Spanning

Markers placed on the application map may sometimes be placed too wide apart, hence the user may be left wondering where the other markers are located. Hence, sometimes it is necessary to span the map to fit all the markers on the map. The API provided function was not particular useful in achieving this result. The pseudo code shown in Figure 25 below describes the technique developed to achieve this.

```
begin
For Points p(x_i, y_i)
     y_max ← Maximum value of p(y_i...n)
     y_min ← Minimum value of p(y_i...n)
     x_max ← Maximum value of p(x_i...n)
     x_min ← Maximum value of p(x_i...n)
          where i = 0...n
     center point = ((y_max + y_min)/2) , ((x_max + x_min)/2)
     yspan = |y_max - y_min|
     xspan = |x_max - x_min|
end
```

Figure 25: spanning algorithm

Basically, what it does is to assume the earth as one big flat rectangle, and each

latitude and longitude pairs represent point y and x respectively on a Cartesian graph. The extremes of both axis is obtained, thus we obtain the smallest rectangle that can fit all the points. The map can then be made to span to the width and height of this rectangle from the centre of the rectangle.

5.4 UI Controls

The screen size of mobile phones limits the number of UI controls such as buttons, check-boxes that can be visible to the user at any particular point. This severely hinders accessibility to frequently used functions. Having exhausted the six menu items Android allows be visible at once [6], an alternative way of making the user access other important controls easily had to be found. It also had to be done without clogging the main map area with the controls.

The solution that was adopted was to make use of the sliding drawer which Android provides. The sliding drawer hides content out of the screen and allows the user to bring it back to the screen [6]. The content of the sliding drawer was then designed to be a layout that can house the needed UI controls. Figure 26 below is a screen-shot of the application when the sliding drawer is expanded revealing previously hidden controls and the image of the asset. The checkboxes in the slider toggles the visibility of the markers shown on the map and the button is used to launch another activity that shows all the information about the asset.

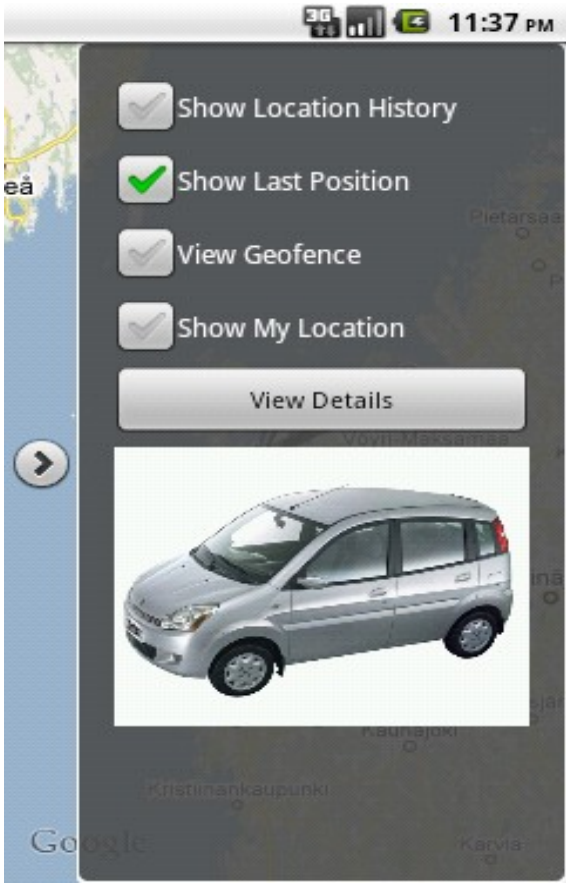

Figure 26: Slider exposing more controls

5.5 Data Security

Obfuscating data described in 4.3.3 in came with another challenge. A reliable source of key had to be found and it had to be done without the user being aware of it. To solve this problem, the key for the encryption is either supplied from a server each time the application is launched or from a unique identifier on the device. The first option can not be used in respect to this application as it is designed to be server-less. The second option is flawed because Android does not provide a stable, unique device identifier [15]. Hence if the application provides a unique identifier, then it is used, if not the application defaults to using the asset unique identifier to encrypt its data. Using the asset unique identifier is a better than nothing comprise because anyone with access to the source code can decrypt it.

6 CONCLUSION

This project shows that smart phones such as Android, can be used as an additional device that can be used for GPS telemetry with SMS as the medium of data exchange, therefore a better alternative to the current model of WRD asset tracking solution.

The challenges found in this project were more than anticipated when drawing-up the project plan. The amount of time needed to write out the thesis document, was particularly underestimated. In retrospect, writing the thesis document should have been included as a unit in the work cycle as part of the iterative Agile approach taken in the project development. A lot of time could have been saved there. Also the simulator used in testing and providing data to the application could have been used to expand the use of the application for tracking another Android mobile phone.

Like any software product or design, there is still room for improvement. Time has not permitted some of the requirements such as the use of Open-Maps, to be implemented. The use of Open-Maps will remove the license constraint attached to the use of Google maps [16] for enterprise application. Due to shortage of resources, testing has not been rigorous enough. While the application was thoroughly tested with the emulator and a personal Android device, the application still needs to be tested on more devices from the plethora of Android devices in the market, which vary on manufacturer, size, speed and version of the operating system. These are definitely source of future improvement if customer's satisfaction is to be guaranteed.

On the tracker side, the cost of maintenance may still be an issue. Consider, a user that chooses to receive constant updates at the longest interval of five hours for a whole month, with an average text message price of 0.069€ per piece [17], that means the user will incur a total of about

$$\frac{24\ hrs}{day} \times \frac{1\ sms}{5\text{hrs}} \times \frac{0.069\ €}{sms} \times \frac{30\ day}{month} \simeq 10\ € / month$$

For a constant update of ten minutes, the user will receive 4,320 messages which

will cost about 300€ a month. However, with telecommunications companies with bulk SMS offerings, the bill may be reduce to about 30€ a month. Therefore the inexpensiveness of the system is dependent on individual usage, frequency of use and the relative value of asset which is being tracked. Another improvement that could be made on the tracker is to add other sensor modules to measure other environmental parameters such as temperature, humidity and pressure, this can then widen the use of the tracking unit to other markets other than asset tracking.

Overall, the project has a substantial business value because it reduces hardware and maintenance cost and increases customer's satisfaction.

References

[1] Cellular News. Available on the Internet: <URL: http://www.cellular-news.com/story/29824.php>.

[2] WRD Systems Ltd Company Website. Available on the Internet: <URL: http://www.wrdsystems.com>.

[3] VisiRun Company website. Available on the Internet: <URL: http://www.visirun.com/>.

[4] Track Peers. Available on the Internet: <URL: http://www.trackpeers.com/>.

[5] BizSpeed Company website. Available on the Internet: <URL: http://www.bizspeed.com/>.

[6] Android developers. Available on the Internet: <URL: http://developer.android.com/guide/>.

[7] Sayed Y. Hashimi, Satya Komatineni. 2009. Pro Android. .

[8] BBC News. Available on the Internet: <URL: http://www.bbc.co.uk/news/technology-12481799/>.

[9] W.Frank Ableson, Charlie Collins, Robi Sen. 2008. Unlocking Android A Developer's Guide. Manning Publications.

[10] Marko Gargenta. 2011. Learning Android. O' Reily.

[11] Sentence description of NMEA. Available on the Internet: <URL: http://www.gpsinformation.org/dale/nmea.htm#RMC>.

[12] SQLite Official webpage. Available on the Internet: <URL: http://www.sqlite.org/about.html>.

[13] Jay A. Kreibich. 2010. Using SQLite. O' Reilly.

[14] Erik Hatcher, Steve Loughran. 2002. Java Development with Ant. Manning Publications.

[15] Android Official blogpost. Available on the Internet: <URL: http://android-developers.blogspot.com/2011/03/identifying-app-

installation>.

[16] Google Maps API. Available on the Internet: <URL: http://code.google.com/apis/maps/>.

[17] Dna Mobile Company. Available on the Internet: <URL: http://www.dna.fi/en/privatecustomers/mobilecommunication/priceplans/Sivut/>.