Rapport de Thèse Professionnelle

Développement logiciel distribué d'un service Internet of Things utilisant Qeo

Ochia Okechukwu Emmanuel

March 2015

Société : Technicolor Connected Home, Rennes Snc 975 av des Champs Blancs – CS

Encadrant dans l'entreprise(Industrial Supervisor) : Fabien BATTINI

Encadrant académique (Academic Supervisor) : Navid NIKAEIN

Thèse Confidentielle: Yes

Filière (Specialisation): Mobile Communications

Résumé

Développement logiciel distribué d'un service Internet of Things utilisant Qeo

Mot de cles: Internet of Things, Voiture Connecte, Qeo, OsmAnd, Android

Mon stage a été effectué à Technicolor Maison Connectée, Rennes entre le 26 Août 2014 et le 20 Février 2015. La portée de mon travail a été axé sur “le Développement logiciel distribué de services dans l'Internet des objets domaine” et en particulier dans “l'écosystème Voiture Connectée”.

Ces services exploitent la présence du protocole Qeo, développée par Technicolor et utilisé pour simplifier et unifier l'échange de données et d'informations entre différents objets et dispositifs intelligents quelles que soient les plate-formes et systèmes d'exploitation fonctionnant sur ces dispositifs.

J’ai participé aux réunions des parties prenantes ont porté sur la définition, la preuve des concepts et des cas d'utilisation de ces services qui devaient être mis en œuvre afin de tirer profit du protocole Qeo.
Dans un deuxième temps, J’ai implémenté ces services sur une application de navigation (OsmAnd) installé sur un dispositif de tablette Android. Mon développement a été fait en utilisant le langage de programmation JAVA, Eclipse IDE et Git.
Les résultats de mon stage se compose d'une démonstration à faire pour les acteurs de l'espace de Connected Car et le monde de l'internet des objets au sens large.

Abstract

Distributed Software Development of an Internet of Things Service Using Qeo

Keywords: Internet of Things, Connected Car, Qeo, OsmAnd,Android.

My internship was carried out at Technicolor, Rennes in the Connected Home Business Unit between 26th August 2014 and 20th February 2015. The scope of my work was on the Distributed Software Development of Services in the Internet of Things domain and specifically in the Connected Car Ecosystem.

These services leverage the presence of the Qeo framework, developed by Technicolor. Qeo simplifies and unifies the exchange of data and information among different smart objects and devices regardless of the platforms and operating systems running on these devices.

I participated in the stakeholder meetings centered on the definition, Proof of Concepts and use cases of these services which were to be implemented in order to take advantage of the Qeo protocol.

As a second phase, I continued on the work achieved by a previous intern who successfully embedded the Qeo framework service and its functions in the target software which is an Open Source navigation application called OsmAnd which is an acronym for ***O**pen**S**treet**M**ap **A**utomated **N**avigation **D**irections*).

I integrated these services into the navigation application targeted for initial installation, testing and usage on an Android tablet and eventually in a Car Navigation system. Development was done using the JAVA programming language, Eclipse IDE and Git for Source and Version Control Management.

The results of my internship will be built upon and eventually demonstrated to the actors and stakeholders in the Connected Car space and the world of the Internet of Things at large.

Acknowledgements

I would like to thank my supervisor, M. Fabien BATTINI for his warm reception and for his relentless support and guidance throughout the period of my internship, and also the Director of Technicolor R & D Snc France for accepting me as an intern in the business.

I thank Mr. Danny Vossen and the stakeholders at Technicolor Hq, Issy-les-Moulineaux for involving me early on in the meetings to define the use cases and service definitions of the project and by extension the Technicolor team in Edegem,Belgium who were also collaborators in this project.

I would like to also thank the entire Technicolor staff with whom I have worked with during the last six months for making it very easy for me to integrate into the company culture and for demonstrating availability and great attention to me throughout my internship.

Table of Contents

Introduction

The Internet of Things or Objects is an extension of “The Internet of Everything" which is based on the idea that devices, objects and systems can be connected in simple, transparent ways to enable seamless sharing of information among all of them. This level of interoperability can address everyday, real-life scenarios including home automation and remote sensing, business intelligence, transport systems, security and so forth. These can only be provided by a cross-industry effort which is needed to deliver new experiences to vast numbers of consumers ,businesses and enterprises.
Technicolor as a multinational company, is focused on delivery of innovative and cutting edge solutions to media creators and content distributors. It’s businesses span across three domains: Technology, Entertainment and Connected Home.

The Connected Home Business Unit develops and supplies hardware and software technologies to the Media and Entertainment Industry. It has a leading position in the market of set-top boxes and gateways worldwide.

This unit has launched several projects in the field of the Internet of Things which have been initiated based on the Qeo framework. Acting as a universal language, Qeo enables full interoperability between ecosystems, applications and devices of all brands and thus transcends the complexity and fragmented nature of the Internet of Things.
By breaking down the barriers of proprietary ecosystems, Qeo unleashes the potential of the Internet Of Things and makes it possible to offer end-users the most innovative and valuable usage scenarios.

The purpose of my internship was “the Distributed Development of an Internet of Things service” to capitalize on the benefits of the common Qeo Language. This was done in the context of the Connected Car Ecosystem. The OsmAnd project was at the centre of the development phases of my work. In the succeeding chapters, I will describe the Connected Car context, the OsmAnd project and the different services which were integrated into OsmAnd to provide more interesting and integrated options to the User. I used Eclipse IDE with Android SDKs for my development with JAVA as the language of choice.

I Company Overview

The company Technicolor, formerly Thomson, was formed on August 24 1985 and was registered on November 7 1985 for a term of 99 years expiring on November 6 2084.

Technicolor is a worldwide technology leader in the media and entertainment sector, and is at the forefront of digital innovation. Its world class research and innovation laboratories and a creative talent pool enable Technicolor to lead the market in delivering advanced services to content creators and distributors. Technicolor also benefits from an extensive intellectual property portfolio focused on imaging and sound technologies, supporting its licensing business.

I.1 Activities

Technicolor's activities are organized into three operating divisions:

- **Technology** includes several activities: Research & Innovation; Intellectual Property & Licensing; M-GO; Virdata etc.
    - The main objective of Research & Innovation is to develop and transfer innovative technology to support the services, software and solutions the Group provides.
    - The Licensing activity – which includes patent, technology and trademark Licensing - is responsible for protecting and monetizing the Group's Intellectual Property and technologies, while managing some iconic brands.
    - M-GO is a new platform aimed at making digital entertainment easier to find, watch, and enjoy.
    - Virdata extends both the lifetime and the applicability of traditional industrialization and automation enterprise applications such as asset management and asset tracking systems.

  2013 Revenues - €485 million representing 14% of total revenues.

- **Entertainment Services** develops and offers video-related technologies and services for the Media & Entertainment industry, notably the motion picture, broadcast and commercial advertising industries. This business is dedicated to delivering solutions for content management (including creation, imaging, finishing,

preparation) and for digital and physical content distribution (including DVD & Blu-ray™ services).

It also includes IZ-ON Media, which provides digital place-based media services

In 2013, Revenues - €1,618 million representing 47% of total revenues.

- **Connected Home** offers a wide range of solutions to Pay-TV operators and network service providers for the delivery of digital entertainment, data, voice, and smart home services. Through the design and supply of products such as set-top boxes, gateways, managed wireless tablets, Technicolor offers full connected life solutions. In 2013, the revenue generated by Connected Home was €1,346 million representing 39% of total revenues.

Technicolor is also supported by Corporate functions:

- Finance
- Human Resources
- Marketing & Communications

I.2 Technicolor Key Figures

- Employees at end of 2013: 14,000 employees in 25 countries
- 2013 key financial figures :
    - Group revenues: €3.45 billion
    - Adjusted EBITDA: €537 million
    - Free cash flow generation: €153 million

I.3 About Technicolor Rennes

Technicolor Rennes is the largest technology centre of Technicolor Group. The Rennes centre is the supplier of innovative solutions for Media & Entertainment Industries. A part of the activities is focused on the compression, protection, transmission, production and management of high definition contents while other technological solutions are developed to respond to the demands of network operators and to meet the needs of Media

Entertainment customers.

Technicolor Rennes has built partnerships with top academic institutes in Europe, public institutes, industrial partners and is involved in all French and European research programs such as RNRT, RNTL, RIAM, MEDEA, ITEA, IST, CNES, ESA …

I.4 Research and Development Themes

- Compression/Decompression with audio/video processing
- Signal and image acquisition and processing
- Mobile and broadband technology
- Content security on the whole image chain
- 3D
- Licensing
- Digital decoder platforms including the most recent technologies such as Personal Video Recorder, MPEG4 coding, Video over Internet Protocol, High Definition etc.

I.5 Key Figures (Rennes)

In December 2013, there are 517 employees in Rennes of which 85% are Engineers

II Internship Description

My internship at Technicolor was centred around the addition and implementation of integrated services into OsmAnd, a map and navigation application which is deployed and used on an Android tablet device and in future, in the navigation system of a car.
The phases of my internship work are categorised below:

- Definition of Services and Use Cases, Proof Of Concept and Market Value
- Study of the OsmAnd sources, dependencies and Helper Applications.
- Design of a suitable User Interface and Model for these Services.
- Development and Testing
- Presentations.

I attended a meeting in company of my supervisor and other stakeholders in the Connected Car Business. This was a workshop and brainstorming session aimed at defining new use cases and the associated services to aid these use cases, the target market as well as identifying the competition. This served as a benchmark to differentiate in the competition by implementing these services to provide additional market value.

These services were to be added on top of OsmAnd which is primarily used for navigation by a driver or pedestrian. By providing extra options to the user of OsmAnd in addition to the prime use for navigation, an added value could be realised.
I implemented these services through development using Eclipse which is an Integrated Development Environment that provides multiple tools to develop applications.

I made periodic presentations to my supervisor on the progress of my development activity and the final result is to be presented to the different stakeholders in the Connected Car project as well as partners alike.

III The Connected Car

III.1 Definition

A Connected car is a car that is equipped with Internet access, and usually also with a wireless local area network. This allows the car to share Internet access to other devices both inside and outside the vehicle. Often, the car is also equipped with special technologies that tap into the Internet access or wireless LAN and provide additional benefits to the driver. Examples include: automatic notification of crashes, notification of speeding and safety alerts.

Increasingly, Connected Cars (and especially electric cars) are taking advantage of the rise of smartphones, and applications are available to interact with the car from any distance. Users can unlock their cars, check the status of batteries on electric cars, find the location of the car, or remotely activate the climate control system.

III.2 Market Challenges

The Connected car concept is considered by the automotive industry as the main innovation for the next decade. In fact, the integration of a broadband connection and a local wireless network in vehicles, is becoming a default standard offering endless possibilities and many opportunities for manufacturers, operators, technology companies, content providers and different start-ups positioned in this market. The connectivity associated with the widespread use of sensors, analysis of real-time data through the cloud and the pursuit of technological innovations are redefining the automotive industry as well as the existing value chains.

This subject is complex because the concept of connectivity is plural. Connectivity means first and foremost a broadband Internet connection in the vehicle, it extends also to the Internet of Things and the concept of Machine to Machine that uses telecommunications and information technology to enable communication between a car and its environment. This turns the car into a communicating object and a new connected platform.

At the highest levels of politics this is already highlighted. In July 2012, the European Parliament adopted a draft resolution providing that, by the end of 2015, the new cars will be equipped with eCall devices to automatically alert relief services in the event of the occurrence of road accidents via the 112 emergency call.

Finally, all these reflections bring a whole new vision of the car. Making a “Connected car” means that radical changes need to be made regarding the hardware and software architecture of electronic cars. Also within a connected environment, there is the need to have “water-proof” security to safeguard against potential hackers. Imagine someone deciding to take action against an individual through his/her carrier, for example by disabling the braking system which can lead to disastrous consequences. Thus, the giants of Internet security like McAfee and other security firms have a key role to play in engineering secure vehicle electronics and software.

IV Qeo and AllSeen

IV.1 Qeo Service

Qeo is a simple, secure and future proof communication framework developed by Technicolor that enables a simple and user-friendly "Connected Life" experience, enabling consumers to enjoy the full potential of their devices and services.

Qeo brings a new world of exciting possibilities to different ecosystems because it enables the easy sharing of data and information between the cloud, multiple accessories and infrastructure of which the Connected car can also benefit from, e.g access to home devices while away, remote monitoring of the home and so forth. Qeo is also agnostic to the operating systems running on these devices and so it is easy to integrate into all these platforms.

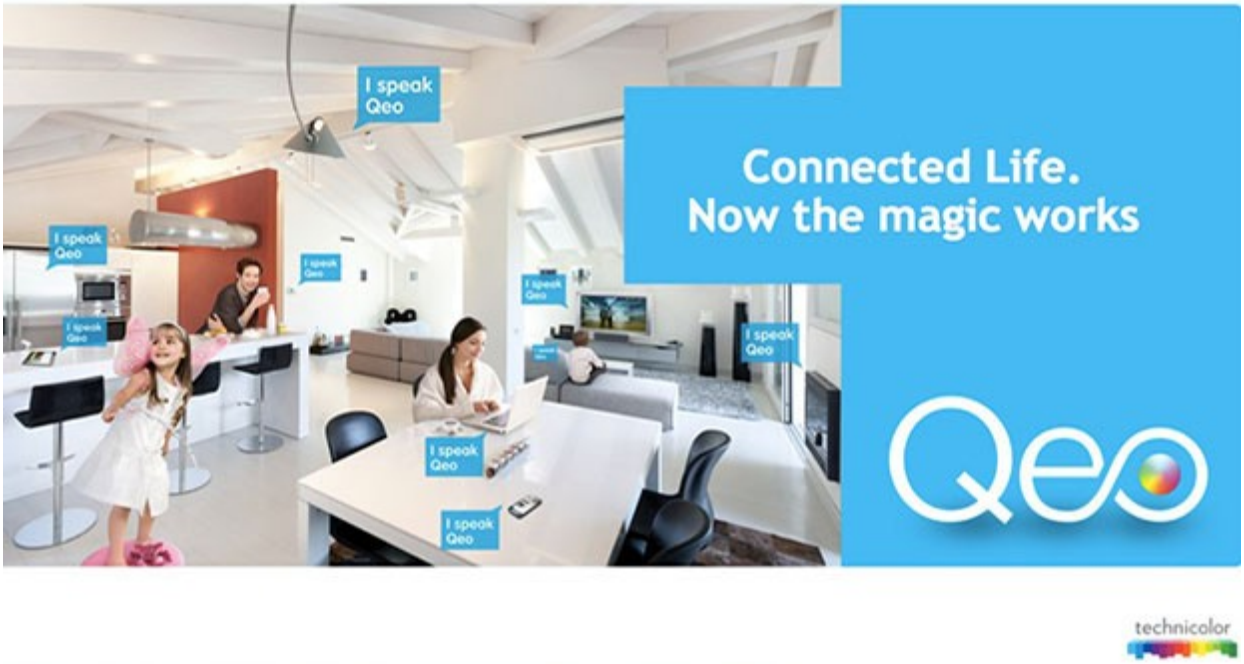

Figure 1: The Connected Life experience powered by Qeo.

IV.2 AllSeen

The AllSeen Alliance is the broadest cross-industry consortium to date to advance adoption and innovation in the “Internet of Everything” in homes and industry.
The Alliance, formed in December 2013, expands upon the “Internet of Things” to include more functionality and interactions across various brands and sectors, such as the Connected home, healthcare, education, automotive and enterprise. The members of the AllSeen Alliance contribute software and engineering resources as part of their collaboration that builds on the open source “AllJoyn” software and services framework, originally developed by Qualcomm Innovation Center, Inc. Technicolor being a Premier-member of this alliance has brought the power of Qeo to augment its capabilities by enabling a more data-centric publish/subscribe set of application programming interfaces (API). By combining Qeo’s APIs and secure architecture with AllJoyn’s software and services framework for discovery, connectivity and interoperable services, developers and device makers have even more options and even greater flexibility in how they

implement a fully distributed Internet of Everything.

V The OsmAnd Project

V.1 Background

OsmAnd (OSM Automated Navigation Directions) is a map and navigation application with access to the free and worldwide OpenStreetMap (OSM) data. All map data can be stored on a device's memory card for offline use. OsmAnd offers routing, with visual and voice guidance, for car, bike, and pedestrian. All the main functionalities work both online and offline.

As a navigation application, OsmAnd was the preferred choice for development purposes because it has many characteristics which include:

Open source: OsmAnd is an open source application that allows access to the source code and thus allows the code to be updated and edited to add more features and services to generate more market value.

Free: There is a free version which can be downloaded from Google Play. This facilitates the marketing of services developed around this application.

Offline Use: Once the maps are downloaded, OsmAnd does not need an Internet connection to work. This is a very important criterion for usage of services not needing an Internet connection.

Simplicity and documentation: The OsmAnd source code is well structured and documented.

This facilitated the understanding of the architecture of the application and allowed easier identification of the functions necessary for the project. Also, the documentation which is readily accessible on the Internet was an asset in the detection and correction of errors at different phases of the project.

I used a modified version of OsmAnd which is based on earlier work done by a previous intern on the OsmAnd source code. This version contains the Qeo framework service integrated into it. This Qeo service embedded in OsmAnd allows users to share their GPS positions in real time and integrate these positions on a map. It also offers a Dialog Interface function on the map enabling more User interactivity, awareness and sharing of useful information among users registered in the same Qeo realm. Since this service provides communication among different users, this gives rise to the emergence of more

marketable services and use cases in the automotive ecosystem.

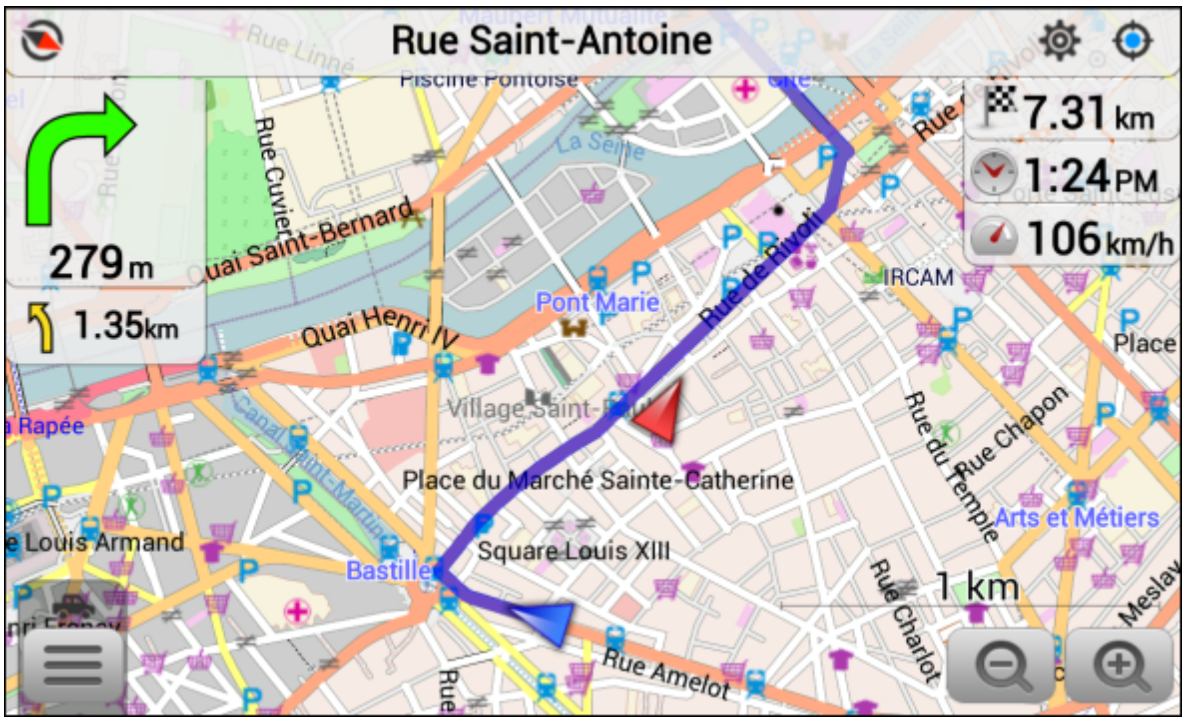

Figure 2 : The Osmand Map Interface.

V.2 OsmAnd Build and Compilation

After downloading the modified OsmAnd source code from its remote repository, there was the need to build it.The project was saved in a convenient location in the file system. The Eclipse IDE integrated with Android SDK tools, Build Tools and Platform tools provides an easy alternative for development, building, compiling, debugging and running android projects without resorting to command line tools like Javac and Apache ANT.

It provides a rich GUI with several view perspectives and a good hierarchical view of the project sources, resources and various sub-directories.

After saving the project source to a location on disk, I imported it into Eclipse and added it to the working directory.

On inspection of OsmAnd, I found that It had some libraries and dependencies which include the OsmAnd-java package and the SherlockBar package.

Hence for the project to be built successfully, some additional steps needed to be taken:

While in Eclipse;

- Right-Click on OsmAnd Project folder:
- Navigate to Build-Path> Configure Build Path > Java Build Path
- Navigate to the Project Tab, Add “OsmAnd-java” and “SherlockBar” to Required projects on the build path.
- In the Libraries section, add the Sherlockbar JAR file to the Android Dependencies folder of the project
- Navigate to the Android tab in the “Configure Build Path” window, Add SherlockBar as a Reference library to OsmAnd.

- Navigate to the SherlockBar package itself, right click on it:
    - Go to Build-Path> Configure Build Path > Android
    - Verify that the “Is Library” box is checked.
    - Go to OsmAnd project, right click and ‘Build’.

The above steps were taken in order to successfully build OsmAnd. It was then installed on the target android tablet device for initial inspection and for making a demonstration of the use cases.

V.3 The Pick Me Up Use Case

The Pick-Me-Up scenario is within the Connected Car context. This was studied to provide the background for further development of Services.

In this use case, remote devices can communicate with each other. One device being used by a driver for instance a father and the other device used by a remote person like the daughter for instance. The presence of Qeo and other helper applications enable the father and daughter to exchange the positions of their locations and can automatically route the father to the location of the daughter if necessary. The steps taken in achieving this demo are discussed below.

The different application packages were installed on the corresponding devices for the purpose of running the demo. These include:

- OsmAnd - The android navigation and map application which is installed on the tablet and two smart phones.

- FakeGPS - A standard android application that simulates GPS on two smart-phones. It provides “fake” positions for the driver and the passenger
- QeoFakeGPS- an android Qeo-based service that runs on the tablet only. It will provide the position of the car out from OpenDS.
- Qeo Service- The Qeo service that interfaces between each device and registers them in the same realm for direct communication even without Internet connectivity
- Qeo Management- Management service for Qeo.
- QFinder- Helper application to verify that Qeo service works correctly
- Connected Car- an android Qeo-based service that runs on the 2 smartphones. It is used to exchange the car positions.
- OpenDS- Application that simulates 3D navigation on a PC. It uses perl scripts and other config files for navigation scenarios. Alternatively, the Android tablet can be used to simulate the Car Navigation System.

In this demo, there are 3 devices; 1 smart phone used by the driver, 1 smartphone used by the daughter and 1 tablet used by the driver during navigation. On the OsmAnd Map UI, when the car engine is started by pushing the simulated START button, the driver can see his GPS position displayed on the map. This activates the communication between OsmAnd and Qeo. At the same time, the button colour becomes Orange and displays STOP.

The color of the Qeo logo in the top left angle changes to blue. Clicking this logo starts or stops the OsmAnd-Qeo interface. The OpenDS navigation simulator was used to simulate navigation. When OpenDS is started on the PC, the car position is sent to the tablet, so should be visible there. Typing key “E” on the PC starts the engine. At the same time, the driver phone should go into “Connected Car mode”(a dark screen with a logo). In this mode, the driver will not be able to use his phone; any attempt to use it will fail- the screen will return to a dark mode.

At this point, the position of the daughter is visible on the Map and the Qeo logo should be blue. The “Favorite” position displayed on the Map can be long clicked and the “Directions To” also clicked. Two things happen after this is done.

1-A trip notification is sent to the Daughter’s phone

2- The arrow starts to move on the navigation screen of the tablet

The daughter can then click on the notification received and click “Yes” and then the driver

is automatically rerouted to navigate to the daughter's position. A trip notification is sent to the daughter's phone and she clicks "Map". She can then see the position of the father getting closer to her location.

The father can click on the STOP engine button upon arrival. His phone now goes out from the "Connected Car" mode. The OsmAnd application opens on the driver's phone and a "Favorite" is set (the last location of the car). A parking notification is also received on the father's phone.

VI OsmAnd Services

VI.1 Scope Of Services

An investigation into more interesting services was launched. The scope of these services was to cover "Entertainment", "Home Presence" and "Security".

These services were to be added to the already existing base functionalities of OsmAnd using a suitable and undisruptive model.

VI.2 Technical Requirements

In integrating these services, some target requirements had to be met, taking into account the constraints of OsmAnd. These are discussed below.

- These services are to be an extension to the base functions of OsmAnd. OsmAnd's main interface is a map of a region, country or the world and it is used primarily for navigation. This function should be maintained and not overridden by the presence of these services. They will be present to add more interesting options to the user.
- The services should be connected to the cloud to enable remote updating of their content. The presence of Qeo framework on the device and in the cloud will implement this function since it will enable secure and real-time data exchange between the cloud infrastructure(a server or similar database repository) and the device and provide secure device to device communication as well.

Building upon these generic requirements, the technical requirements for these services are described below:

- The new UI elements will be layered on top of the map and the layer should be transparent so that the portion of the map interface underneath the layer is visible

and also accessible to the user.

- The new UI elements will be responsive to user generated events such as clicking, dragging etc and can also pass such events not intended for the elements to the OsmAnd map interface, i.e transparent to events directed at OsmAnd.
- The new UI elements can be easily updated.
- The new UI elements will be able to communicate with OsmAnd such as:
    - receiving notifications from OsmAnd and sending messages to OsmAnd
    - executing OsmAnd native code.

Further development in OsmAnd was done to reflect these requirements.

VI.3 Music Service

VI.3.1 Background

The purpose of this service is to provide the option of playing an audio track or song saved in the device or tablet media storage system while simultaneously navigating with the OsmAnd map.

Historically, this was not initially present because the user had to exit the map application to be able to use a media playing application to play audio files. This service was to merge these two functions; navigating with the map and interacting with a music player at the same time.

I studied the Android FrameLayout Application Programming Interface to understand its properties and attributes because this is the parent Layout upon which the OsmAnd map is structured.

I customised a space within the OsmAnd screen which will host the Music player UI.

I integrated a “ListView” widget into this space and provided playback control buttons(play, pause, previous, next etc) for user action. The purpose of this ListView was to hold the list of songs on the device and also provide play back controls for user action.

In this new configuration, one-third of the screen hosted the ListView and its playback controls which were positioned on the left side of the screen while the remaining two-thirds of the screen held the Map view and was positioned on the right. The xml layout was designed to support both landscape and portrait configurations to enable seamless resizing when the tablet switches between landscape and portrait orientations

during run time.

After adding the music player, I had to take into account that the music player has states which needed to be managed correctly and that there are multiple applications that can potentially use the Android Music Service stream while the music player is using it so I had to manage this in order to prevent application crashes and exceptions during run-time.

VI.3.2 The Audio Manager and Audio Stream

The Audio Manager is the main class that manages audio sources and audio output on an android device. In implementing the music player, I made use of APIs provided by this class for the music player to perform typical operations such as to request “Audiofocus” which I will describe in a subsequent section.

The Audio Stream provides a predictable audio experience to the user of the music player. Android maintains a separate audio stream for playing music, alarms, notifications, the incoming call ringer, system sounds, in-call volume, and DTMF tones. This is done primarily to allow users to control the volume of each stream independently.

The other stream types are mainly restricted to system events so media playing applications typically use the STREAM_MUSIC stream.

VI.3.3 Volume Control with Hardware Buttons

Having identified the Audio stream,I set it as the volume stream target. This was done very early in the MapActivity’s OnCreate() because there is only a need to call it once during its lifecycle, This was to ensure that whenever the OsmAnd application is visible, the volume controls function as expected.

```
setVolumeControlStream(AudioManager.STREAM_MUSIC);
```

VI.3.4 Handling PlayBack

The MediaPlayer class is the primary API for playing audio and video in android. It is an important component of the media framework. I used an object of this class to fetch, decode and play audio files from the device storage system.

In order to retrieve songs and handle playback, I added a Song class to the OsmAnd project. I used this class to model the data for a single audio file. It contains instance variables for important data for each track such as the name of the artist, title of the song and so forth. I added constructor methods to instantiate these variables and get methods to return the instance variables.

```
public class Song{
	// Instance variables for the data to be stored for each track
	private long id;
	private String title;
	private String artist;
	// Constructor to instantiate the  variables
	public Song(long songID, String songTitle, String songArtist) {
		  id=songID;
		  title=songTitle;
		  artist=songArtist;
		}
	// get methods for the instance variables;
	public long getID(){
		return id;
		}
	public String getTitle(){
		return title;
		}
	public String getArtist(){
		return artist;
		}
}
```

I provided a way for the songs to be stored in an ArrayList which will eventually be mapped to the ListView using an Adapter Instance.

```
ArrayList<Song> songList;
ListView songView;
```

I also created a layout in the xml subdirectory which is contained in the resources folder of the OsmAnd project. It contains a ListView widget that provides the UI for the song list.

In the OnCreate method of the Map activity,I retrieved the ListView layout using its specified id.

This enabled the MapActivity to display the ListView which will host the list of songs.

In the song.xml layout;

```
<ListView
    android:id = "@+id/song_list"
…/>
```

In the MapActivity's OnCreate();

```
songView = (ListView)findViewById(R.id.song_list)
```

I instantiated the songList as an instance of an ArrayList which will hold the songs.

```
songList = new ArrayList<Song>();
```

I created a helper method whose function is to retrieve the list of songs

```
public void getSongList() {
  //retrieve song info
}
```

This method can be called to get the List of tracks. Within this method, I made use of a ContentResolver and a Cursor object to retrieve audio files from the device,

Below is the implementation of the ContentResolver and Cursor instances used to query for songs on the device. The Cursor instance takes a URI(Uniform Resource Identifier) as one of its parameters. This parameter points to the media storage system on the device

```
ContentResolver musicResolver = getContentResolver();
Uri musicUri = android.provider.MediaStore.Audio.Media.EXTERNAL_CONTENT_URI;
Cursor musicCursor = musicResolver.query(musicUri, null, null, null, null);
```

I iterated over the results of the query to check that there is valid data.

```
if(musicCursor!=null && musicCursor.moveToFirst()){
   //get columns
    int titleColumn = musicCursor.getColumnIndex
    (android.provider.MediaStore.Audio.Media.TITLE);
    int idColumn = musicCursor.getColumnIndex
    (android.provider.MediaStore.Audio.Media._ID);
    int artistColumn = musicCursor.getColumnIndex
(android.provider.MediaStore.Audio.Media.ARTIST);
       //add songs to list
     do {
        long thisId = musicCursor.getLong(idColumn);
        String thisTitle = musicCursor.getString(titleColumn);
        String thisArtist = musicCursor.getString(artistColumn);
        songList.add(new Song(thisId, thisTitle, thisArtist));
        }
   while (musicCursor.moveToNext());
}
```

VI.3.5 Displaying Audio Tracks

After calling the helper getSongList() method in the OnCreate(), I provided a means to sort the data so that the songs are presented alphabetically:

```
Collections.sort(songList, new Comparator<Song>(){
    public int compare(Song a, Song b){
       return a.getTitle().compareTo(b.getTitle());
    }
});
```

VI.3.6 Mapping The List Of Tracks

I created a SongAdapter class to extend the BaseAdapter as its superclass. Basically, this class maps the list of songs from the ArrayList into the ListView.

```
public class SongAdapter extends BaseAdapter {

    private ArrayList<Song> songs;
    private LayoutInflater songInf;

    public SongAdapter(Context c, ArrayList<Song> theSongs){
       songs=theSongs;
       songInf=LayoutInflater.from(c);
    }

    @Override
    public int getCount() {

        return songs.size();
    }
    @Override

    public View getView(int position, View convertView, ViewGroup parent) {
     //map to song layout
    LinearLayout songLay = (LinearLayout)songInf.inflate
      (R.layout.song, parent, false);

    //get title and artist views
    TextView songView = (TextView)songLay.findViewById(R.id.song_title);
    TextView artistView = (TextView)songLay.findViewById(R.id.song_artist);

    //get song using position
    Song currSong = songs.get(position);
   //get title and artist strings
    songView.setText(currSong.getTitle());
    artistView.setText(currSong.getArtist());
   //set position as tag
   songLay.setTag(position);
   return songLay;
   }
...
}
```

In the MapActivity's onCreate, after sorting the List, the Adapter is called and is set on the ListView instance

```
Collections.sort(songList, new Comparator<Song>(){
  public int compare(Song a, Song b){
    return a.getTitle().compareTo(b.getTitle());
  }
});
….

SongAdapter songAdt = new SongAdapter(this, songList);
songView.setAdapter(songAdt);
```

VI.3.7 AudioFocus

I investigated the Android Audio System APIs and I implemented Audio Focus.

The idea behind this is to ensure that no other service or application uses the Audio stream when it is being used by the music player.

The music player when in operation, requests for Audio Focus using the Audio Manager APIs. It is then granted focus and has sole use of the Audio Manager stream while it is active.

I have presented a snippet of the code used in achieving this. The audio focus is requested immediately before playback begins, such as when the user presses the play button.

```
AudioManager am = mContext.getSystemService(Context.AUDIO_SERVICE);
// Request audio focus for playback
int result = am.requestAudioFocus(afChangeListener,
                                  // Use the music stream.
                          AudioManager.STREAM_MUSIC,
                                  // Request permanent focus
                                  AudioManager.AUDIOFOCUS_GAIN);


if (result == AudioManager.AUDIOFOCUS_REQUEST_GRANTED) {
    am.registerMediaButtonEventReceiver(RemoteControlReceiver);
```

```
    // Start playback.
}
// Abandon audio focus when playback completes
am.abandonAudioFocus(afChangeListener);
```

The AudioManager class handles the AudioFocus function. The Music player can request for audio focus, that is for sole use of the AudioManager Music stream during playback by calling requestAudioFocus() which returns AUDIOFOCUS_REQUEST_GRANTED if the request is successful.

The parameter “AudioManager.AUDIOFOCUS_GAIN” handles the request for permanent focus.

When the music player stops playing or its service is killed by the system after the player disconnects from it, it abandons focus and drops its use of the Audio Manager stream. If on the other hand, another application for instance an incoming call requests use of the audio stream, it is abandons focus temporarily and remains in a transient state by lowering its volume while the user answers the call. After the call is dropped, it resumes control of the Audio Manager stream.

The following snippet illustrates the audio focus implementation.

```
int result = am.requestAudioFocus(afChangeListener,
                         // Use the music stream.
                          AudioManager.STREAM_MUSIC,
        // Request permanent focus, lower volume if focus is lost temporarily.
                          AudioManager.AUDIOFOCUS_LOSS_TRANSIENT_MAY_DUCK);
```

Here the parameter, AUDIOFOCUS_LOSS_TRANSIENT_MAY_DUCK allows the music player to lower its output volume if there is a temporary loss of the music stream due to a higher process occurring in the Android system and requiring the stream.

VI.3.8 Repositioning OsmAnd MapView Control Objects

In this phase, I changed the positions of some map control buttons in order for them not to obstruct the UI of the music player.

I examined the classes, methods and png image files holding the properties of the map objects like the “Qeo” logo, Satellite logo and the menu button in the OsmAnd source code. All the map controls properties and methods were located in a MapControlsLayer class.

Prior to repositioning, they were situated on the left side of the screen where the music player is positioned. This was not ideal because they obstructed the music player song list. I changed their positions on the map interface from right to left by changing their layout attributes and specifically, the layout gravity attribute and method parameters which specify their relative positions on the screen with respect to their parent views.

As an example of the repositioning of the Menu button, I have shown a code snippet below.

```
private void initBackToMenuButton(final OsmandMapTileView view, FrameLayout
parent) {
            android.widget.FrameLayout.LayoutParams params;
            Context ctx = view.getContext();

             backToMenuButton = new Button(ctx);

backToMenuButton.setContentDescription(ctx.getString(R.string.backToMenu));
            backToMenuButton.setBackgroundResource(R.drawable.map_btn_menu);
             params = new FrameLayout.LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT, Gravity.BOTTOM | Gravity.LEFT);
            parent.addView(backToMenuButton, params);
            backToMenuButton.setEnabled(true)
}
```

In the menu button initialization method above, there is a Gravity parameter which the FrameLayout.LayoutParams class instance takes. It controls the position of the menu

button within it's parent View which is the FrameLayout that hosts the MapView itself. I altered this parameter to place the button in a new location.

VI.3.9 User Experience

I made an effort to improve the user experience of the Music Player and make it more appealing.

I drew up a checklist of requirements to measure the performance of the music player and ensure it works in a predictable manner.

These include:

- Each Music Player Track has the title and artist information displayed correctly.
- The information for each track is displayed in Human readable characters
- Every List Item of the player has the same dimensions(width/height) i.e. it has a consistent UI.
- The title and artist text each appear on one line.
- If there is an overhead in text length, it is truncated to one line and ellipsis added to indicate continuation.
- A User can scroll through the list of songs seamlessly.
- Color for each state of the music player song (default, pressed, active) okay.
- Music Player control buttons customized within the bounds of the Music Player UI.
- Control buttons: 'Play', 'Pause', 'Forward', 'Rewind', 'Next', 'Previous' all working correctly. Toggling between Play and Pause also functional.

VI.4 Music Player Architecture

VI.4.1 MusicService and MapActivity Interaction.

I will now describe the architecture of the different components of the audio playing service and how they interact within OsmAnd to make the music player function appropriately.

The "MapActivity" is the main activity of the application. It hosts the Map which has "OsmandMapTileView" as its associated xml layout.

I added a MusicService Class which extends the Service class as its superclass. This controls the playback functions of the music player.

From an Android perspective, a Service is an application component that can perform long running processes or operations in the background and does not provide a user interface. Another application component like an activity, can bind to a service and interact with it.

In the case of the MusicPlayer, playback is started by using the MediaPlayer class instantiated within the MusicService class. The MusicService class executes playback continuously even when the user is not directly interacting with it or when using the Map Interface. It also adds user control functionality which the user calls within the MusicPlayer UI to control its various states. The MusicService is started within the MapActivity's OnCreate() method after which the player is initialized in an initMusicPlayer() method where its properties are set.

The playback methods are also implemented in the Music Service.

playSong() is a method that allows a song to be played. Within this method the song is retrieved by setting a URI as the data source for the MediaPlayer. However since an exception may be thrown as a result of an error or an invalid data source, a try/catch block is used to test for validity of the data source.

After this try/catch block, a new thread for the MediaPlayer is spawned using the prepareAsync() method. This is to keep it separate from the Application's main thread and avoid runtime errors like ANR(Application Not Responding)

onPrepared() method starts the player and provides a notification for the current player state e.g. current song playing, name of artist and so forth.

onError() resets the player when there is a system error.

onCompleted() is called when a track has finished playing. It resets the player and plays the next track.

I set various Listeners to handle the different player states. The Music Service is able to control playback by implementing these listener interfaces which include:

OnPreparedListener- to be notified when the player preparation is complete and the user can start playing.

OnErrorListener-for Error management.

OnCompletedListener -to signal completion of playback.

These interfaces aid the process of interacting with the media player. They also implement accompanying methods which perform their control functions.

The list of songs is passed from the MapActivity to the MusicService. This forms part of the interaction between the MapActivity and the Music Service class. A MusicBinder instance serves as the Interface binding the MapActivity and Service classes.

VI.4.2 MusicBinder and IBinder

Essentially, the music is going to be played in the MusicService class and controlled in the MapActivity class which provides the Music Player's User Interface. To accomplish this the MapActivity class has to bind to the MusicService class . A MusicBinder is instantiated within an onServiceConnected() method in the MapActivity class. The service is retrieved by the Binder and the songList array is passed into it. When the connection to the service is lost the onServiceDisconnected() method is called.

The Service will also be started by the MapActivity by overriding the onStart() method, declaring an Intent object within the method, binding to it and passing the intent as an argument to the startService() method. When the connection to the intent is made, the list of songs is passed to it making it possible to interact with the Service instance in order to control playback.

A MusicBinder class gets the MusicService class and returns it to start the binding process.

The binding is completed by an IBinder onBind() method which returns the musicbinder instance. An onUnbind() releases resources when the Service is unbound i.e when the player is stopped and released.

VI.4.3 Song

This class is used to model the data for a single audio file. These data include the id of the song, its title and the name of the artist.

I provided a constructor method to instantiate a song's data and I added methods to retrieve each of these data.

VI.4.4 SongAdapter

An Adapter acts as a bridge between a ListView and the data that backs the List which usually comes from a Cursor. The ListView is able to display any data that is wrapped with a ListAdapter.

I created a SongAdapter Class which extends the BaseAdapter superclass to provide this mapping function i.e. mapping the songList to the ListView. In this class, the variable for the songList is declared as an ArrayList to contain the songs and I declared a Layout Inflater which will inflate this List.

The song list is passed from the MapActivity class and the LayoutInflater is used to map the title and artist strings to the Textviews in the song layout which is an xml file that I added to the project's resources folder.

After the song variables are instantiated, I then retrieved them in a constructor method.

The title and artist text are set by retrieving the correct instance from the list using its position index and mapping these strings to the views which were added to the song layout file. The position is also set as the view tag, which will let a user play the correct song when the user clicks an item in the list. The song layout xml file contains an onClick attribute which is used in the MapActivity class to retrieve the tag.

Within the MapActivity, the songs are stored in a list and displayed in a ListView. In the onCreate() method, the songView is retrieved using the id given to it in the main layout file. The setAdapter() is called which populates the ListView with the song information.

The above classes, methods and interfaces which I have discussed in the previous sections, interact to provide a functioning Music player within the OsmAnd application.

I have highlighted the architectural components of the MusicPlayer in the succeeding figures. They show the classes, their principal methods and the interactions which enable the Music Player to function.

Figure 3: The MusicService class hierarchy showing its main methods and an inner MusicBinder instance that gets this service.

Figure 4: The MapActivity class.This is the main activity of OsmAnd that bounds to the MusicService class.

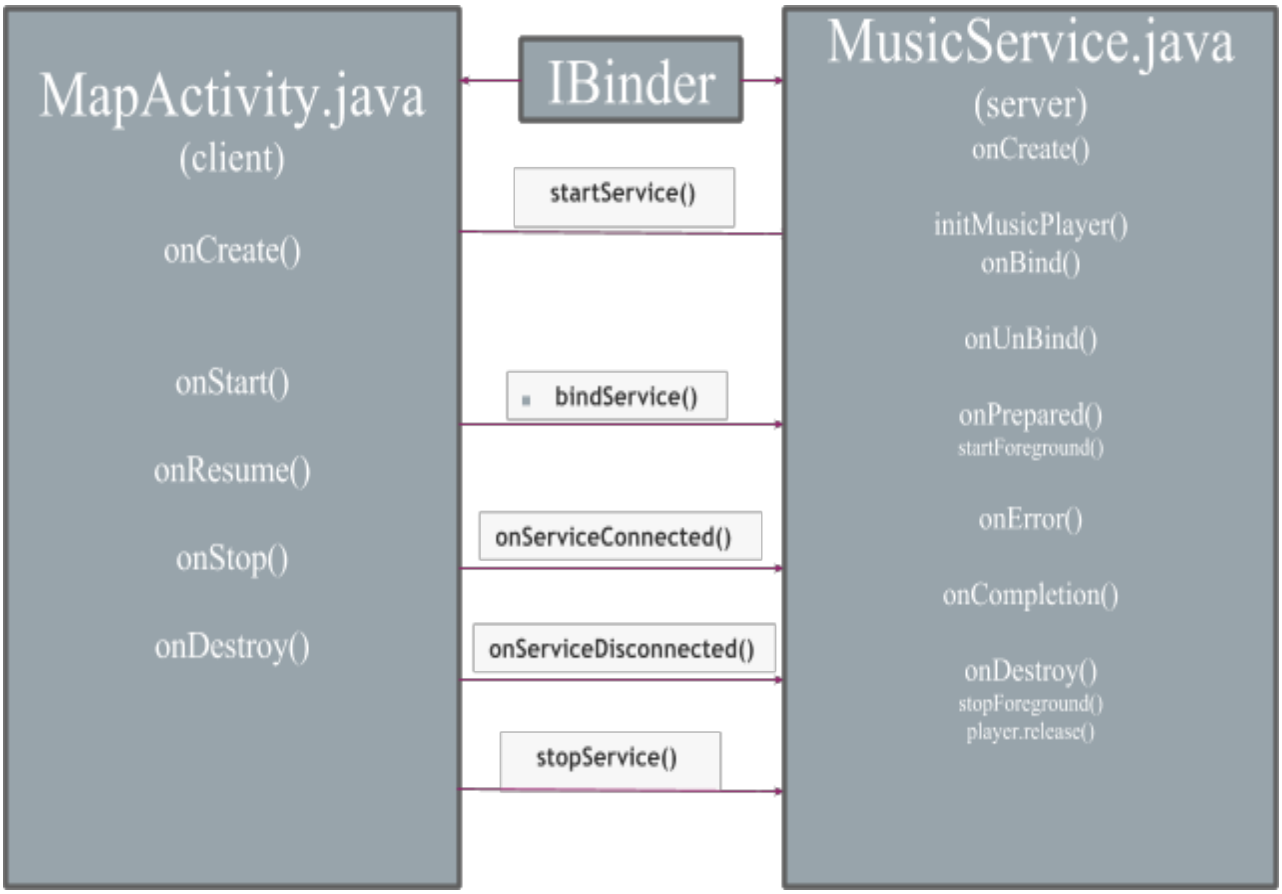

Figure 5: Client-Server Interaction between the MapActivity class and the MusicService class facilitated by an IBinder Interface.

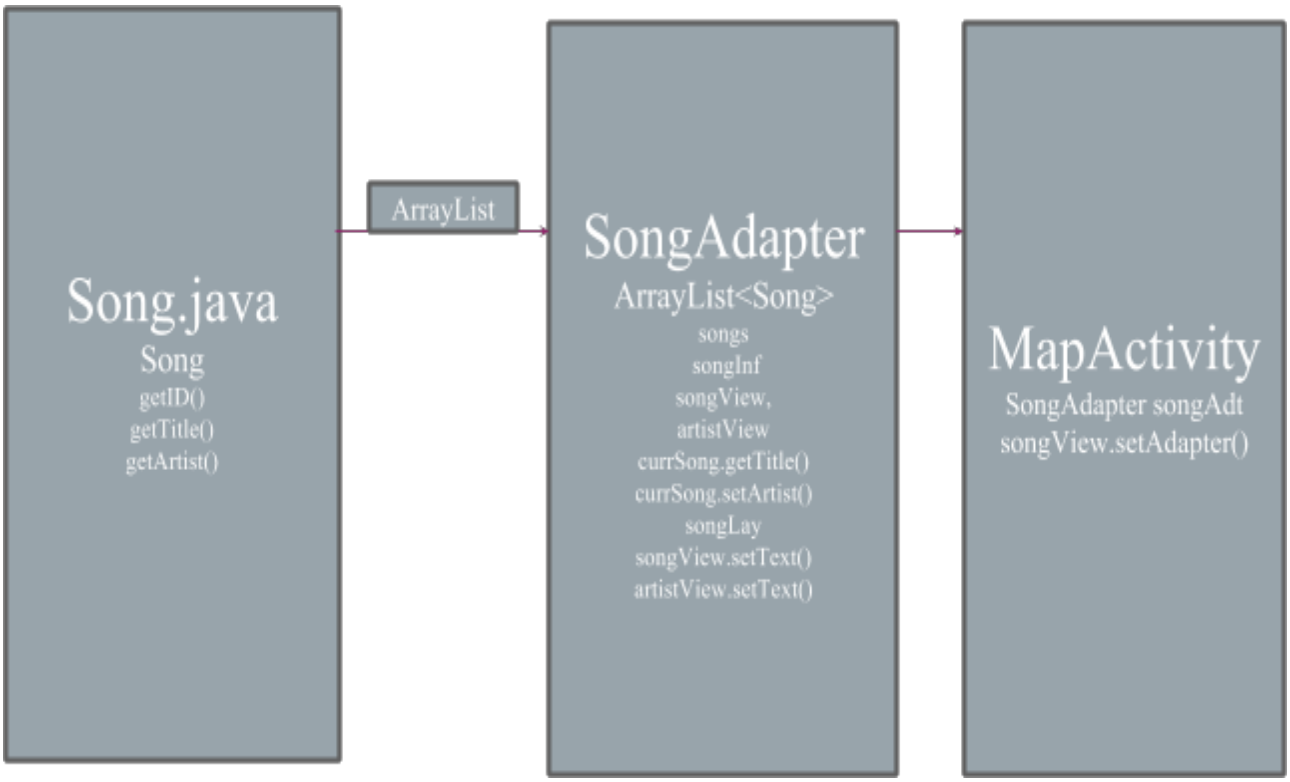

Figure 6: Flow Of Control between the Song, SongAdapter and MapActivity Classes.

VI.5 Design Model for Other Services

Up to this point, I examined and tested the OsmAnd Map application to verify that it's UI components display correctly for a large screen size and pixel density as well as in landscape mode on a tablet device since this is the target device for using OsmAnd.

I contacted the SCM team at Technicolor to request for access to its git servers to enable me push the stages of my work to the Connected Car repository which is hosted on a remote server. In order to get access, I used Putty-gen software to generate an SSH-2 RSA key pair (private and public keys). I exported the private key and saved it in a suitable location. I also saved the public key and sent it to the SCM for integration into the public keys directory.

After pushing the initial stage of my work, I launched an investigation into the design model for other services which were to be integrated into the OsmAnd User Interface.I examined a couple of design options, taking into account some factors such as ease of access to these services, minimal driver distraction and accessibility

of the Map View while using these services. These options were narrowed down to three and I discussed with my supervisor to select the most optimal option.

VI.5.1 First Design Option

The First idea was to have a button positioned strategically on the OsmAnd UI. This will provide a means to access each service. It will be the entry point and interface through which a user can access any of the services available. When a user/driver clicks on it, an enlarged UI will be displayed. This UI will provide a list of options available and the user can then navigate to the service of interest, while still within the Map Application. Also, if necessary, a notification can be provided to the user through this button as a text or voice message.

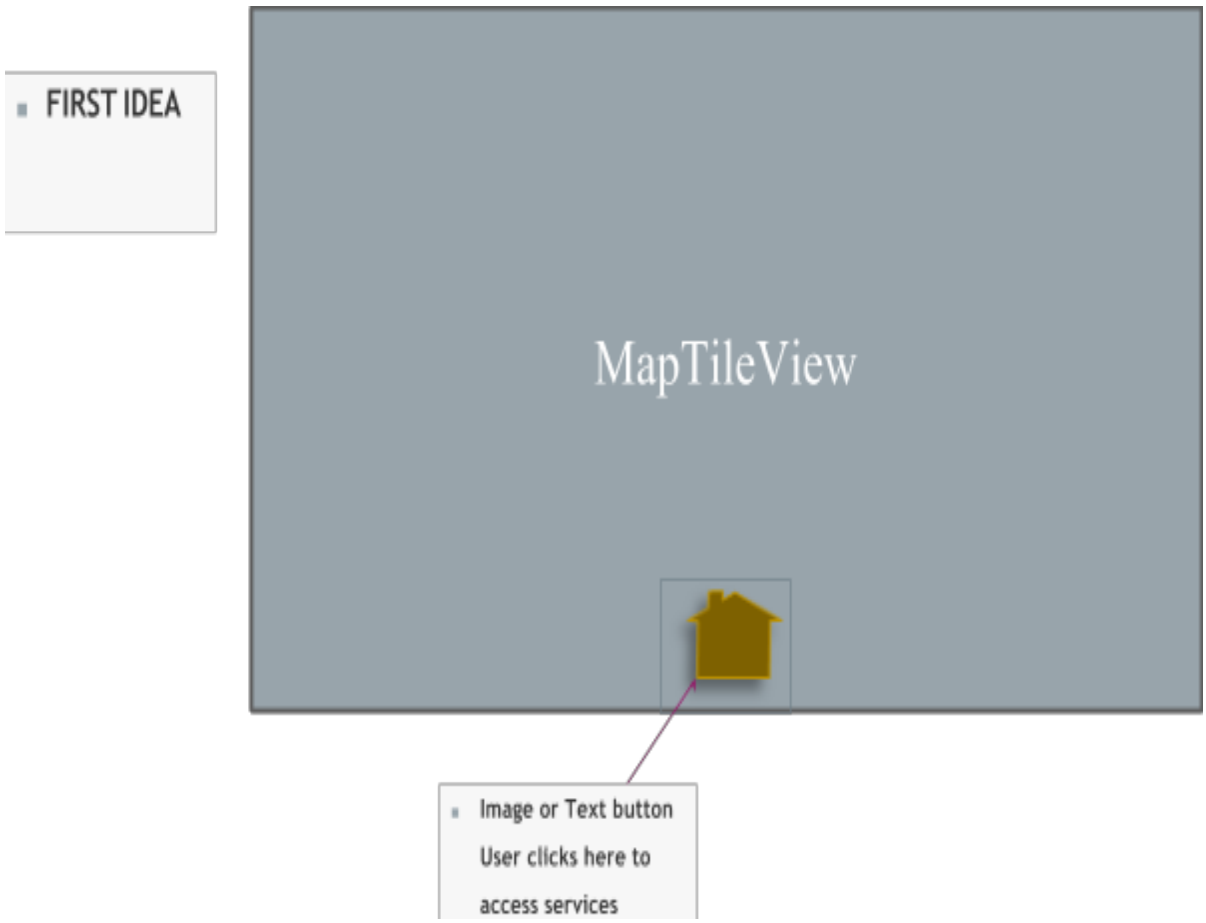

Figure 7 : The User clicks on the button to access the services.

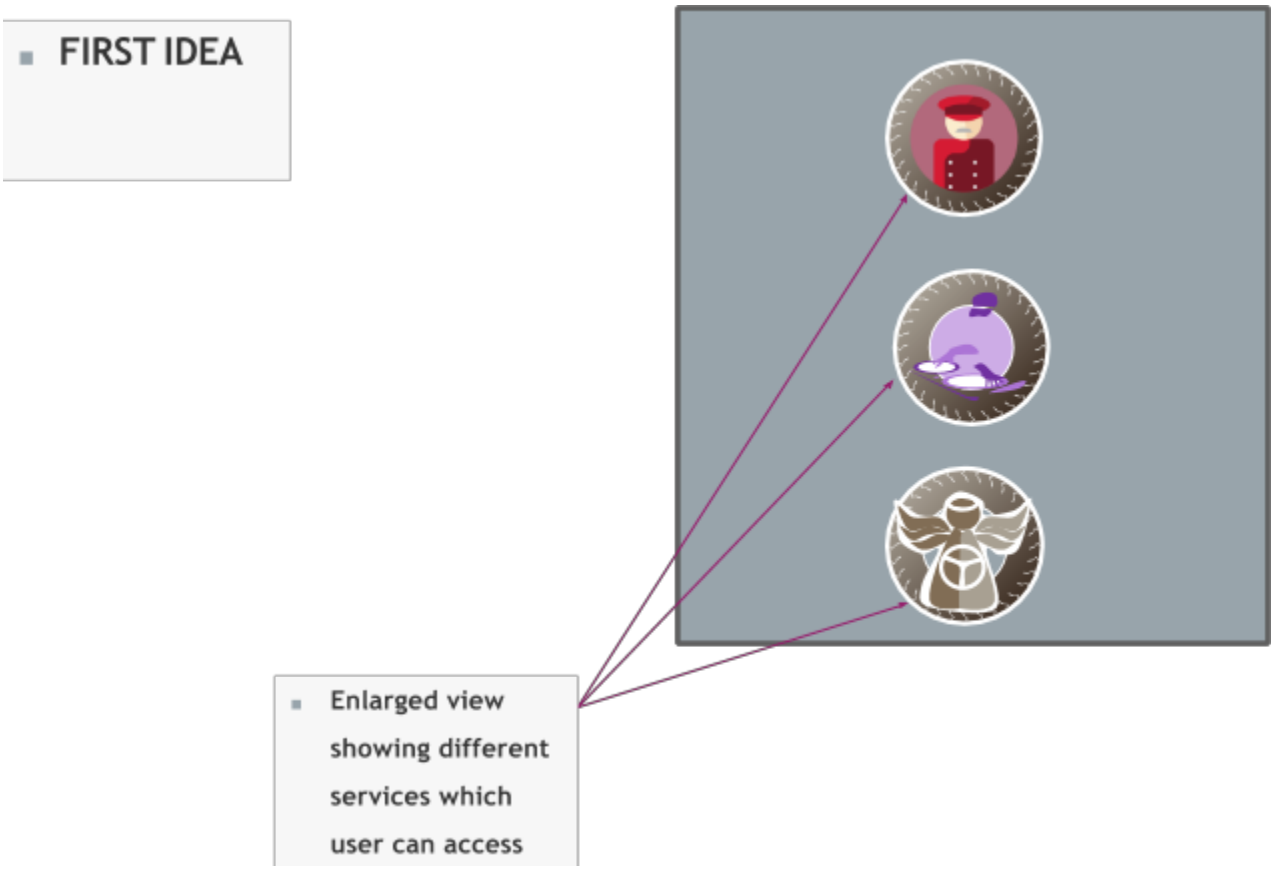

Figure 8: After clicking the button, an enlarged view is shown with the different services which the user can access.

VI.5.2 Second Design Option

The second idea was to have a dedicated button for each service to be integrated. These buttons will be positioned at different points on the screen(bottom, left, next to each other etc.). They will serve as entry points to each service and a user/driver can click on any of them depending on what service needs to be accessed. A notification is provided to the driver via text/voice as well.

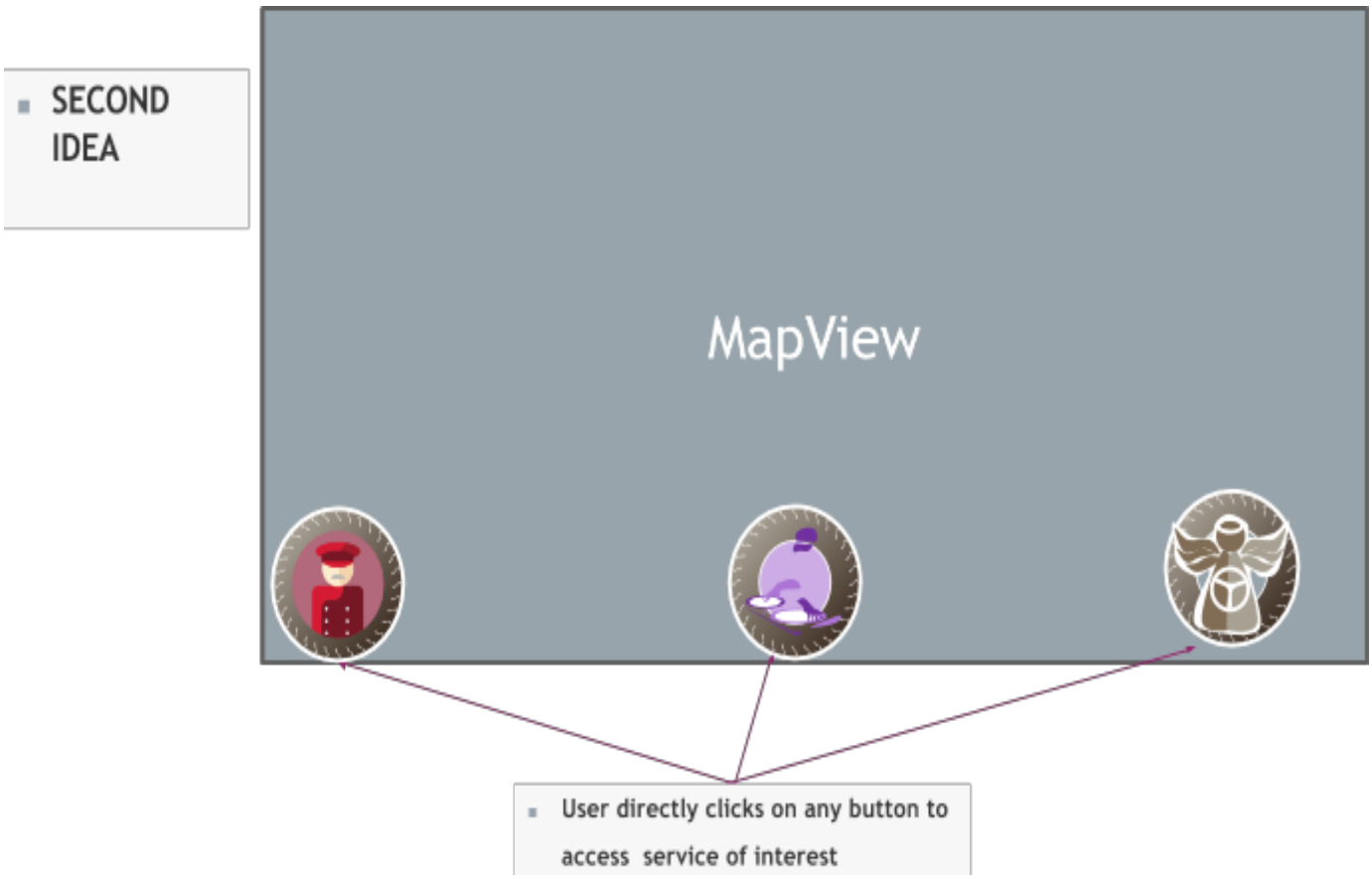

Figure 9 : The User directly clicks on any button to access a service of interest.

VI.5.3 Third Design Option

The third idea was to provide a sliding function. The user clicks on a button to make the service context screen slide from left to right or bottom to top and over the map UI. When the user is through with interacting with the service, the button is clicked again to withdraw the view.

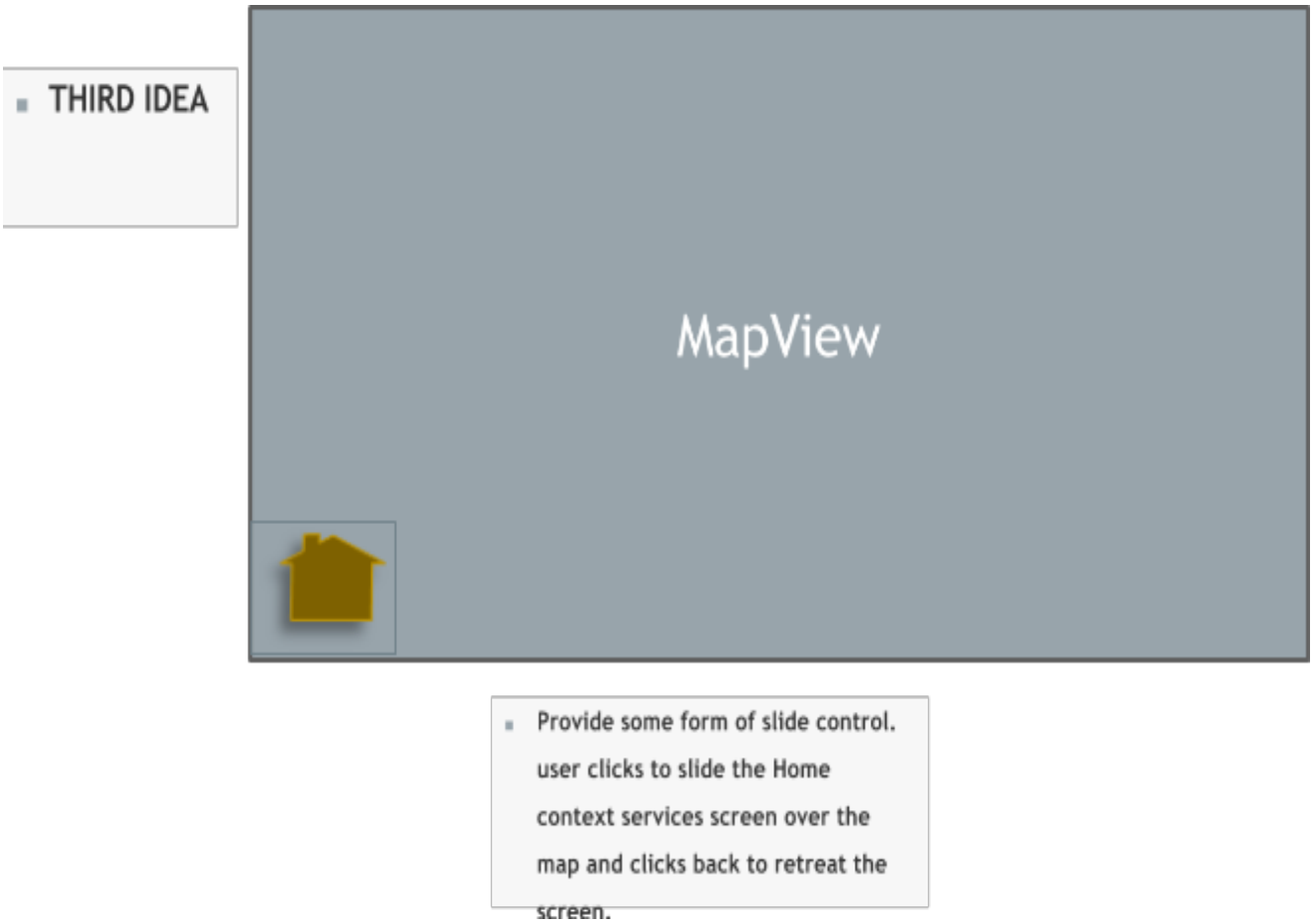

Figure 10: The user clicks the service button and a service context screen slides over the MapView. Clicking the back button or clicking the button again retreats this view.

After careful analysis with my supervisor, it was decided that the second option provides the most intuitive and easiest option for a driver or user of these services on the Map.

Also, it was decided to develop the UI elements of these services using HTML to aid more rapid development time and provide a first prototype. In line with this, the services were configured using html and css elements and interactivity was added to them using javascript.

In this instance a .html file containing html and css elements which define all these services was configured .The page also contained links to a database of pictures,images and other resources which make up the visual component of the Services UI. There were also different onClick functions implemented in Javascript to handle user interactivity with this interface.

The code snippet in the next page shows a portion of the UI content of one of the services.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
<style type="text/css">

//CSS
body {
    margin: 0;
    font-family:  Century Gothic;
    font-size: 25px;

}
….
#status     {
   position: absolute; top:90%; left:25%; width:75%; height:10%;
background-color:transparent; border:0; opacity: 0.8; color: black;
          seamless:"seamless";   frameborder:"0" ; marginwidth:"0";
marginheight:"0";
            padding:"0"; scrolling:"no";
}

</style>
…
// Javascript

<script type="text/javascript">

var CurUrl = "";
function LoadIframe(Url){
       var MyFrame = document.getElementById("panel");
       if (MyFrame){
            if (CurUrl == Url){
              if (MyFrame.className == "up" ){
                  MyFrame.className = "down";
               } else {
                MyFrame.className = "up";
               }
            }else {
               if (MyFrame.className == "up" ){
                  MyFrame.className = "down";
               }
             MyFrame.src = Url;
             MyFrame.className = "up";
          }
          CurUrl = Url;

          }
}

</script>
//div element for one table data in a row in the status table

<div id="status">
```

```
            <table>
            <tr>

            <TD  onclick="return LoadIframe('Ize.html');">
//onclick returns a UI  element
                              <IMG class="logo" src="Pictures/Home-presence.png" >
            </TD>

…

</body>
</html>
```

VI.6 Mixing HTML with Native

In order to integrate these Services into OsmAnd, there was the need to create an interface between the Android Native code and the dynamic HTML which makes up the Services. Android provides a way to wrap HTML content by using a “WebView”. The WebView class is an extension of Android's View class that enables the display of web pages as a part of an activity layout or as a component of an application’s UI. I utilised this approach in embedding the services into OsmAnd.

VII The OsmAnd Layer Architecture

VII.1 Background

In order to integrate these services into OsmAnd, I studied its architecture to understand the best approach to do this.

The OsmAnd architecture is structured such that it has many layers which host the different map objects, views, control buttons, Location services and so forth.

These layers combine to implement the functions of the Application.

Some of these layers are listed below:

- BaseMapLayer;
- GPXLayer;
- MapControlsLayer;
- MapInfoLayer;
- OsmandMapLayer;
- POIMapLayer;
- QeoPointLocationLayer;

Each Layer holds a different view and therefore needs to be initialised and configured. The OsmandMapLayer class performs this initialisation.
It is an abstract class that contains methods and inner abstract classes which help in layer configuration such as the initialisation of a layer, how a layer is drawn using a Canvas, the DrawSettings for each layer and so forth.

The class code fragment is illustrated below.

```
import java.util.Map;
import net.osmand.data.RotatedTileBox;
import net.osmand.plus.ContextMenuAdapter;
import android.graphics.Canvas;
import android.graphics.PointF;
import android.os.AsyncTask;
/// imports
public abstract class OsmandMapLayer {
      public abstract void initLayer(OsmandMapTileView view);
      public abstract void onDraw(Canvas canvas, RotatedTileBox tileBox,
DrawSettings settings);

      public void onPrepareBufferImage(Canvas canvas, RotatedTileBox tileBox,
DrawSettings settings) {
      }
      public abstract void destroyLayer();

///…...
      }
      public abstract boolean drawInScreenPixels();

      public static class DrawSettings {
            /// initialisation of settings goes here.
      }
      public DrawSettings(arg1, arg2,...) {
                  ///draw settings.
      }
}
```

Each Layer of the Map extends this OsmandMapLayer and inherits its properties and methods.

The Layers are added to the Map's UI and positioned based on a relative depth or "zorder" so that layers with higher zorders in the layer hierarchy are placed above layers with lower zorders. These zorders are integer values starting with 1 as the lowest layer with as many layers as needed.

This is illustrated for the QeoPointLocationLayer, the Layer which initialises Qeo,its position, UI and it's draw properties.

```
public class QeoPointLocationLayer extends OsmandMapLayer {
protected final static int RADIUS = 7;
      // Qeo Icon
      private Paint QeolocationPaint;
      private Paint Qeoarea;
      private Paint QeoaroundArea;
      private Paint textPaint;
      private Bitmap QeolocationIcon;

      // Global Var
      private OsmandMapTileView Qeoview;
      private ApplicationMode appMode;
      protected OsmandApplication app;
      protected final MapActivity map;

       private void initUI() {
            // init the User Interface
            QeolocationPaint = new Paint();
            QeolocationPaint.setAntiAlias(true);
            QeolocationPaint.setFilterBitmap(true);
            QeolocationPaint.setDither(true);

            textPaint = new Paint();

textPaint.setColor(Qeoview.getResources().getColor(R.color.color_black));//.getColor
(R.color.color_qeo));//ARGB(0, 255, 255, 1);//
            textPaint.setTextAlign(Align.CENTER);
            textPaint.setTypeface(Typeface.DEFAULT_BOLD);
```

```
            float sp = Resources.getSystem().getDisplayMetrics().scaledDensity;
             textPaint.setTextSize(sp * 16);

            Qeoarea = new Paint();
            Qeoarea.setColor(Qeoview.getResources().getColor(R.color.color_white));
            QeoaroundArea = new Paint();

QeoaroundArea.setColor(Qeoview.getResources().getColor(R.color.color_black));
           QeoaroundArea.setStyle(Style.STROKE);
           QeoaroundArea.setStrokeWidth(2);
           QeoaroundArea.setAntiAlias(true);

            app = map.getMyApplication();
           checkAppMode(Qeoview.getSettings().getApplicationMode());
           resourceManager = Qeoview.getApplication().getResourceManager();
           filter = map.getMapLayers().getPoiMapLayer().getFilter();
      }

      @Override
      public void initLayer(OsmandMapTileView view) {
            this.Qeoview = view;
            initUI();
      }
…
}
```

The initUI() initialises the UI properties of the Qeo icon including the text, color and background image, The initLayer() sets the layer and loads the UI of the Qeo service on that layer.

VII.2 Adding the WebView Layer

Building upon the layer structure of OsmAnd, I created a WebViewLayer class to host the WebView which will be the container to wrap the html content.

```
public class WebViewLayer extends OsmandMapLayer {
      private OsmandMapTileView view;
      protected final MapActivity map;
```

```
      public WebViewLayer(MapActivity map) {
            //create a layer on the map for the WebView
                        this.map= map;
      }
…
}
```

I included the initLayer method where the layer is initialised and its UI configured

```
@Override
      public void initLayer(OsmandMapTileView view) {
            this.view= view;
                      initUI();
}
```

Within this method , the initUI is called which draws the UI.

```
private void initUI(){
// initialise the UI for this layer

}
```

I added a new Instance of a WebView in this method which is the view to host the xml layout. I also provided the FrameLayout instance which is the view's parent.

```
private void initUI(){
    // initialise the UI for this layer
    FrameLayout frameLayout = (FrameLayout) view.getParent();
    WebView wv = new WebView(view.getContext());
...
}
```

The layout id is referenced from its web.xml file;

```
                        wv.findViewById(R.id.webview);
```

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:orientation="vertical">

   <WebView
       android:id="@+id/webview"
       android:layout_width="520dp"
       android:layout_height="match_parent"
       android:layout_gravity="left" />

</LinearLayout>
```

The local html file is retrieved from the device storage system where it is saved in its Download folder together with its associated resources like images and local link files. I used instances of Android's File and Uri APIs to retrieve the html file as illustrated below.

```
    // Retrieve the local html file from the device storage and load it on the webview

String fileName= "Default.html";

String completePath = Environment.getExternalStorageDirectory() + "/Download/HTML/"+
fileName

File file = new File(completePath);

Uri fileUri = Uri.fromFile(file);
```

With this, the html content could be loaded into the webview using the loadUrl() method

```
wv.loadUrl(fileUri.toString());
```

I set the WebView to be transparent so that the part of the map under it is visible.

```
wv.setBackgroundColor(Color.TRANSPARENT);
```

I made the layer type to be "hardware" to take advantage of Android's accelerated hardware rendering and increase the rendering quality of the WebView.

```
wv.setLayerType(WebView.LAYER_TYPE_HARDWARE, null);
```

Finally, I set the layout parameters, the size and position of the WebView and added it to its parent view.

```
final FrameLayout.LayoutParams lp = new FrameLayout.LayoutParams(500,
                  ViewGroup.LayoutParams.WRAP_CONTENT);

                       lp.gravity = Gravity.LEFT;

                       frameLayout.addView(wv, lp);
```

In order to add this layer to the existing layer hierarchy, I examined the OsmandMapTileView class. This class contains the list of layers and their positions in the hierarchy

```
public class OsmandMapTileView extends SurfaceView implements
IMapDownloaderCallback, Callback {

...

    private BaseMapLayer mainLayer;

    private List<OsmandMapLayer> layers = new ArrayList<OsmandMapLayer>();

    private Map<OsmandMapLayer, Float> zOrders = new HashMap<OsmandMapLayer,
Float>();
```

The ArrayList holds all the layers of the map while the HashMap holds the object/value mapping between the layers and their relative zorder positions.

This class also contains a constructor to initialise its UI

```
public OsmandMapTileView(Context context) {
            super(context);
            application = (OsmandApplication)
context.getApplicationContext();
            initView();
         }
```

The initView() method models the UI of the Map.

```
public void initView() {
            paintGrayFill = new Paint();
            paintGrayFill.setColor(Color.GRAY);
            paintGrayFill.setStyle(Style.FILL);
            // when map rotates
```

```
            paintGrayFill.setAntiAlias(true);
            paintBlackFill= new Paint();
            paintBlackFill.setColor(Color.BLACK);
            paintBlackFill.setStyle(Style.FILL);
…
}
```

After studying this class, I discovered that there are methods which controls the addition and removal of map layers within the map,

```
public synchronized void addLayer(OsmandMapLayer layer, float zOrder) {

        //adds new layers to the  map
            int i = 0;
    for (i = 0; i < layers.size(); i++) {
        if (zOrders.get(layers.get(i)) > zOrder) {
                        break;
                }
            }
            layer.initLayer(this);
            layers.add(i, layer);
            zOrders.put(layer, zOrder);
      }

public synchronized void removeLayer(OsmandMapLayer layer) {
    while(layers.remove(layer));
            zOrders.remove(layer);
            layer.destroyLayer();
      }
```

The addLayer does an iteration through the list of layers and searches for their corresponding zorders. If the zorder of a particular layer is greater than the existing zorder value, it creates a new layer and adds this layer to the list of layers. It also maps the layer to its corresponding zorder value.

Similarly the removelayer removes a layer and destroys it in the hierarchy.

The OsmandMapTileView class also provides a get method to retrieve the list of layers.

```
public List<OsmandMapLayer> getLayers() {
            return layers;
      }
```

After studying the class, I provided a means of adding the WebViewLayer to the List of Layers and mapping it to its corresponding zorder so that the addLayer could handle its addition to the map hierarchy.

For doing this, I examined the MapActivityLayers class. This is the class responsible for maintaining the Layers used by the MapActivity.

```
package net.osmand.plus.activities;

// imports...

/**
 * Object is responsible for maintaining layers used by map activity
 */
public class MapActivityLayers {

       private final MapActivity activity;
      // the order of layers should be preserved  when you are inserting a new
layer

       private MapTileLayer mapTileLayer;
      private MapVectorLayer mapVectorLayer;
       private GPXLayer gpxLayer;
       private QeoPointLocationLayer qeoLayer;
…..// other layers
```

The creation of the layers is done by createLayers method within the MapActivityLayers class.

```
public void createLayers(final OsmandMapTileView mapView){

            OsmandApplication app = (OsmandApplication) getApplication();
            RoutingHelper routingHelper = app.getRoutingHelper();

            // mapView.addLayer(underlayLayer, -0.5f);
            mapTileLayer = new MapTileLayer(true);
            mapView.addLayer(mapTileLayer, 0.0f);
            mapView.setMainLayer(mapTileLayer);

            // 0.5 layer
            mapVectorLayer = new MapVectorLayer(mapTileLayer);
            mapView.addLayer(mapVectorLayer, 0.5f);

            downloadedRegionsLayer = new DownloadedRegionsLayer();
            mapView.addLayer(downloadedRegionsLayer, 0.5f);

            // 0.9 gpx layer
            gpxLayer = new GPXLayer();
            mapView.addLayer(gpxLayer, 0.9f);


// other layers created...

}
```

Each layer is instantiated within this method and then added to the mapView List which accepts the layer as its object/key and its corresponding zorder as the value

In line with this observation, I declared a private instance of the WebViewLayer inside the MapActivityLayers class

```
private WebViewLayer webViewLayer;
```

I added it to the mapView in the createLayers method setting the MapActivity as its context

```
public MapActivityLayers(MapActivity activity) {
            this.activity = activity;
      }

...


public void createLayers(final OsmandMapTileView mapView){

           //WebView layer
            Log.d("Edebug", "WebViewLayer is created and added to existing
layers");

             webViewLayer = new WebViewLayer(activity);

          //add webView layer to map layer hierarchy.

             mapView.addLayer(webViewLayer, 13);
}
```

There were 12 other layer zorders visible on the mapView so I specified the zorder of the WebViewLayer as 13.

There is also an updateLayers method to update the layer hierarchy and refresh the map.

```
public void updateLayers(OsmandMapTileView mapView){
            OsmandSettings settings = getApplication().getSettings();
            updateMapSource(mapView, settings.MAP_TILE_SOURCES);
…
if(mapView.getLayers().contains(webViewLayer) != settings.SHOW_WEB_LAYER.get()){
                  if(settings.SHOW_WEB_LAYER.get()){
                        mapView.addLayer(webViewLayer, 13);
                  } else {
                        mapView.removeLayer(webViewLayer);
                  }
OsmandPlugin.refreshLayers(mapView, activity);
```

```
}
```

The WebViewLayer was now available for the OsmandMapLayer class to add it to the existing layer hierarchy.
I have provided some figures which highlight the different classes and how they interact to implement the WebViewLayer in OsmAnd.

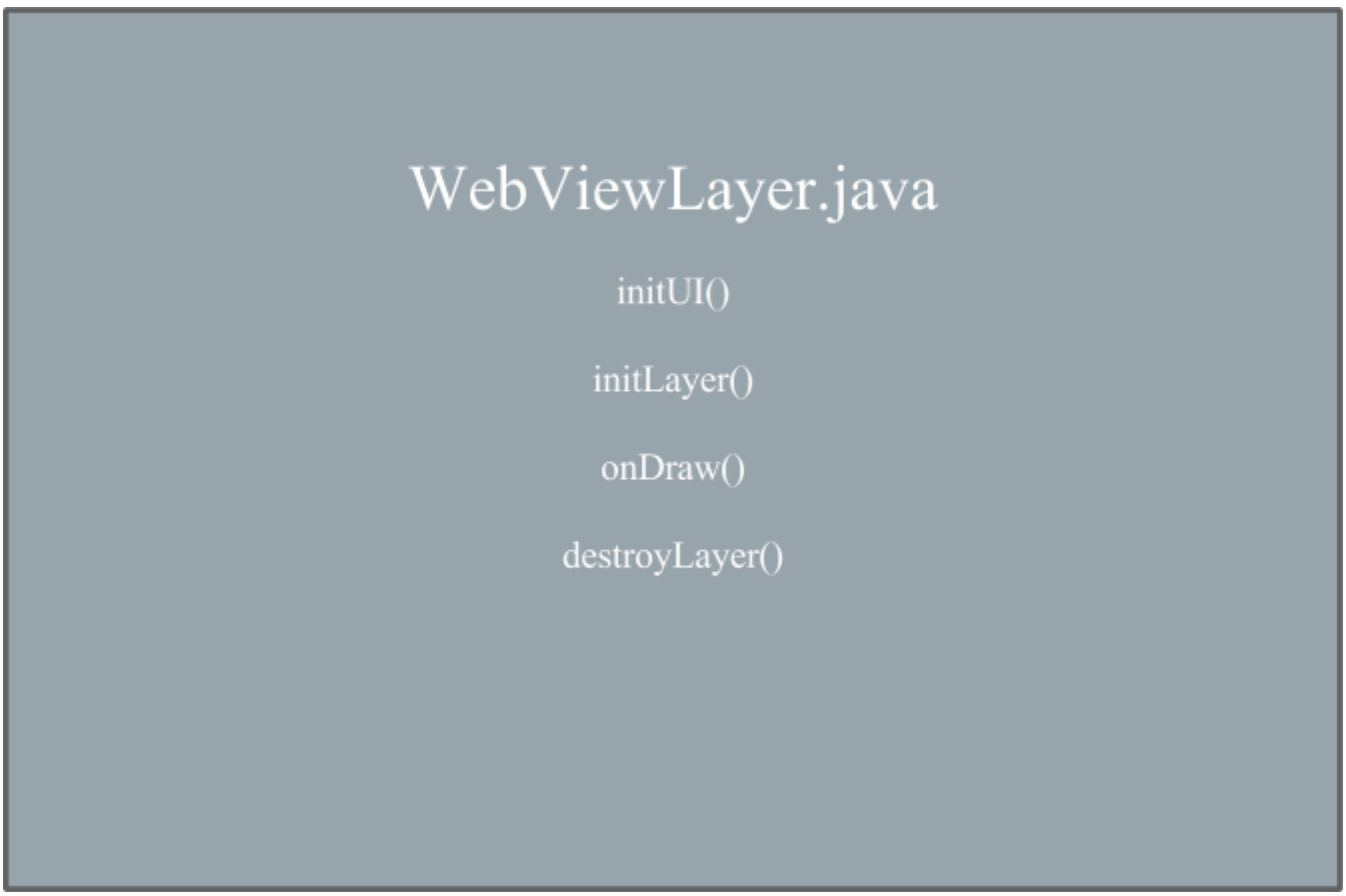

Figure 11: The WebViewLayer class and its principal methods.This is where the WebView is initialised and where its UI is also modelled.

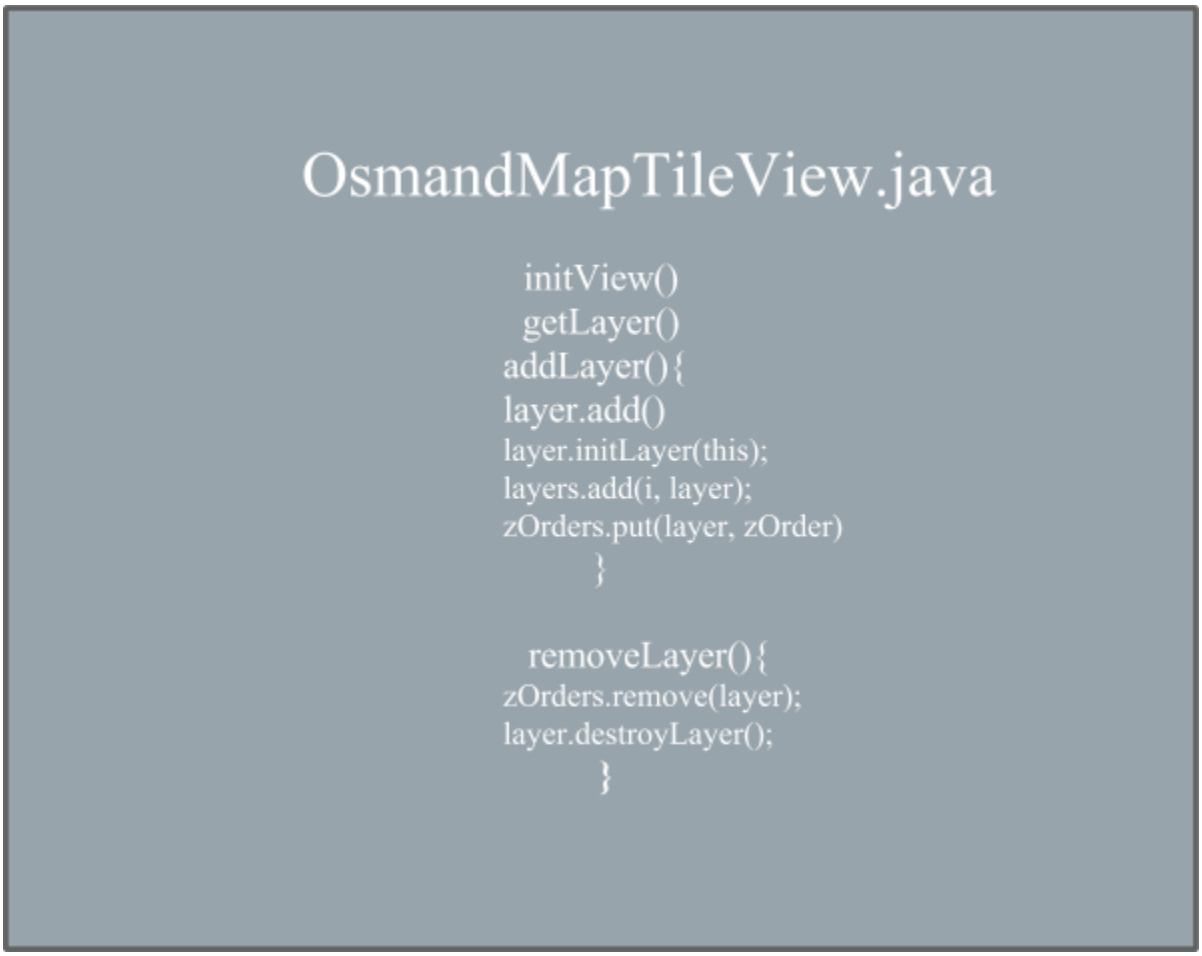

Figure 12: The OsmandMapTileView class. This is the class that adds and removes layers in OsmAnd.

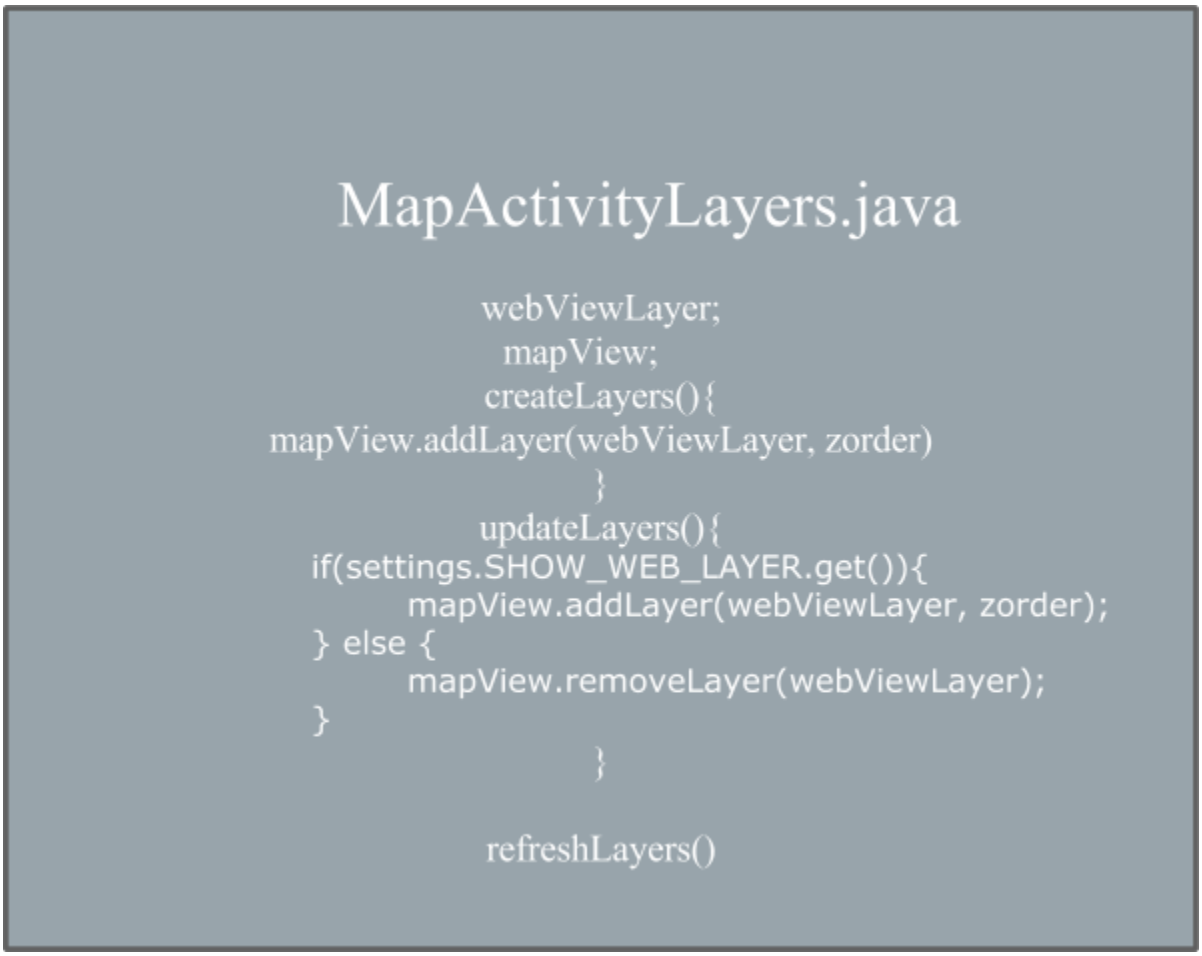

Figure 13: The MapActivityLayers class maintains the Layers used by the MapActivity in OsmAnd.

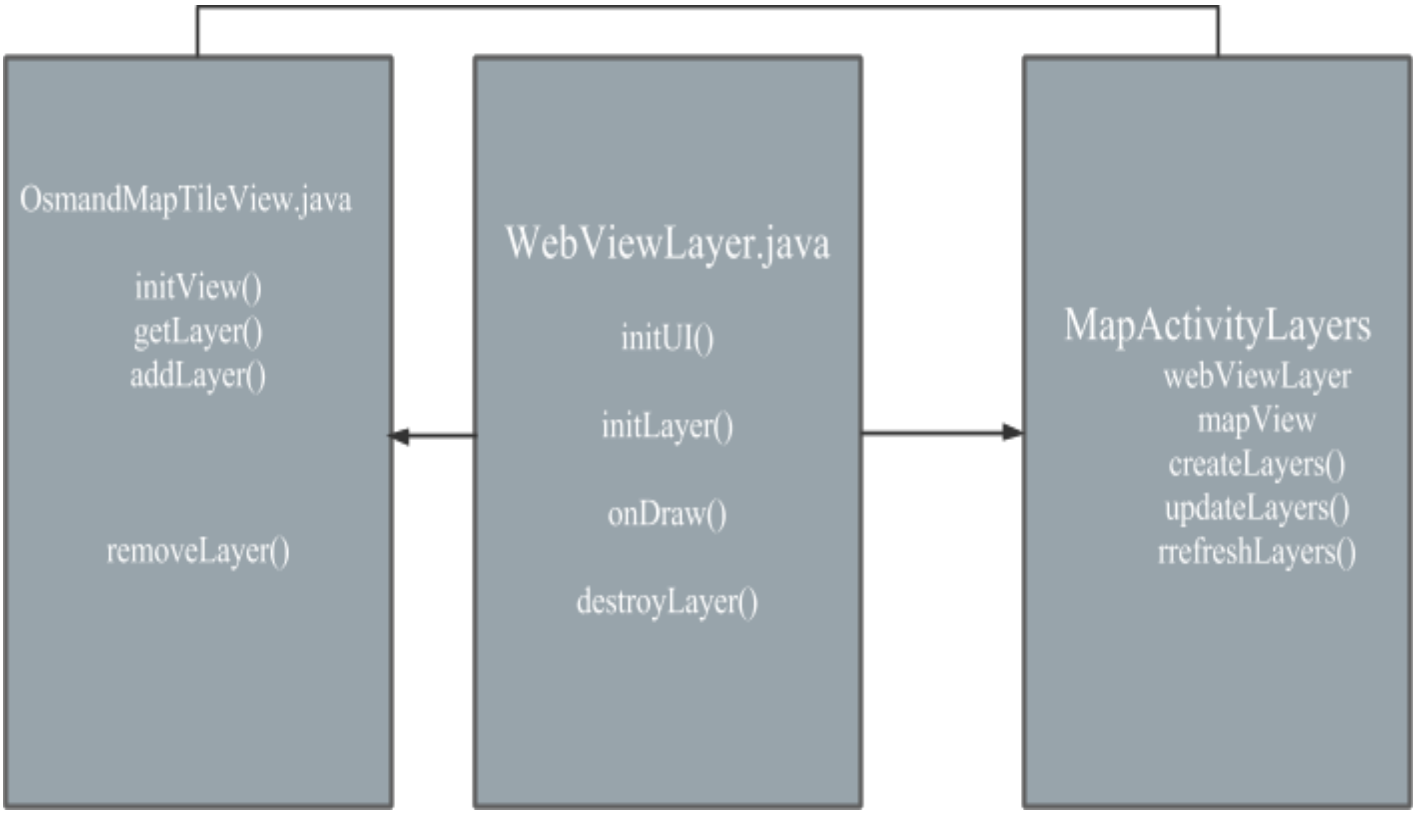

Figure 14: The WebView class components and how they interact.

I have included some screenshots of the new OsmAnd interface on the next page.

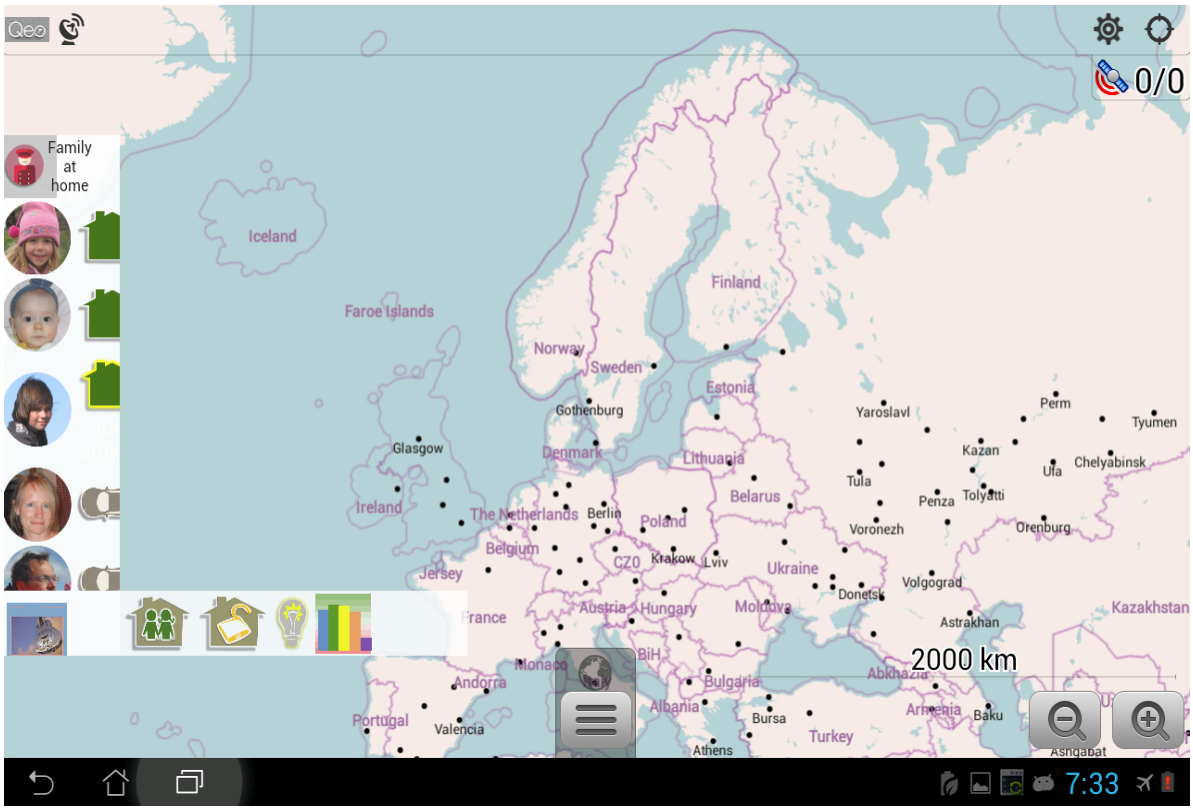

Figure 15: The Osmand Map showing the WebView Layer hosting the Services implemented in a web page. The portion of the map underneath the webview is still visible and there are control buttons at the bottom of the webview which can be clicked to load and unload these services.

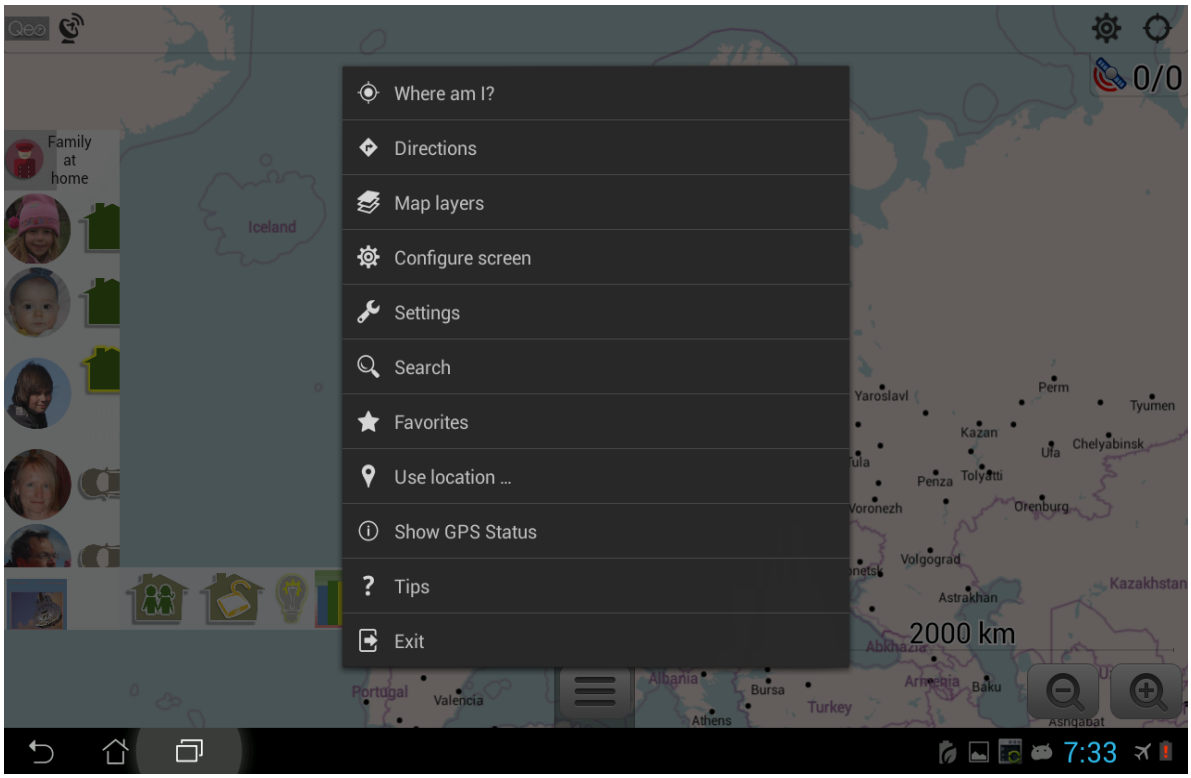

Figure 16: The menu list showing the basic map functions. These are still accessible and not disrupted by the presence of the WebView layer.

VII.3 Javascript, Native Communication

I added a Javascript Interface class named WebInterface.java that exposes native methods of android and allows javascript functions in the webpage to call these native methods. These allows Javascript-native communication and enables several possibilities for example to view the driver location by clicking an element in the webview and sending this location to the OsmAnd-Qeo interface to change direction to a new destination.
The WebInterface class snippet is shown below;

```
package net.osmand.plus.activities;

import android.webkit.JavascriptInterface;
public class WebInterface {

  /** Show a toast to change direction on the map from the web page */
        @JavascriptInterface
          public void showToast(String toast) {
              Toast.makeText(mContext, toast, Toast.LENGTH_LONG).show();
          }
      }
```

The Android framework specifies that for a method to be accessible from javascript, the annotation “@JavascriptInterface” has to be explicitly added immediately before the method call in the interface. This is to ensure security and eliminate vulnerability of the application to hacking or security breaches.
In the WebViewLayer.java, the javascript interface is bound to android and an alias is given to it. This alias together with it’s associated toast method is then called within a javascript function in the webpage.

I added the binding method in the WebViewLayer class as shown in the snippet below.

```
                        //Bind javascript to android
            wv.addJavascriptInterface(new WebInterface(), "Android");
```

Here is a snippet of the android toast method called in the Javascript on the webpage,

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title></title>
...

<script type = text/javascript>

function showAndroidToast(toast){
      Android.showToast(toast); // webinterface method
}
…
</script>
</head>

<body>

<div id="status">
         <table>
            <tr>

                  <TD onclick="showAndroidToast('Do you want to change your
destination?');">
                  <img src="Pictures/Map.png">
                  </TD>
             </tr>
         </table>
</div>
…

</body>
</html>
```

The function showAndroidToast() calls the toast function in the WebInterface class. This allows a button on the webpage to return a message on the map that asks if the driver wants to change destination.

VIII Code Metrics Review.

This section gives an outline of the number of lines of code used in each class and the aggregate number involved in the development activity.

Class	# Lines of Code
MusicService	261
Song	26
SongAdapter	60
WebViewLayer	121
OsmandMapTileView	37
MapActivityLayers	60
WebInterface	14
Dynamic HTML	426

Aggregate code base in Java: 579

Aggregate code base in Dynamic HTML: 426

Total Code Base: 1005

Conclusion

The framework for the extra services have been built into OsmAnd during the course of this internship. The results will be used as a basis for future work such as using Javascript to communicate with and execute OsmAnd native methods and vice versa which will give rise to interesting use cases such as parking lot notifications, automatic meter payments as well as enabling dynamic updates of these services by syncing with the cloud.

During the past six months of this internship project , I have had the opportunity to enhance my creativity and develop more competence in software development principles and requirements engineering.

My time at Technicolor has also given me a great experience and a good stepping stone into the world of work.

Bibliography

- http://www.technicolor.com/fr/ General Information on Technicolor and Qeo
- http://www.qeo-app-development.com/ Developer Information on Qeo
- https://allseenalliance.org/ Information on AllSeen
- http://osmand.net/ Official Site for OsmAnd.
- https://groups.google.com/forum/osmand OsmAnd Google groups forum
- http://developer.android.com/ Android Developers site.

Table Of Figures

Glossary of Terms

Activity

In Android, an Activity is an active screen of an application.

Class.

In Object Oriented Programming, a class declares properties common to a set of objects. The class declares attributes representing the state of objects and methods represent their behavior.

Eclipse.

Eclipse is an integrated development environment (**IDE**). It contains a base workspace and an extensible plug-in system for customizing the environment. Written mostly in Java, Eclipse can be used to develop applications and has plugin functionalities to support different languages.

Framework

In computer programming, a framework is a coherent set of components which is used to create the foundations and the outline of all or part of a software (architecture).

GUI

A GUI (Graphical User Interface) is a graphical display in one or more windows containing controls, called components, that enable a user to perform interactive tasks. The user of the GUI does not have to create a script or type commands at the command line to accomplish the tasks.

LAN

Local Area Network, refers to a local computer network which connects computers within a limited area such as a home, school, computer lab, or office building.

SCM

Source control management is the management and tracking of changes in config files and source code within a project. Git, CVS and Subversion are popular implementations of scm software.

SDK

A development kit and software development kit is a set of tools
allowing developers to create different types of applications for different operating systems (eg iOS,Android, BlackBerry 10, Symbian, Bada, Windows Phone 7, Windows Phone 8 and Windows 8).

SSH

Secure Shell (SSH) is a cryptographic network protocol for securing data communication. It establishes a secure channel over an insecure network in a client-server architecture, connecting an SSH client application with an SSH server.

Appendix

Appendix 1: Role and purpose of OsmAnd

OsmAnd (OSM Automated Navigation Directions) is a map and navigation application with access to free data OpenStreetMap (OSM). OsmAnd provides GPS routing with visual and voice guidance for car, bike, or pedestrian.

Key features:

- Navigation - Voice Guidance
- Optional display of roads, street names and estimated time of arrival
- Support for different stages of a route
- Automatic re-route of an initial route
- Destination search by address, point of interest (e.g restaurant, hotel, gas station, museum, golf course etc.)

Map display

- Display your position and your direction on the map
- Aligning the optional card based on the compass or direction of your movement
- Backup your important places in favorites
- Display of points of interest (POI) around you
- Display satellite views (Bing)
- Display of additional layers such as navigation routes GPX and maps

Additional customization with transparency

- Optional display of place names in the English spelling, local or phonetic

Uses OpenStreetMap and Wikipedia data:

- High-quality information from the best global collaborative projects
- Global maps of OpenStreetMap, available by country or region
- Points of Interest Wikipedia
- Free unlimited downloads, directly from the application
- maps are always updated (updated at least once per month)
- Choice of the whole map or only the road network
- Compact Vector Maps

Optional security features

- Automatic switch day / night
- Display of speed limits with display of speeding
- Automatic Zoom function of the traveling speed
- Sharing your location with your friends

Pedestrians and cyclists

- The cards include footpaths, pedestrians and cyclists
- Navigation and specific display for pedestrians and cyclists
- Optional display of public transport stops (bus, tram, train) to include the names of line
- Optional display of speed and altitude
- Optional display of contour lines and relief shading

Appendix 2: FlowChart of OsmAnd's Main Classes and the Class-Layer hierarchy