An intelligent Smartphone application

Combining real-time with static data in pursuit of the quickest way to travel by bus

Magnus Raaum

Student at the Department of Computer and Information Science

NTNU Trondheim

CONTENTS

ABSTRACT

The goal of this project is to make provide accurate and helpful information to bus travelers in Trondheim through an Android application. By combining real-time data (delays, etc), user location, the location of bus stops and a powerful natural language bus route expert system the application can fulfill every user's information need. The method used for user location is a built in GPS receiver in the smartphone, while the bus stops are saved in the software as a static list.

A connection to BusTuc, the natural language bus route expert system, is created through its web interface by taking advantage of the smartphone's ability to make HTTP requests.

The real-time data was unfortunately not available this semester, but was taken into account while the development was ongoing so extending the software to accommodate these data's should not be a problem.

The temporary implementation provided in this paper includes successful communication with BusTuc, a graphical representation of the user's location and bus stop locations and a way to extract bus stops within a given radius.

This paper also includes notes on current and future problems, methods and suggestions for solving those problems and the implementation so far.

1. INTRODUCTION

This report describes the motivation, method and technologies needed to create an application which intelligently calculates the best possible travel route for bus users in Trondheim. By using Global Positioning System (GPS) coordinates, real-time data and a natural language bus route expert system the application should be able to give the user information describing the fastest way to get from point a to point b.

The primary motivation behind this research is to not to create new data, but collect data from many components or systems and create most accurate, reliable and usable application up to date. The problem with information about bus routes is not that it is hard to find, but that it is spread over multiple system. Consider yourself a guest in someone's house where you have not been before. You do not know where the closest bus stop is, what the route times are or even which bus you should take. A simple press of a button will answer all your questions.

This report will also contain a description of the underlying technologies located in smartphones and how to utilize these technologies in order to provide a system with all the required functionality.

As of today the only tool available for information about the bus network is a web interface at ATB's homepage. ATB, the company responsible for the bus network, is launching a real-time system in the end of December where travelers will be able to get information about delays online, at the bus stops and on the bus. I hope this information will be made available for further development on this application.

In a world where users require instant information with minimal effort, high expectations are set for information value and correctness, usability and stability. These requirements are especially important because people will trust the information given and plan their route accordingly.

The core requirements in this project are:

1. Find the user's position by utilizing GPS.

2. Use this position to find the relevant bus stops surrounding the user.

3. Use the relevant bus stops to find the fastest way to the destination provided by the user.

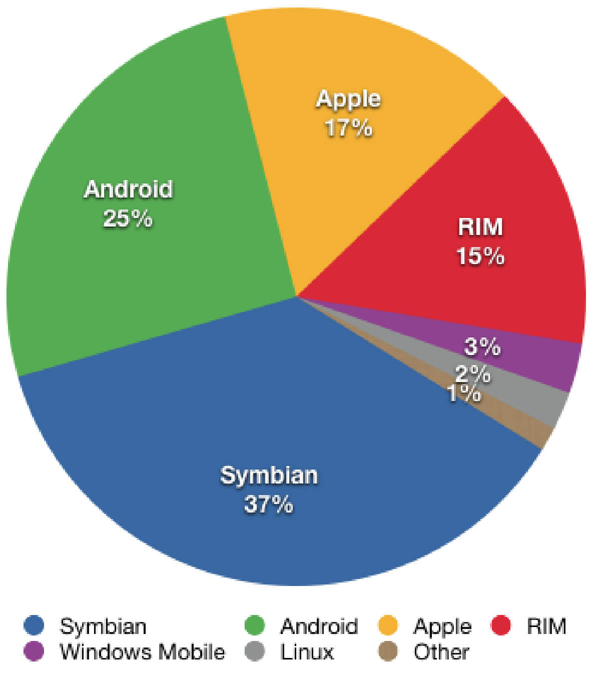
4. Present this information in natural language and with a graphical representation of the suggested route.

While this report is originally meant for Professors and students at NTNU with a technical background, most of it (excluding details on implementation) can be understood by people in other fields.

2. UNDERLYING TECHNOLOGIES

2.1 SMARTPHONES

Smartphones are defined as a cell phone that offers more connectivity and advanced computing ability than a 'regular cell phone'. With the ability to install and run more advanced applications, smartphones can be thought of as handheld computers integrated within a mobile telephone. These types of phones run a complete operating system which provides a platform for application developers.



Sales of smartphones by operating system

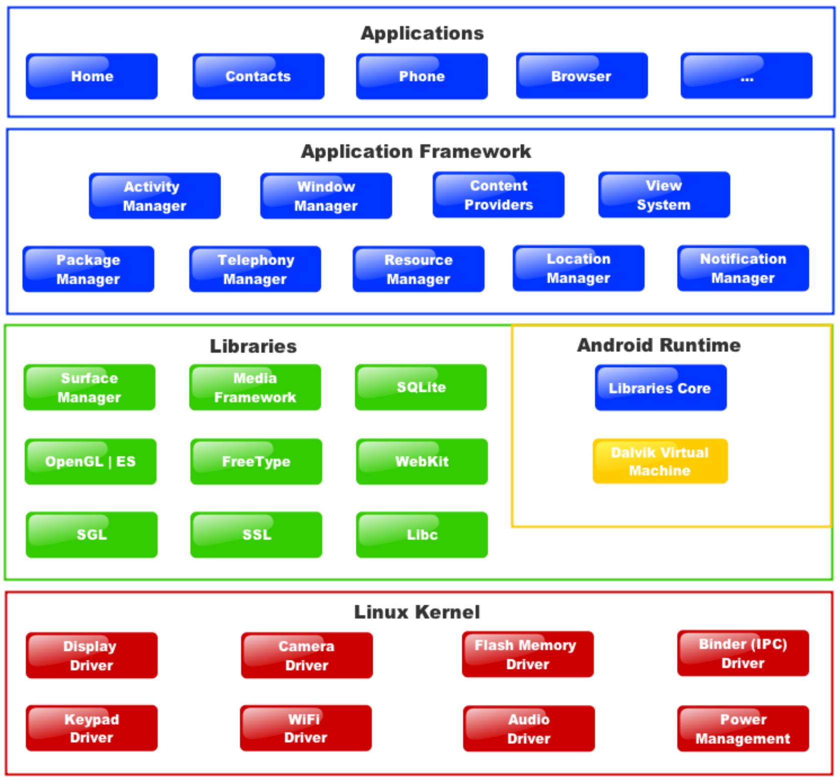
The demand for more computational power, larger screens and open operating systems has made smartphones the dominant product in the mobile phone market.

The smartphone used in this project was HTC Wildfire. It runs Android OS 2.1 on a Qualcomm MSM7225 528 MHz CPU, supported by 512 MB Read-only memory and 384 MB Random-access memory. Wildfire was chosen primarily because of its relatively low price and full GPS support.

HTC Wildfire

2.2 ANDROID OS

Android OS is an open-source operating system initially created by Android Inc, but purchased by Google in 2005. It is based on a modified Linux kernel and consists of Java application running on an object oriented application framework on top of Java core libraries. These core libraries run on Dalvik VM (register-based virtual machine) featuring JIT compilation (just-in-time compilation. A method to improve the runtime performance of computer programs by translating high-level language continuously). The core is generally written in C and the user interface in Java. Some third party libraries are written in C++.



The Android OS Architecture

It allows for developers to write application programs that extend the devices' standard functionality. The development occurs in Java, using Google's Java libraries.

Development on Android OS is quite popular and the official market for applications (Android Market) has over 100,000 applications which extend the devices' standard functionality.

Developers must use Android software development kit (SDK), which includes a comprehensive set of development tools.

2.2.1 DEBUGGER

The Android debugger works like a normal debugger. It shows the position in the original code if the program crashes or reaches a preset condition. It is also possible to set run restrictions on the application by inserting a wait-for-debugger call in the code. This blocks the selected application from loading until a debugger attaches. You want to add this call as soon as possible because it is important to debug the startup process. The debugger also shows how often elements on the screen is being redrawn by flashing a pink rectangle, which is useful for discovering unnecessary screen drawing.

An important part of the debugger is the Android Debug Bridge (adb). This is a tool that lets you manage the state of the Android-powered device or an emulator instance. The tool can be described as a client-server program which includes three different components.

- A client, which runs on the machine you develop on.
- A server, which runs on the machine you develop on as a background process. This server handles the communication between the adb daemon on an emulator or device and the client.
- A daemon, which runs on each emulator or device instance as a background process.

At startup, the client will check if a server already exists. If it does not, it will create one on the development machine. The server then binds to local TCP port 5037 and listens for instructions sent from clients. The server proceeds to scan all odd-numbered ports in the range 5555 to 5585, the range used by emulator or device instances to identify them and set up a connection. Each emulator or device instance gets a pair of sequential ports – and even-numbered port for console connections and an odd-numbered port for adb connections. For example

- HTC Wildfire, console: 5554
- HTC Wildfire, adb: 5555

The console can be used to send commands and debug if you do not have another tool (e.g Eclipse). Examples of commands:

- Adb –e pull /system/app/Development.apk ./Development.apk
- Adb –d install Development.apk

The first one copies the development package from a running emulator to a connected device.

The second one installs it on the connected device.

Another part of the debugger is the Dalvik Debug Monitor Server (DDMS). It is a graphical program that provides a lot of functionality including:

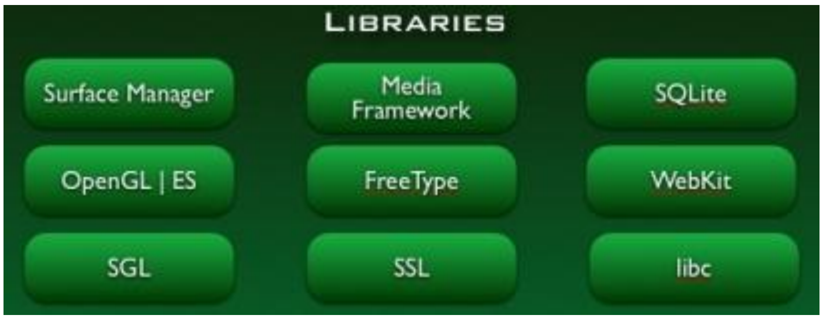
- Port-forwarding services
- Screen capture on the device
- Thread and heap information on the device
- LogCat (Responsible for dumping a log of system messages. The messages include a stack trace when the device throws an error, as well as Log messages the developer writes from the application)
- Process and radio state information
- Incoming call and SMS spoofing

- Location data spoofing

The DDMS connects the IDE to the applications running on the device. Android hosts an individual virtual machine for each application which runs in its own process and each process listens for a debugger on a different port. The DDMS connects to adb and the process and creates a monitoring service between the two. The DDMS will then receive the virtual machine's process ID, via adb, and opens a connection to the virtual machine's debugger, through the adb daemon on the device using a custom wire protocol.

The DDMS also includes a file explorer for the device or emulator instance.

2.2.2 LIBRARIES



The available libraries

Android includes libraries written in C/C++ used by different components of the Android system. The functionality provided by these libraries can be used by developers through the Android application framework. The primary libraries are:

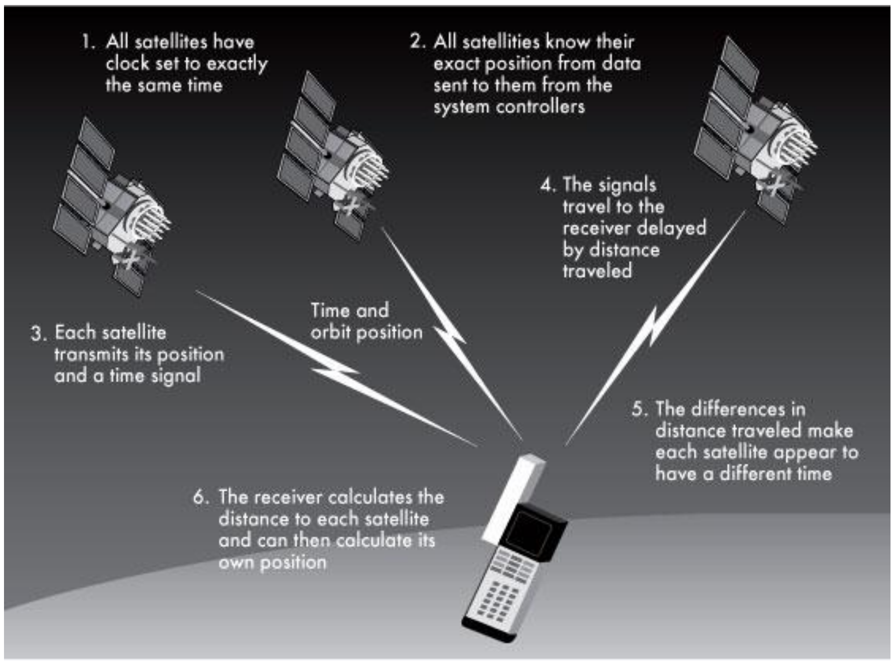
- Surface Manager: handles access to the display subsystem and composites 2D and 3D layers of graphic from multiple applications.
- Media Libraries: support playback and recording of many image files, audio and video formats, including MPEG4, H.264, MP3, AAC, AMR, JPG and PNG.
- SGL: underlying 2D graphics engine.
- FreeType: vector font and bitmap rendering.
- SQLite: lightweight relational database engine.
- 3D libraries: use either hardware 3D acceleration or the included, highly optimized 3D software rasterizer
- System C library: a BSD-derived implementation of the standard C system library (libc), modified for embedded Linux-based devices.

2.3 GLOBAL POSITIONING SYSTEM

2.3.1 GENERAL DESCRIPTION

The Global Positioning System (GPS) is a satellite-based navigation system created by a network of 24 satellites. The satellites are controlled by the U.S Department of Defense and GPS was originally intended for military applications. The system provides reliable location and time information as long as there is an unobstructed line of sight to four or more GPS satellites. No matter what weather, hour of the day or where you are. In the 1980's the U.S Department of Defense made GPS available for everyone with a GPS receiver.

The satellites circle the earth in a very precise orbit twice a day and sends signal information to earth. This information is picked up by the GPS receivers who use triangulation to calculate the user's exact location. Triangulation works like this: The GPS receiver compares the time a signal was transmitted by a satellite with the time it was received. This difference allows the GPS receiver to calculate the actual distance to the satellite. When combining these distance measurements from multiple satellites, the receiver can pin point the user's position. The accuracy of the position will naturally increase with the amount of available satellites. 3 satellites is enough to calculate a 2D position (longitude and latitude), but with 4 satellites you'll be able to get the altitude as well.


1. All satellites have clock set to exactly the same time
2. All satellites know their exact position from data sent to them from the system controllers
4. The signals travel to the receiver delayed by distance traveled
Time and orbit position
3. Each satellite transmits its position and a time signal
5. The differences in distance traveled make each satellite appear to have a different time
6. The receiver calculates the distance to each satellite and can then calculate its own position

GPS steps

The network of satellites is orbiting the earth at about 20,200 kilometers in the Medium Earth Orbit. They make two complete orbits in less than 24 hours traveling at speeds of roughly 11,300 km/h. They gather energy from the sun, but carry backup batteries onboard in case of a solar eclipse. Small rocket boosters on each satellite keep them on the correct path.

Most receivers now have a parallel multi-channel design. This means that they have 12 parallel channel receivers which are quick to lock onto satellites and maintain strong locks. They are considered extremely accurate even within urban environments with tall buildings.

The signals that satellites send out are called L1 and L2. These are low power radio signals. L1 is dedicated for civilians and has a frequency of 1575,42 MHz in the Ultra High Frequency (UHF) band. As mentioned the signals travel by line of sight, meaning they will pass through obstacles like clouds and plastic but struggles more with solid objects such as mountains and buildings. The signal contains 3 different elements. Ephemeris data, almanac data and a pseudorandom code. The ephemeris data contains important information about the status of the satellite (healthy or unhealthy), current time and date. The almanac data contains information about every satellite, showing the orbital information for all the satellites in the system. The pseudorandom code is a code that indentifies which satellite is transmitting information.

2.3.2 CELL PHONES AND GPS

After September 11 2001, the demand for GPS technology in cell phones increased. The U.S government pushed for implementing enhanced emergency calling which would show the location of the person in distress. There are basically two different ways of locating a cell phone. One is to use the towers and base stations which are arranged into a network of cells. A cell phone contains a low-power transmitter that allows it to communicate with the nearest tower. The base station then tracks your movement as you move from one cell to another by monitoring your phone's signal strength. So even without a GPS receiver, the location of the cell phone can be calculated based on its angel of approach to the cell towers, how long it takes the signal to travel to multiple towers and the strength of your signal when it reaches the towers. This is less accurate than GPS.

The cell phones with GPS usually have something called assisted GPS. This is a system which can improve the startup time of a satellite-based positioning system. It does this by using network resources to utilize the satellites faster as well as better in poor signal conditions. Signals bouncing off buildings, walls or trees are examples of poor signal conditions. This makes downloading the almanac and ephemeris data very difficult and time consuming because the receivers will only get fragmented signals.

Assisted GPS uses data available from a network to:

1. Quickly acquire satellites

The network can supply orbital (almanac) data for the satellites to the GPS receiver, which requires less transfer and quicker satellite locks. The network can also provide precise time.

2. Help calculate the position.

The server always has a good satellite signal and a lot more computation power than the cell phone. So it helps to calculate the position by comparing the fragmented signals it gets from the cell phone, with the satellite signal it receives directly.

The A-GPS also helps devices because the computational power required by the GPS device is reduced due to the fact that more calculations are done on the assistance server.

Cell phones also have the options to only use standalone GPS.

2.4 BUSTUC

This application will take advantage of a natural language bus route expert system developed at the Department of Computer & Information Science, NTNU. It was implemented first by Team Trafikk and later by ATB, the companies responsible for the buses in Trondheim and is considered to have reached the intelligence of a savant. Savant is defined as a term used for a person with certain social disabilities, but has some extraordinary skills when it comes to memory or speed calculations. Their natural language understanding is excellent within their domains of expertise, but lacking when it comes to other conversation topics. BusTuc is indeed very robust when it comes to queries about Trondheim's bus routes. The following examples are in Norwegian.

'Når går bussen fra Solsiden til Dragvoll?' This will return a travel route together with departure or arrival times. It will assume you want to travel as fast as possible unless you specify arrival or departure times.

'Når går første buss etter 15.00 fra Solsiden til Dragvoll?' This will return the first available travel router after 15.00.

The BusTUC system is separated into 3 different components:

- A parser system which consists of a parser, a grammar, a lexical processor and a dictionary
- A knowledge base (KB), divided into an application KB and a semantic KB
- A query processor, containing a routing logic system, and a route database.

The dictionary contains about 4590 words, 1204 station names, 80 buses and 9970 name variants only for the Norwegian part of the system. There are also over 5000 grammar rules, a semantic net with 6500 entries and 1500 more rules which help translate the output from the parser to a route database query language. The semantic knowledge base contains 960 nouns, 1100 verbs and 450 adjectives. All in all there are about 130500 lines of Prolog code. The parser uses a generalization of Definite Clause Grammars, called Consensical Grammar (CONtext SENSItive CompositionAL Grammar). Compositional grammar means that the semantics of the subphrases creates the semantics of a phrase.

The semantic knowledge base creates the foundation and the logic is generally generated by it. When there is a need for changes, these will be made in the semantic knowledge base, while the general grammar and dictionary remains untouched. The semantic knowledge base contains a restricted list of legal combinations of verb, nouns, adjectives and prepositions.

The query processor translates natural language into a form called Temporal Query Language which is a first order event calculus expression. The TQL expressions consist of predicates, functions, constants and variables. The TQL is then translated to BusLog, a route database query language. The goal of BusTUC is to automate the creation of new natural language interfaces for a well defined domain and with a minimum of precise programming.

3. METHODS

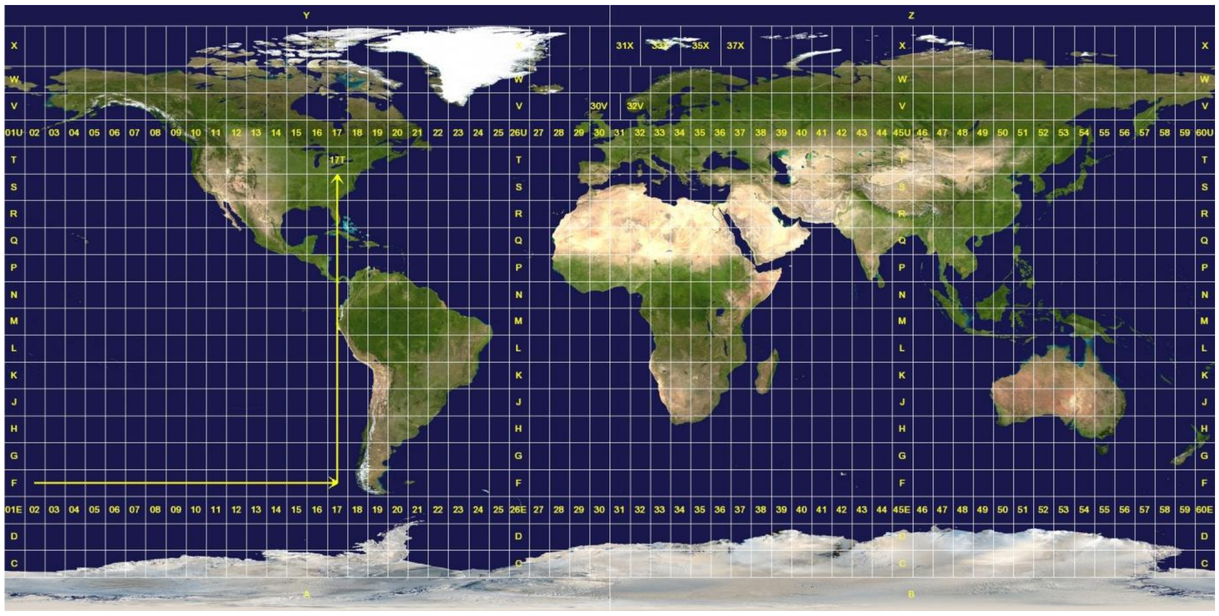
3.1 GPS

3.1.1 GEOGRAPHIC COORDINATE SYSTEM

The application uses Google Maps as a map which again uses the World Geodetic System (WGS) as a geographical coordinate system. WGS is a standard within cartography, geodesy and navigation. It comprises:

- A standard coordinate frame for the Earth
- A standard spheriodal reference surface (the datum or reference ellipsoid) for raw altitude data
- A gravitational equipotential surface (the geoid) that defines the nominal sea level

The coordinate frame is a standard Cartesian coordinate frame as pictured below.



3.1.2 PROVIDER

When calculations are done with real-time values, the process of deriving these values should be carefully analyzed. For the first part of this project the values we are concerned with is Global Positioning System (GPS) coordinates. To obtain our location the Android SDK offers 3 different ‘providers’ (as defined by the Android SDK). Each has their own advantages and limitations.

3.1.2.1 GPS

Derives a location by utilizing the device’s GPS chip and is highly accurate in doing so, but a line of sight is necessary for the satellites to provide a fix.

3.1.2.2 NETWORK

This provider uses the cellular network to provide a fast initial fix before utilizing the GPS chip. It is still very accurate and works without provide the satellites a line of sight.

3.1.2.2 PASSIVE

This method derives the location by Cellular ID / Wifi MAC ID look up. This does not require a GPS chip, but is significantly less accurate.

There is one problem with the methods given by the Android SDK to choose the correct provider. It will normally choose ‘GPS’ as the best possible provider, even though you are currently located inside a building. This will make the application search for a satellite link up indefinitely.

3.2 LOCATION MANAGER

This is the class which provides access to the system location services. These services allow applications to obtain periodic updates of the device’s geographical location. It makes it possible to specify the name of the provider with which to register, a minimum time and distance interval for notifications. This class is combined with a listener.

3.3 HTTP

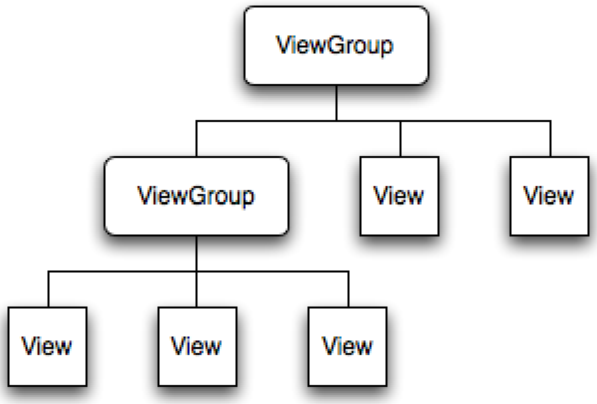
The method used to communicate with BusTuc is a HTTP Browser provided by Java. The common way for people to use BusTuc is through the web interface where they type in a query using the provided form. The form then sends the query to a perl script. This application, on the other hand, creates name-value pairs to simulate the form and sends a request directly to the perl script. The benefits of doing this instead of integrating the whole html page are many. It requires less traffic and takes up less space on the screen because only the answer is returned and shown.

3.4 BUS STOPS

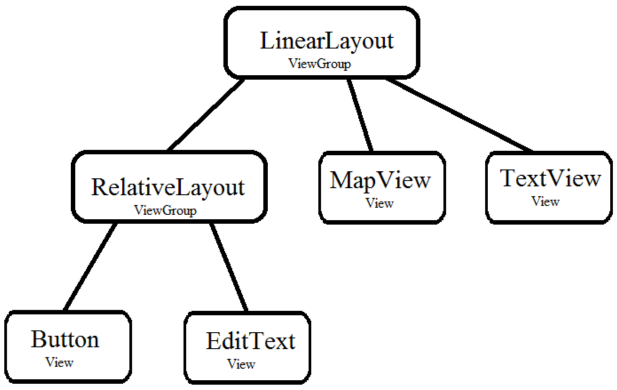
The list of bus stops used in development of this application was provided by ATB. It is included as a XML file in the package, and the application creates an array on startup. One problem with this list is that there is only 1 stop registered per location. So while almost every bus stop has its counterpart across the street (which handles traffic in the opposite direction), only one of these bus stops are registered with coordinates. The method for calculating the air distance between location A (where the user is) and location B (where the bus stop is) in meters is provided by Android SDK. These distances are also important when determining how much walking is required for the user. The walking speed used is 1.7 meters per second.

3.5 GUI

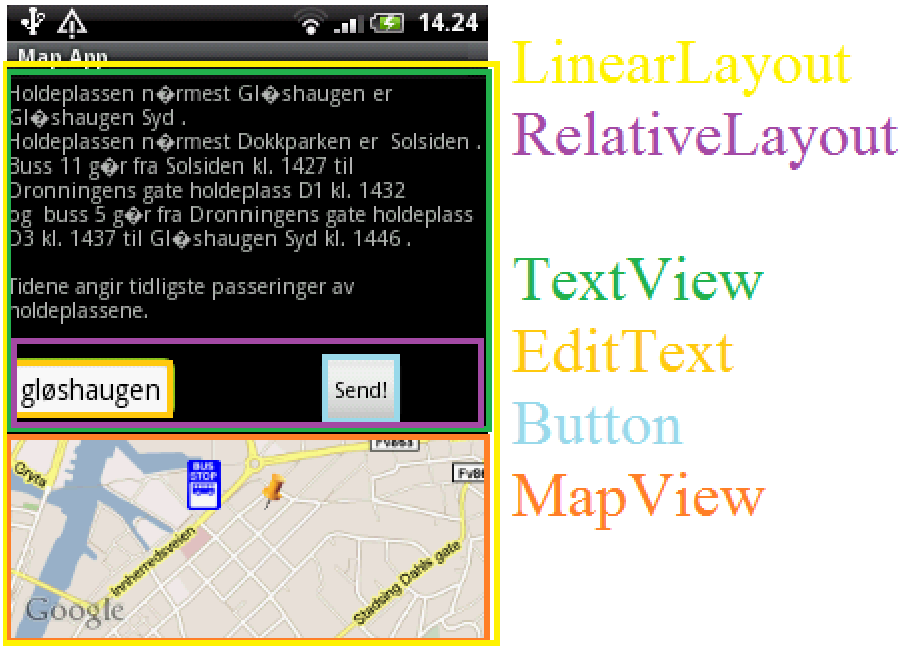
The user interface is defined by a hierarchy of View and ViewGroup nodes.



The view hierarchy and is expressed with an XML layout file. Each element in XML is either a View or a ViewGroup object (or a descendant thereof). View objects are leaves in the tree, ViewGroup objects are branches in the tree. In this application the view hierarchy is defined as such:



When implementing this view hierarchy as XML in the application, the result will look like this:



Holdeplassen n�rmest Gl�shaugen er
Gl�shaugen Syd .
Holdeplassen n�rmest Dokkparken er Solsiden .
Buss 11 g�r fra Solsiden kl. 1427 til
Dronningens gate holdeplass D1 kl. 1432
og buss 5 g�r fra Dronningens gate holdeplass
D3 kl. 1437 til Gl�shaugen Syd kl. 1446 .

Tidene angir tidligste passeringer av
holdeplassene.

One of the benefits Google's MapView is that it can be used as a canvas. The pin and bus stop sign is drawn upon MapView to create a visual representation of the locations in question. The icons used are located within the application package and represented as a variable within the development environment. The only necessary operation before drawing is translating the geographical coordinates to screen pixels.

4. RESULTS AND DISCUSSION

4.1 DISPLAYING MAPS

This application uses Google Maps API as the visual representation of personal position and the location of bus stops. The integration of Google Maps in Android applications is relatively easy (procedures explained in Appendix) and the functionality it provides is well documented. There weren't many problems at this point. The only thing required was internet access, may it be WiFi or cellular.

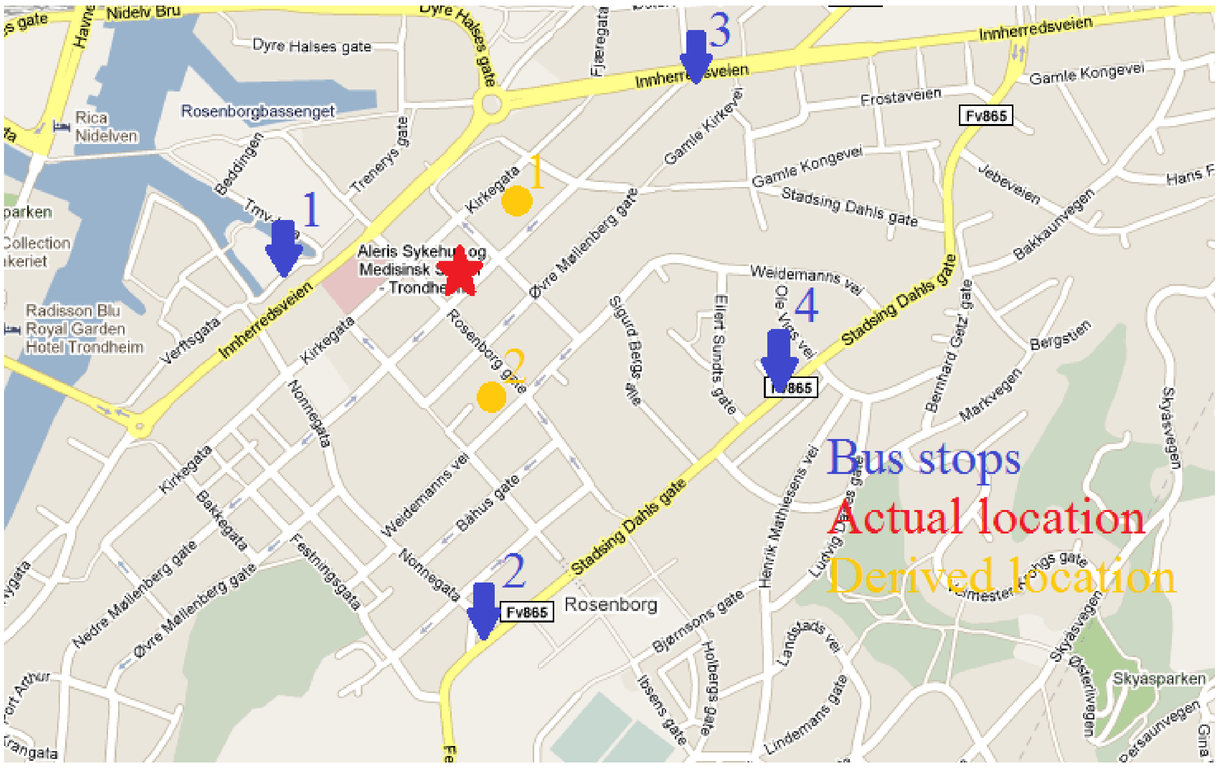
4.2.2 BUS STOPS

The 500 bus stop locations available in this application are described by 4 different characteristics. A unique integer, address, longitude and latitude are provided for each stop. As mentioned earlier the list of bus stops was formatted as an XML file located in the application package. The process of translating and parse the XML file into Location objects is handled when the application starts. The Location objects contain longitude, latitude and a name. These objects are needed for calculating the distance between the bus stops and the device's current location. The problem of only having 1 longitude and latitude pair for each bus stop location (e.i not one for each direction) is not really restricting the intelligence of the application at this point. I will explain the need for a more detailed bus stop list under the chapter of further research.

4.2.2 DEVICE LOCATION

As mentioned earlier, there are three different location providers available in the framework. This version of the application uses the network provider, which requires WiFi access. This is due to the fact that making an application which would prioritize providers after what type of resources it had at a given time, requires more testing than I had the time to do. Additional information about this is available under the chapter of further research.

There are some inconsistencies in deriving the device location due to the fact that the application uses the network provider. These happened when the device was inside and used assisted GPS for the initial satellite fix.

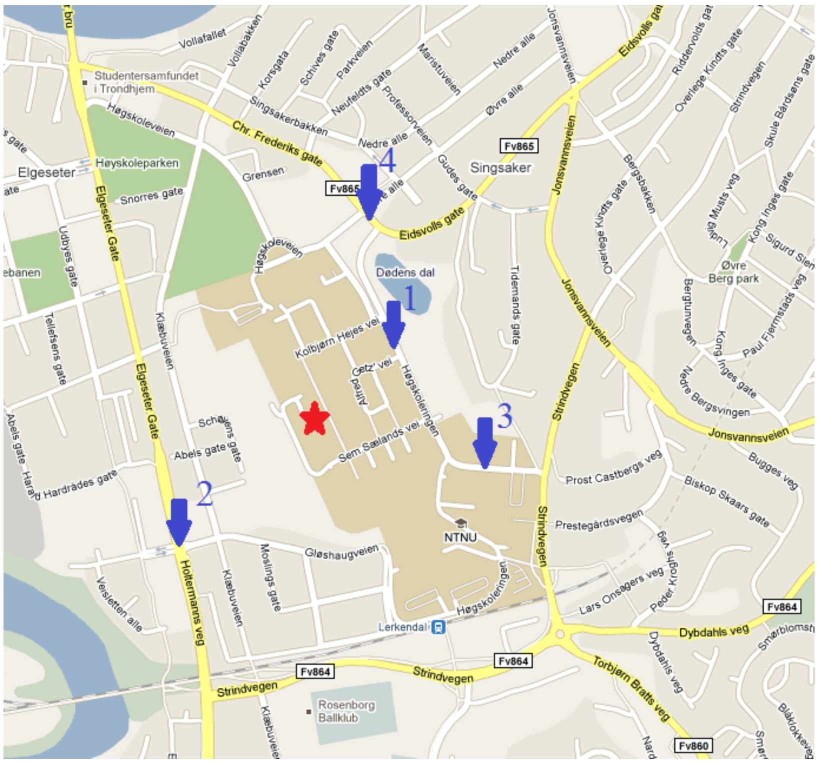


This is an overview over the results I've received when using this application from Nedre Møllenberg Gate 68, Trondheim. The red star represents the actual location. My returned locations (d1, d2) are represented by the orange dots and the bus stops (bs1,..,bs4) are the blue arrows.

Location d1 was returned 80 % of the time, and d2 20 %. This had implications on which bus stop the application regarded as the closest one. When the application reported d2 as its location, bs2 was considered the closest bus stop which is simply not true for my actual location. One thing worth mentioning here is that if my desired destination is 'Gløshaugen NTNU', the fastest way for me to travel is to walk to bus stop 2 due to the fact that bus stop 1 does not offer a direct route. These dilemmas further justify the need for better route advice intelligence. A new example will be provided below together with an intelligent solution.

4.3 FINDING RELEVANT BUS STOPS

As mentioned above, finding which bus stops that are relevant requires more than just locating the closest one. Consider point A (where the user is located) being Gløshaugen NTNU. There are here 4 different bus stops within a 500 meter radius.



The objective is then to create an algorithm robust enough to:

a. Get N closest bus stops.

b. Get all bus stops within a specified radius M.

This is required because not every bus stop offers the same route. In this example you would be at IDI's offices on NTNU campus. Your desired destination is Solsiden (North-East in Trondheim). The bus stops available to you within a 500 meter radius is Gløshaugen Nord (1), Professor Brochs gate (2), Gløshaugen Syd (3) and Høyskoleringen (4). What variables do our calculations need to have in order to find the quickest route possible? First would be distance to the bus stop. This varies from bus stop to bus stop and definitely impacts a user's decision. Second would be bus travel time. Would a penalty to a route's total score be in order if you would need two take multiple buses? The third value we would need to consider is time from the final bus stop to your wanted destination. In this example it could be an option to take the bus from Høyskoleringen (4) to Rosenborg skole, which is about 7 minutes walking time from

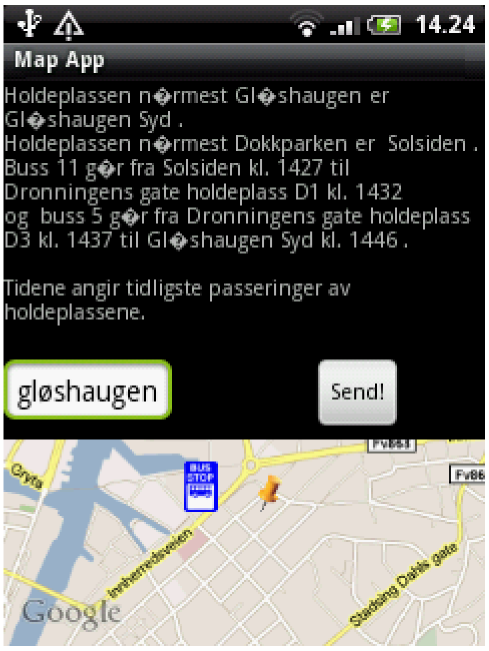
Solsiden. The third criteria though, are tougher and will be discussed in the chapter on further research.

Walking time in this application assumes that a user walks at 1.7 meter per second which is defined as the average walk speed. And since we already have air distance from current position to the bus stops in question (restricted by N closest or radius M), the calculation is not hard. To calculate distance by air and not actual travel distance is a clear limitation, and there are examples where this could provide false results, but this dilemma is not under consideration at this point.

The travel time by bus is information held by BusTuc. The application will be responsible to send a formatted list with relevant bus stops together with walking time, and BusTuc will be responsible for calculating each bus stop's total score. Discussion on how this will be done and how real-time data will affect the score is written in the chapter on further research.

4.4 INTEGRATING BUSTUC

The application integrated BusTuc by using the HTTP methods discussed earlier. The simplest use of BusTuc in this application is to provide BusTuc with two bus stops (A and B) and ask for a route. This is done by simply taking the closest bus stop and the destination typed in by the user and forms a natural language query 'from A to B'. You can specify the query by adding restrictions or limitations to the query (e.g 'fra A til B før (before) 15.00). This is useful because it does not require the user to specify where he wants to travel from. The application assumes he wants to travel from the nearest bus stop.


14.24
Map App

Holdeplassen n�rmest Gl�shaugen er Gl�shaugen Syd .
Holdeplassen n�rmest Dokkparken er Solsiden .
Buss 11 g�r fra Solsiden kl. 1427 til Dronningens gate holdeplass D1 kl. 1432
og buss 5 g�r fra Dronningens gate holdeplass D3 kl. 1437 til Gl�shaugen Syd kl. 1446 .

Tidene angir tidligste passeringer av holdeplassene.

gløshaugen
Send!

But this is not always the smartest or fastest way to get from one point to another as previous examples have shown. The application needs to provide BusTuc with more information about the user's current situation. Therefore the query sent to BusTuc will be formatted like this:

('B1'+W1,'B2'+W2,....Bn+Wn) destination

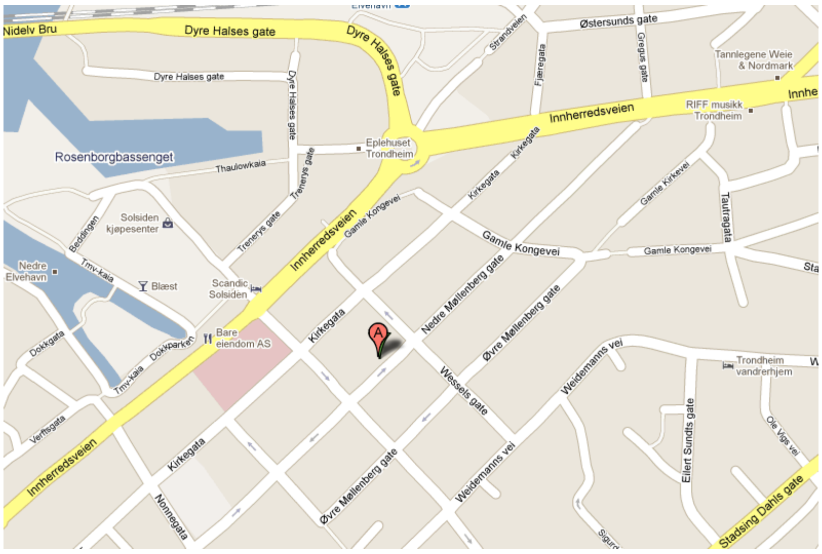
Where B1,B2,...Bn are relevant bus stops limited by distance radius and W1,W2,..,Wn is walking time from the user's current location to the bus stop. The list of bus stops is sorted after shortest walking time.

BusTuc will need an extension to parse this query. It currently handles it as ‘from B1 to destination’.

One problem with BusTuc and the current bus stop list is that the names of the bus stops are not always the same. For instance, the bus stop list calls the bus stop in the picture above for ‘Dokkparken’, while BusTuc calls it ‘Solsiden’. This has not created any problems while testing the application, but it is not unimaginable to think that it can.

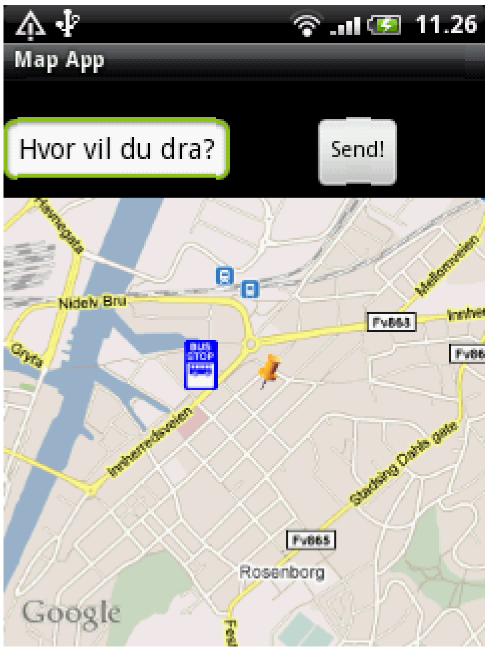
4.5 EXAMPLE

In this example the initial position of the device is at Nedre Møllenberg Gate 68, 7043 Trondheim. This is in the North-East region of Trondheim.



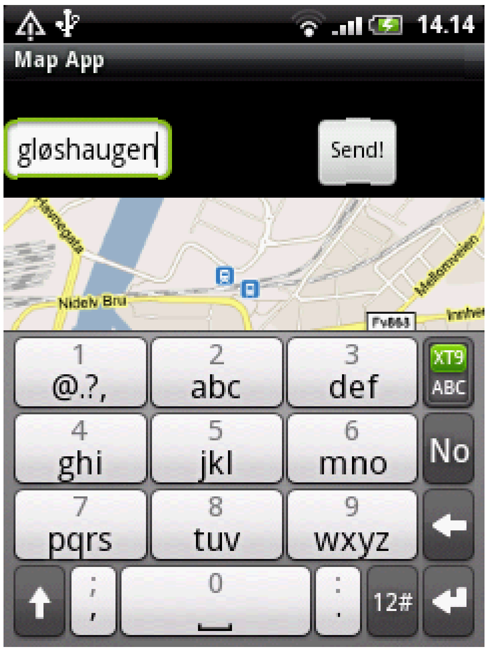
The precise location shown on Google Maps

After starting the application, it will automatically pin point the location of the device. This with the accuracy previously discussed.



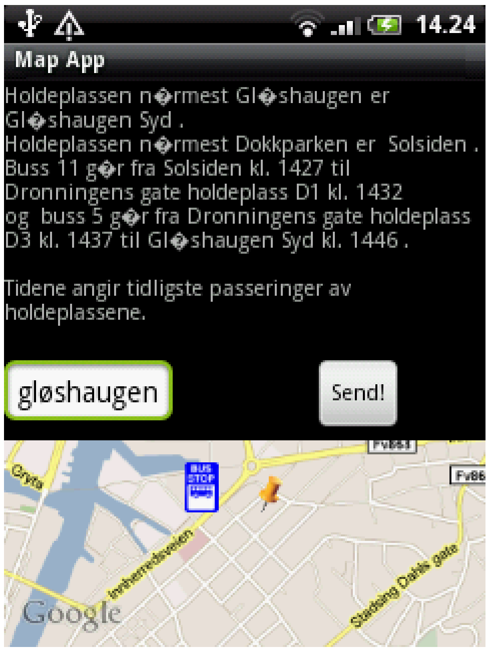
The location shown on the application

We can see here that the location is off about 50 meters. The importance of taking multiple bus stops into consideration is again evident. At the top of the screen the user interface towards BusTuc is visible with the caption 'Hvor vil du dra?' (Where do you want to go?). The closest bus stop is also shown on the map, represented by a Bus Stop sign. In this case the bus stop is 'Dokkparken'.



The user types in his destination

After the user has provided his destination and pressed 'Send!' the application will now create a query in natural language (Norwegian) for BusTuc. In this example the query will be 'fra Dokkparken til Gløshaugen'. It is also possible to add time restrictions to the query by saying 'før 15.00' or 'etter 15.00'.



Holdeplassen n�rmest Gl�shaugen er Gl�shaugen Syd .
Holdeplassen n�rmest Dokkparken er Solsiden .
Buss 11 g�r fra Solsiden kl. 1427 til Dronningens gate holdeplass D1 kl. 1432
og buss 5 g�r fra Dronningens gate holdeplass D3 kl. 1437 til Gl�shaugen Syd kl. 1446 .

Tidene angir tidligste passeringer av holdeplassene.

The application receives and formats BusTuc data.

5. FURTHER RESEARCH

5.1 REAL-TIME DATA

ATB (the company responsible for bus transport in Trondheim) will present their real-time system in December 2010. A goal should be to incorporate these real-time values in this Android application. How these values will be made available is tough to say, but there is probably possible to make a HTTP call and acquire them. The real-time data will then have an impact on the total score a route will get. For example, a user stands on a bus stop and asks for the quickest way to get across town. The bus was supposed to go 2 minutes earlier, but there's been a delay. Without having any knowledge about the delay, the system will give the user wrong information or even advice him to go to another bus stop.

5.2 PROVIDERS

Some testing with providers is required to always equip the application with the best available device location. There are a lot of variables which interferes or restricts the availability of services like line of sight, wifi connection, etc.

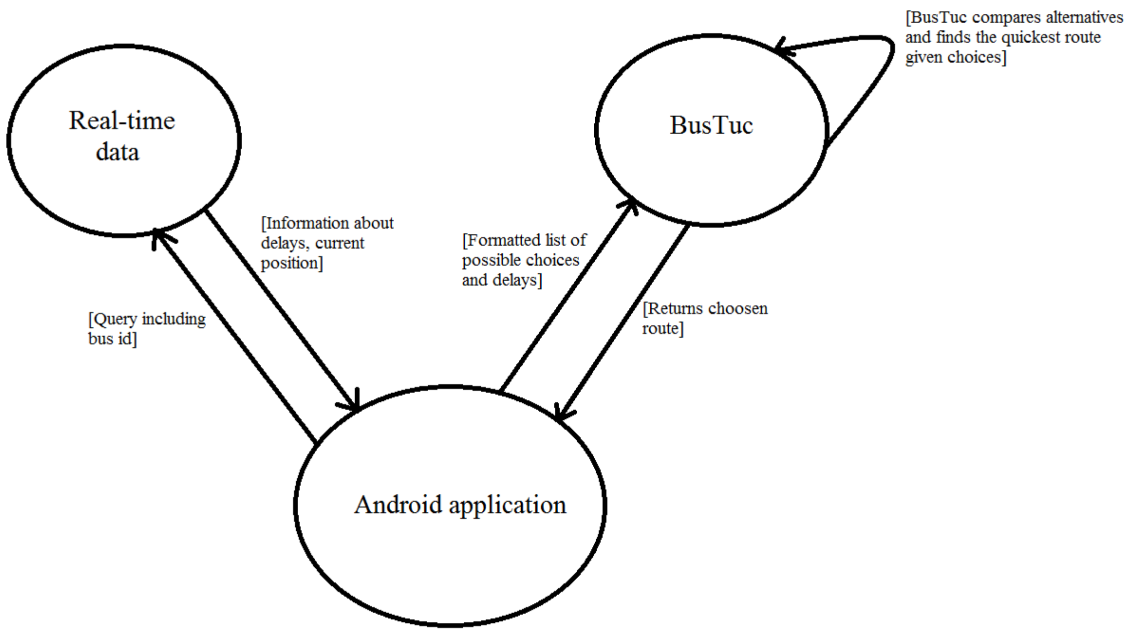
5.3 BUS STOPS

When extending the functionality of this application a demand for a better detailed bus stop lists. They should contain direction, route number, etc.

5.4 COMPARE ROUTES

When ATB makes real-time data available, the method for finding the best route will have to be extended. This information will probably be sent to BusTuc for comparison, so it is necessary to implement methods for handling the data sent from the application.

One way of handling the communication is provided below.


Real-time data
BusTuc
Android application
[BusTuc compares alternatives and finds the quickest route given choices]
[Information about delays, current position]
[Query including bus id]
[Formatted list of possible choices and delays]
[Returns choosen route]

I can only assume that the interface ATB will provide in the future, will give the option of checking delays along a bus route or with a special bus id. The bus stop list used in the application would have to be adapted to fit ATB's requirements. There also might be necessary to send multiple queries to the ATB server. If you are considering routes from 5 different bus stops, all of these bus stops would have to be checked for delays. Then again, it is natural to assume that a lot of the same buses will stop at adjacent bus stops. An intelligent way to handle this will be necessary. After data is received from ATB a formatted query can be produced for BusTuc. An example would be:

('Bus stop 1'+walking distance + potential delay, 'Bus stop 2'+ walking distance + potential delay) query string.

5.6 TESTING

This application has not undergone a great deal of testing. Exceptions will generally be caught, but a lot more effort and time must be invested before it can be considered a commercially ready product. Testing for mobile application is worse than regular software, since emulators and API docs are only marginally useful. They do not give you any idea what really happens to software running on a real network or even a real device. Experienced mobile application developers say that if your application works perfectly on the emulator, you have only just begun.

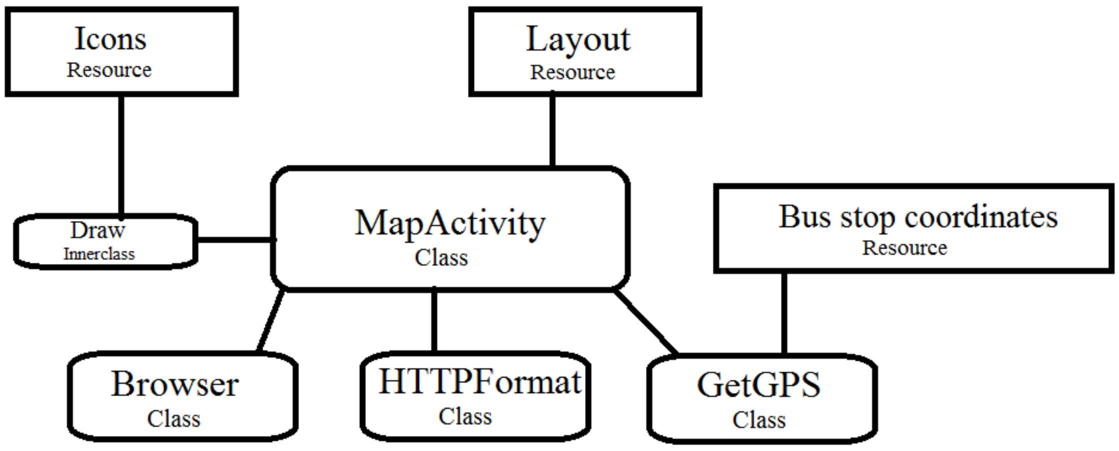
This is mostly because of the vast differences between computers and cell phones. Screen size & form factor divergence, carrier limitation, etc.

So an extensive and systematic testing procedure is needed before unleashing it into the big world of mobile applications.

6. IMPLEMENTATION

The implementation of the application is split into two different categories, code and resources. The code contains the java files, while the resources contain the XML file for the bus stops and layout.

6.1 ARCHITECTURE



This is a rough overview of the architecture. The architecture is modeled after the Model-View-Controller, where the support classes and resources is model, MapActivity is controller and view is defined by MapActivity and manipulated by the Draw innerclass.

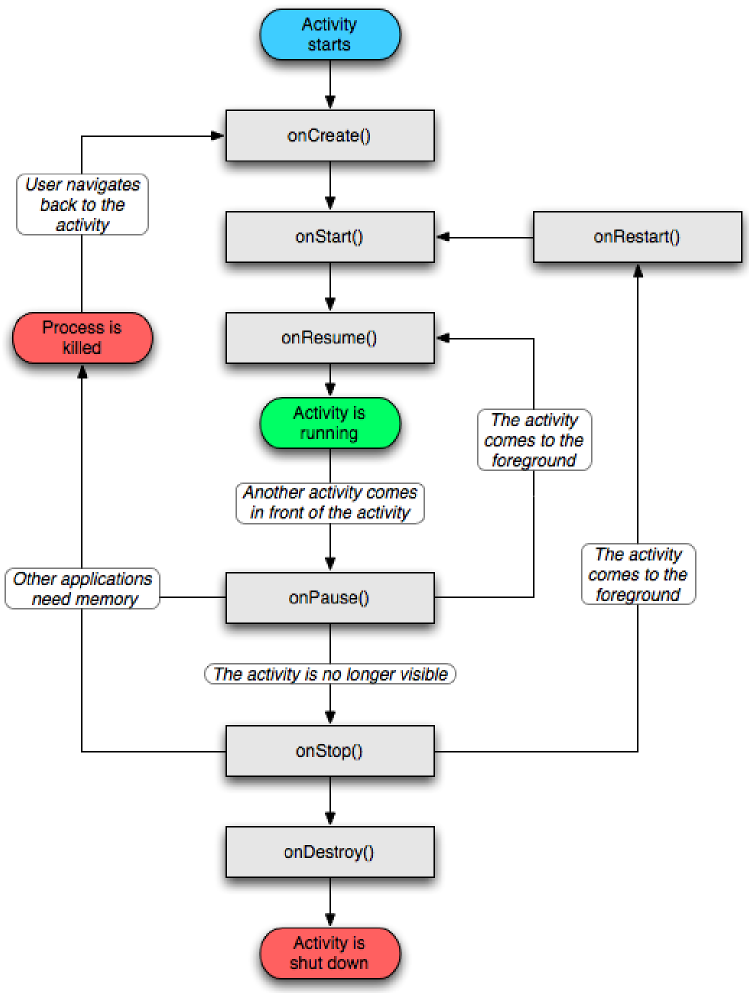
6.2 MAPACTIVITY

MapActivity is responsible for the communication between the objects and runs the ‘Activity’. An activity is defined by Android as a single, focused thing that the user can do. So an application can have multiple activities, but only one running at the time. Activities in the system are managed as an activity stack. When a new activity is started, it is placed on top of the stack and becomes the focused activity and previous activities will be pushed down until the new activity dies.

6.2.1 ACTIVITY

The activity lifecycle is defined by 4 different states:

- Active or running. The activity is in the foreground of the screen
- Paused. It has lost focus, but is still visible. The activity is considered alive, but can be killed by the system in extreme low memory situations.
- Stopped. The application is no longer visible to the user, but retains all state and member information. Generally killed when memory is needed elsewhere.
- Killed. The system can drop the activity from memory by either asking it to finish, or simply killing its process. When displayed again, the program is completely restarted.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Activity lifecycle

The MapActivity class contains some of these methods to make sure the application behaves properly. A quick overview of the different methods follows:

- onCreate() : This method is called when the activity is first created. This is where the application should bind data to lists, create views, etc.
- onStart(): This method is called when the user can see the activity.
- onResume(): This method is called when the activity interacts with the user. The activity is at the top of the activity stack and input from user is available.
- onPause(): This method is called when a previous activity is about to be the focus of the system. Here you can save data, stop animations etc.
- onStop(): This method is called when the activity is no longer visible to the user, because another activity is covering it. Followed either by onRestart() or onDestroy().
- onDestroy(): This method is called right before the activity is destroyed.

In this application

- onCreate() creates the views, sets the layout and binds the data from the bus stop XML file to a String array. The string array which contains bus stop id, name, latitude and longitude is then split so these values have their own string[x][y] position. This method also starts the GPS location manager, which configures the location service. An event

listener for both location and button is created. You can find more information about these listeners further down.
- onResume() set the restrictions on how often the locationManager should request location updates. In this project it checks if the position has changed more than over one meter, every second. If the position fulfills the criteria set by location manager, the location listener will trigger an event.

6.2.2 METHODS

Most of the functionality in this application is event based. The two events are physical movement and button push.

LocationListener contains 4 different methods.

- onLocationChanged(): This method runs when the device moves more than 1 meter (also runs when there is not an intial position). It gets the current location and retrieves the closest bus stop. These two coordinates will be converted into geographical points, which again are used by the draw class. A HashMap is created with all the coordinates and used by the method responsible for selecting all the relevant bus stops. Those bus stops are then saved in a new HashMap to be used by the button listener.
- onProviderDisabled(): writes to log.
- onProviderEnabled(): writes to log.
- onStatusChanges(): writes to log.

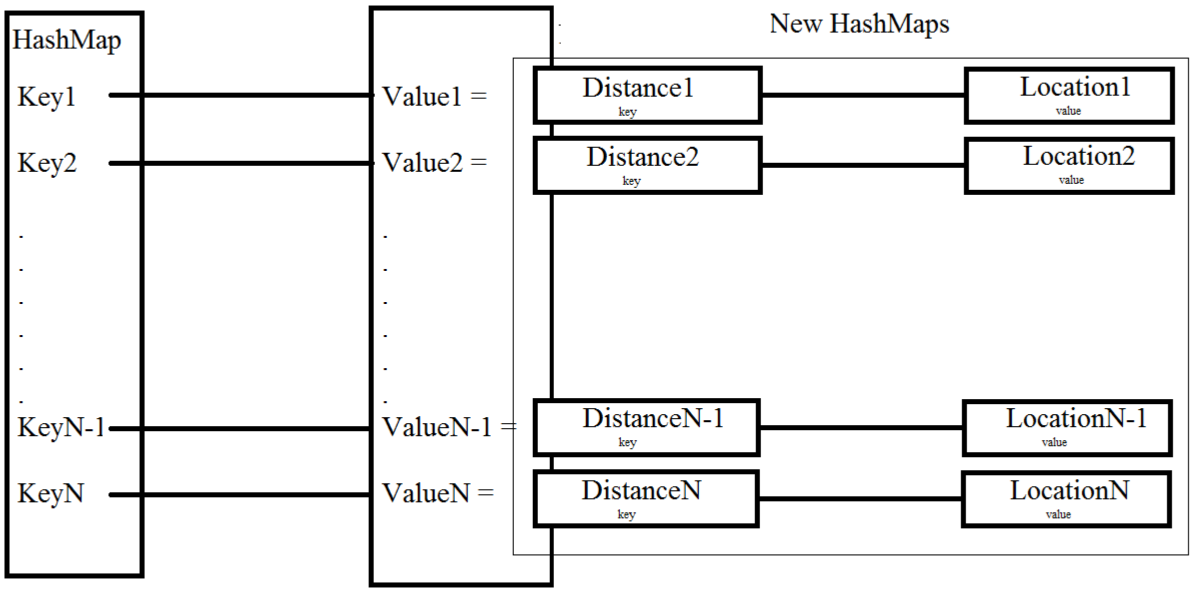
The buttonListener only contain one method.

- onClick(): This method starts the communication with BusTuc. It creates a Browser object, which receives the HashMap with the relevant bus stops and the text written in the edittext box. The browser object runs the HTTP request and returns the answer from. The answer is then formatted, added to the TextView and made visible to the user.

Another method from MapActivity which I would like to explain is the one used by locationListener onLocationChanged, which sorts out the relevant bus stops and puts them into a HashMap used for the BusTuc query.

It is called m_partialsort and takes a HashMap, two integers (one for specifying N closest bus stops and one for specifying radius R which sets the maximum distance a bus stop can be located from the user's position) and a Boolean.

A HashMap is a key-value pair. Where you can find values by using get(key). In this application the HashMap maps a unique integer (key) to a new HashMap object (value). This new HashMap object has the air distance between the bus stop and the user's current position as key and the location object as value (containing latitude, longitude and name of the bus stop).



This HashMap will be iterated through. For each key, the value will be extracted, which is a new HashMap. The key in that hashmap is the distance we want to compare. So we either save all the HashMaps which has a key under the pre-set radius R, or add the N HashMaps with the smallest key. TreeMap has been used to get rid of the biggest value added when a smaller value appears. The benefit of this 'partial sort' is that we only have to iterate through the HashMap once.

APPENDIX

HOW TO SET UP ANDROID DEVELOPMENT

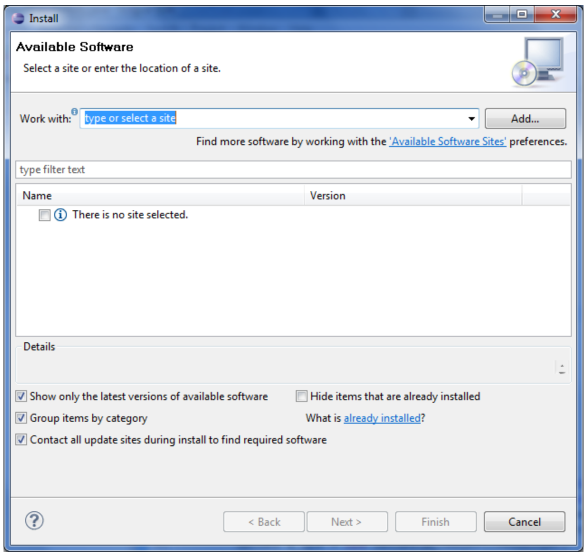
1. Setting up system

You will need to have JDK (http://java.sun.com/javase/downloads/index.jsp . Get version 5 or 6) and Eclipse (http://www.eclipse.org/downloads/ version 3.4 ++).

2. Installing Android SDK

Download Android SDK from http://dl.google.com/android/android-sdk_r07-windows.zip . Unzip and add the location to your PATH.

3. Open eclipse. Press 'Help' and 'Install New Software'. This will open the 'Install window' as shown below.


Install

Available Software

Select a site or enter the location of a site.

Work with: type or select a site

Find more software by working with the 'Available Software Sites' preferences.

type filter text

Name	Version
☐ There is no site selected.	

Details

☑ Show only the latest versions of available software
☐ Hide items that are already installed
☑ Group items by category
What is already installed?
☑ Contact all update sites during install to find required software

Now add https://dl-ssl.google.com/android/eclipse/ and install the Android Development Tool (ADT). Restart eclipse.

4. Point eclipse to the Android SDK folder by pressing 'Window' -> 'Preferences' -> 'Android' and add your SDK folder.

5. Open 'Android SDK and AVD Manager', press 'Available packages' and add https://dl-ssl.google.com/android/repository/repository.xml to get access to the latest Google API's. This is necessary for Google Maps.

6. Create a new virtual drive where 'Google API' is the target.

7. Create or load an Android project.

CODE

MapActivity.java

```java
package test.Map;

import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.TreeSet;

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

import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Canvas;
import android.graphics.Point;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
```

```
public class MapsActivity extends MapActivity
{
        MapView mapView; // Google Maps
        String[][] gpsCords;  // Array containing bus stops
        MapController mc; // Controller for the map
        List<String> prov; // List of providers
    GeoPoint p,p2; // p is current location, p2 is closest bus stop.
    GetGPS k_gps; // Object of the GetGPS class.
    Location currentlocation, busLoc; // Location objects
    HashMap<Integer,Location> tSet; // HashMap used for finding closest locations
    LocationManager locationManager; // Location Manager
    String provider; // Provider
    TextView myLocationText;
    LocationListener locationListener;
    /** Called when the activity is first created. */

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        mapView = (MapView) findViewById(R.id.mapView);
        myLocationText = (TextView)findViewById(R.id.myLocationText);
        myLocationText.setText("");
        LinearLayout zoomLayout = (LinearLayout)findViewById(R.id.zoom);
        // Gets the coordinates from the bus XML file
        String[] myImageFileEndings = getResources().getStringArray(R.array.coords);
        GetGPS k_gps = new GetGPS(myImageFileEndings);
        // Formats the bus coordinates
        gpsCords = k_gps.fCords();

        View zoomView = mapView.~~getZoomControls~~();

        zoomLayout.addView(zoomView,
            new LinearLayout.LayoutParams(
                LayoutParams.WRAP_CONTENT,
                LayoutParams.WRAP_CONTENT));
        mapView.displayZoomControls(true);

        mc = mapView.getController();
        // Creates a locationManager.
        locationManager = (LocationManager) getSystemService(LOCATION_SERVICE);
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        provider = locationManager.getBestProvider(criteria, true);
        Log.v("provider","provider:"+ provider);

        // Creates a locationListener
        locationListener = new LocationListener() {

         // This method runs whenever the criteria for change is met.
            public void onLocationChanged(Location location) {
                 currentlocation = location;

                 Log.v("currentLoc","PROV:LOC=" +
currentlocation.getLatitude()+":"+currentlocation.getLongitude());
               // finds the closest bus stop
                 busLoc = closestLoc(gpsCords);
                 // creates a HashMap containing all the location objects
                HashMap<Integer,HashMap<Integer,Location>> locationsArray = getLocations(gpsCords);
                Log.v("sort","returnedHmap:"+locationsArray.size());
                // creates a HashMap with all the relevant bus stops
                tSet = m_partialSort(locationsArray,0,500,false);
                Log.v("sort","returnedtSet"+tSet.size());
                // adds the closest bus stop as a GeoPoint
                p2 = new GeoPoint(
                        (int) (busLoc.getLatitude() * 1E6),
                        (int) (busLoc.getLongitude() * 1E6));
                Log.v("GEOPOINT","GEO1" + p2.getLatitudeE6() + ":" + p2.getLongitudeE6());
                // add the current location as a GeoPoint
                p = new GeoPoint(
                        (int) (currentlocation.getLatitude() * 1E6),
                        (int) (currentlocation.getLongitude() * 1E6));
                Log.v("GEOPOINT","GEO2" + p.getLatitudeE6() + ":" + p.getLongitudeE6());

                MapOverlay mapOverlay = new MapOverlay();
                List<Overlay> listOfOverlays = mapView.getOverlays();
                listOfOverlays.clear();
                listOfOverlays.add(mapOverlay);
                mc.animateTo(p);
                mc.setZoom(15);

            }

                          @Override
                          public void onProviderDisabled(String provider) {
                                  Log.v("PROV","PROV:DISABLED");
                                  // TODO Auto-generated method stub
                        }

                        @Override
                        public void onProviderEnabled(String provider) {
                                Log.v("PROV","PROV:ENABLED");

                        }

                        @Override
                        public void onStatusChanged(String provider, int status,
                                        Bundle extras) {
                                Log.v("PROV","PROV:STATUSCHANGE");

                        }
         };
        // adds button
        final Button button = (Button) findViewById(R.id.Button);
        // adds edittext box
        final EditText editTe = (EditText) findViewById(R.id.eText);
        // binds listener to the button
        button.setOnClickListener(new OnClickListener() {
             public void onClick(View v) {
                 // Perform action on clicks
                  if(!tSet.isEmpty())
                        {
                    Browser m_browser = new Browser();
                    // Creates a request to BusTuc
                   String[] html_page = m_browser.getRequest(tSet,editTe.getText().toString(),false);

                   StringBuilder str = new StringBuilder();
                   // Parses the returned html
                   for(int i = 0;i<html_page.length;i++)
                   {
                   Log.v("CONTENT"+i, html_page[i]);
                   if(!html_page[i].contains("</body>"))
                   {
                   str.append(html_page[i] + "\n");
                   }
                   }
                   // Sets the html in the text view
                   myLocationText.setText(str.toString());
             }
             }
         });
    }
    @Override
        protected void onResume() {
                super.onResume();
            // Sets the restrictions on the location update.
                locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, 100, 1,
locationListener);

         }
    // Method used to find the closest BusStop.
    private Location closestLoc(String[][] k_gpsCords)
    {
       String tempCords[][] = k_gpsCords;
       int clength = tempCords.length;
       Log.v("CORDL", "C:"+clength);
     Location closestLocation[] = new Location[clength];
       for(int i = 0;i<clength;i++)
       {
            closestLocation[i] = new Location(provider);
            closestLocation[i].setProvider(tempCords[i][1]);
            closestLocation[i].setLatitude(Double.parseDouble(tempCords[i][3]));
            closestLocation[i].setLongitude(Double.parseDouble(tempCords[i][2]));
       }
       Location closLocation = new Location(provider);
       Location secClosLocation = new Location(provider);
       Log.v("currentLoc", currentlocation.getLatitude()+":"+currentlocation.getLongitude());
       float mindis = 100000000;
       for(int y = 0; y<clength;y++)
       {
            float dis = closestLocation[y].distanceTo(currentlocation);
            if(dis <= mindis)
            {
                     mindis = dis;
                     secClosLocation = closLocation;
                     closLocation = closestLocation[y];

         //          Log.v("newClosestLoc", closLocation.getLatitude()+":"+closLocation.getLongitude());
            }
            dis = 0;
       }
       mindis = 100000000;

       return closLocation;
    }
    // Creates the HashMap for the locations.
    public HashMap<Integer,HashMap<Integer,Location>> getLocations(String[][] k_gpsCords)
    {
         String tempCords[][] = k_gpsCords;
        int clength = tempCords.length;
        Log.v("CORDL", "C:"+clength);

        HashMap<Integer,HashMap<Integer,Location>> newMap = new
HashMap<Integer,HashMap<Integer,Location>>();
        HashMap<HashMap<Integer,Location>,Integer> testMap = new
HashMap<HashMap<Integer,Location>,Integer>();
        Location closestLocation[] = new Location[clength];
        for(int i = 0;i<clength;i++)
        {
           closestLocation[i] = new Location(provider);
           closestLocation[i].setProvider(tempCords[i][1]);
           closestLocation[i].setLatitude(Double.parseDouble(tempCords[i][3]));
           closestLocation[i].setLongitude(Double.parseDouble(tempCords[i][2]));
           int distance = (int)closestLocation[i].distanceTo(currentlocation);
           HashMap<Integer, Location> hMap = new HashMap<Integer,Location>();
           hMap.put(distance, closestLocation[i]);
           newMap.put(i, hMap);

        }
        return newMap;
    }
    // Finds either N closest bus stops or the bus stops within a radius M
    public HashMap<Integer,Location> m_partialSort(HashMap<Integer,HashMap<Integer,Location>> lHMap, int i,
int m, boolean maxLoc)
    {
         HashMap<Integer,HashMap<Integer,Location>> newMap = lHMap;
         Log.v("sort","lHMap:"+lHMap.size());
         HashMap<Integer,Location> finalMap = new HashMap<Integer,Location>();
         TreeSet<Integer> minValues = new TreeSet<Integer>();
         for(int y = 0;y<newMap.size();y++)
         {
                 int currentValue = Integer.parseInt(newMap.get(y).keySet().toArray()[0].toString());
                 if(minValues.size() < i && maxLoc)
                 {
                    minValues.add(currentValue);
                    finalMap.put(currentValue, newMap.get(y).get(currentValue));
                 }
                 else if(!maxLoc && currentValue < m)
                 {
                         minValues.add(currentValue);
                         finalMap.put(currentValue, newMap.get(y).get(currentValue));
                 }
                 else if(maxLoc)
                 {
                         if(currentValue < minValues.last())
                         {
                                 minValues.remove(minValues.last());
                                 finalMap.remove(minValues.last());
                                 minValues.add(currentValue);
                                 finalMap.put(currentValue, newMap.get(y).get(currentValue));
                         }
                 }
         }
         Iterator<Integer> iter = minValues.iterator();
         while(iter.hasNext())
         {
            int next = iter.next();
            Log.v("sort","TreeValue"+next);
            Log.v("finalmap","finalmap="+finalMap.get(next).getProvider()+":"+next);
         }
         return finalMap;
    }

    @Override
    protected boolean isRouteDisplayed() {
        return false;
    }
    // Method used to find the coordinates to the current location
    public String[] showLocation(Location location) {

         String[] currentLocation = new String[2];

                         if (location != null) {
                                 String lat = Double.toString(location.getLatitude());
                                 String lng = Double.toString(location.getLongitude());
                                 Log.v("LAT", "latitude:"+lat);
                                 Log.v("Longitude", "longitude:"+lng);
                                 currentLocation[0] = lat;
                                 currentLocation[1] = lng;
                                 Log.v("GPS", currentLocation[0]+":"+currentLocation[1]);
                         } else {
                                 currentLocation[0] = "-1";
```

```
                                  currentLocation[1] = "-1";
                                  Log.v("LOC","Location not available.");
                          }
                          return currentLocation;
         }
     // The class which draws on the map
     class MapOverlay extends com.google.android.maps.Overlay
     {

         @Override
         public boolean draw(Canvas canvas, MapView mapView,boolean shadow, long when)
         {
             super.draw(canvas, mapView, shadow);
             Point startPts = new Point(0,0);
             Point busPts = new Point(0,0);
             try
             {
             // translate the GeoPoint to screen pixels
             startPts = new Point();
             // Adds the current location
             mapView.getProjection().toPixels(p, startPts);
             busPts = new Point();
             // Adds the closest bus stop
             mapView.getProjection().toPixels(p2, busPts);
             } catch (Exception e)
             { Log.v("DRAW", "MapView getProjection:"+e.toString()); }

             // add the marker
             Bitmap bmp = BitmapFactory.decodeResource(
                 getResources(), R.drawable.pp);
             canvas.drawBitmap(bmp, startPts.x, startPts.y-50, null);
             Bitmap bmpBus = BitmapFactory.decodeResource(
                     getResources(), R.drawable.s_busstop2);
                 canvas.drawBitmap(bmpBus, busPts.x, busPts.y-50, null);
             return true;
         }

     }

}
```

GetGPS.java

```
package test.Map;

public class GetGPS
{
        String[] unFormatedGPSCords;
        String[][] formatedGPSCords;
        public GetGPS(String[] p_UnFormatedGPSCords)
        {
                unFormatedGPSCords = p_UnFormatedGPSCords;
        formatedGPSCords = new String[unFormatedGPSCords.length][4];
        }
        public String[][] fCords(){

         for(int i=0;i<unFormatedGPSCords.length;i++)
         {
                 String startString = unFormatedGPSCords[i];
                 String[] line = startString.split("\\,");
                 for(int y=0;y<line.length;y++)
                 {
                 formatedGPSCords[i][y] = line[y].trim();

                // Log.v(TAG, line[y]);
                 }
         }
     return formatedGPSCords;
        }
}
```

HTTPFormat.java

```
package test.Map;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;

import org.apache.http.HttpResponse;

public class HttpFormat {
    HttpFormat()
    {
```

```java
    }
    public String[] request(HttpResponse response){
        String result = "";
        String[] contentArray = null;
        try{
            InputStream in = response.getEntity().getContent();
            BufferedReader reader = new BufferedReader(new InputStreamReader(in));
            StringBuilder str = new StringBuilder();
            StringBuilder content = new StringBuilder();
            String line = null;

            while((line = reader.readLine()) != null){
                 if(line.endsWith("</body>"))
                 {
                         contentArray = line.split("<br>");
                         content.append(line + "\n");
                 }
                str.append(line + "\n");
            }
            in.close();
            result = str.toString();
        }catch(Exception ex){
            result = "Error";
        }

        return contentArray;
    }
}
```

Browser.java

```java
package test.Map;

import java.io.IOException;
import java.text.DecimalFormat;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;

import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.message.BasicNameValuePair;

import android.location.Location;
import android.util.Log;
import android.webkit.WebView;

public class Browser
{
        WebView web;
        HttpClient m_client;
        HttpFormat httpF;
        Browser()
        {
                m_client = new DefaultHttpClient();
                httpF = new HttpFormat();
        }
        String[] getRequest(HashMap<Integer,Location> startMap, String stop, Boolean formated)
        {
                String[] html_string = null;
                Log.v("BUSTUCS","startmapsize:"+startMap.size());
                Iterator<Integer> iter = startMap.keySet().iterator();
                String start = "(";
                DecimalFormat decifo = new DecimalFormat("###");
                int highestNumber = 1000;
                String wantedStart = "";
                String start2 = "(";
                Object[] keys = startMap.keySet().toArray();
                Arrays.sort(keys);
                int hSize = startMap.keySet().size();
        for(int i = 0;i<hSize;i++)
        {
           String output2 = decifo.format(Math.ceil((Double.parseDouble(keys[i].toString())/1.7)/60));
            start2 = start2 + "'" + startMap.get(keys[i]).getProvider()+"'"+"+"+output2;
            if(i+1<hSize)
            {
                    start2 = start2 + ",";
            }
            Log.v("OUTPUT","outputStart2:"+start2);
        }
        start2 = start2 + ")";
                String wanted_string = start2 + stop;
                String wanted_string2 = "fra "+wantedStart+" til "+stop;
                Log.v("BUSTUCSTR", "wanted_string:"+wanted_string);
                HttpPost m_post= new HttpPost("http://www.idi.ntnu.no/~tagore/cgi-bin/busstuc/busq.cgi");
                try {
                        List<NameValuePair> nameValuePairs = new ArrayList<NameValuePair>(2);
                nameValuePairs.add(new BasicNameValuePair("lang", "eng"));
                nameValuePairs.add(new BasicNameValuePair("quest", wanted_string));
                m_post.setEntity(new UrlEncodedFormEntity(nameValuePairs));

                        HttpResponse m_response = m_client.execute(m_post);
                        html_string = httpF.request(m_response);
                } catch (ClientProtocolException e) {
                        Log.v("CLIENTPROTOCOL EX", "e:"+e.toString());
                } catch (IOException e) {
                        Log.v("IO EX", "e:"+e.toString());
                }

                return html_string;
        }
}
```

Main.xml (Responsible for implementation)

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

    <TextView
    android:id="@+id/myLocationText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text=""
    />
    <RelativeLayout
         android:id="@+id/widget61"
         android:layout_width="264px"
         android:layout_height="59px"
         android:layout_x="17px"
         android:layout_y="50px"
         >
         <Button
         android:id="@+id/Button"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Send!"
         android:layout_alignTop="@+id/eText"
         android:layout_alignParentRight="true"
         >
         </Button>
         <EditText
         android:id="@+id/eText"
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Hvor vil du dra?"
         android:textSize="18sp"
         android:layout_centerVertical="true"
         android:layout_alignParentLeft="true"
         >
         </EditText>
         </RelativeLayout>
    <com.google.android.maps.MapView
        android:id="@+id/mapView"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="0ID7ig0s2JJRqiNQqd5s9VwQUBvoNoFy-NXRGKw"
        />

    <LinearLayout android:id="@+id/zoom"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_centerHorizontal="true"
        />

</LinearLayout>
```