**COMP8790 Software Engineering Project**
**Semester 1, 2010**

# TaxiShare Client on Windows Mobile Platform

Melvin Yong

## Abstract

Taxi sharing can help reduce traffic congestion in cities and as well as carbon emissions by enabling several passengers to take a single taxi to the same destination, instead of several separate trips. It also benefits passengers by lowering the overall cost of taking a taxi. Moreover, taxi sharing also provides unique networking opportunities.

TaxiShare is a taxi sharing system developed by CSIRO to facilitate taxi sharing that aims to encourage more commuters to share a taxi by streamlining and simplifying the taxi-sharing process.

This report describes and documents the development of a Windows Mobile client for the TaxiShare system which is a smaller subset of the parent TaxiShare Project. The client aims to increase the ubiquity of the TaxiShare system as part of the overall strategy of targeting multiple mobile and social networking platforms.

At the conclusion of the project, a proof of concept Windows Mobile TaxiShare client was constructed, with Google Maps support and location awareness capability through GPS or cell tower triangulation using Google location services.

# Acknowledgments

First and foremost, I would like to thank my project supervisor: Dr Ken Taylor for giving me the opportunity to work on this project and the freedom to experiment with various ideas. His insight, knowledge and guidance were instrumental in the completion of this project.

Many thanks also go to Herman for taking time to explain to us how the system works. Without your help we would be buried under the mountain of code.

I would like to thank Yingyi. I appreciate your intellect, and your ability to deconstruct the many implementation problems we faced so easily. I also enjoy our brainstorming sessions and I hope they helped you as much as they helped me.

Last but not least, I would like to thank Dr. Uwe Zimmer, the overall project co-ordinator, who provided us with the "Community of Practice" Meetings. The advice and guidance he gave on doing a research project as well as presentation, and scientific writing skills opened my eyes to what is really out there.

# Contents

# 1. Introduction

This section provides a brief introduction to this project. The first part is attempts to explain how the TaxiShare system works and how the project ties in with the overall TaxiShare system. The second part gives a brief overview of what the project aims to achieve and its objective.

## 1.1. Motivation

In recent years, there has been increased awareness in environment and there have been many governmental initiatives (The Department of Climate Change and Energy Efficiency, 2010) to attempt to reduce the country's carbon footprint. Transportation accounts for significant amount of greenhouse gas emissions (EPA Victoria, 2010) and has a huge overall impact on global emissions.

## 1.2. Background

In 2008, CSIRO started an initiative called the "Director's Challenge"**.** The aim of the challenge was to find a project that would provide environmental benefits. Taxi sharing, one of the ideas considered, seemed to be a relatively easy and cost-effective way for individuals and institutions to reduce their impact on the environment.

With that in mind, a taxi sharing system called TaxiShare was proposed to assist CSIRO employees in sharing taxis from airports to CSIRO sites and conferences. By increasing the number of shared taxi rides, the system promises to reduce carbon footprint, save money and provide opportunities for networking (CSIRO ICT Centre, 2010).

### 1.2.1. How TaxiShare Works

The TaxiShare System is currently SMS based and works using the following procedures below.

1. Send a SMS text message with the current location and intended destination to the TaxiShare system in the following format:

   "FROM <current location> TO <intended destination>"

2. The TaxiShare system will try to interpret the address and match it to another traveller.
3. The system will then SMS text message in reply of the other traveller going to the same destination and their mobile phone number.
4. The 2 travellers then attempt to negotiate the taxi sharing details.

The diagram in figure 1.1 shows in greater detail how it works.

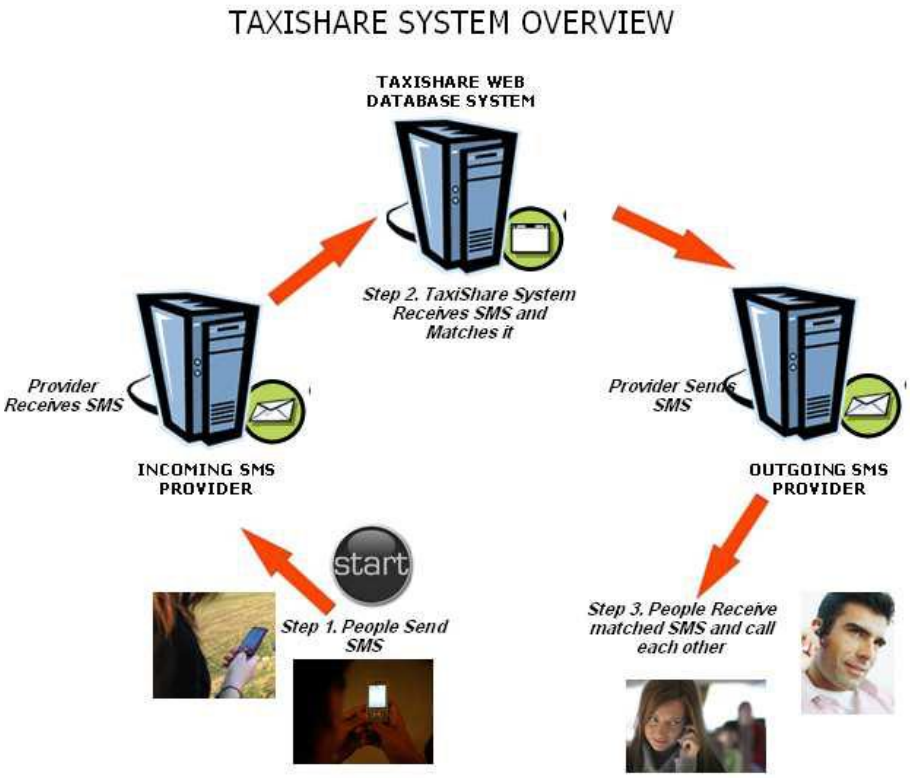


**Figure 1.1:** TaxiShare System Overview.

Further information regarding the TaxiShare System can be found in the paper, "A System for Sharing Taxis with Multiple Interfaces" by Herman Jaya (Jaya, Taylor, Yong, & Chen, 2010).

## 1.2.2. Other Interfaces

Listed below are the interfaces that have been implemented for the TaxiShare system.

### 1.2.2.1. Online Map

TaxiShare has a web interface (Jaya, TaxiShare, 2010), with an embedded map powered by Google Maps. The map displays the intended destinations of the TaxiShare users. Phone numbers that have been matched are shown in red while phone numbers that are matched are displayed in green.

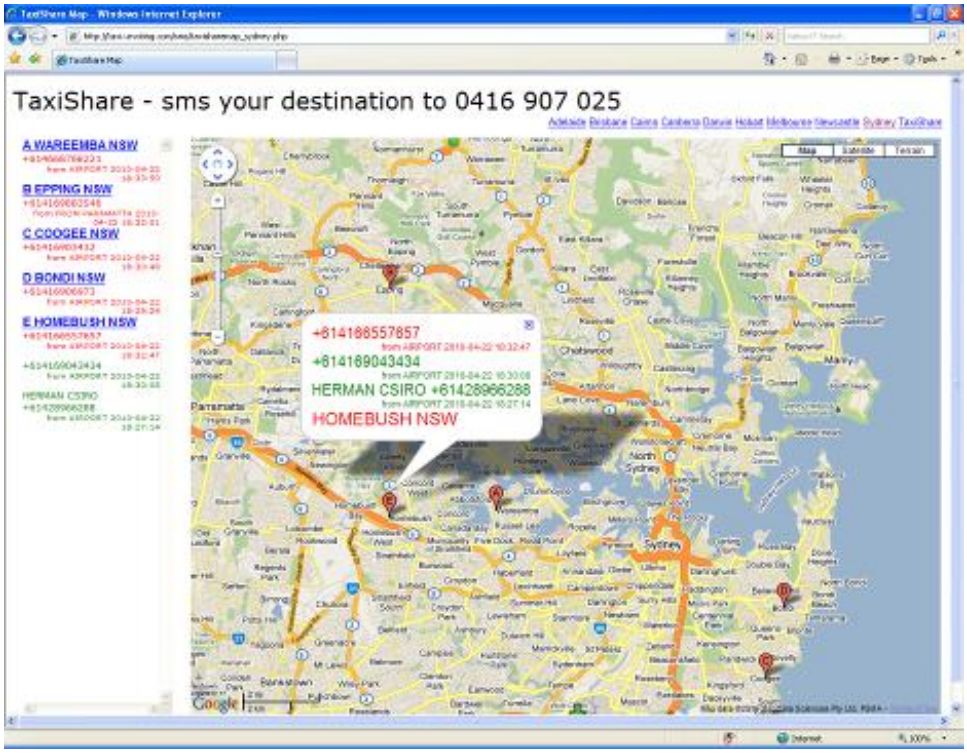


**Figure 1.2:** TaxiShare web interface.

#### 1.2.2.2. Twitter

TaxiShare requests can also be sent through Twitter. To send a request, users can 'tweet' the destination to TaxiShare's Twitter account. This interface has an advantage over the SMS in that users are able to view the matched user's Twitter profile and make a decision as to whether to share a taxi.

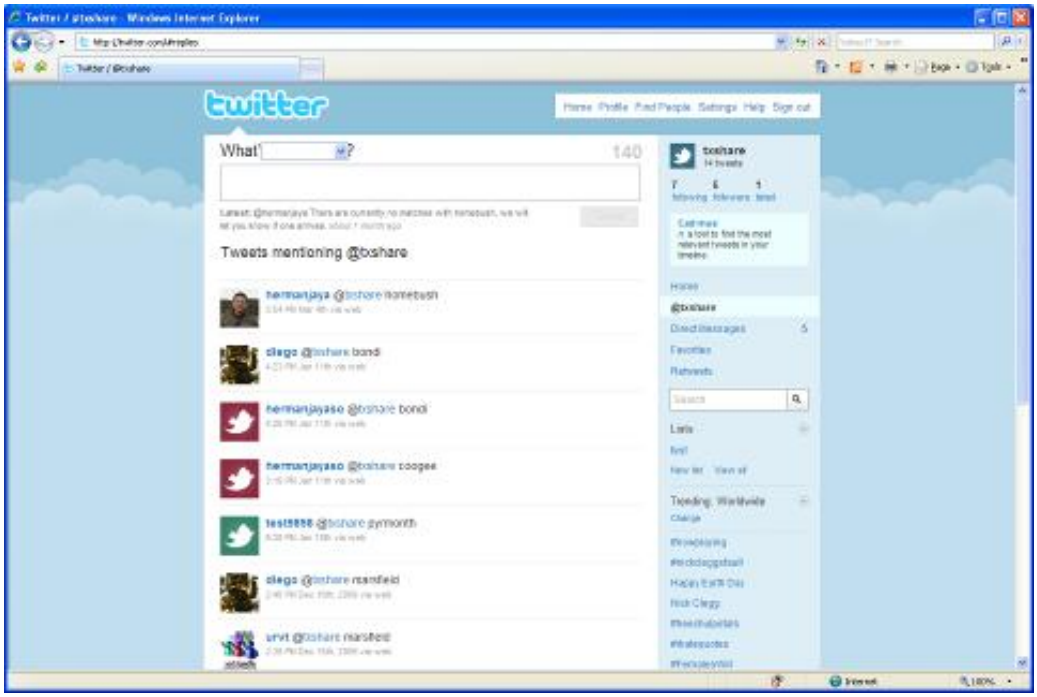


**Figure 1.3:** TaxiShare Twitter interface.

### 1.2.3. My Work

There is a limitation in the current interfaces for the TaxiShare system: the potential for ambiguous location and destination input.

For example, if a user enters “Lonsdale Street” as the intended destination, there is no way for the system to know if the user meant the Lonsdale Street in Melbourne or the Lonsdale Street in Canberra.

In addition, there is no context for the user as to where they are and their intended destination (Map interface displays the intended destination only after sending a TaxiShare request).

To address these limitations as well as increase the reach of the TaxiShare system, a client on a mobile platform would be developed which aims to improve the accuracy of input and reduce the ambiguity of the TaxiShare requests sent.

# 2. Preparation and Research

This section describes the preparation done before implementing the application. Design decisions made in the implementation will also be explored.

## 2.1. Initial Project Parameters

While the TaxiShare Client was to be a proof-of-concept of how another platform can interact with the existing TaxiShare System, and as such without strictly defined requirements, there were 2 key design parameters by which the WM Client had to abide by:

1. **Compatibility with existing system**. The backend for the TaxiShare system is written in PHP and as such is web based. This meant that the client is restricted to 2 methods of communicating with the TaxiShare system - Through a HTTP request or an SMS message. The Windows Mobile client would also have to send the TaxiShare requests in the same format in order for the system can process them.
2. **Increased accuracy.** The user interface and design of the client had to be an improvement in accuracy over texting in the input of location and destination details to the TaxiShare server in order to justify the creation of the client.

## 2.2. Choice of Platform

The platforms considered for TaxiShare client include Android, iPhone OS, .NET compact framework (Windows Mobile). To determine their suitability, the platforms were to be compared on the following criteria:

1. **Developer familiarity.** This determines how familiar the developer is with the platform environment. The familiarity of the platform environment would impact the development time of the client application.
2. **Availability of test device.** This indicates whether an actual device is available for testing. Having an actual device to test is important as the application might behave differently on an emulator as compared to an actual device. It was also important to be able to demonstrate the client
3. **Availability of development tools.** This indicates whether the prerequisite development tools are or will be available at the time of development.
4. **Market share.** This indicates the percentage market share that the particular platform has at the moment (Tudor & Pettey, 2010). A higher market share would be preferable to maximize the potential user base.

The most important metrics in the selection of the platform is the availability of a test device and development tools. The test device being crucial as the TaxiShare client would need to be demonstrated while the availability of the development tools would impact the implementation of the client.

An assessment table was created based on the above factors for comparison purposes:

| Platform | Familiarity | Test Device | Development Tools | Market Share |
|---|---|---|---|---|
| BlackBerry | Java. Familiar. Low learning curve. | No. | Open source. Yes. | 19.4% |
| Android | Java. Familiar. Low learning curve. | Yes. | Open Source. Yes. | 9.6% |
| iPhone OS | Objective C. Variant of C. Somewhat familiar. Slight learning curve. | No. | Proprietary. No. | 15.4% |
| Windows Mobile | .Net Compact Framework. C#, VB. Familiar. Low learning curve | Yes. | Proprietary. Yes. | 6.8% |

**Table 2.1:** Platform comparison table.

After weighing the pros and cons of each individual platform, it was decided that the target platform would be Windows Mobile.

### 2.2.1. Reason for Selection

Developer familiarity was major factor in the selection of the .NET compact framework, as was the availability of a device for testing, which would aid in the testing, development and demonstration of the mobile application. The development tools, which consist of a Windows based (XP, Vista or 7) desktop and the Visual Studio IDE were also readily available.

The Android platform was another suitable platform. However, there is an existing Android based TaxiShare client currently being developed.

The other considered platforms were eventually not adopted either because the actual devices were not available for testing in the case of the BlackBerry or that the development tools were not complete in the case of the iPhone. An OSX based computer was not available for development, at time of the selection.

It is worthwhile to note that even though Windows Mobile has only 6.8% of the market share, there are still significant numbers of users of windows Mobile based Smartphones.

## 2.3. Review of Google Mapping APIs

Based on the increased accuracy parameter described in section 2.1, it was decided that some form of mapping functionality would be implemented in order to give the end-user some context to the selected location and destination, with the intention of improving input accuracy and reduce ambiguity. As Google Maps was already used in the TaxiShare web interface, the map feature would utilize Google maps APIs, if possible to maintain consistency across the platforms.

### 2.3.1. Static Maps API

There are several Google Maps APIs available for developers to mapping functionality into their applications, and they can be found at the Google code website (Google, 2010). Due to the lack of full JavaScript native support in Windows Mobile (Jeyes, 2008), the chosen API must not depend on JavaScript and should be capable of generating map images in a format readable by the .NET compact framework.

Based on the above criteria, the Static Maps API was chosen.

### 2.3.2. How it works

The Static Maps API is web based service that creates a map based on URL parameters specified by Google. The URL is sent via a HTTP request and returns an image (which can be in the GIF, PNG or JPEG format) of a map, which can be embedded into a website or mobile application (in the case of the TaxiShare client).

Each request contains specification of the map location, the size of the image, the zoom level, the type of map, and/or the placement of optional markers at locations on the map.

### 2.3.3. Input Parameters

This section describes the parameters that can be used to generate a map from the Static Maps API. Not all the parameters are required and only the ones relevant to the development of the client would be specified below.

The Static Maps API defines map images using the following URL parameters:

*http://maps.google.com/maps/api/staticmap?parameters*

Location Parameters:

- ***center*** (required if markers not present) defines the centre of the map, equidistant from all edges of the map. This parameter takes a location as either a comma-separated latitude, longitude pair (e.g. "40.714728,-73.998672") or a string address (e.g. "Australian National University") that identifies a unique location on Earth.
- ***zoom*** (required if markers not present) defines the zoom level of the map.

Map Parameters:

- ***size*** (required) defines the rectangular dimensions of the map image. This parameter takes a string in the form of [value]x[value] where horizontal pixels are denoted first while vertical pixels are denoted second.
- ***format*** (optional) defines the format of the resulting image. By default, the Static Maps API creates PNG images. There are several possible formats including GIF, JPEG and PNG types. Which format you use depends on how you intend to present the image. JPEG typically provides greater compression, while GIF and PNG provide greater detail. For more information, see Image Formats.
- ***maptype*** (optional) defines the type of map to construct. Map types include roadmap, satellite, hybrid, and terrain.
- ***mobile*** (optional) specifies whether the map will be displayed on a mobile device. Valid values are true or false. Maps displayed on mobile devices may use different tile sets optimized for those devices.

- ***language*** (optional) defines the language to use for display of labels on map tiles. This is only supported for some language tiles.

Google has a developer guide (Google, 2010)that has more detailed information on the parameters described as well as the optional parameters not described in this section. A portion of this section is adapted from the developer guide.

### 2.3.4. Static Map Example

To get a map image of the Australia National University located in Canberra with a marker labelled "A", the following URL can be input into the web browser.

*http://maps.google.com/maps/api/staticmap?center=Australian+National+University&zoom=14&size=512x512&markers=color:red|color:red|label:A|-35.276280597872876,%20149.1227102279663&maptype=roadmap&sensor=false*

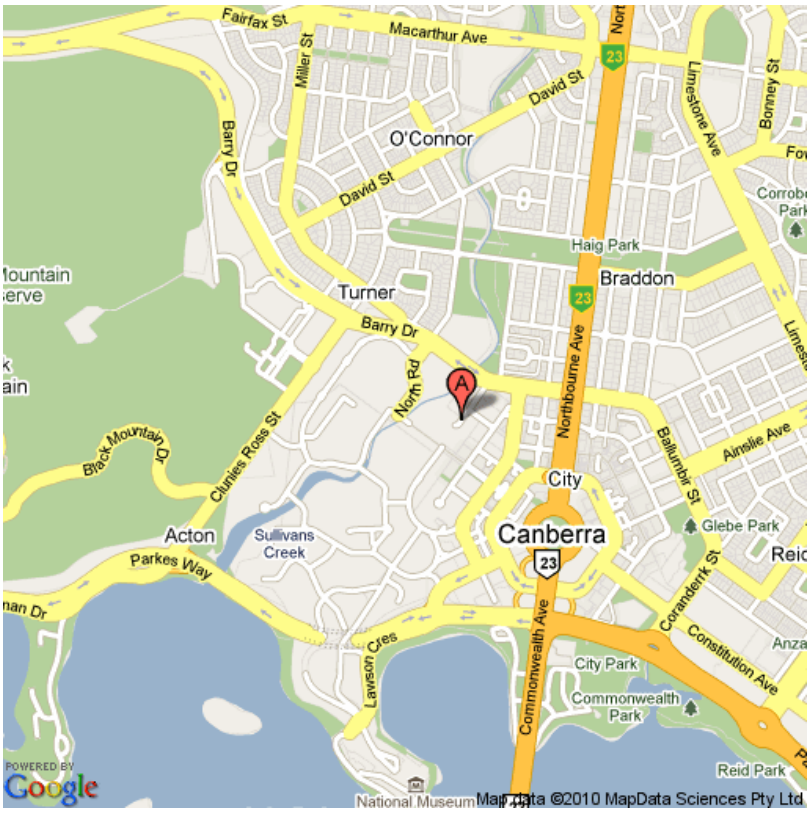


**Figure 2.1:** Example of a map generated by the Google Static Maps API

### 2.3.5. Static API Limitations

It is worthwhile to note that the Static Maps API would not give the same level of interactivity afforded in Google's other JavaScript based Google Maps API. The features that most end-users have come to expect in a mapping application: direct interaction or manipulation with the map interface are not possible with a map image. This is a limitation of the Windows Mobile platform, a result of the lack of true JavaScript support (Jeyes, 2008).

## 2.4. Review of Static Map Interactivity

This section describes how interactivity might be achieved on the TaxiShare client by mapping device screen coordinates to the real world coordinates represented by the map image. Adding this feature would allow more interactivity with the map image. It would enable end-users to place a location marker directly by tapping on the map interface instead of manually inputting the location details.

To find the real world coordinates from the map image coordinates and the reverse, the map projection must first be known as it would impact the way the coordinates are calculated. Several developers have noted that Google Static Maps uses the Mercator projection for their maps (Tim, 2008). With that mind, a set of equations can be derived to calculate the image and real world coordinates.

Developers have considered the translation problem and some of them have come up with solutions that are based on the Mercator projection (Tuupola, 2010).

### 2.4.1. How it might be Implemented

Shown below is one possible way of how interactivity with the static map might be achieved within the context of the TaxiShare System:

The user taps on a particular area on the map. The application stores the coordinates of where the user tapped.


**Figure 2.2:** Example of map interactivity implementation.

When the placing a location marker is selected from the context menu of the `picturebox` object, the application retrieves the coordinates that was stored where the user tapped and uses the coordinate translation methods described earlier to translate the tapped coordinates to real world coordinates.

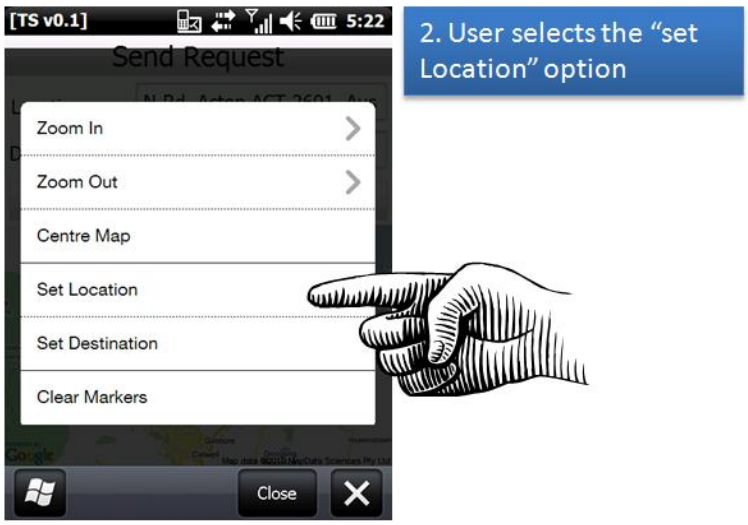


**Figure 2.3:** Example of map interactivity implementation.

The real world coordinates are then sent to Google's Static Maps servers. The resulting map image generated with the location marker and displayed in the mobile application.

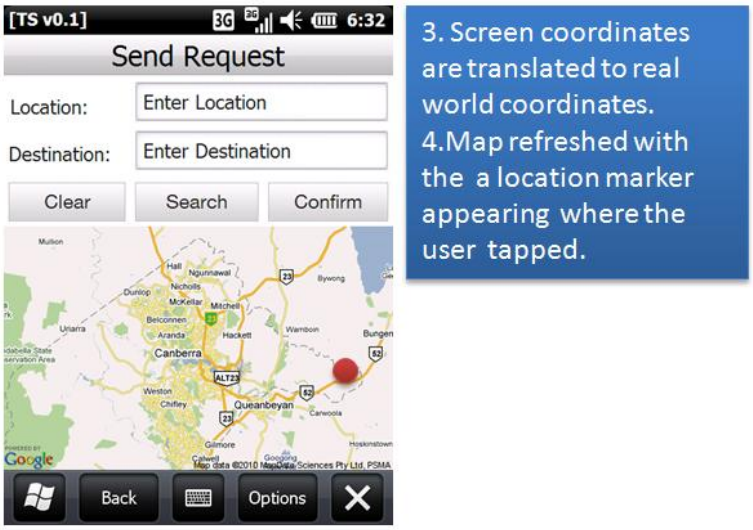


**Figure 2.4:** Example of map interactivity implementation.

## 2.5. Review of Alternative User Interfaces

A review of the custom user interfaces was conducted in an effort to improve usability and overall application aesthetics. 3 solutions for Windows Mobile devices were studied and compared for implementation feasibility.

The criteria used to assess the suitability of the solutions are as follows:

1. **Cost.** The solution should also be free to use and be, preferably open-sourced.
2. **Ease of use.** The solution should have sufficient implementation examples and be easy to implement.
3. **Features.** How fully featured the solution is: whether it supports kinetic scrolling for example (this is something that cannot be implemented in a windows mobile application).

### 2.5.1. Resco MobileForms Toolkit

The MobileForms Toolkit is a set of custom controls for version 3.5 of the .NET compact framework developed by Reso (Resco, 2010)

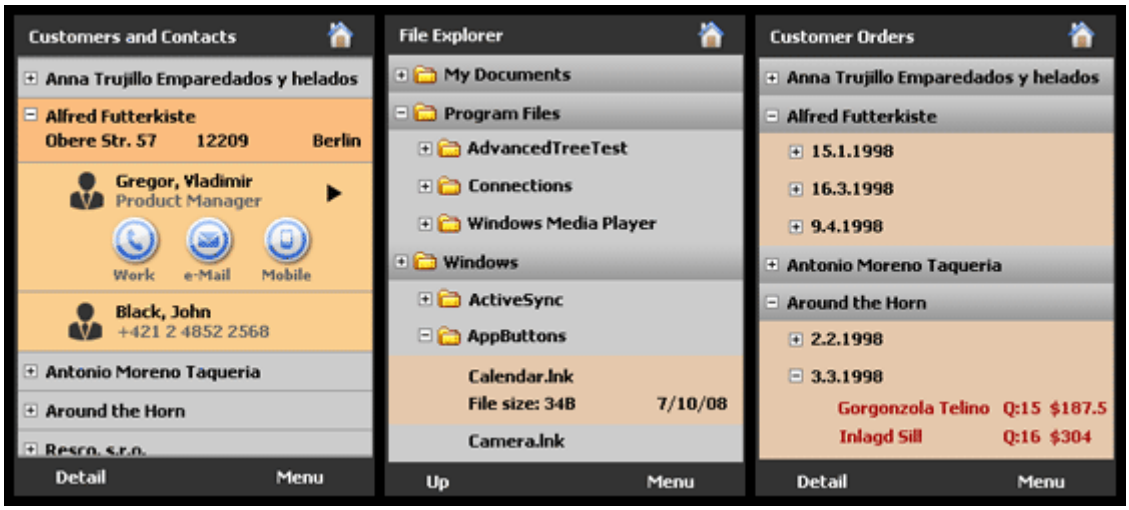


**Figure 2.5:** Screenshots of Resco MobileForms Toolkit.

#### 2.5.1.1. Advantages

The set of custom controls for user interface design is easy to use, professional looking. The controls are also fully featured and support features like kinetic scrolling (touch scrolling) and improved image management (Resco, 2010). Help is available in the form of a user forum (Resco Forums, 2010) where implementation examples and technical support can be found.

#### 2.5.1.2. Disadvantages

While the controls are professional looking, and easy to use, they are not free (the cost for the standard edition version of the controls is USD$749.95).

### 2.5.2. Fluid

Fluid is a set of Windows Mobile 6.x Touch Controls for version 2.0 of the .NET compact framework developed by Thomas Gerber (Gerber, Fluid - Windows Mobile .NET Touch Controls, 2009)


**Figure 2.6:** Screenshots of Fluid.

#### 2.5.2.1. Advantages

Many of the user interface elements are similar to the Apple iPhone/iPod Touch UI conventions and users familiar with them would have a lower learning curve to learn how to use the application. It is open-source and as such, it is free for use.

#### 2.5.2.2. Disadvantages

The implementation example provided was complicated and there is a lack of documentation regarding how to implement the controls. Support was also lacking as many requests for assistance on the discussion board are unanswered (Gerber, 2010). The controls can only be implemented on version 2 of the .NET compact framework.

### 2.5.3. Sense Interface SDK

The Sense Interface SDK is a set of custom controls developed by eboelzner for version 3.5 of the .NET compact framework (eboelzner, 2010). The look and feel of the controls is based on HTC's sense UI interface for Windows Mobile devices.

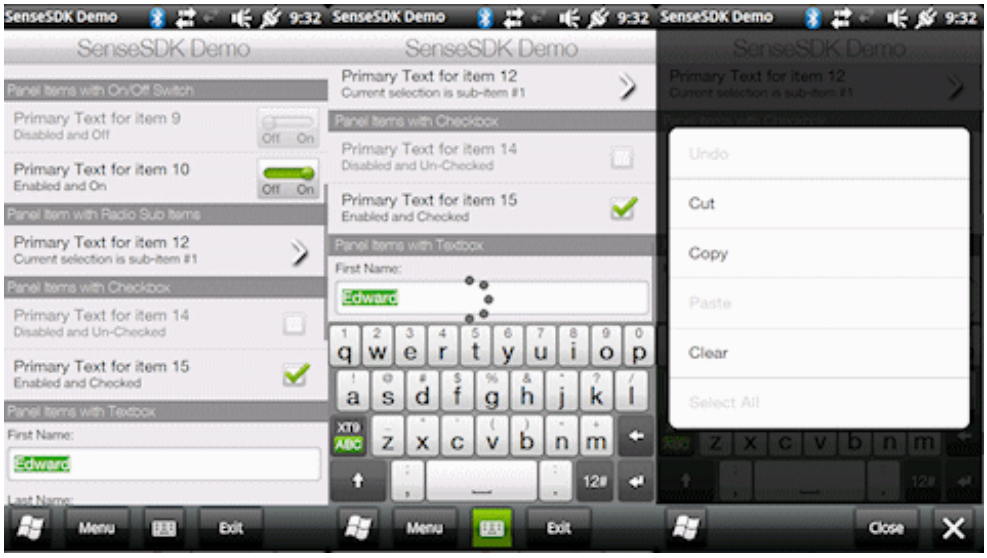


**Figure 2.7:** Screenshots of Sense Interface SDK

#### 2.5.3.1. Advantages

The provided example project is fairly easy to understand. While the documentation is not as comprehensive compared to the other solutions such as the Resco MobileForms Toolkit, there is strong developer support in the project forum (eboelzner, 2010). The controls are also free for use.

#### 2.5.3.2. Disadvantages

The controls are in an early stage of development and as such there are many bugs in implementation of the controls. For example in the earlier versions there was a bug in the controls that prevented Visual Studio from showing the custom controls in the toolbox and the controls had to be manually added into the source code (eboelzner, 2010).

### 2.5.4. Manila Interface SDK

The Sense Interface SDK is a set of custom controls developed by michyprima for version 3.5 of the .NET compact framework (michyprima, 2009).The look and feel of the controls is based on HTC's manila handset user interface, which in turn is based on the HTC's sense UI for Windows Mobile devices.

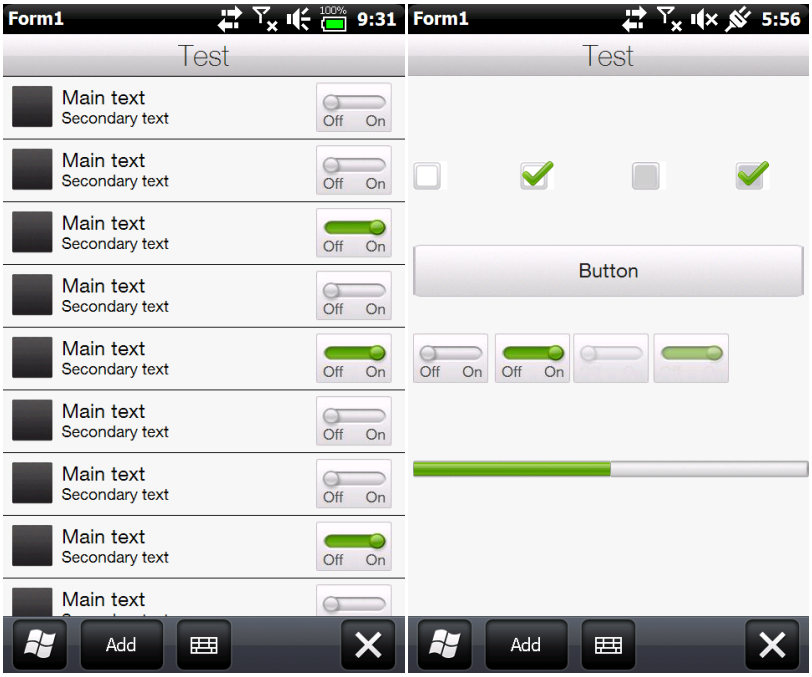


**Figure 2.8:** Screenshots of Manila Interface SDK.

#### 2.5.4.1. Advantages

The user base of these custom controls is similar to the Sense Interface SDK in section 3.6.3 and the user support is large as well and assistance can be found easily in the project forum. The controls are also free for use.

#### 2.5.4.2. Disadvantages

Improvements and bug fixes for the controls have slowed as the developer of the SDK focuses on other project.

### 2.5.5. Choice of Custom controls

Based on the review of the various solutions for custom controls, it was decided that the Sense Interface SDK and Manila Interface SDK would be used in the development of the TaxiShare client.

While both projects contain bugs, there is a support base to get help. Both solutions are also relatively easy and free to use.

## 2.6. Review of Location Awareness Technologies

In an effort to improve input accuracy and usability, location awareness technologies available to the Windows Mobile were explored. By automatically acquiring location details, end-users can reduce the time to input location information.

GPS and cell tower triangulation were suggested as possible methods to retrieve the current location and studies were done to determine if it was feasible to implement.

## 2.6.1. GPS

According to the United States Government's National Executive Committee for Space-Based Positioning, Navigation, and Timing (PNI), The GPS (Global Positioning System) is a

"...space-based global navigation satellite system that provides reliable location and time information in all weather and at all times and anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites."

It is possible to retrieve location information from Windows Mobile Smartphone's with built-in GPS hardware. This section will mainly focus on how to retrieve GPS location data through the GPS Intermediate Driver (GPSID) provided Microsoft.

### 2.6.1.1. GPSID Overview

The GPS Intermediate Driver is not a driver as the name suggests, but rather a layer of abstraction that that sits in between the application and drivers for the GPS hardware (Microsoft, 2006). It is found in Windows Mobile 5 and 6 devices and developers can find a sample project containing an implementation example as well as the wrapper classes bundled with any windows mobile SDK or DTK after 5.0.

The main advantage of the GPSID is that it enables multiple applications on the device to utilize the GPS hardware at the same time, which was not possible in earlier versions of Windows Mobile devices. In those earlier devices, only one application at a time could access the GPS hardware. At a low level, the GPSID is the only application that accesses GPS hardware and is a higher level interface for multiple applications to access location data.

From a developer point of view, it provides simpler way to utilize GPS functionality, compared to using native code.

The following class diagram shows the classes of the GPSID wrapper.


**Figure 2.9:** Class diagram of GPSID classes.

Further information regarding the GPSID can be obtained from the Microsoft MSDN site (Microsoft, 2006).

### 2.6.1.2. Implementation Example

Shown in here is an example of how an application can retrieve GPS location data. This can be done both synchronously and asynchronously. To get the data synchronously, only the `getPosition` method is required to be called. Asynchronous is accessed though the `LocationChanged` event which constantly updates location data through the `getPosition` method.

In the context of TaxiShare client, retrieving the GPS location asynchronously is not required so this example will only focus on synchronous access.

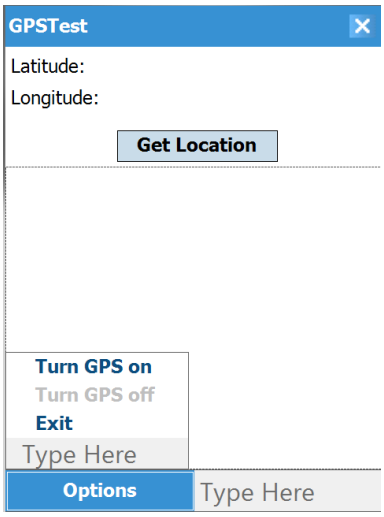


**Figure 2.10:** GPS Implementation Example.

```
if (gps.GetPosition().LatitudeValid)
{ //Has position: do something with the data
    lbLat.Text = "Latitude: " + gps.GetPosition().Latitude;
    lbLon.Text = "Longitude: " + gps.GetPosition().Longitude;
    genMap(gps.GetPosition().Latitude, gps.GetPosition().Longitude);
}
else
{
    MessageBox.Show("GPS is not ready...");
}
```

**Figure 2.11:** Code snippet of GPS Example

As the above code illustrates, to retrieve a location data, the `Open` method of the `GPS` object needs to be called first to enable the GPS hardware. Once GPS hardware enabled, the `GetPosition` method can be invoked to retrieve the current location data.

#### 2.6.1.3. GPS Limitations

Even though, GPS location data is by far the most accurate of all the methods examined, there are many environments in which its performance is limited. For example, GPS reception is often poor or not possible in sheltered environments like buildings. There are also factors such as the effect of signal multi-path (Garmin, 2010) which reduces the accuracy of GPS in areas like cities or places with many tall buildings.

### 2.6.2. Cell Tower Triangulation

While other methods like GPS can give more accurate location data as compared to cell tower triangulation, not all Smartphones have GPS hardware built-in (although a significant number of devices have them). Cell Tower Triangulation has the advantage of retrieving relatively accurate location data from any Smartphone.

#### 2.6.2.1. How it Works

The location of the Smartphone can found by measuring the radial distance or direction of the received signal from 2 or more cell towers. The distance is derived from the lag time when the cell tower sends a ping and receives a ping from the phone. The following illustration is simplification of how a triangulation can be achieved.

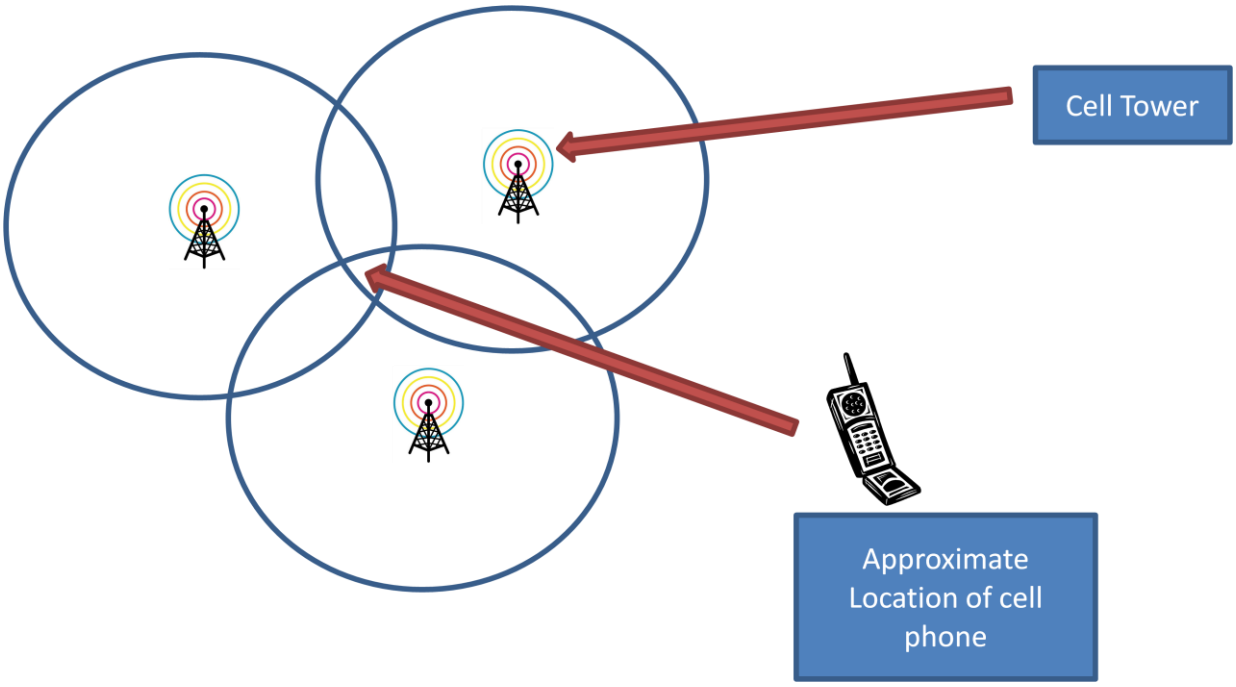


**Figure 2.12:** How cell tower triangulation works.

Factors that impact the accuracy of the triangulation include the number of cell towers (the more towers the phone is connected to, the more accurate the location data) and the type of cell towers that the phone is connected to (if the phone is connected to directional cell towers, the location data would be more accurate).

There are several web based location services using cell towers that include OpenCellID (OpenCellID, 2010) and Yahoo ZoneTag (Yahoo, 2010) and Google My Location (Google, 2010). It was eventually decided that Google's location service would chosen for implementation.

While Google's location service requires more effort to implement as there are not documented, they have a more comprehensive database of cell tower information as compared to the other services.

#### 2.6.2.2. Google Maps for Mobile

Google Maps for Mobile is an application for a variety of mobile devices to use Google Maps service (Google, 2010). One of the key features is the ability of the application to get location data without the use of the GPS.

Cell tower data sent by the phone is sent to Google via a HTTP request (Chu, 2007)when the user accesses the Google Maps mobile application. Google then references the data on their database (which contains cell tower IDs and the corresponding location information) and sends back the location of the tower in latitude and longitude format.

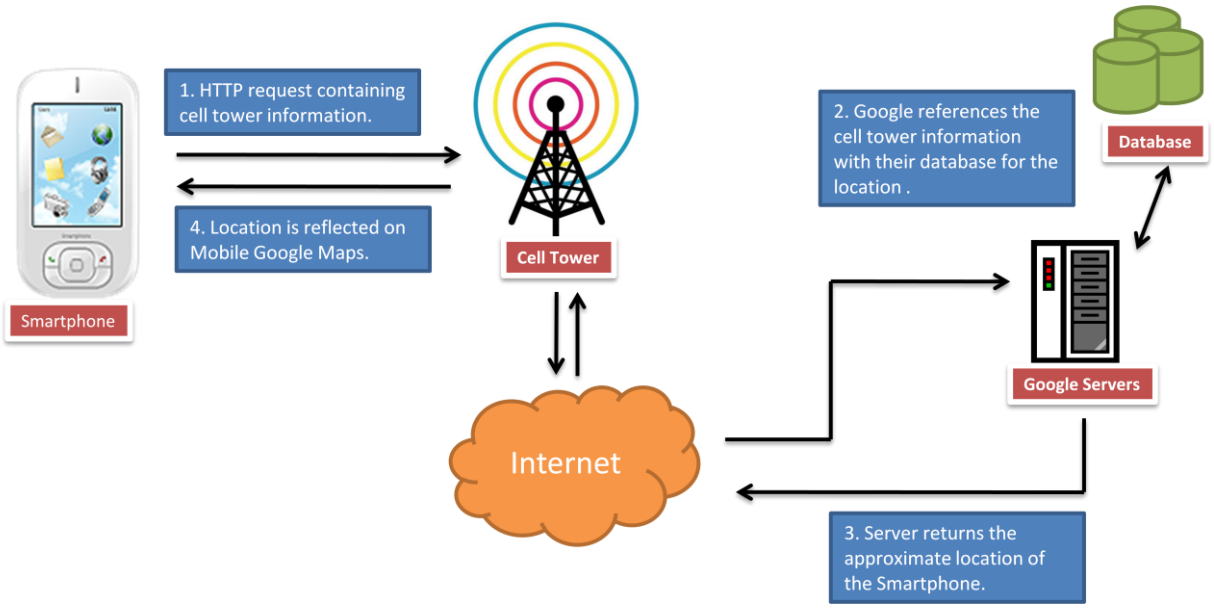


**Figure 2.13:** How Google Maps for Mobile derive approximate location of the Smartphone.

#### 2.6.2.3. Google Location Service

Google's API for determining location is not publicly documented, and as such some packet analysis would be required to figure out how exactly to send a request for location data.

Wireshark (WireShark, 2010) was used to monitor the packets sent from the windows mobile emulator to Google's servers. Preliminary inspection of the packet reveals that the HTTP requests are being sent to: *http://www.google.com/glm/mmap*

Other developers have gone a step further (Young, 2010)and have identified the 4 Parameters that are sent along with the request and their respective functions:

- Cell Tower ID
- LAC (Location Area Code)
- MNC (Mobile Network Code )
- MCC (Mobile Country Code)

Further information regarding the above codes can be found at the Nokia Developer Wiki (Nokia, 2010).

#### 2.6.2.4. Radio Interface Layer

Data regarding 4 parameters that need to be passed to the Google service can be extracted from RIL (Radio Interface Layer), an interface that handles the communication between the CellCore system software and the radio hardware (Microsoft, 2010).

To use RIL, the following P/Invoke signatures must be added:

```
[DllImport("ril.dll")]
private static extern IntPtr RIL_Initialize(uint dwIndex, RILRESULTCALLBACK pfnResult, RILNOTIFYCALLBACK pfnNotify,
    uint dwNotificationClasses, uint dwParam, out IntPtr lphRil);
[DllImport("ril.dll", EntryPoint = "RIL_GetCellTowerInfo")]
private static extern IntPtr RIL_GetCellTowerInfo(IntPtr hRil);
[DllImport("ril.dll", EntryPoint = "RIL_Hangup")]
private static extern IntPtr RIL_Hangup(IntPtr hRil);
[DllImport("ril.dll")]
private static extern IntPtr RIL_Deinitialize(IntPtr hRil);
```

**Figure 2.14:** Code snippet of RIL P/Invoke signatures.

The 4 APIs used in the above code snippet are:

- `RIL_Initialize`. Initializes the Radio Interface Layer.
- `RIL_Deinitialize`. Frees up any resources after an application finishes using RIL.
- `RIL_GetCellTowerInfo`. Retrieves cell tower information that the phone is connected to.

With the above 4 methods and the `RILRESULTCALLBACK` and `RILNOTIFYCALLBACK` structures, managed code can be used to retrieve cell tower data from the mobile device.

### 2.6.3. Other Methods Considered

Further study of Google geo-location API's revealed that it is possible to triangulate the current position of the user using WiFi signals as well (Google, 2010).

The WiFi triangulation works by sending the MAC addresses and signal strengths of the surrounding access points to Google through a HTTP request in JSON format. Google has a database of MAC addresses with their corresponding SSIDs and location data. The request is then processed with the help of the database and an approximate location of the user is returned.

The following illustrates an example JSON request taken from the Google Gears API documentation (Google, 2010).

```
<script type="text/javascript" src="gears_init.js"></script>
<script type="text/javascript">
var geo = google.gears.factory.create('beta.geolocation');

function updatePosition(position) {
  alert('Current lat/lon is: ' + position.latitude + ',' + position.longitude);
}

function handleError(positionError) {
  alert('Attempt to get location failed: ' + positionError.message);
}

geo.getCurrentPosition(updatePosition, handleError);
</script>
```

**Figure 2.15:** Example of JSON request.

WiFi triangulation appears to be more accurate as compared to cell tower triangulation in informal testing around the Australian National University campus. It was, however not implemented due to time constraints.

# 3. Approach and Methodology

This section details the approach applied in the implementation of Windows Mobile TaxiShare client. Also described are some issues faced during the development of the application.

The approach to building of the Windows Mobile TaxiShare client consists of 2 main phases: the preparation and research phase and the implementation phase.

## 3.1. Preparation and Research Phase

In this phase, the research is done on the necessary tools and technologies required to construct the application. The knowledge and technical know-how from this phase reduces the time it takes to implement the mobile application. Further details regarding this phase can be found in section 2.

## 3.2. Implementation Phase

The implementation of the TaxiShare client follows an Agile style development. In this phase, the basic functionalities of the TaxiShare client are finalized through negotiations and discussions with the project supervisor, Dr Ken Taylor. Shown below are the major functionalities identified:

1. Sending and receiving a TaxiShare request.
2. Mapping Controls.
3. Location Awareness.

The actual implementation of the functionalities was then grouped into sprints. The following describes the activities in each sprint:

1. Story boards are drawn to illustrate how the application was to behave with some text describing briefly what the application does at the backend.
2. Prototypes are then made to explore how the business logic would function.
3. The business logic of the prototype would then be added to the main application.

## 3.3. Issues Faced in Development

Listed here are the issues faced in the development of the TaxiShare client.

### 3.3.1. Project Scope

Some features planned for in the stages of development had a larger scope than initially thought and had to be scaled back or abandoned.

The early iterations of the TaxiShare client application, TaxiShare request were sent to the TaxiShare server via HTTP requests. The web based method had a performance advantage over the SMS based method, whose reliability is dependent upon the current load of the SMS gateways. If a match is found, the server would send a message back to both the user that sent the request and the user that is matched.

However, when a user that uses the SMS method to submit a request, the TaxiShare servers would only send a response back to the SMS user but not the user that submitted the request via HTTP due to the architecture. To solve this would require modifications to the server side code that which would have required time and effort beyond the scope of the project, as well as possibly breaking the operation of the live system if something went wrong.

The scope was eventually reduced, and as a compromise, the TaxiShare client now runs on top of the SMS system, ensured interoperability, although at the cost of reduced performance.

### 3.3.2. Effort Estimation

The effort spent on implementing the mapping controls for the client took longer than expected due to the lack of managed code or APIs and had to be built from scratch. While the controls were eventually implemented, time allotted for development of the location awareness functions was re-diverted to help meet the schedule.

This resulted in less time to work on other features. For example, The WiFi triangulation feature had to be left out of the client final feature set.

# 4. Results

This section details the final iteration of the Windows Mobile TaxiShare client including its features, a typical usage scenario as well as some known issue with regards to the application.

## 4.1. Application Architecture

The following diagram describes the high level architecture of the Windows Mobile TaxiShare client.

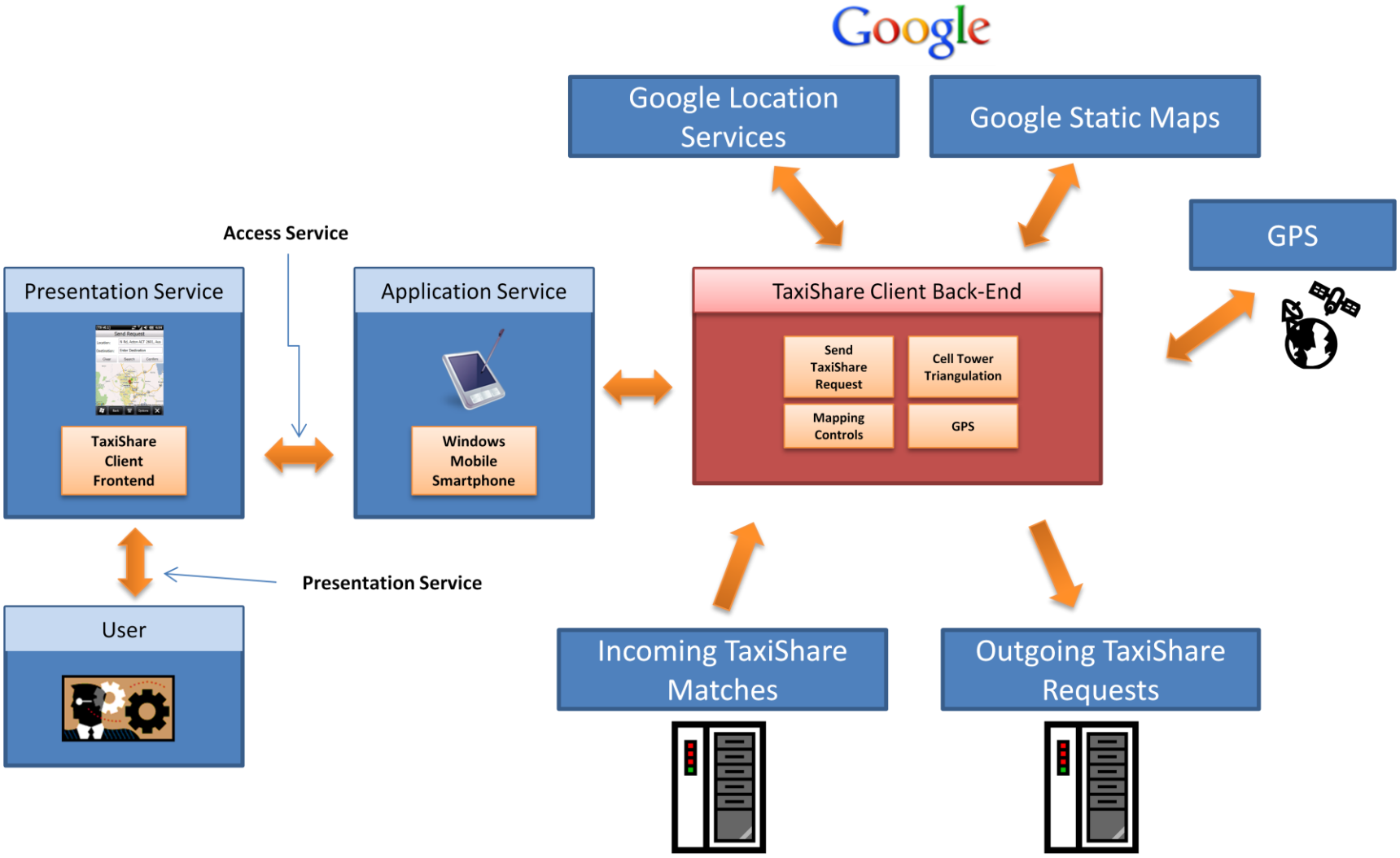


**Figure 4.1:** High level architecture of Windows Mobile TaxiShare client.

## 4.2. Application Features

Here are some of the notable features in this application:

### 4.2.1. Google map view

Users are able to view their current location and destination on a map image provided by Google maps. They are also able to place location markers directly on the map making it faster to input the route as compared to manually inputting the location and destination of their journey.

### 4.2.2. Location Awareness

Users can get their current location accurately through GPS or through cell tower triangulation, which would serve to reduce the time it takes to input the current location.

## 4.3. Typical Usage Scenario

Shown below is how a user would send a TaxiShare request:

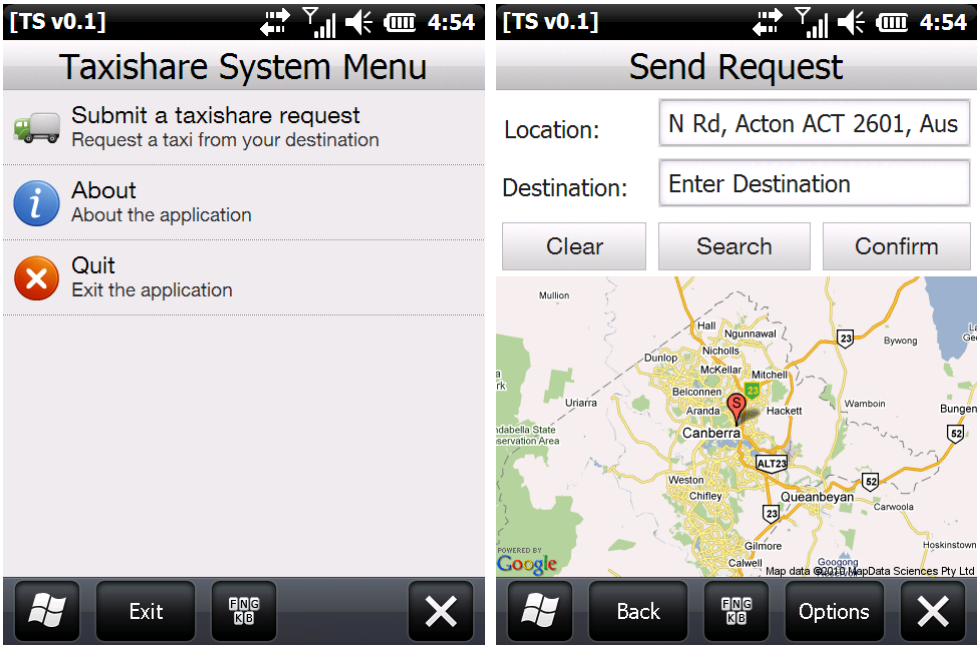


**Figure 4.2:** UI walkthrough 1.

First, tap on the **"Submit a TaxiShare request"** panel to submit a taxi share request. The System would then bring up the TaxiShare request interface.

On loading the interface, the application would try to get the current location of the user (through cell tower triangulation) and display them onto the map interface.

If the current location is not accurate, the user can also opt to get their current location through the phone built-in GPS.

Users can also choose to enter location details through the **Location:** and **Destination:** textboxes on the top of the application and clicking on the **Search** button or by tapping and holding on the map interface.

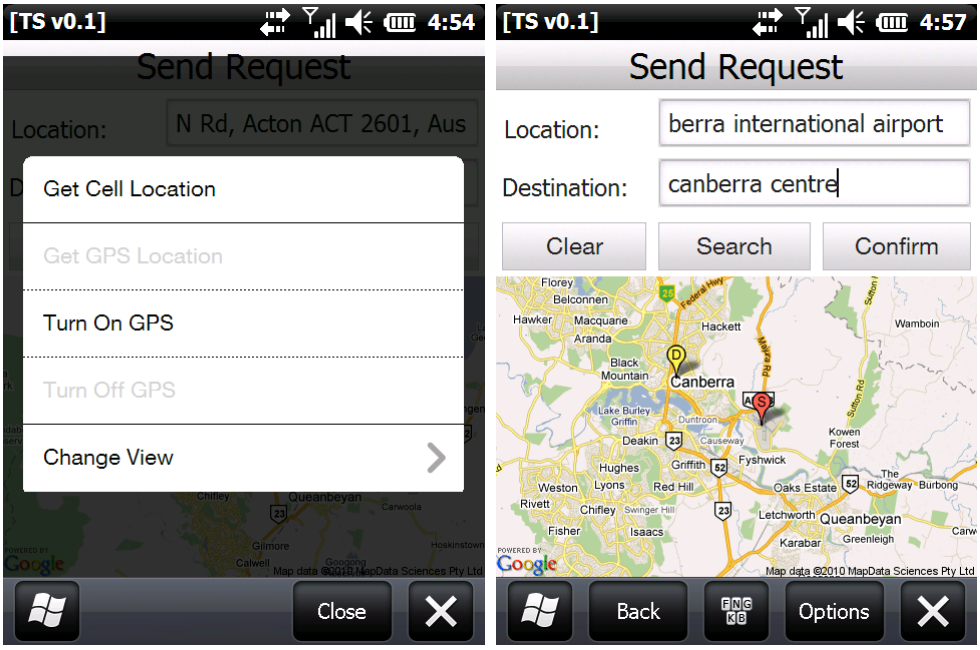


**Figure 4.3:** UI walkthrough 2.

Tapping and holding on the map interface yields the ability to: **zoom in and out**, **centre the map**, **set the location** (this action will put a current location marker on the map), **set the destination** (this action will put a destination marker on the map) and **clear location markers**.

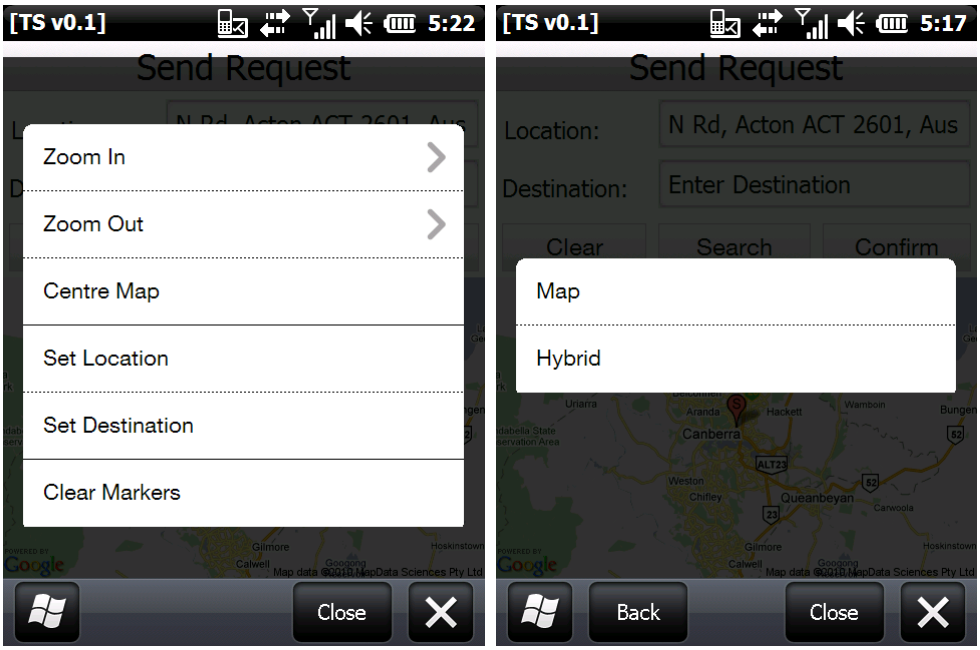


**Figure 4.4:** UI walkthrough 3.

The map view can also be changed should the user desire a different view.

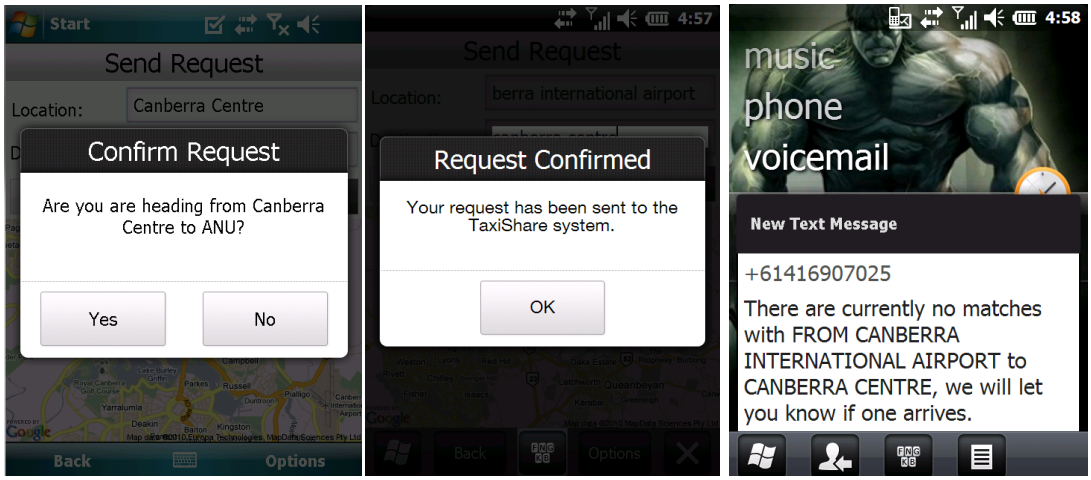


**Figure 4.5:** UI walkthrough 4.

To send the request, click on the **OK** button. The system would display a confirmation message and send an SMS message to the TaxiShare system. The rest of the user interactions would be similar to the SMS interface.

## 4.4. Known Issue

This section lists a known issue regarding the TaxiShare client.

### 4.4.1. Location Matching

The TaxiShare system does not always match the TaxiShare requests sent from the Windows Mobile client to the requests via the SMS.

An example scenario:

1. User A and B's intended destination is the Australian National University and they have the same location.
2. User A submits a TaxiShare request via SMS to go to "anu".
3. User B submits a TaxiShare request via the Windows Mobile client to go to "Australian National University, Canberra City ACT 2601, Australia".
4. User A and B will not be matched due to the difference in the precision of the location details.

## 4.5. Thoughts regarding the platform

This section attempts to describe the perceived advantages and disadvantages of developing for the windows mobile platform (in comparison to the Google Android, since they belong to the similar category of devices). The basis of these points was derived from experience gained through the development of the Windows Mobile TaxiShare client.

### 4.5.1. Advantages

#### 4.5.1.1. Portability

Source code that contain business logic can be re-used in most cases from different windows platforms (windows mobile, windows XP, vista etc.), provided they use libraries that present in both .NET framework or .NET compact framework. This would make porting the key classes with the exception of UI elements in the current windows mobile client to a windows based desktop .NET application or even the upcoming Windows Phone 7 relatively trivial, with minimal modifications to the existing code.

While it is also possible for Java based Android to port source code to J2SE or other Java-based platforms like J2ME, given that each platform have frameworks that can differ significantly the amount of source code that can be re-used across platforms is much more limited. It is however worthwhile to note that most source code migration is between versions of same platform so this might not be that much of a negative point on the Android platform.

#### 4.5.1.2. Cross Language Support

Windows Mobile .NET applications can be written in any CLR-compliant language such as Visual Basic, C# or C++. The code would be translated into the Common Intermediate Language or CIL which is eventually assembled, verified and compiled. In the context of the Windows Mobile client, portions of the application could be written in another language if need be. A new developer that is not familiar with C#, the language that the client is written in can write in Visual Basic or any other CLR-compliant language.

In Contrast, the Android platform is tied to the Java language specifically. While there is a large developer base for Java, the cross language support of the Windows Mobile .NET compact framework still makes the platform more flexible as compared to Android.

## 4.5.2. Disadvantages

### 4.5.2.1. SDK

Standardization and ease of use of APIs in Windows Mobile is one key area that is behind the android platform. Here are 2 examples of where this can be seen.

### 4.5.2.2. Access to GPS hardware

For Windows Mobile 6.x:

GPS hardware can be accessed managed code through the .NET compact framework and Microsoft has included code samples (titled "GpsSample") to help developers get started on developing location awareness applications. The sample project contains a collection of classes that wraps the native GPS API GPS hardware.

While it is possible to add these classes into a project that requires interaction with GPS hardware, the files needed to be packaged into a DLL file for referencing in the development of TaxiShare client.

For Google Android:

Getting location data is handled by the `LocationManager` system service which is part of the `android.location` package.

All an Android developer needs to do to get location information is the following lines of code:

```
locationManager =
(LocationManager)getSystemService(Con
text.LOCATION_SERVICE);
```

```
Location location =
locationManager.getCurrentLocation("g
ps");
```

### 4.5.2.3. Interactive Maps support

For Windows Mobile 6.x:

There is currently no interactive maps support for the Windows Mobile platform and most custom 3rd party solutions are expensive (ThinkGeo, 2010). It is possible however, for static maps to be downloaded and displayed from map providers like Bing Maps or Google Maps (which was used in the development of the WM TaxiShare client).

So while the .NET compact framework might be stronger than Android, Android's API support is far superior to Windows Mobile, in my opinion.

For Google Android:

Interactive maps support from Google Maps is one of the core features of the Android platform and was part of the platform right from its first release. The `com.google.android.maps` package provides developers with the ability to download, render, and cache of Maps tiles from Google maps servers as well as create other visual features that involve direct manipulation and interaction with the map interface.

# 5. Conclusion

The main objective of this report is to document the construction of Windows Mobile Client for the TaxiShare system, the results of which are documented in section 5.

This report also provides the background and context behind the construction of the TaxiShare: the Windows Mobile client application was meant to increase the reach of the TaxiShare system, alongside the other interfaces that interacted with the system.

In addition to the background, the preparation and research that went before implementing the system was also explored. The decisions regarding platform, Mapping APIs, location aware technologies, mapping controls and alternative user interfaces were discussed and explained.

Furthermore, a high level view of how the project was tackled was presented in the Approach section of the report to help illustrate how the client was actually implemented.

## 5.1. Future Work

The following section lists some possible future work to be carried out in the future development of the project.

### 5.1.1. Migration to Windows Phone 7

This iteration of the TaxiShare client on the Windows Mobile platform could very well be the last and only iteration, serving as a proof of concept due to a new mobile platform released by Microsoft (Microsoft, 2010)**.** Microsoft has already released an SDK for the platform (Microsoft, 2010) and business logic can be ported over to the new platform.

### 5.1.2. Server Matching Improvements

The current logic of matching destination names relies on name matching. This could cause potential mismatches described in section 4.4. Further work can involve implementing some form of database that keeps a list of alternate names for a particular location or switching it over to some NLP (Natural Language Processing) solutions to improve matching.

# 6. Works Cited


- Chu, M. (2007, November 28). *New magical blue circle on your map*. Retrieved June 1, 2010, from Google Mobile Blog: http://googlemobile.blogspot.com/2007/11/new-magical-blue-circle-on-your-map.html

- CSIRO ICT Centre. (2010). TaxiShare. *TaxiShare* . CSIRO ICT Centre.

- eboelzner. (2010, March 19). *[.netCF 3.5] Sense Interface SDK v1.37 [HTC's Sense UI Look 'n Feel]*. Retrieved June 1, 2010, from xda-developers: http://forum.xda-developers.com/showthread.php?t=648906

- EPA Victoria. (2010). *Australia and Victoria's greenhouse gas emissions*. Retrieved June 1, 2010, from EPA Victoria: http://www.epa.vic.gov.au/greenhouse/australia-victoria-emissions.asp

- Garmin. (2010). *What is GPS?* Retrieved June 1, 2010, from Garmin: http://www8.garmin.com/aboutGPS/

- Gerber, T. (2010). *Discussions for Fluid - Windows Mobile .NET Touch Controls*. Retrieved June 1, 2010, from CodePlex: http://fluid.codeplex.com/Thread/List.aspx

- Gerber, T. (2009, Febuary 15). *Fluid - Windows Mobile .NET Touch Controls*. Retrieved June 1, 2010, from CodePlex: http://fluid.codeplex.com/

- Google. (2010). *Geolocation API*. Retrieved June 1, 2010, from Google Code: http://code.google.com/apis/gears/api_geolocation.html

- Google. (2010). *Google Maps API Family*. Retrieved June 1, 2010, from Google Code: http://code.google.com/apis/maps/

- Google. (2010). *Google Maps for mobile*. Retrieved June 1, 2010, from Google Mobile: http://www.google.com/mobile/maps/

- Google. (2010). *Static Maps API V2 Developer Guide*. Retrieved June 1, 2010, from Google Code: http://code.google.com/apis/maps/documentation/staticmaps/

- Jaya, H. (2010). *TaxiShare*. Retrieved June 1, 2010, from TaxiShare - SMS your destination to 0416 907 025: http://taxi.urvoting.com/

- Jaya, H., Taylor, K., Yong, M., & Chen, Y. (2010). *A System for Sharing Taxis with Multiple Interfaces.* Canberra.

- Jeyes, D. (2008, April 14). *Windows Mobile 6.1: "Full Desktop Browsing?"*. Retrieved June 1, 2010, from theregoesdave: http://theregoesdave.com/2008/04/14/does-windows-mobile-61-provide-full-desktop-browsing/

- michyprima. (2009, September 30). *[.NetCF Library] Manila Interface SDK*. Retrieved June 1, 2010, from xda-developers: http://forum.xda-developers.com/showthread.php?t=566188

- Microsoft. (2006). *GPS Intermediate Driver Application Development*. Retrieved June 1, 2010, from MSDN: http://msdn.microsoft.com/en-us/library/ms894898.aspx

- Microsoft. (2010). *Microsoft Windows SDK for Windows 7 and .NET Framework 3.5 SP1*. Retrieved June 1, 2010, from Microsoft Download Center: http://www.microsoft.com/downloads/details.aspx?FamilyID=c17ba869-9671-4330-a63e-1fd44e0e2505&displaylang=en

- Microsoft. (2010, April 8). *RIL Application Development*. Retrieved June 1, 2010, from MSDN: http://msdn.microsoft.com/en-us/library/aa923037%28v=MSDN.10%29.aspx

- Microsoft. (2010). *Windows Phone*. Retrieved June 1, 2010, from Windows Phone: http://www.windowsphone7.com/

- Nokia. (2010). *How to get info on cell location*. Retrieved June 1, 2010, from Nokia Community Wiki: http://wiki.forum.nokia.com/index.php/How_to_get_info_on_cell_location

- OpenCellID. (2010). *OpenCellID*. Retrieved June 1, 2010, from OpenCellID: http://www.opencellid.org/

- Resco. (2010). *.NET CF controls - Special features*. Retrieved June 1, 2010, from Resco .Net Developer: http://www.resco.net/developer/mobileformstoolkit/features.aspx

- Resco Forums. (2010). Retrieved June 1, 2010, from Resco .Net Forums: http://www.resco.net/forums/default.aspx?ForumGroupID=6

- Resco. (2010). *Resco MobileForms Toolkit Overview*. Retrieved June 1, 2010, from Resco .Net Developer: http://www.resco.net/developer/mobileformstoolkit/overview.aspx

- The Department of Climate Change and Energy Efficiency. (2010). *A-Z of Government initiatives*. Retrieved June 1, 2010, from The Department of Climate Change and Energy Efficiency: http://www.climatechange.gov.au/government/initiatives.aspx

- ThinkGeo. (2010). *Map Suite Pocket PC .Net Components*. Retrieved June 1, 2010, from Map Suite GIS .Net Components: http://gis.thinkgeo.com/Products/GISComponentsforNETDevelopers/MapSuitePocketPC/tabid/163/Default.aspx

- Tim. (2008, October 3). *How to make Google static maps interactive*. Retrieved June 1, 2010, from RevaHealth.com Blog: http://blog.revahealth.com/2008/10/how-to-make-google-static-maps-interactive.html

- Tudor, B., & Pettey, C. (2010, May 19). *Gartner Newsroom*. Retrieved June 1, 2010, from Gartner: http://www.gartner.com/it/page.jsp?id=1372013

- Tuupola, M. (2010). *Clickable Markers With Google Static Maps*. Retrieved June 1, 2010, from Mika Tuupola: http://www.appelsiini.net/2008/6/clickable-markers-with-google-static-maps

- WireShark. (2010). *Wireshark*. Retrieved June 1, 2010, from Wireshark: http://www.wireshark.org/

- Yahoo. (2010). *ZoneTag Location Services: Cell Location*. Retrieved June 1, 2010, from
- Yahoo Developer Network: http://developer.yahoo.com/yrb/zonetag/locatecell.html

- Young, N. (2010). *Koders Code Search*. Retrieved June 1, 2010, from Koders: http://www.koders.com/csharp/fid1082DBFC02F278AB454637F47D905067E093FA6B.aspx?s=zoom