Speedflirt for Android, a study of porting a mobile application and its effects on user experience

Erik Jansson, Niclas Jonasson

May 5, 2011
Master’s Thesis in Computing Science, 30 credits
Supervisor at CS-UmU: Håkan Gulliksson
Examiner: Jerry Eriksson

UMEÅ UNIVERSITY
DEPARTMENT OF COMPUTING SCIENCE
SE-901 87 UMEÅ
SWEDEN
Abstract

Smartphones have become increasingly popular and sales numbers from the third quarter of 2010 report that an estimated 80.5 million smartphones were sold worldwide. With this, the aftermarket for smartphone applications has grown explosively and the applications have become a part of the user’s everyday life. As companies want their mobile applications to reach a broader audience the demand for cross-platform applications is increasing.

This master thesis was conducted at The Mobile Life in Gamla Stan, Stockholm, with the objective of porting an iPhone application called Speedflirt to the Android platform. Speedflirt is a location based mobile dating application developed by Speedflirt Pte Ltd and promoted by Metro Sweden.

This thesis investigates the techniques and frameworks that are currently available for development of cross-platform mobile applications, identifying the most suitable technique for porting Speedflirt; furthermore, the effects of porting Speedflirt between the platforms was investigated with respect to how the user experience was affected.

The result of this thesis is a ported version of Speedflirt available on the Android Market; furthermore a set of guidelines aimed at the preservation of user experience during porting is presented in this thesis report.
CONTENTS

5 Implementing Speedflirt for Android 23
  5.1 Implementing a REST-client for Android .......................... 23
    5.1.1 RestService .............................................. 24
    5.1.2 RestProcessor and JSONParser .......................... 24
    5.1.3 RestClient .............................................. 25
  5.2 Implementing a "Coverflow" gallery .................................. 25
  5.3 Mimicking iPhone UI ................................................ 28
    5.3.1 Implementing iPhone components on the Android platform ...... 29

6 The effect of porting applications on user experience 33
  6.1 Which factors affect UX ............................................ 34
  6.2 The UX of Speedflirt ................................................ 34
    6.2.1 The impact of consistency on usability and user experience .... 35
    6.2.2 Platform specific design guidelines .......................... 35
    6.2.3 How brands and social identification affect the UX ............ 37
    6.2.4 iPhone vs Android .......................................... 39
  6.3 Conclusions .......................................................... 40

7 The resulting application 43

8 Conclusions 49
  8.1 Future work ........................................................ 50
  8.2 Acknowledgements ................................................ 50

References 51

A Loading images 55
List of Figures

2.1 ER-diagram of Speedflirt UI ............................................. 6
2.2 Speedflirt UI 1 .......................................................... 6
2.3 Speedflirt UI 2 .......................................................... 7
2.4 Similar applications ....................................................... 7
3.1 Skatteverket mobile application ....................................... 12
4.1 Android architecture[37] ................................................ 18
4.2 Illustrates an example of the structure of a tree hierarchy that forms the Android UI[34]. ........................................... 19
4.3 Illustration of the concept of NinePatch images .................... 20
4.4 Illustration of the concept of patches within NinePatch images [23] .......... 21
5.1 REST-Client Flowchart .................................................. 24
5.2 Sketch of a “cover flow” styled gallery .............................. 25
5.3 Standard gallery component within the Android library .......... 26
5.4 Cache and preloading of images to display next in the gallery .... 27
5.5 Architecture of the cover flow styled gallery component and its assisting classes 28
5.6 Static elements in Android ................................................. 29
5.7 AgeSpinner comparison .................................................. 30
5.8 iPhone dialog for using the accessing camera or gallery .......... 30
6.1 The appearance of the menu that becomes visible when Menu button is pressed 36
6.2 Standard component differences ...................................... 37
6.3 A selection of standard components with a predefined meaning. .... 38
6.4 Platform rivalry ........................................................... 40
7.1 Coverflow screenshots .................................................... 44
7.2 Incoming flirts screenshots .............................................. 44
7.3 Ongoing flirts screenshots ................................................. 45
7.4 Chat menu screenshots ................................................... 45
7.5 Registration screenshots .................................................. 46
7.6 Settings screenshots ......................................................... 46
7.7 Chat conversation screenshots ........................................... 47

A.1 Flow chart for the process of loading an image into the gallery component ......................................................... 55
List of Tables

3.1 Native development .................... 10
3.2 Mobile web application architecture ..................... 11
Chapter 1

Introduction

Smartphones have become increasingly popular and sales numbers from the third quarter of 2010 report that an estimated 80.5 million smartphones were sold worldwide [24]. The usefulness of smartphone devices now partly comes from their ability to extend their functionality with applications chosen by the user. With this, the aftermarket for smartphone applications has grown explosively where the applications have become a part of the user’s every day life. A study conducted by Hossein et al. shows that a regular user have up to 200 distinct interactions with a smartphone device per day[13].

The market for the applications that the smartphones utilizes are thriving, Kris Van Bruwaene states that there were 7 billion application downloads in the year of 2009 resulting in a revenue of 4.1 billion dollars[47]. He further concludes that the number of downloads grows by 92 percent a year and are estimated to 50 billion in 2012[47]. Although Kris Van Bruwaene acknowledge the fact that the numbers varies between sources, the figures indicate an impressive growth in the smartphone industry as well as in the smartphone application market. Along with these impressive numbers, research are reporting case studies and surveys where smartphones are tested against and integrated into various professions showing a specific interest in health care[4] and education[42]. The integration of smartphones into work professions as well as the reported growth of the smartphone industry highlights the impact that smartphones have, and further will have on users everyday life. Reaching out to the majority of employees and customers becomes crucial regardless of which platform or device they are using.

The mobile world is getting more and more fragmented where several popular mobile platforms are present today, and it is very likely it they will be available in the future as well. The platforms that constitutes the majority of the smartphone market are Symbian, Windows Mobile, Windows Phone 7, Palm OS, Web OS, BlackBerry, iOS, Linux Mobile and Android[40]. Because of this diversity, an application developed for one specific platform will only reach a segment of smartphone users connected with that platform. In order to reach such a wide audience as possible with a mobile application, it needs to be available on all possible platforms. Cross-platform applications are becoming more common as developers tend to port their applications to several platforms in order to reach a bigger audience.

This Master’s Thesis in the field of interaction design was conducted at The Mobile Life in Gamla Stan, Stockholm. The Mobile Life is a design- and technology firm specialized in software solutions for mobile platforms. Founded in Singapore in 2006, The Mobile Life opened their first office in Sweden in 2010 and because of the rise in popularity of smartphones (such as the iPhone and Android), smartphone platform applications constitutes a
majority of the project proposals received by The Mobile Life. One of these projects is Speedflirt which is a collaborative venture between The Mobile Life, Speedflirt Pte Ltd and Metro (the metropolitan newspaper).

Speedflirt is a location based mobile dating application developed by Speedflirt Pte Ltd and promoted by Metro Sweden. Speedflirt was initially launched for the iPhone platform in 2010, but in order to reach a larger audience, the current plan is to implement the application on a wider array of mobile platforms. Due to the increasing number of smartphone platforms and devices, porting mobile applications is becoming an inevitable part of mobile development. Furthermore, the popularity of iPhone as a smartphone has made it the most common first-choice-platform to develop a mobile application. As the Android platform is growing in popularity, an expansion to this platform is a typical next step in the search for a broader audience.

A common request is to create an exact copy of the original application on the new platform. Due to differences in performance of devices as well as in the underlying technology, creating an exact copy of an application on a new platform can be a difficult and time consuming task. As The Mobile Life has noticed an increased number of request of this nature, a study of different techniques examining this issue was desired. Furthermore, as design idioms and user expectations differ between platforms, an investigation on how user experience is affected by porting was conducted. Porting of Speedflirt represented a typical porting request and was therefore used as a point of origin for this investigation; by outlining important factors that should be considered when porting, this investigation could be used as a framework to ensure a preserved user experience for future porting projects. During the thesis, the Android version of Speedflirt was implemented and analyzed with respect to user experience.

1.1 Problem Statement

The problem to be solved during this thesis was to implement an iPhone application named Speedflirt on the Android platform. In addition, the resulting application was investigated in order to analyze how the user experience was affected by the port. The project was carried at The Mobile Life in Stockholm during January to May in 2011.

1.2 Goals

The main goal of this Masters thesis was to investigate how the user experience of a mobile application is affected when porting the application to new platforms and construct guidelines for important factors that should be considered when porting. The porting of Speedflirt would be used as a base for this investigation. In order to achieve this, a number of subordinated goals are described below:

- Investigate which techniques and frameworks that are currently available for development of cross-platform mobile applications, identifying the most suiting technique for porting Speedflirt.
- Port Speedflirt to the Android platform according to the requirements given.
- Investigate what constitutes user experience of mobile applications.
- Applying the findings from the investigation to Speedflirt and investigate how the user experience was affected from the port.
1.3 Porting requirements

In porting Speedflirt the new versions should make use of the same server as the original iPhone version of Speedflirt. By using the same server, users from all platforms would be mixed, forming a larger community on the existing Speedflirt server allowing Android users to flirt and chat with iPhone users and vice versa.

The other requirement was that the Android version should look and behave just like the iPhone in order to have a unified user interface across the platforms. This would entail making an exact copy of the interface which would prove to be a challenge. As many iPhone native GUI elements where used in the iPhone version, the Android equivalent components had to be created from scratch.

1.4 Limitations

Testing the Android version of Speedflirt against a test group could not be conducted in the timeframe of this thesis. Thereby all assumptions and conclusions are based on feedback from the Android Market and findings from literature studies.

1.5 Paper outline

- Chapter 2. Speedflirt: Describes the concept of the original version of Speedflirt, the structure of the GUI, similar applications and the requirements that were set for the porting of Speedflirt.

- Chapter 3. Porting mobile applications: This section investigates the idea of developing cross-platform applications and identifies which technologies that are available today. The most suitable technique for porting Speedflirt is identified by comparing these techniques in relation to the requirements of Speedflirt.

- Chapter 4. The Android framework: Describes and clarifies different parts of the Android framework and the techniques that was used when porting Speedflirt.

- Chapter 5. Implementing Speedflirt for Android: Describes the implementation process when porting Speedflirt to the Android platform.

- Chapter 6. Results: This section illustrates the results from the porting of Speedflirt. Screenshots covering the final GUI are listed and described.

- Chapter 7. The effect of porting applications on user experience: This chapter will identify the consequences of porting applications and how these might affect the user experience.

- Chapter 8. Conclusions: This section clarifies important aspects when porting applications between platforms. The errors made within the project of Speedflirt are identified and suggestions on solutions of these errors are proposed.
Chapter 2

Speedflirt – the example application

The practical part of this thesis involved porting a mobile application called Speedflirt which was released on the iPhone platform in 2009. The assignment was to port Speedflirt to the Android platform with the goal to create an exact copy of the functionality and interface. This chapter will briefly explain the concept of Speedflirt, the included functionality, the graphical interface and the porting requirements for Speedflirt.

2.1 Concept

Speedflirt can be described as a mobile, location based dating service aiming at being as fast and accessible as possible. While most dating services use elaborate questionaries to match singles, Speedflirt match singles that are located in the same vicinity with the only matching parameters being physical attributes; by this approach, Speedflirt tries to mimic the social interaction taking place in a regular nightclub.

A user is presented with a gallery of singles which the user can choose to send a flirt to. If the flirt is accepted by the recipient, a chat conversation is established between the two. The only information revealed about the users are their pictures and some physical attributes such as height and body type; it is then up to the users themselves to engage in the conversation in order to get to know each other.

Technically this is done by connecting every mobile client to a cloud server that stores the location, profile picture and information about each user. The server acts as the central hub in Speedflirt, thus it can be seen as the nightclub that the Speedflirt users are visiting. All activities such as flirts and conversations are managed and stored by the server, enabling a user to access his or hers Speedflirt session from any device just by logging in to the server.
2.2 User interface

The Speedflirt user interface is based around the iPhone tab-bar, meaning that one static navigation-bar is always present at the bottom of the screen. The user can then switch back and forth between the four main parts of the Speedflirt UI as seen in figure 2.1.

Figure 2.1: ER-diagram of Speedflirt UI

![ER-diagram of Speedflirt UI](image)

The application will always start with the find-flirts tab selected. Here the user will be presented with an iPhone-style cover-flow gallery of the singles matching the user’s preferences that are in close proximity of the user’s current physical location. As the user scrolls through the gallery there is the possibility to click a picture to view detailed information about that person as well as the option of sending a flirt.

The second tab will display all incoming flirts for a user. The interface is similar to finding flirts, the main portion of the screen is a cover-flow gallery displaying all incoming flirts. The user can scroll through the gallery, viewing information about the user who sent the flirt, and has the ability to accept or decline incoming flirts.

Figure 2.2: Speedflirt UI 1

(a) Coverflow gallery  (b) Profile information  (c) Incoming flirts

![Speedflirt UI 1](image)

The third tab will display a list of the user’s currently active flirts. Clicking one of the cells within the list will display the ongoing conversation between the two users in a style resembling the native SMS application for the iPhone.
2.3 Similar applications

As seen in figure 2.2 and 2.3 some iPhone standard components are used such as the tab-bar at the bottom of the screen and the navigation-bar at the top of the screen. Also some standard components have been slightly modified for use in Speedflirt, one example of this is the accept / ignore flirt buttons which are standard components for accepting and ignoring incoming calls. Using standard components with similar functionality like this will allow a first time Speedflirt user to recognize interface elements and easier understand the functionality of these elements.

2.3 Similar applications

There exist a couple of applications that are very similar to Speedflirt, more focused around casual flirting than strict matchmaking. The most popular alternatives to Speedflirt were identified as Zoosk, SpeedDate and SprayDate.

All of these applications make use of the built in GPS in the smartphone and use proximity as the main matching parameter, discarding the questionaries that form the base of
matching in traditional dating services. The functionalities and way of usage are very similar in these applications with a typical flow of use being:

1. Browse through potential flirts in your vicinity.

2. Send some sort of notification that you want to engage in a conversation.

3. If your invite to a conversation is accepted some sort of chat conversation is engaged.
Chapter 3

Cross-platform development frameworks

Because of the advances in mobile technology over the last decade, mobile applications have become more useful and also more common for both private use and within corporations. There exist several popular mobile platforms today, e.g. iPhone, Android WP7 and Symbian that all have the ability to be extended with mobile applications. A mobile application developed for a certain platform, for instance the iPhone, will only reach users of that particular platform. This problem is becoming a bigger challenge today when companies and developers want their application to reach users on other platforms. This has led to the rising requests for cross-platform applications, The Mobile Life receives regular requests to make applications that covers several platforms, the most popular today being Android and iPhone.

The main reason of this study is to investigate the idea of reusable code, meaning that code written for an application on a particular platform can be reused when implementing the same application on a different platform, and to what extent this could be applied when porting Speedflirt. The ability to reuse code in porting could drastically reduce development time for a cross-platform application. Building a mobile-web application has traditionally been the solution to achieve support for multiple platforms but as a mobile-web applications have certain shortcomings, which will be discussed later in this chapter, building native applications for every platform has been the most common choice for developers.

Several frameworks aimed for mobile cross-platform development have emerged in the last years, trying to bridge the gap between mobile-web applications and native applications. This study will investigate the potentials of these frameworks and how a cross-platform application will differ from a natively developed application.

By discerning the strengths and weaknesses of each framework this study could provide information of the most suitable framework to use when developing future cross-platform applications. These findings will then be used to decide which method or framework is best suited for porting Speedflirt.
3.1 Current technologies

This section will investigate some of the most popular frameworks for cross-platform mobile applications today and compare them against native development and mobile web development. Interesting variables are code portability/reusability between platforms, performance and hardware access.

3.1.1 Native applications

The mobile world is getting more and more fragmented, there are several popular mobile platforms today, and it is very likely that they will be available in the future as well. Most smartphones sold today converge around the same core design paradigms, such as Internet access, an online marketplace for downloading and installing applications, GPS-positioning, built in camera and a touchscreen interface. But, while the capabilities of mobile phones are getting more and more similar, the APIs used by each platform are very different. As seen below the mobile platforms popular today differ in many aspects.

<table>
<thead>
<tr>
<th>Platform</th>
<th>Language(s)</th>
<th>IDE</th>
<th>OS X</th>
</tr>
</thead>
<tbody>
<tr>
<td>iPhone</td>
<td>Objective-C, C</td>
<td>Xcode</td>
<td>OS X</td>
</tr>
<tr>
<td>Android</td>
<td>Java, C, C++</td>
<td>Eclipse/Netbeans</td>
<td>Windows, Linux, OS X</td>
</tr>
<tr>
<td>Blackberry</td>
<td>Java</td>
<td>Eclipse</td>
<td>Windows</td>
</tr>
<tr>
<td>Symbian</td>
<td>C, C++, QT, Java</td>
<td>Any text editor</td>
<td>Windows, Linux, OS X</td>
</tr>
<tr>
<td>Windows Phone</td>
<td>C#, VB.NET</td>
<td>Visual Studio</td>
<td>Windows</td>
</tr>
<tr>
<td>webOS</td>
<td>HTML, Javascript, CSS</td>
<td>Any text editor</td>
<td>Windows, Linux, OS X</td>
</tr>
</tbody>
</table>

Developing a native application will grant access to all native features such as GUI elements and hardware features. This allows for creating very advanced applications which are very hard or even impossible to create using mobile-web technologies.

The diversity of these frameworks, on the other hand, put a heavy burden on developers when an application should run on more than one platform. Many platforms utilizes the same languages, such as Java, but as they still have their own frameworks and libraries the code-reusability when switching platform very low, i.e. a developer cannot reuse code from a Blackberry application to make an Android application even though the programming language is the same. Because the low degree of code reusability, porting applications between platforms is both time consuming and costly for developers.
3.1.2 Mobile-Web Applications

Implementing an application as a mobile web-application would have several advantages compared to using native porting. Where the main advantage would be that the codebase created can be reused for other platforms with minor changes. A web application is built using standard web technologies such as HTML5, JavaScript and CSS3.[8]

Table 3.2: Mobile web application architecture

<table>
<thead>
<tr>
<th>Meta Configuration</th>
<th>XML, PList, Json, Meta Tags</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Access</td>
<td>JavaScript APIs</td>
</tr>
<tr>
<td>Data Storage</td>
<td>HTML 5, Gears, JavaScript APIs</td>
</tr>
<tr>
<td>Network Access</td>
<td>AJAX, Script Loading</td>
</tr>
<tr>
<td>Logic</td>
<td>JavaScript</td>
</tr>
<tr>
<td>User Interface</td>
<td>HTML, CSS, DOM, Images, Canvas</td>
</tr>
</tbody>
</table>

As mobile web application is essentially a web page and not a native app the possibility to use the native appstore as a distribution channel will not be available. Appstores for mobile web applications exist today but they are essentially a web page that provides a collection of links to mobile web applications. The user therefore first need to know the address to this appstore and later find the correct application. The implemented native appstores have the advantage of simplicity, by just starting the appstore the user is presented with all apps available on that platform. As a mobile web application will run in the web browser installed on the telephone no porting is necessary for a mobile-web application as most mobile platforms today have an installed browser that supports the technologies used.

A web application will run in the web browser installed on the telephone which will entail certain complications as different platforms use different browsers, such as MobileSafari, Chrome, Fennec and Opera Mobile. Since browsers have a built in JavaScript engine performance fragmentation occurs. Some browsers run JavaScript faster than others[7]. In addition to the performance fragmentation the CSS interpretation may differ slightly between browsers as well. These factors reduce the portability of code slightly as minor changes may be needed in order to support specific browsers.

Another drawback is that the performance of a mobile web application is not very good compared with a native application. When implementing CPU-intensive processes the best solution today is a native application. A mobile web application creates more overhead-code and cannot take advantage of native hardware acceleration and multithreading which decreases performance.[15]

Access to of hardware specific features such as the camera and accelerometer is limited in a web application. There exist certain JavaScript APIs that grant access to these features. The most common API is PhoneGap, which will be discussed in the next section.

3.1.3 PhoneGap

Phonegap[43] is an open source framework that provides mobile-web applications access to hardware and platform specific features such as camera, accelerometer and GPS. By providing an API that is an abstraction of the native mobile platform the same code can be used on different mobile platforms with minor or no change. However, not all hardware specific features are available for all platforms[44]. Developing a Phonegap application is very similar to developing a web application, using HTML, CSS and JavaScript with the main addition that the developer can make JavaScript calls to the PhoneGap API to access
Chapter 3. Cross-platform development frameworks

PhoneGap supports development for a number of different mobile platforms including iPhone, Android, Blackberry and Symbian [1].

PhoneGap also provides the ability to compile your web application as a native app and distribute it on the native app market. This does not mean that the application is recompiled completely into a native app. The mobile-web application developed is only wrapped in a native shell. In practice this means that a user can download the application from the native market, start it just like any native application but the application actually starts the default web browser for the phone and loads the corresponding mobile web application. This additionally means that the performance of the application relies heavily on the default browser of the phone or more specifically on the JavaScript engine used by the default browser.

The ability to easily create applications for several platforms has made PhoneGap a popular choice for developing multi platform applications. One recent example is the Skatteverket mobile application, shown in figure 3.1, which has been developed as a web application and packed in PhoneGap. As the browsers differ between the platforms some small differences can be seen between the applications which is something that a developer should have in mind.

Figure 3.1: Skatteverket mobile application

(a) iPhone Version
(b) Android Version

![iPhone Version](image1)
![Android Version](image2)

3.1.4 Appcelerator Titanium

Appcelerator Titanium[3] is a framework that, like PhoneGap allows for platform specific features through a JavaScript API[2]. The big difference lies in that Titanium also provides access to the native UI components, allowing developers to make applications that feel more native than a mobile web application wrapped in PhoneGap[1].

Titanium does not make use of HTML and CSS to create and specify the interface but instead uses XML to specify the layout and JavaScript to access native UI elements. This makes for a faster and more native feel but also entails certain restrictions. One restriction is that only the standard version of the native UI elements for each platform are supported by the Titanium API, making customization of these UI elements more complicated than with regular native development. In order to create customized elements the developer needs to code a native module that can be called via JavaScript in Titanium.

When accessing UI elements through the Titanium API the developer must specify both the element and platform such as `Titanium.UI.Android.Slider` to access the Android native slider or `Titanium.UI.iOS.Phone.Slider` to access the iPhone native slider. Because
of these calls a developer cannot reuse the UI specific code when porting an application
developed in Titanium making the degree of reusable code lower than in a mobile-web
application or a PhoneGap application. But, as all code handling logic and backend can be
reused the degree of code reusability is much higher than with native porting.

3.1.5 Finding the most suitable framework

The one common factor among all the frameworks is the use of web technologies such as
JavaScript. So it may seem like the only available solution in order to gain code reusability
over mobile platforms tends to lie in the use of JavaScript. Having the fastest JavaScript
engine has become somewhat of a power struggle among browser developers so Google,
Apple, Microsoft, Opera and Mozilla are continuously updating their implementations to
outperform the competitors. This struggle pushes the performance of JavaScript to the point
where it is starting to become a usable alternative for developing mobile applications[7]. And
with the use of frameworks such as PhoneGap that also allow access to hardware specific
features and access to the native appstore the divide between a mobile-web application and
a native application is closing.

The area where mobile-web applications comes up short is the user interface. As most
mobile platforms have advanced abstractions of common UI elements which cannot be
reached from a mobile-web application since developers must create their own interface
elements. When the user-interface design idioms can be very different between platforms
this can be very challenging task. The Titanium framework can be used to bridge native
UI elements, thus enabling the creation of a user interface that follow the design paradigms
of the platform.

But as all frameworks have their advantages and drawbacks, the requirements of the in-
dividual application must be considered in order to find the most suiting framework for the
application being developed. Determining which platforms the application should support,
which hardware features are required and how the interface should look and behave can give
a good indication of which framework is best suited. Some general indications of framework
suitability are as follows.

Mobile-web applications have the advantage that they can be implemented for a large
array of platforms in a short time using only one codebase. As a mobile-web application is
outside the native appstore updates does not have to be submitted and reviewed before they
come into effect. This makes the administration of the application faster and simpler. Not
having access to the native appstore is also a large drawback in terms of monetization as
Internet services are mostly free and the most reliable revenue model is advertisement. As
all appstores have an implemented business model with support for payments a mobile-web
application is not the best choice if plans are to charge money for the application.

Adding the PhoneGap framework to a mobile-web application will grant the application
access to the native appstore making it a good choice if plans are to charge money for the
application. PhoneGap is also a good choice of framework if the application will utilize
hardware specific features of the phone such as the GPS or the camera. When looking at
the user interface a web-application or PhoneGap does simply not have the same capabilities
as a native application. As a PhoneGap application runs in the Internet browser a complex
user interface that still will run smoothly on the phone will be harder to achieve than with
a native application. Therefore mobile-web and PhoneGap are best suited for applications
with a simple web-like interface.

For applications that require a more complex interface than a mobile-web application
can offer either a Titanium or a native application is the best choice. Titanium has the advantage that much of the codebase can be reused for several platforms, but the drawback that only the standard version of the native UI elements are supported. If the application will incorporate an UI with many custom elements a native application is preferable.

3.2 Porting Speedflirt

Deciding which hardware specific features to make use of in the application is an important step in finding the most suitable porting technique, since certain hardware features are difficult or impossible to access from certain porting frameworks. The Speedflirt application will make use of several hardware features and functionalities. The features needed for Speedflirt are as follows.

**Location.** As Speedflirt concentrates on finding singles near the users location, GPS or phone triangulation will be used frequently in the application. In order to provide the user with an up to date set of singles the location of the user will be sampled with regular intervals.

**Camera.** The camera will be used for taking profile pictures. The registration process can be much more streamlined if the profile picture can be taken directly with the built in camera of the phone. Without camera access the user would need to manually upload his or hers profile picture to the Speedflirt database via a computer which is far from ideal.

**File Storage.** Accessing the filesystem of the device in order to store user settings and images on either the SD-card or in the internal memory of the device. Downloading images in the coverflow before they are displayed and buffer them in local memory would increase performance and user experience, when downloading in the background the user will be presented with a more responsive coverflow much like the iPhone version. Having pictures stored locally enables the users to go backwards in the coverflow, viewing pictures multiple times while only downloading them once, thus reducing overall bandwidth required by the application.

**Phone Settings.** In order to extract the unique identifier (UDID) of the phone the application needs permissions to access the phone settings on the Android platform. Speedflirt will make use of the UDID of every phone as the identifier for subscriptions and payments. This unique identifier is also needed for Android cloud-to-device messaging.

**Multithreading.** In order to have a fluid and responsive user interface time-consuming tasks such as server calls and image processing should run on separate background threads. Complete control of these background threads is desired, including the possibility to abort execution of code in a thread that is no longer needed.

**Notifications.** Some feature for notifying the user that a new flirt or new message has been received when the application is not running is needed. The Android platform already has a system for notifications that will be used. The two main ways to implement notifications are either implementing a custom background service on the phone that check for updates with a regular interval or to implement Android cloud-to-device messaging.

3.2.1 Choosing framework for Speedflirt

Because the original iPhone version of Speedflirt is implemented for iPhone platform using the Objective-C programming language no code could be reused regardless of the chosen framework. So, in deciding which framework to use when porting Speedflirt, code reusability for the first version was not a factor.
3.3 Conclusions

The number of desired hardware and native features to use for Speedflirt rules out implementing Speedflirt as a mobile-web application, since access to these features is far too limited. Using the PhoneGap framework would allow access to the hardware and native features required for Speedflirt. However it was decided that with the lack of multithreading and utilization of hardware acceleration, the creation of a interface that was fluid enough to meet the requirements could not be ensured.

Using the Titanium framework would require the implementation of native modules for the customized UI elements needed for Speedflirt but would provide a logic and backend codebase that could be reused for further porting of Speedflirt other platforms.

As Android is a very popular platform with many users doing a native port of Speedflirt would ensure the creation of an application with maximum utilization of platform specific features. Also with Android users being such a big target group, this would allow for the creation a user interface that would be as fast and responsive as possible for that target group.

Ultimately the decision was to make a native port of Speedflirt to the Android platform. For the other platforms portability of code was favored so it was decided that a PhoneGap application would be developed for the remaining platforms e.g. Blackberry, Symbian and WP7. By using PhoneGap all hardware features needed for Speedflirt would be available and one codebase could be used for all platforms. Some simplifications of the UI would be required for this version, but this was accepted as code portability was deemed more important. This thesis will only involve the creation of the Android version of Speedflirt.

3.3 Conclusions

As the fragmentation of the mobile world does not show any tendencies to decrease, rather it increases as more mobile platforms are launched, there will be a continued demand for cross-platform mobile applications.

In order to be able to create code that have the ability to execute on several platforms some abstraction is needed between the actual platform specific and hardware features and the code written. The abstraction used by the frameworks investigated in this study is the web browser, as they all utilize web techniques available on all platforms.

It seems that the higher the abstraction, the more code can be reused. But a high abstraction also restricts access to platform-native and hardware features. However, as these frameworks are still being developed and new native features are added with every iteration this trade off is becoming smaller and smaller.

As the frameworks included in this study all have different amounts of abstraction a developer must make the choice between fast deployment and a native feel prior to choosing a framework. If the interface and native feel is important a native application or a Titanium application is the right choice, but if, on the other hand, fast deployment on several platforms is deemed more important a mobile web application or perhaps a PhoneGap application is the right choice.

The main conclusion of this study is that the the individual requirements of an application should determine which framework to use for development. There will be a loss of native features when developing cross-platform applications, but as these frameworks are developing the gap between native and cross-platform applications is closing.
Chapter 4

The Android framework

This chapter will describe and discuss some of the different parts of the Android framework, focusing on the parts that were of greater importance during the development process. The chapter firstly introduces the Android framework, continuing with sections such as xml layouts, fragmentation, 9-patch graphics and the REST protocol. The parts covered in this chapter were tools and concepts that were a prerequisite to successfully achieve the port of Speedflirt.

4.1 Android

Android is an open-source operating system for mobile platform developed by Google. Initially released in October 2008, it has become the best selling smartphone platform of today[21]. Applications that extend the capabilities of the Android platform are provided through the Android Appstore, which today has more than 150,000 individual applications. Applications for the Android platform are developed in the Java language utilizing the Java libraries provided by the Android API.

4.1.1 Architecture

The architecture of the Android platform is very different from that of the iPhone platform. An iPhone applications is built using the CocoaTouch API and the Objective-C programming language while an Android application is built using the Android Java API and the Java programming language. Further differences lies in memory handling where the iPhone requires explicit memory handling while Android utilizes the Java garbage-collector for memory handling. The platforms also differs in the way the GUI is created, while the iPhone utilizes a graphical drag and drop application called Interface Builder the GUI for an Android application is defined in separate XML files. Because of these differences, Speedflirt had to be completely redesigned with respect to the new underlying architecture.

The architecture is illustrated in figure 4.1, and is a software stack based on a Linux kernel consisting of five different layers.

The application layer consist of the applications installed on the device, written in Java using the underlying framework API’s. Android ships with a set of core applications installed such as SMS and email, a web browser, and applications for contacts, maps, calendar etc.
The application framework exposes all the framework APIs to a developer. Through these API’s a developer can access a set of underlying systems and services used by all applications, including:

- **UI elements.** A set of generic UI elements, such as buttons, views, grids, text boxes that can be used to build the user interface of the application.

- **Resource Manager.** The resource manager provides access to non-code resources for example to graphics, localized strings and layout files that are used by applications.

- **Notification Manager.** The notification manager allows access to the static Android status-bar, allowing applications to send information to the user even though they are not currently being displayed.

- **Activity Manager.** The activity manager controls the lifecycle of an application.

These components are specifically designed for re-usage and customization allowing a developer to build rich and innovative applications from simple and generic parts.
4.1. Android

The libraries are a set of C/C++ libraries used by various components throughout the Android platform. These libraries consist of low level implementations of CPU and memory intense processes such as playback and recording of media, 2D and 3D graphics engines, vector rendering and database engines. A developer can access these libraries through the API's exposed by the application framework.

The Android runtime contains a set of libraries that provides most of the functionality of the Java core libraries and the Dalvik Virtual Machine. Every application is run within a separate instance of the Dalvik VM. Designed specifically for the Android platform the Dalvik VM is designed so a device can run several virtual machines efficiently. Dalvik runs applications that are compiled to a .dex format which is optimized to leave a small memory footprint. For multithreading Dalvik relies on the underlying Linux kernel.

The bottom layer consists of a Linux kernel that acts as an abstraction between the hardware and the rest of the software stack. This layer is responsible for memory and power management, multithreading, hardware drivers, network access and inter-process-communication.

4.1.2 XML – Layouts

A common way of building UIs in Android is through XML layout files where they have the advantage of separating the presentation from the application code that controls its behavior. Each graphical component in Android originates from the ViewGroup or View class and together forms a tree hierarchy of graphical elements. The elements defined within the xml files each corresponds to a node within a tree hierarchy that later forms the Android UI. A ViewGroup is a group of Views that have a specific relation to each other, for instance, laid out in a predefined way. An example of a ViewGroup element is the LinearLayout, which places its children in a linear fashion (horizontally or vertically) where each ViewGroup can have nested ViewGroups to form complex UIs. The name of the element is respective to the corresponding Java class, so for instance, a <TextView> element creates a TextView in the UI and a <LinearLayout> element creates a LinearLayout view group[34].

Figure 4.2: Illustrates an example of the structure of a tree hierarchy that forms the Android UI[34].

![Diagram of a tree hierarchy](image)

4.1.3 Fragmentation

One thing that separates the Android platform from the iPhone platform is fragmentation. The iPhone platforms consists of three different phones, iPhone, iPhone3g/s and the
iPhone4, which all have the same amount of physical buttons, the same UI and screen size although the iPhone 4 have twice the resolution. The Android platform is more fragmented including different screen sizes and resolutions, and in addition several different phone manufacturers have implemented their own user-interface such as HTC Sense, Samsung TouchWiz or the Sony Ericsson Rachael. In order to provide a consistent user experience, Android applications must be flexible enough to run well across different devices.

4.1.4 9-Patch graphics

NinePatch images, or NinePatchDrawables are graphic elements that have the ability to resize its dimension according to the desired content [20]. Because of the variety of screen sizes among Android devices the framework included this feature to produce nice looking graphics to a dynamic context.

More specifically, to produce a component that should be able to adjust to a dynamic size, the component is equipped with a NinePatch image as its background element. Depending on layout preferences supplied by the developer, the element will adjust to screen size, screen density and component characteristics. A typical example within Android applications is the use of custom images in button components. Depending on the title, the button will adjust its dimensions in order to accommodate the space needed to enclose the entire text of the title.

The NinePatch image is a regular PNG file that have 9 patches (regions of the image), defining which part of the image that should be able to scale in the appropriate area. By providing the image with patches, it prevents the image from scaling as one unit which often results in compromising the aesthetics. Android will scale the components according to the preferences provided by the developer and the screen size, by providing NinePatch graphics with defined, scalable regions, the component will look better on different screen sizes.

Figure 4.3: Illustration of the concept of NinePatch images

The NinePatch image, apart from its content, includes a 1-pixel wide transparent border on each side. In total, 4 lines can be drawn within this border to inform Android of how this scalable component should behave. By defining a 1-pixel black line at the top and left side, the component defines which part of the image that should be able to be scaled leaving some areas outside the black line static. By defining lines on the right or bottom side of the border, the component informs Android of how much of the image that should be left for
the actual content.

As seen in Figure 4.4 by drawing the top and left lines, the image is automatically divided into 9 patches, each scaled separately. In the figure, the rectangle created by the top and left lines constitutes the middle area, scaled in both directions. The resulting patches will all scale in different ways, helping the component to preserve its appearance.

Figure 4.4: Illustration of the concept of patches within NinePatch images [23]

For the earlier example with the button component, the top-left lines defines 9 patches that will be scaled (or not scaled) to accommodate the required space where the center patch is the only area that will be stretched both horizontally and vertically. If the graphics provided were not 9 patch, the entire image would have been stretched both horizontally and vertically in order to fit the required size, resulting in a compromised appearance.

As a summary, the top-left lines divides the graphic into patches that Android can stretch in order to scale the component while the bottom-right lines defines the area that the button’s content can be laid out (the button title) [20].

4.2 REST

Representational State Transfer Protocol (REST) is a software architectural principle for distributed hypermedia systems, first introduced by Roy Fielding in 2000[14]. This principle describes how services should be provided in a client-server relationship. The Speed flirt server is implemented using the principles of REST and this section will discern the core principles of this architecture.
One of the core principles of the REST-architecture is the existence of resources, each identified with a global URI. In order to manipulate these resources one must explicitly use HTTP-methods that follow the RFC 2616 protocol[46]. The REST design principle establishes a one-to-one mapping between create, read, update and delete operations and HTTP methods.

- Create a resource on the server, HTTP-POST
- Retrieve a resource from the server, HTTP-GET
- Update a resource on the server, HTTP-PUT
- Remove a resource from the server, HTTP-DELETE

With the HTTP-request the client can attach the parameters needed for executing the request. The parameters are attached to the URI for the desired resource. Below is an example for a request with two attached parameters

Example:

- URI: http://www.example-service.com/login
- Parameters: username=john pass=doe
- Complete request: http://www.example-service.com?username=john&pass=doe

The response data from the server is packaged inside the HTTP-body of the response. Following the REST-architecture this data should be put in the HTTP-body as any valid Internet media type, but is most often sent as either JSON or XML. Also certain requests contain data packaged inside the HTTP-body.

A webserver based on the REST-architecture should be stateless, meaning that no session-state or client context is stored on the server. The client should always send complete and independent requests to the server meaning that the HTTP header and body of the request should contain all the parameters, context and data needed for the server to generate a correct response. Because the server can process all request without retrieving any context or session data the server performance and scalability is increased.

As the REST-architecture is based around standard HTTP-methods it is easy to implement clients on different platforms, therefore it is suitable to implement the server for multi-platform mobile application according to the REST-architecture. In the case of Speedflirt, the iPhone and Android version will be able to use the same interface and methods to interact with the server.
Chapter 5

Implementing Speedflirt for Android

The following chapter will describe the practical work of building Speedflirt for the Android platform. The beginning of the implementation was initially divided into two parts, the REST-client and the Coverflow gallery, which will be discussed in detail in this chapter. As these two parts formed the backbone and the main UI element of Speedflirt, further implementation where mostly focused around mimicking the behaviour and look of the iPhone version of Speedflirt. This entailed creating custom "iPhonesque" elements, creating graphics that followed the iPhone version of Speedflirt but still where scalable in order to fit the variety of screen sizes on the Android platform and mimicking the navigation of a iPhone application.

Problems that occurred was mostly fragmentation issues as phone manufacturers often implements their own version of specific elements that do not follow the Android API. Another time consuming issue was memory management, as some parts of Speedflirt are quite memory consuming Java's garbage collector was not sufficient some explicit memory management had to be implemented.

5.1 Implementing a REST-client for Android

One of the early essential parts of the Speedflirt project was to implement a REST-client for the Android platform. Because Speedflirt relies heavily on client-server communication one of the first tasks of the project was to build the REST-client. The purpose of this client is to create and handle all requests and responses sent back and fourth between the Speedflirt server from the client. The requirements of the client were that it should be able to handle asynchronous calls from both the Speedflirt application and callbacks from the Speedflirt server. During Google-IO, May 19-20, 2010, there were discussions on the architectural considerations that should be made when developing a REST-client on the Android platform[10]. With some modifications to suite the Speedflirt application a design pattern was established. The different classes in the flowchart below will be examined in the following sections.

23
5.1.1 RestService

The main task of the RestService class is to create and start threads that the request can execute, call the correct method for the request in the RestProcessor class and finally create a callback to the UI when the server has responded to the request.

The RestService is started when one of the UI classes broadcasts an Intent that informs other parts of the application that they have a request to the Speedflirt server. This Intent contains all information the server needs to generate a correct response, thus following the REST architecture. When the Intent is received by the RestService, the service will create and start a new thread that is responsible for the request execution. Because the time it takes for a request to be processed and returned by the server can be several seconds, the creation of a thread is important. Running the request on a background thread ensures that the main UI-thread does not lock up during these time consuming operations. The RestService also handles thread safety meaning that if a current thread is running, all incoming requests will be queued making sure only one thread is executing at any given time.

After a thread has been started the RestService retrieves all information about the request from the Intent, calls the correct method in the RestProcessor, passing all information given by the Intent and awaits the response. When the response comes back it is packaged inside a new Intent and broadcasted back to the UI class that sent the request.

5.1.2 RestProcessor and JSONParser

The RestProcessor is the main hub in the REST-client. It has a corresponding method for every request that can be sent to the server, which is called by the RestService. When any
method is called, the RestProcessor takes the information provided by the call and converts it to either header or body parameters depending on the type of request; these parameters are then passed to the RestClient class.

Upon receiving a response from the RestClient class the RestProcessor passes this response through the JSONParser. The purpose of the JSONParser is to take the raw response from the server, which is in JSON, and convert it to a format better handled by the Android platform. The JSONParser also handles eventual errors from the server where the response is packaged in a custom wrapper and sent back to the RestService. In order to handle both errors and correct responses from the server a custom wrapper was used. The wrapper basically contains an error integer and a response object. If the response is correct the error integer is set to 0 and the response is set. In the case of an error the integer is set to the corresponding error-code and the response is set to null.

5.1.3 RestClient

The RestClient is the class that handles all communication with the server. Its main task is to create the HTTP-methods used to send request and set the correct URI, header, parameters and body to any given request. The RestClient has corresponding methods for all the HTTP-methods the REST-architecture supports, namely GET, PUT, POST and DELETE. When one of these methods gets called the RestService will create a HTTP-request and send that request to the server. When the response comes back from the server the RestClient write the response to a JSON-string and passes it back to the RestProcessor.

5.2 Implementing a ”Coverflow” gallery

The main functionality of Speedflirt is to browse through profiles in the vicinity, matching the user’s preferences as well as browsing through flirt proposals from other users. Because of the gallery being a crucial component it was also one of the first tasks that was attended to in the project. The gallery component on the IPhone was implemented with the widely recognized style called ”cover flow”. Cover flow is a collection of e.g. images that is displayed in a certain style where the image currently in focus will reside in the center of the screen with it’s adjacent images tilted in a predefined angle (see Figure. 5.2 for a graphical illustration of the cover flow concept).

Figure 5.2: Sketch of a “cover flow” styled gallery

Within the android framework there is a standard component named “Gallery” [31] that is commonly used for displaying a series of images. The gallery component does not have the ability to behave as a cover flow without customization (See 5.3 for the standard gallery
component), so the inner workings needed to be adjusted in order for the component to fit the requirements. Due to the existence of useful functionality such as horizontal scrolling and selections, the component was an ideal module to use as a base for the final gallery.

Figure 5.3: Standard gallery component within the Android library

Because of Android framework being open source, the source code for the gallery component was openly available. One way of reusing existing components within Java coding language is simply to make sub classes of an already existing class and add the desired behavior that is needed for the new, custom component. Due to the importance of the gallery component, it was critical to ensure a high user experience when browsing through potential matches for a user (or flirt proposals). Factors that had an impact on perceived performance, such as sufficient loading times of gallery images and smoothness in image transitions were given a high priority. To achieve the wanted behavior with sufficient performance, sub classing was not enough. The need for private fields, alternations in the components inner layout and image cache functionality within the gallery component implied that the complete source code for the gallery component was required and needed modifications.

All images are retrieved through the Speedflirt web service where each image has a dimension of 480x480 pixels. These large images exposed the mobile client to heavy operations of downloading and displaying images as the user browsed through the gallery. In order to provide the user with a smooth user experience without waiting for images to load properly, the gallery needed to preload images to speed up the perceived performance. To achieve this, the gallery class implemented a separate caching of items (See 5.4 for a graphical illustration of the cache). The items before and after the items currently displayed was stored in memory to provide fast population of new items. The drawback of this approach was the allocation of memory for objects not currently in use. The gain of “flow” when browsing the gallery was considered to outweigh the drawbacks of memory consumption and ensured a better user experience. To prevent the client from crashing due to memory loss, the cache was limited to a maximum of one view on each side. To further improve the loading performance, the client stored already downloaded images on the SD-card or internal storage. By doing this, the client could access already browsed images much faster without connecting to the web service. This approach was specifically efficient whenever the user browsed back
5.2. Implementing a "Coverflow" gallery

and forth through the gallery.

Figure 5.4: Cache and preloading of images to display next in the gallery

The architecture of the final gallery component is illustrated in 5.5. Below a brief description will be provided for the functionality and the responsibility of each class.

The custom implemented gallery consists of four different classes that together form the component: NativeAdapterView, NativeAbsSpinner, NativeGallery and SpeedflirtGallery. NativeAdapterView is an abstract ViewGroup subclass, which directs the behavior of its children to ensure a wanted behavior.

NativeAbsSpinner incorporates the wanted behavior of NativeAdapterView such as filling the layout with data (from external adapters) and handling user selections; furthermore, it also has the responsibility of managing recycled views during layout operations. NativeAbsSpinner also needed to implement new behavior not already defined in the standard component, namely functions capable of appending data without causing an unwanted update behavior and preloading a image buffer of subsequent images, which is explained further below.

NativeGallery inherits the behavior from its parents and hence has the possibility to access selected views, its content (provided by adapters), preloaded views etc. Among others, the distinct responsibility of NativeGallery was to layout its content in a horizontal fashion and handle views that has been scrolled outside the view. The class also enables the behavior of scrolling by detecting user inputs such as button or touch events. The functionality of clearing and repopulating the gallery’s content was also added due to extensive memory problems further explained below.

SpeedflirtGallery is the final class within the gallery component which takes care of the actual transformation of images such as rotations and translations on the x, y and z axis. The class has other responsibilities such as determining the resulting behavior of flings, distance between images, the properties of shadows etc. SpeedflirtGallery applies the transformations needed to mimic the style of a cover flow and is the most important part of the custom component.

The Speedflirt web service provided the mobile client with chunks of image data, e.g. 20 images per request. Whenever the users browse through all images, the client appends another 20 images to display next. Instead of using default behavior by clearing the data content and redrawing the entire gallery with new data (resulting in a few millisecond
flickering), a new method was added to the gallery. The method enabled the component to append new data without redrawing the entire data source and therefore did not invade the user experience by exposing the user to annoying flickering or interrupted scrolling.

Outside the scope of the gallery component the system utilizes a couple of help classes to efficiently achieve the task of populating the gallery with images: ImageAdapter, ImageProcessor and ImageFactory. ImageAdapter is the adapter that is coupled with the NativeAdapterView class and is used as an external data source that populates the gallery with data. ImageProcessor is a threaded instance that handles an ordered queue of images to load into the adapter used by the gallery instance. ImageProcessor uses a sliding window to abort the loading of images that have moved outside the screen (turned obsolete). The queue implements the LIFO (Last in first out) protocol, meaning that the last image request to arrive gets the highest priority. Each request was dispatched to another class, ImageFactory where the class was responsible for the actual streaming of data; furthermore, the class also handled manipulation of the images such as scaling and the creation of image reflections.

When loading, ImageProcessor will first attempt to fetch each image (via ImageFactory) from the local storage. If the attempt fails, the processor will send a request to load the image from Internet. The flow chart shown in Appendix:A illustrates the sequence of tasks required in order to load one image into the adapter, and further into the gallery.

### 5.3 Mimicking iPhone UI

As most of the functionality and logic of Speedflirt was situated at the Speedflirt server, implementing the REST-client along with some logic on the client side, completed the functionality part of the port. The functionality and logic of Speedflirt was already available on the Android platform, which meant that presenting this functionality in the same manner as the iPhone version would finalize the port.

One problem is that, since the iPhone platform has a different system for presenting the transition between views than Android, the standard Android architecture for presenting
new views could not be used.

Also, the UI of Speedflirt is based around an iPhone tab-bar where some of the tabs contain a static navigation controller to flip between views. The Android platform has a native tab-bar that could be reused for Speedflirt but the concept of a navigation controller does not exist on Android as the physical buttons of the Android device fills that purpose.

The navigation controller of an iPhone application is static, meaning it is always displayed at the top of the screen while new views are animated in and out of the screen below it. As an Android application is based around Activities, all which contain a ContentView that take up the whole portion of the screen, maintaining static elements in the UI could not be done. A solution would be to put an identical copy of the static elements in every Activity creating static appearance of these elements, but as shown in figure 5.6 the “static” navigation bar would be animated when entering or leaving the screen, revealing that they are, in fact, not static.

Figure 5.6: Static elements in Android

This problem was solved by the use of a native Android class called a ViewFlipper[35], which has the ability to animate the transitions between two or more Views. Defining the ViewFlipper as a portion of an Activities ContentView allowed for animating custom views in and out of that portion of the screen, thus mimicking the behavior of an iPhone.

All the graphics such as logos, buttons and icons created for the iPhone version of Speedflirt were available for reuse in the Android version. Two versions of the graphics existed, iPhone standard version and a retina display version (iPhone4 with twice the resolution). As the Android platform includes devices with a vast array of resolutions, the iPhone graphics could not be reused directly. In order to support a greater variety of screen resolutions all graphics were converted into 9-patch bitmaps, enabling all graphics to scale according to the resolution of the device.

5.3.1 Implementing iPhone components on the Android platform

In order to fully match the interface of the iPhone version of Speedflirt some UI components had to be implemented on the Android platform. As the components provided by the Android API are both open source and flexible, these components could be used and reconfigured in order to look and behave like their iPhone counterparts.
AgeSpinner

When a user is asked to choose one value from a set of predefined values on the iPhone platform a component called UIPickerView should be used according to Apple’s design guidelines. A typical example when using this component is the alarm clock, where the user has to choose which hour and minute the alarm should go off. The appearance of this component is that of a white wheel that is spinning around a horizontal axle with the active selected value in the center. To select a new value, the user flings his finger to spin the wheel. In Speedflirt this component, as seen in Figure 5.7a, was slightly modified in the way that the “wheel” spun around a vertical axle instead of a horizontal.

The Android platform does not have a standard component filling the same purpose as the UIPickerView, but instead has specialized pickers depending on what type of input is expected from the user e.g. DatePicker and TimePicker for picking dates and time. Also these components does not look or behave like the iPhone UIPickerView and was thereby not ideal to use for creating the AgeSpinner.

One native Android component that did behave exactly as desired was the image gallery. By redefining the gallery to contain numbers instead of pictures, a horizontal scrolling list containing numbers was created. By taking the graphical look of the iPhone AgeSpinner, converting it to 9-patch bitmaps, and setting it as the background for the converted image gallery an Android version of the AgeSpinner was created.

Figure 5.7: AgeSpinner comparison

![AgeSpinner comparison](image)

Camera

During the registration process the user is asked to attach a profile picture which can either be taken directly using the onboard camera or the user can choose to pick a photo from the phone’s photo library. The iPhone version of Speedflirt uses a menu that slides up from the bottom of the screen which displays these choices to the user as shown in figure 5.8.

Figure 5.8: iPhone dialog for using the accessing camera or gallery

![Camera](image)

Creating this menu on the Android platform, the same approach was taken as in the creation of the AgeSpinner. Firstly, an Android native component that could be used had
to be identified. In this case no component existed with the required properties, so the basic building block of any Android UI, the View class was chosen. A View object represents a rectangular area of the screen and handles drawing and event handling inside that area. As a View can contain any number of child components the View became the menu and buttons where placed inside it. Once again the graphics where taken from the iPhone version and converted into 9-patch bitmaps in order to handle the fragmentation of screen resolutions and a custom animation was scripted to mimic the menu sliding up from the bottom of the screen as in the iPhone version.

For taking or viewing pictures, preinstalled camera and gallery applications that are part of the Android API are used. By calls to the API these applications are launched and when finished return a picture back to the application who called it. This procedure of using native preinstalled camera and gallery applications through the API is also used for picture capturing on the iPhone platform.

During the implementation of this feature the fragmentation of the Android platform became noticeable. Apparently it is quite popular among phone manufacturers to implement their own version of the camera and gallery applications to distinguish themselves from their competitors. However these implementations do not always adhere to the Android API. Among the devices that Speedflirt was tested on, Sony Ericsson, Samsung and HTC none of them worked in accordance to the API, having different return types and different input parameters when starting the application. As these implementations do not adhere to the API and also differ from one another, the developer is faced with a huge challenge. In order to support all devices a developer must write code that can handle every variation of these implementations and as they do not follow the Android API, the only way to test this is directly on the device. This entails having access to all devices that do not use the standard implementation of the preinstalled applications.

Another component that is frequently modified by manufacturers is the software keyboard. During the implementation of Speedflirt it was found that the keyboards from Sony Ericsson and Samsung does not follow the Android API. This problem also required extensive and time consuming testing over a variety of devices in order to establish the differences between the actual functionality of these keyboards and the documented functionality of the Android API.
Chapter 6

The effect of porting applications on user experience

When applications are transferred from one platform to another with the requirement of preserving the graphical look, problems might arise. Standard components and interaction styles might not be as appropriate on other platforms and the resulting user experience might be compromised. What will be the effect of porting applications with a predefined design idiom onto another platform?

In this study, the questions in focus are:

- What is User experience?
- Which factors affect user experience?
- How does porting of Speedflirt affect User experience?

Modern interface designers have acknowledge the fact that in order to achieve appreciated design solutions, usability is not enough. As a response to the dominance of task-related usability paradigm the interest for user experience design has increased [18].

While usability focus on utilitarian aspects such as goals, task efficiency and cognitive information processing, user experience (UX) is directed to more abstract goals such as designing an interface to be stimulating, relaxing, efficient, or fun[45]. Despite the increasing popularity of UX, the actual meaning of the buzzword “user experience” remains vague and redefined from a variety of perspectives [45] [22].

A recent seminar performed in February 2011 by leading UX experts made an attempt to produce a united view on what UX is and what it is not as well as which factors that affect UX [12]. They preferred to distinguish UX into three categories, each from a different perspective: experiencing, concerning the constant flow of perceptions and the appraisal of these perceptions (specific interaction events); a user experience, targeted at a period in time with a beginning and an end (focus on outcomes and memories); and co-experience, socially constructed experiences where situations are experienced together [12]. What is common for all these types of UX is that all, in some way, evoke emotions as a consequence of interacting with a system or product.

As defined by Hassenzahl, when looking at experience as related to interactive products, UX is described as an “ongoing reflection on events” where the events refer to human-product interactions [17]. He further defines UX as “a momentary, primarily evaluative feeling (good-bad) while interacting with a product or service.”[17]
Chapter 6. The effect of porting applications on user experience

So by these definitions UX clearly is a phenomenon that relies heavily on the user’s inner self. Aspects such as mood, current context (stressful environment, noisy surroundings, social and cultural contexts etc) and personal values should play a vital role on how users react to different systems.

6.1 Which factors affect UX

At the seminar, three different sources that affect UX were identified: Context, user and system. Context refers to everything surrounding the user and the system, aspects such as Internet connections or social environments play an important role. User refers to the dynamic, internal state of the user such as mood, experience from prior interactions, expectations and mental resources etc. The final factor that affects UX is the system; the properties of a particular system will influence UX where aesthetics, functionality, responsiveness and interactive behavior all play an important part in forming the users UX [12]. Furthermore, properties added by users (personal photos, personally customized interfaces) as well as brand or manufacturer images are factors that could affect the users experience.

Before the UX expert seminar made the attempt to bring some clarity to the concept of UX, attempts were made to outline important aspects leading to a better UX. Virpi Roto et al. [45] made an attempt to outline similarities between different perspectives of UX and define elements that were of great importance. It was found that common for all examined theories were the incorporation of pragmatic attributes such as functionality/utility and usability. It was noted that in order to achieve good UX, not only the interface is of interest but also the functionality. This implies, as Virpi Roto et. al puts it, “The product must be both useful and usable to result to great UX” [45]; these findings further suggests that old fashioned usability still apply in order to aid the goal of achieving good UX. It should be noted that usability is a good means of increasing the likelihood of better UX, although usability alone is not enough to engage a user. As imposed by Hassenzahl, pragmatic quality only contributes indirectly to positive experience by “making fulfillment more easy and likely” [17]. This conclusion is also confirmed from the UX seminar where it was stated that usability is not the same as UX, but indeed is found to be a typical aspect contributing to a better UX [12].

Virpi Roto et. al further found similarities among the emotional (experience related) attributes where, common for all, was the notion of identity. It describes the concept of relating to a product and how the product reflects the users self-perceived identity; except for this attribute, the different theories/perspectives varied. Virpi Roto et. al identifies the fact that different products and companies might aim at different emotional goals as a probable cause of why this is the case. As a result, it is stated that common for all considered definitions there exists a pragmatic side, dealing with utility and usability as well as an experiential side where the only common factor was found to be that of identity [45]. Although this factor does not necessarily need to be the most important one, it is highly probable that it will have an impact on UX within a majority of systems.

6.2 The UX of Speedflirt

In order to make an assumption of how porting applications will affect UX, it first must be clarified which factors that actually changes due to the switching of platform architecture. The UX of Speedflirt as a concept remains the same despite the change of platform. This means that UX coupled with aspects related to the content of the application such as the
6.2. The UX of Speedflirt

happiness of receiving a response from a flirt, the number available functions and how these appeal to the user, surprising elements supplied within the interaction of the application etc. does not change as a consequence of porting. Factors that relate to the internal state of the user such as mood and mental resources as well as factors related to the context (internet connections, work environments, stress etc) will not be affected by the action of porting; although, prior experience and expectations could affect the UX because of the user expecting a typical “Android experience” and instead encounters a more iPhone-like experience.

6.2.1 The impact of consistency on usability and user experience

When porting applications from one platform to another with the requirement of preserving the exact look and feel, the ported application will most likely violate platform specific design guidelines and standards. Users cannot rely on prior experience with similar applications since consistency within the platform framework will not apply. The concept of UX included usability as a contributing factor in achieving good UX [12] [17], therefore the issues stated above concerning the violation of platform standards should not only negatively impact usability, but also the overall UX for Speedflirt as an application.

A study conducted by Sjöberg et. al examined whether violating design guidelines and standards affected usability [16]. A program was deliberately changed to violate GUI standards and was tested in relation to programs following appropriate standards. Sjöberg et. al found that GUI standards significantly increased usability and provided the user with a sense of stability and predictable functionality. However, it was also concluded that unfamiliar GUIs were perceived as more pleasant to use indicating that the violation of standards decrease usability and at the same time increase user experience [16].

These findings were further supported by Patrick Langdon et al., which concluded that prior experience with similar products, specifically products within the same brand, was a strong predictor of usability [41]. Furthermore, they draw the conclusion that in order to facilitate learning in new products one should make use of previously learned functional features and use key visual elements associated with that functionality [41].

6.2.2 Platform specific design guidelines

As mentioned earlier, when developing applications towards a specific platform there usually exists different guidelines that the developers should follow [33] [27]. These guidelines are more or less controlled depending on the policy of the corresponding company behind the questioned platform and therefore can be more or less attained to. In the case of this thesis, dealing with the IPhone and Android platforms, the corresponding companies were Apple and Google [25] [32]. Both Apple and Google stress the fact the developers should consider using standard components to leverage users existing knowledge when interacting with applications. Within the documentation for IPhone’s interface guidelines one can find: “Consistency in the interface allows people to transfer their knowledge and skills from one application to another. A consistent application is not a slavish copy of other applications. Rather, it is an application that takes advantage of the standards and paradigms people are comfortable with.” [26]. Similarly, Android’s guidelines include the following: ”You can make it easier for users to learn how to use your application by developing a user interface that complies with Android’s standard interaction patterns, instead of creating your own or using interaction patterns from another platform.” [30]

Specifically, Android highlights three design guidelines that are of greater importance:
– Use the platform’s built-in widgets and layouts whenever possible

– Use the Options Menu as an alternative to complex touchscreen tasks

– Make sure the BACK button correctly moves the user back one logical step in the task’s back stack

The two latter guidelines refer to the (mostly) physical buttons that all android devices are required to implement. It is written within the CCD (Compatibility Definition Document): “The Home, Menu and Back functions are essential to the Android navigation paradigm. Device implementations MUST make these functions available to the user at all times, regardless of application state. These functions SHOULD be implemented via dedicated buttons.” [36]. By these requirements, the back, home and menu buttons are consistent with what Android users are familiar with and thus are prioritized within the Android guidelines.

Figure 6.1: The appearance of the menu that becomes visible when Menu button is pressed

Clearly, both platforms emphasize the benefits from using standard components and familiar interactions patterns. Despite the fact that these guidelines exist, it doesn’t mean that developers follow them. When it comes to Apple, in order to go public with an application, it first needs to be approved by a controlling instance at the publishing site. By providing this governing instance, guidelines and eventual usability following these guidelines can be maintained and the intended user experience can be ensured. This is not the case for Android where nothing controls the quality of the application or whether it follows appropriate guidelines. Due to Google’s policy and the fact that Android is open source, nothing really should control how developers create their applications or what they do with their components, even if the consequence is poor usability and bad UX.

Apple’s control of violations of design guidelines mostly prohibits the use of standard components for purposes outside the scope for which they were intended. Best practices and design guidelines still works more as a suggestion than a requirement. Nothing prevents the developer from creating his/her own components and interaction patterns. When it comes to Android, all design guidelines are suggestions and nothing forces the developers to obey the guidelines. All standard components are available due to the open source policy and the developer can use them for whatever they want or change their behavior.
Figure 6.2: Illustration of the differences between standards components. The standard behavior of the tab bar component for Android is located in the top of the UI while the tab bar on IPhone is located at the bottom.

The problem caused by the absence of consistency and standards is acknowledged by Norman [11] who criticizes the new interfaces and interaction patterns that the smartphone era has brought with it. He identifies the problem partly being “the developer community’s apparent ignorance of the long history and many findings of HCI research” [11]. The fact that developers themselves become interaction designers and software experts for their own applications does indeed impose a level of uncertainty when it comes to usability issues. Norman further concludes that both platforms should maintain the same guidelines to insure a consistency between devices: “The different operating-system developers have provided detailed interface guidelines for their products. Unfortunately, the guidelines differ from one another, [...]. But whatever the reason, proprietary standards make life more difficult for everyone.” [11]

It is unlikely that two rival companies will agree upon a united set of UI guidelines. The user interface and the overall user experience is one of the primary weapons the companies have to persuade the uses to buy their product, therefore it is more important that consistency exist within the bounds of the platform. In this way, typical Android or IPhone users will be familiar with standard components and functionality within their platform of choice.

6.2.3 How brands and social identification affect the UX

As noted earlier, when talking about UX in relation to the user, the most likely factor that will be affected by porting is that of identity. Because smartphones nowadays can be used as accessories they will not only provide the user with handy features and simplified communications, they will also provide a means of supporting the user’s perceived identity. By porting an application from one platform with the requirements of preserving the exact look, it will result in brand associated design idiom being applied on a rival platform and therefore could affect the perceived personal or social identity of a user.

The concept of identity is related to several research practices such as social identification
Figure 6.3: A selection of standard components with a predefined meaning.

and brand community research. Social identity theory states that personal identity includes specific attributes such as personal interest and values while social identity involves a social category or group, such as “Swedes” or Mac-users[5]. The process of social identification takes place whenever individuals associate themselves to a group of desirable characteristics. It is noted that the preference for, or the consumption of, a brand may serve as common factor that enables a group of individuals to perceive themselves as part of a social group (or non-members of an unwanted social group)[5]. It should be emphasized that not only the characteristics of a desired brand is of interest but also the characteristics of other individuals using that particular brand[5]. If for instance an individual perceive Android users as technically superior and professional while perceiving iPhone users as fashion enthusiasts or mainstream users, he/she might assign negative values to iPhone as a device (or Apple as a brand) and positive values to Android devices. Whether the user perceives mainstream and trendy as something negative or positive depends on his/her perceived identity and therefore will have an impact on whether he/she relates to the community of Android or iPhone users.

This conclusion is further supported by Brad D. Carlson et al., who examined the effect of brand communities on consumer behavior [6]. He identifies psychological brand communities, or a “community that exists in the mind of the individual” to have a similar effect as social identity. Both the attractiveness of the brand itself and the consumers of the brand are of great importance. Brad D. Carlson et al. further establish the conclusion that consumers that felt a sense of community with other users turned out to be highly committed to the brand. These findings support the hypothesis that brand communities have a positive effect on brand loyalty and brand attitude. A user that has bought an Android device as a result of social identity (e.g. participant in a developer community) or personal identity (identifies with the product attributes such as novel, trendy, technically superior etc.) will most likely be expecting design idioms that are associated with that particular brand or device. When design elements from a competing brand (such as iPhone) are glued upon the device, values or attributes that are associated with the competing brand might interfere with the users perceived social- or personal identity.

As noted, extensive research suggests that individuals that perceives themselves as members of a brand community will indeed affect the attitude towards a product within a par-
ticular brand (or a product within a rival brand) [6][5][9]. Is it valid to make the assumption that Android and iPhone users are members of brand communities and that the brand of a smartphone will have an effect on the overall evaluation of user interfaces? Does it in fact exist a rival culture between Android and iPhone users? Because of the extensive differences between these two companies in their approach to availability of source code for developers, the ability to control and create custom applications as well as the support for third party products, the issue of “openness”, “freedom of choice” and “mainstream versus super users” becomes potential conflict of interest between the opposing brands.

[19] states that it is predicted by social cognitive theory that users that identify with a particular group have a desire to portray the group in a positive light. It is further established that social identity theory also works the other way around, meaning that individuals who identify themselves as members in a brand-specific group have a tendency to portray competing brands in a negative light, something that is known as “oppositional brand loyalty” [19]. He uses the example of Mac and PC users where Apple users seemed to identify themselves as individualistic and creative which were in line with the Apple brand image; they viewed themselves as different, even better that PC users and differentiated from, what at that time was perceived as an ordinary, mainstream group [19]. It was found that inter-group rivalry was specifically apparent when brand communities, with a smaller market share, felt threatened by a dominant competitor and their “mainstream” users. This rivalry further resulted in a hostile view towards the competing brands [19]. In-groups members (e.g. members of a particular brand) were found to be very passionate about developing negative views about an opposing brand where their negative views had a tendency of being discussed and confirmed within group communications. These negative views are denoted as “trash talk” and the phenomenon is used to “positively differentiate themselves from rivals” [19].

6.2.4 iPhone vs Android

There is no research available on whether iPhone and Android user perceive their rival counterpart as something negative, although evidence exists about rivalry between their desktop equivalents (if you chose to perceive PC- as corresponding to android users) [19]. Will Android users be affected by iPhone components “infecting their platform” or vice versa?

In the absence of literature in the subject, an informal study on different websites, Facebook pages and groups, blog post and other sources have been made in order to “feel the pulse” on a possible ongoing “war” between iPhone and Android communities.

When it comes to question of whether or not there exists communities for these brands, one does not need to search long. Numerous fan pages and groups were found on the community site Facebook [28]. Each page or group contained ongoing discussions about good and bad aspects of respective platform (or a device targeted at the platform). Other activities found were evaluations of applications or devices informing the “community” of best and worst purchases. Even negative comments were found targeted at the opposing brand (iPhone or Android respectively) indicating that there exists a strong sense of “we against them” [29]. Furthermore, there are several communities and forums for developers as well as regular and novice users outside Facebook, all providing the ability to discuss anything related to the platform, applications or a particular device [29]. It should be noted that both Apple and Google provides a developer community to support the developers in their work of creating applications [27][33]. These developer communities will further strengthen the bonds between developers and increase the sense of belonging to a group
of "special" individuals; which in turn might increase the likelihood of rivalry between platforms.

A recent study performed by Business Insider indicated the same results as the above findings [38]. The study consisted of questionnaires measuring customers overall tendency for switching platform (targeted at, among others iPhone, Android, Windows phone 7). The study made several interesting discoveries: Android users will not likely change to iPhone, the majority of Android users will upgrade to a smartphone within the same brand and iPhone and Android appears to be over-represented by Smartphone users [38]. When non-iPhone users were faced with the question "What might make you buy an iPhone instead?" 55.7% of the respondents chose the alternative "Nothing, I hate Apple" [39]. The statistics supports the hypothesis that there in fact exists a level of rivalry between the two leading platforms in the smartphone industry.

6.3 Conclusions

The above findings indicate that components that doesn’t already have a predefined functionality or behavior can be customized to increase the sense of novelty and be more pleasant to use. By using familiar components and icons, the developers leverage the experience of the user to learn how to use a new application. It should be noted that it is important that familiar components act as expected and icons are coupled with their anticipated functionality to avoid confusion [41].

Drawing from the conclusion of extensive research as well as design guidelines from both Google and Apple, it can be established that changing the behavior of predefined components should be discouraged [41] [26] [30]. Among Android devices, due to the CCD (Compatibility Definition Document), each device is forced to implement the back and menu function, most likely in the form of physical buttons on the device. Changing the behavior of these buttons or removing expected behavior by leaving the buttons unimplemented will most likely cause confusion and decreased usability. This in turn will interrupt the “flow” of using the product and therefore also intervene in the user’s experience. Furthermore, by preserving the appearance of the application originally designed on a specific platform, a platform dependent design will be transferred and glued upon another design idiom resulting in a design-mismatch.
It is further established that the brand of a particular smartphone device, as well as brand associated GUI components, will in fact affect the appraisal of the product. The social identity theory as well as research within the field of brand community conclude that users within a community will most likely favor a particular brand in front of others. Designing the device’s interface with design idioms from a rival brand/product will therefore most likely affect users negatively. Although no scientific evidence support the hypothesis that there exists a rivalry between the smartphone platforms, extensive material on the web indicates that this is the case.
Chapter 7

The resulting application

Although the resulting Android version of Speedflirt has the same functionality as the original iPhone version of Speedflirt, it was built completely different. An Android application is based around Activities and Services, where an Activity is an abstraction of "something a user can do" and a Service is a component that "supplies functionality", the actual code of Speedflirt is divided in this way. For every interface functionality of Speedflirt, such as registering a new user, browse trough singles in the same vicinity, chat with other users and editing one’s settings, there exist a corresponding package of classes that provide that functionality. For all background functionality in Speedflirt like communication with the server, picture and memory handling, providing user notifications and animations there is a corresponding package that provides that functionality.

Mimicking the interface of the iPhone version constituted a large part of the porting of Speedflirt. The task of making an exact copy could not be completely achieved because of differences between platforms that could not be modified, such as screen resolutions. Therefore even though all the graphical elements are in the scalable 9-patch format, differences can be seen between different Android devices due to resolution and screen ratio differences.

Below are examples of the interface taken from the iPhone4 and the HTC Desire, which is the dominant device among the Swedish Android users of Speedflirt. As seen in the pictures the screen ratio differs slightly between the devices. Images have been blurred in order to preserve the integrity of the Speedflirt users within this report.
Figure 7.1: The coverflow that is used for browsing through potential flirts. As seen on the Android version one extra button is added for search filtering. This was requested as an additional feature by the client during the implementation.

(a) iPhone  
(b) Android

Figure 7.2: Browsing through the incoming flirts. The bottom menu for accepting or ignoring flirts has similar characteristics as the iPhone counterpart. The animations and overall appearance of this menu is that of the iPhone standard component for answering or denying incoming calls.

(a) iPhone  
(b) Android
Figure 7.3: This list display a user’s ongoing flirts, showing the latest message in every conversation. Clicking on a list element will display the screen shown in Figure 7.7.

(a) iPhone  
(b) Android

Figure 7.4: Clicking on the top right button will display the chat menu for ending or reporting this flirt. The animation and look mimics that of a standard iPhone menu.

(a) iPhone  
(b) Android
Figure 7.5: The first registration screen made use of two custom UI elements. Firstly the AgeSpinner described earlier. Secondly, the button for choosing gender was also a custom element consisting of two buttons connected by code, allowing only one to be active at any given time.

(a) iPhone  
(b) Android

Figure 7.6: The settings screen differs between the versions due to requirement from the client during the implementation. In order to mimic the navigation of the iPhone settings a ViewFlipper, as described in section 5.3, was implemented in the settings.

(a) iPhone  
(b) Android
Figure 7.7: The chat conversation between two users of Speedflirt. As seen in the figure the bottom tab-bar is missing on the Android version. This is due to that the chat conversation is a separate Activity and not a part of the tab-bar Activity. The chat bubbles are not exactly the same, due to the use of bitmaps as well as the scaling behavior of NinePatch graphics. The point of each bubble was located at the top unlike the iPhone copy where each point was located at the bottom. When defining the NinePatch images, the area where the point was located had to remain static in order to preserve the look of a bubble. By leaving the bottom part static and scaling the area above, the gradient of the scaled bubble became distorted, therefore the point was relocated to the top where the gradient was not affected to the same extant.
Chapter 8

Conclusions

It has been established that when porting applications between platforms, standard behavior and design guidelines should be carefully considered and followed whenever possible. Mimicking foreign design idioms not applicable with the target platform will result in longer developing times; furthermore, by considering appropriate design idioms and platform standards, the application is less likely to be rejected due to the effect of users perceived social and personal identity. By using graphical elements and behavior that are associated with a rival platform, the application is exposed to the risk of receiving negative word-of-mouth reports and bad reviews as suggested by the theory of oppositional brand loyalty.

When considering the Android platform, two key functionalities that should be implemented are the back and menu buttons. These functions are consistently present on the majority of Android devices that ships today and therefore constitutes an expected behavior that the user relies on in order to navigate and explore the application UI. Due to the fact that Android users are likely to be unfamiliar with iPhone design idioms and standards, mimicking the UI from the iPhone platform will result in a mismatch between expected behavior and system behavior. The resulting design will indeed facilitate iPhone users in the task of using the Android equivalent of their application, although considering the fact that the actual users are Android users, this might not be an appropriate approach.

When developing the Speedflirt application, expected behavior associated with the standards of iPhone navigation were implemented. This resulted in the physical back button being unimplemented and replaced by a graphical back button in the top-left corner, which represents a standard behavior of the iPhone navigation controller. The Android application also relied on the standard navigation bar used within several iPhone applications; transitions between views were based on the behavior of the navigation controller instead of the activity paradigm that is standard behavior on Android. Furthermore, a graphical replica of the iPhone tab-bar was implemented which is highly associated with the iPhone platform. Other graphical components also differs, e.g. buttons implementing the standard, glossy gradient look associated with the iPhone design idiom instead of a more faint gradient, which is more associated with standard Android components.

The project was completed successfully within the given timeframe. However, the subsequent literature study indicated that the approach taken, where an exact copy of the interface was implemented, was perhaps not the ideal approach of porting with respect of preserving the usability and user experience across platforms. Furthermore, the task of mimicking the iPhone interface was one of the most time consuming tasks during the project which further supports the conclusion that copying design idioms not applicable to
the target platform should be avoided.

As cross-platform support is becoming more common for mobile applications, frameworks such as PhoneGap and Titanium, which are aimed at this are becoming stable alternatives to native development. Developers should consider that the design idiom of choice should be unique and not heavily associated with a specific smartphone platform if the usability and user experience is to be maintained across all platforms.

Drawn from the conclusions from this thesis, the following guidelines highlights important aspects to consider when porting applications between platform:

– Do not transfer platform/brand specific design idioms onto rival platforms/brands.

– Carefully consider platform specific consistencies and leverage prior experience of the users. For Android applications, always implement the standard physical buttons to their appropriate behavior, specifically the back and menu button.

– When developing cross-platform with web application techniques, consider using designs that do not have a strong associations to a particular brand or platform.

8.1 Future work

In this thesis an informal study was performed examining the level of rivalry between iPhone and Android users. The study, as well as the material used within the study was not of scientific validity. Future research within this field is required in order to confirm the hypothesis that it in fact exist a sense of rivalry between iPhone and Android users. Furthermore, future research is needed to examine the effect brand associated design elements have on an equivalent product with a competitive brand, as well as its affect on usability and user experience.

8.2 Acknowledgements

We thank our supervisor at The Mobile Life, Christian Svaetichin Paulsen, for guiding us through the project. We would also like to thank Håkan Gulliksson for support and help with our report writing. To all involved with the project and our colleagues at The Mobile Life for inspiration and contributing to an enjoyable workplace, Thank You!
References


51


REFERENCES


[39] Business Insider. The truth about smartphones: Our exclusive survey on iphone vs android, 2011. [online], http://www.businessinsider.com/smartphone-survey-results-2011-4 among android users most say they will never buy an iphone because they hate apple most of the rest say they would buy an iphone if it worked better with non apple products and there is the risk of the closed system that apple is selling 7 accessed 2011-04-23.


Appendix A

Loading images

Figure A.1: Flow chart for the process of loading an image into the gallery component