Masterarbeit

Evaluierung von Cross-Plattform Entwicklungsmöglichkeiten für Mobile Applikationen

Krasimir Hristov

hristov@student.tugraz.at

Institut für Softwaretechnologie (IST)
Technische Universität Graz
Inffeldgasse 16B/II,
8010 Graz, Österreich

Begutachter und Betreuer: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, Mai 2013

Master's Thesis

Evaluation of Cross-Platform Development Approaches for Mobile Applications

Krasimir Hristov

hristov@student.tugraz.at

Institute for Software Technology (IST)
Graz University of Technology
Inffeldgasse 16B/II,
8010 Graz, Austria

Assessor and supervisor: Univ.-Prof. Dipl.-Ing. Dr. techn. Wolfgang Slany

Graz, May 2013

Abstract

The availability of mobile devices like smartphones and tablets is growing almost exponentially. Users can choose from a large variety of devices with distinct operating systems. Generally the market is mainly dominated by the Android and iOS platforms. A large number of downloads of an application can only be achieved by being available on as many operating systems as possible. The required development process to accomplish this is cost and time intensive. One ideal solution for this is to write code once that runs on all systems. In the last couple of years, multiple cross platform development frameworks became available that either try to accomplish this or try to fulfill it as much as possible. Two of these techniques and their possibilities are presented in this thesis and compared to the Android native development process, where the fundamental usage of the Android Software Development Kit (SDK) and its system structure are explained. Moreover, the time consuming development for a wide range of Android devices is evaluated and possible solutions are presented. Applications developed natively in the appropriate programming language take full advantage of the platform and the supported SDK. Since users are attached to their devices and expect a pleasing experience, this can only be fulfilled with full access to the native platform.

However, one possibility to create an application is to wrap a web application into it. Another possibility brings cross platform development of mobile applications to a new level through cross compiling source code to the respective platforms.

Hybrid applications developed with the presented Cordova framework provide a great opportunity to extend a web application with native features. These are however limited only to the supported features by Cordova. The extension of them and the associated complexity with it is examined and evaluated. The user interface of such an application is created using web technologies such as HyperText Markup Language Version 5 (HTML5), Cascading Style Sheets Level 3 (CSS3) and JavaScript. In addition two popular web development frameworks used in combination with Cordova are presented. Through analyzing and comparing them, their advantages and disadvantages are stated.

Another presented framework, Xamarin, provides the possibility of writing native applications for various platforms in the programming language C#. This cross compilation solution brings native experience with almost no downfalls. Through the clever usage of modern design patterns, a high percentage (60% - 80%) of code sharing can be accomplished. As a result, applications with their full native experience for Android, iOS and Windows Phone can be realized. Since the SDK with all its graphical elements of each platform can be used, the user does not notice the difference between an application produced with this technique

and a native application. An overview is provided on how this can be done, and a detailed look behind the support of Android particularly is given.

Moreover, testing possibilities and frameworks are evaluated through all three development techniques with respect to the agile development principles.

Keywords: Cross-Platform, Hybrid, Cross-Compilation, iPhone, iOS, Android, Mono, Cordova, PhoneGap, JavaScript, HTML5, CSS3, Test, Test Driven Development (TDD), Behavior-driven development (BDD), C# , Objective-C, Java

Kurzfassung

Die Verbreitung mobiler Endgeräte, wie beispielsweise Smartphones oder Tablets, unterliegt einem fast schon exponentiellen Wachstum. Benutzer können dabei aus einer Reihe von Geräten mit unterschiedlichen Betriebssystemen wählen. Der Markt wird jedoch von den zwei Plattformen Android und iOS dominiert. Eine weitreichende Distribution einer App kann nur erreicht werden, wenn diese für möglichst viele Betriebssysteme erhältlich ist, wobei der hierfür nötigte Entwicklungsprozess kostspielig und zeitaufwändig ist. Idealerweise sollte Code nur einmal geschrieben werden und auf allen Plattformen funktionieren. Daher sind in den letzten Jahren mehrere plattform-übergreifende Entwicklungswerkzeuge entstanden, welche versuchen genau dies zu ermöglichen. Zwei dieser Werkzeuge werden vorgestellt und mit dem traditionellen Android Entwicklungsprozess verglichen, wobei die Verwendung des Android Software Development Kits (SDK) sowie die Systemstruktur genauer erläutert wird. Weiteres wird die Problematik, die durch der Vielfalt an Android-Geräten entsteht, diskutiert, sowie mögliche Lösungen bei der Entwicklung. Native Applikationen nutzen die gesamten Vorzüge des Betriebssystemes und dem angeboten SDK aus, welche die Erwartungen der Benutzer in Bezug auf das App-Erlebnis erfüllen kann.

Eine Möglichkeit die native Entwicklung zu umgehen ist es eine Webapplikation in die native App einzubinden. Eine zweite Technik bringt die mobile Cross Platform Entwicklung auf einen neuen Level, indem sie die plattformübergreifende Kompilierung von Code ermöglicht.

Hybridapplikationen, die mit der Hilfe von Cordova erstellt wurden, erlauben es Webapplikationen in Native Programme umzuwandeln und erweitern diese mit nativen Funktionalitäten. Die Erweiterungsmöglichkeiten von solchen Funktionen, sowie die bei der Entwicklung entstehende Komplexität, wurde in dieser Arbeit betrachtet und evaluiert. Die Benutzeroberfläche wird bei solchen Applikationen mittels Webtechnologien wie Hyper Text Markup Language Version 5 (HTML5), Cascading Style Sheets Level 3 (CSS3) und JavaScript realisiert. Zwei weitverbreitete Web Entwicklungsframeworks, welche in Kombination mit Cordova oft zum Einsatz kommen, sind auf deren Vor- und Nachteile untersucht und präsentiert worden.

Xamarin bietet dem Entwickler die Möglichkeit mittels der Programmiersprache C# Applikationen, die den nativen gleichkommen, für mehrere mobilen Betriebssystemen zu erzeugen. Diese Cross Plattform Kompilierung ermöglicht es ein natives Erlebnis für den Benutzer zu erzeugen, während es kaum Nachteile mit sich bringt. Durch die Verwendung von Design Patterns kann zwischen 60% und 80% des Codes plattform-übergreifend verwendet werden. Durch den Einsatz von den SDKs vom jeweiligen Betriebssystemen, können alle grafischen Elemente der jeweiligen Plattformen verwendet werden, wodurch für die Benutzer kein Unterschied zu nativen Apps mehr erkennbar ist. Innerhalb der Arbeit wurde darauf eingegangen

wie eine App mittels Xamarin entwickelt werden kann und wie diese technisch in Bezug auf Android tatsächlich funktioniert.

Weiteres werden unter Betrachtung auf alle drei Entwicklungsmethoden mögliche Test-Frameworks unter Berücksichtigung agiler Softwareentwicklungsmethoden evaluiert.

Schlüsselwörter: Cross-Plattform, Hybrid, Cross-Kompilierung, iPhone, iOS, Android, Mono, Cordova, PhoneGap, JavaScript, HTML5, CSS3, Test Driven Development (TDD), Behavior-driven development (BDD), C# , Objective-C, Java

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort	Datum	Unterschrift

Acknowledgements

First, I want to thank my girlfriend Roxane who not only supported me through the whole writing process but also encouraged me through my entire journey as a student.

In addition, I want to thank Martin Mrvka who was my advisor at Up to Eleven Digital Solutions GmbH for his support and help during the working process. I would also like to thank my advisor Wolfgang Slany for his great input and support through the last stage of my studies.

I also want to show appreciation for my beloved family which always supported and backed me up wherever they could.

Moreover, I would like to thank family Koitz for their help and courtesy.

Last but not least, thanks to all my friends.

Graz, May 2013 Krasimir Hristov

Contents

List of Tables

List of Figures

1 Introduction

The smartphone and tablet market has been growing almost exponentially over the last couple of years. The hardware is becoming more and more powerful and main functionalities such as making and taking phone calls are ranked behind other aspects of the devices. They are used more often than regular computers or laptops. Usage possibilities are continuously expanding.

In the meantime, there is a big variety of devices with different operation systems users can choose from. The market in 2012 was dominated by Android (68.3%) and iOS (18.8%), both of them combined making out over 87 % of the devices sold. The rest of the market is shared between BlackBerry OS (4.7%), Windows Phone (2.6%), Linux (mostly made up of Bada and MeeGoo) (2.0%) and Others (3.6%) [Int13b].

Operation systems like Symbian, RIM and Windows Mobile dominated the market a couple of years ago, but as you can see in Figure 1.1 have been outgrown.

Windows Phone gained a few percent points since it was introduced in 2010, but it is still struggling, and hoping to gain more popularity through the introduced Windows Phone 8.

As of this writing, the tablet is dominated by Android (41.5%) and iOS (51%).

Pretty much every platform uses different programming languages and has it's own API that allows it to interact with the hardware of the device and access elements of the system. Therefore, developing an application that instantly works on every system is not possible. [Int13a]

In particular this means that to sell a high number of one's app, it must be available at least on the Android and the iOS platforms. The development process of applications for one mobile platform is cost and time intensive. This means that a lot of resources need to be invested in this process. However, these resources are not always available. One ideal solution for this is to write code once that runs on all different systems.

In the last couple of years, multiple cross platform development approaches have evolved that either try to accomplish this or try to fulfill it as much as possible. Two of these techniques and their possibilities are represented in comparison with the Android native development process within this thesis.

In Section 2 the fundamentals of the Android OS and its application development process are described.

This thesis focuses on the Android point of view and will cover only basic information about the iOS development process in Section 3. Other operating systems will not be covered, since their market shares currently are fairly small.

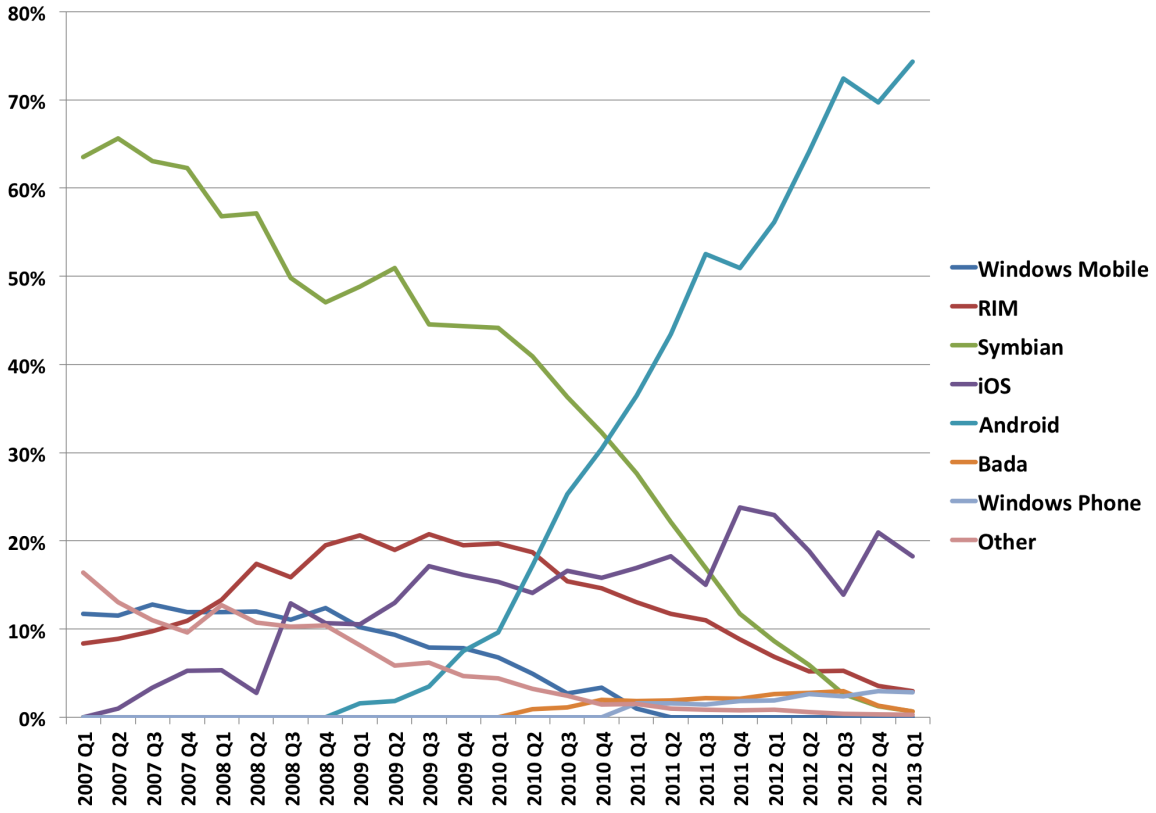

Figure 1.1: World Wide Smartphone Market Shares [Gar13]

Section 4 is covering two unique cross platform development approaches. The first one, Cordova, wraps a web application based on HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript into a native application. It provides further communication facilities between the web and native parts. This allows access to the native features that are not available in pure web applications. The second technique, Xamarin, provides the possibility of writing native applications for various platforms in the C# language. This is accomplished through cross compilation against the other platform specific programming languages. It provides among other features the full access to the platform specific APIs. Using this approach a native application can be developed for Android, iOS and Windows Phone. A general overview is provided on how this can be performed and a detailed look behind the support of specifically Android.

Section 5 describes testing approaches and used frameworks through all introduced developing techniques.

A conclusion is given in Section 6 where all findings are discussed.

2 Android Development

The Android Operating System (OS) is an open source platform designed specially for mobile devices. Android, Inc. was founded in October 2003 by Andy Rubin and Rich Miner, and was purchased by Google later in 2005 [Yag13].

The operation system was announced to the world by the Open Handset Alliance (OHA) in November 2007. OHA is a consortium of companies, including Google and others, with the mission to develop open standards for mobile devices. The first Android SDK (1.0) and Android Phone HTC G1 were released in 2008. [Yag13]

2.1 Architecture

The Android OS is built of various components. Figure 2.1 shows the different operation system layers. The Android system architecture is based on five main sections, explained below.

2.1.1 Linux Kernel

Android is based on the Linux Kernel version 2.6 that is available at `http://android.git.kernel.org/`. It contains all the necessary hardware drivers and is managing power, memory, process, network and security tasks.

2.1.2 Libraries

This component acts like a translation layer between the Application Framework and the Linux Kernel. It contains a set of libraries that are written in C/C++ but can be accessed through a Java API from the Application Framework. Some of the main core functionalities are located here. For example, it holds the SQLite library, a lightweight database engine, or LibWebCore library, a web browser engine.

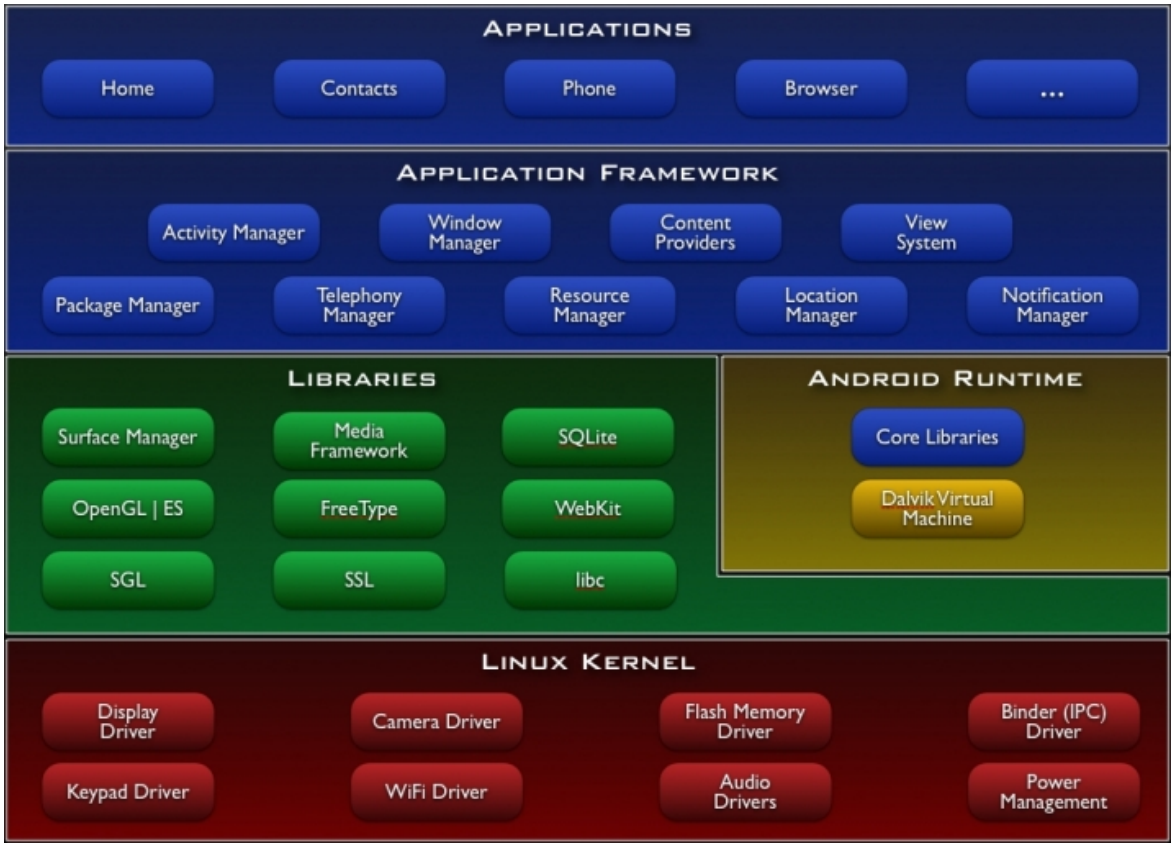

Figure 2.1: Android Architecture Layers [And13n]

2.1.3 Android Runtime

This section lies in the same layer as the Libraries and contains the two parts developers are constantly working with. Core Libraries provide the functionality of the Java libraries and therefore enable developers to write their apps in Java. The Dalvik Virtual Machine (VM) is specially designed to run code efficiently on mobile devices that have limited resources like memory and CPU. Moreover, every application runs in its own process and has its own instance of the Dalvik VM. Java compiled classes are being transformed into the dex format by the dx tool that is included in the Android SDK. This process reduces the size of the file significantly.

2.1.4 Application Framework

This section consists of different Android services and system components that are useful to create an application. They can be accessed by the Android API. For example, it contains the Activity, Window, Resource Manager just to name a few of them.

2.1.5 Applications

This is the app, the user interacts with at the end. It can also be a pre-built one, like the contacts manager, the phone application, or a downloaded one from the Google Play Store. [Gun12], [And13n], [Lee12]

2.2 Versions Overview

Since the first version of Android was released in 2008, the OS has evolved as quickly as the smartphone. The latest Android version is 4.2 (Jelly Bean), at the current time of this writing. Since Android is open, it gives everybody the possibility to customize it. All the major phone manufacturers take advantage of this possibility. Therefore, all of them are shipping their phones with a customized Android build that differs from the point of view of the user by having a different UI. This is on the one hand a big advantage for the manufacturers and users, since phone models can be personalized. Special components and design effects can be included that provide a unique experience compared to other Android devices. But on the other hand this is probably Android's biggest disadvantage. When companies put custom builds on their devices, they are also responsible for maintaining them.

That means every time the Android OS is being updated they need to implement their customized version using the new released Android Version and provide this to the end user. Some of the device manufacturers are known for good support, at least for some of their devices and some of them are know for bad support. Since this is a time-consuming process, some of them do not even bother providing new updates.

Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	1.9%
2.2	Froyo	8	7.5%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	43.9%
3.1	Honeycomb	12	0.3%
3.2		13	0.9%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	28.6%
4.1	Jelly Bean	16	14.9%
4.2		17	1.6%

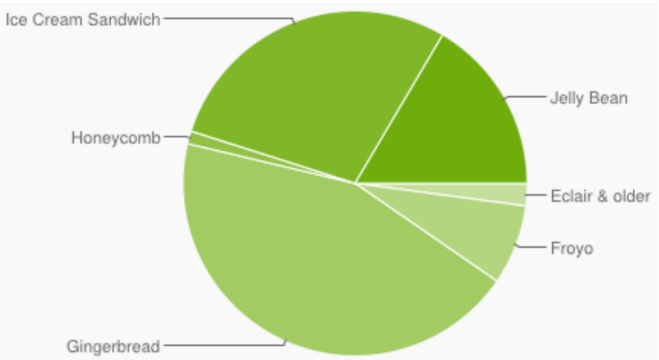

Figure 2.2: Overview Android Versions and Distribution [And13q]

The result in the end is, that even though new Android Versions are released the end user is either getting them with a big delay or never receiving them at all. This debacle results in having a lot of different Android versions "out" there. In Figure 2.2 the current Android versions distribution is depicted. Gingerbread is the most widely spread one, although its original release date was late 2010/ early 2011.

While developing applications for Android, one always needs to be aware of the wide range of different versions out there. For more details about compatibility and the best way to support different Android versions, please have a look at Section 2.7.1 [And13q].

2.3 Fundamentals

In order to start developing for Android, the latest Java version needs to be installed. The latest Android SDK can be downloaded from the Android's developer website[1]. The easiest way would be to get the Eclipse Bundle that contains the Eclipse IDE, ADT Eclipse Plugin, Android SDK Tools, Android Platform Tools, the latest Android Platform and an image of it for the emulator.

However, it is not necessary to use the Eclipse IDE. In fact, every IDE can be used since all tools can be executed through the Command Line. But still, the usage of IDE that has an Android integration should be pursued, since it makes the workflow much easier. Another popular IDE used by the Android community is IntelliJ IDEA[2].

[1] http://developer.android.com/sdk/index.html
[2] http://www.jetbrains.com/idea

2.3.1 Android Developing Tools

The Android SDK includes, besides the OS platform, several tools that can help during the development process.

Android SDK and Virtual Device Manager The Android SDK Manager gives the support to manage and download different SDK versions and packages. The Android Virtual Device (AVD) manager provides the ability to create virtual devices with different specifications and manages them.

Android Emulator This is an implementation of the Android VM that can be run in an Android Virtual Device. With it one can debug and test applications in the emulator without the need of an actual Android device.

Dalvik Debug Monitoring Service (DDMS) This provides several options to monitor and control a device during debugging. Thread, heap, and Logcat information are a few of them.

Android Debug Bridge (ADB) This is a client-server command line tool that provides the functionality to communicate with a connected Android device or an Android emulator instance.

Logcat This is a mechanism for viewing Android debugging outputs.

Android Asset Packaging Tool (AAPT) This creates the Android APK package files.

SQLite3 This is a simple tool for accessing the SQLite databases.

ProGuard This tool helps to shrink the size of the APK file and obfuscates the source code, making reverse engineering much harder.

Others These are a few of the most used tools. In addition, more information about all tools included is provided at the official Android documentation[3].

[Mei12], [And13o], [MDMB12]

[3] http://developer.android.com/guide/developing/tools/index.html

2.3.2 AndroidManifest.xml

Every Android application needs to have an `AndroidManifest.xml` file in the root of the application's directory. This manifest includes essential information about the application's components, requirements, and structure. Pretty much everything important needs to be defined here, such as Broadcast Receivers, Activities, Services and Content Providers. Among others it specifies metadata such as the title, version number, used theme, and the icon of the app. The used SDK and the supported SDK version can be defined here by setting the attributes `android:minSdkVersion` and `android:targetSdkVersion`. This is strongly recommended, since there are many devices running different Android versions. This said, there is a big variety of display sizes that also need to be handled in a proper way. Therefore, display sizes and limits can be specified here. The application might become unusable if the Android OS puts the app in compatibility mode and scales it. Generally, all permissions the application requires to access protected parts of Android need to be specified here. Each one of these permissions will be displayed to the user before the installation of the app. For the most part, these are all imported value nodes that are usually used in Android applications. More information and detailed insight into the structure can be found in the Android documentation[4].

[Mei12], [And13p], [MDMB12]

2.3.3 Files Structure

An Android project created by the Eclipse IDE will posses the file structure as described below.

Android This holds the `android.jar` library that contains all the necessary Android classes that are needed for the development process. In Eclipse, this is presented as a library folder and it is named in respect to the Android target version. Google API files are also listed in here, if they are included in the project, like the `maps.jar` file that provides functionalities of Google Maps.

Android Dependencies All other included libraries are listed in here. Targeting a lower `minSdkVersion` will also eventually list the `android-support-v4.jar` that holds the Android Support Library, as described in Section 2.7.1.

src This folder holds all the `.java` source files in their respective package folders.

[4] http://developer.android.com/guide/topics/manifest/manifest-intro.html

gen (Generated Java Files) Here, the `R.java` file is located, which is generated by the compiler and is referencing the projects resources. Getting compilation errors concerning this file should be resolved by editing the source files or resource files. It should not be modified or imported into source files.

assets All assets of the project like HTML5 files, databases, pictures, etc. are stored in here.

bin In this folder, all build files generated by the ADT can be found and particularly the produced `.apk` file.

res This folder contains all resource files as different drawable images for the support of multiple screen sizes and other Android XML resource files.

AndroidManifest.xml This is the manifest file as discussed earlier in Section 2.3.2.

2.4 Components

There are four main components in an Android application and each one of these exists as its own entity. Some of these components can be entry points of the application. Each one has a specific purpose and a distinct lifecycle. These four application blocks are:

- Activity
- Service
- Content Provider
- Broadcast Receiver

2.4.1 Activity

An Activity fills the whole screen of the mobile device with some kind of interface and also responds to interaction with the user interface. When starting an application the main activity of it is being opened. Activities are strictly separated from each other and are interchangeable parts of the Android user interface across different applications. The navigation between Activities is being handled by Intents. Activities never communicate with each other directly. Passing information and getting results back are all done by Intents; they are described in detail in Section 2.5.

```
1  import android.os.Bundle;
2  import android.app.Activity;
3
4  public class MainActivity extends Activity {
5
6    @Override
7    protected void onCreate(Bundle savedInstanceState) {
8      super.onCreate(savedInstanceState);
9      setContentView(R.layout.activity_main);
10   }
11
12 }
```

Listing 2.1: Simple Activity

Since the Activity class is probably one of the most important ones in Android, having a brief look at an example is helpful.

Listing 2.1 shows how a simple Activity is being created, and a View that is defined in the `activity_main.xml` file, is being setup. A created Activity class is always a subclass of Activity and has a specific lifecycle.

Activity Lifecycle The lifecycle describes all events an Activity goes through from being created until it ends. There are seven methods implemented in the Activity class, that support managing it. In Figure 2.3 a diagram shows the detailed lifecycle and when which method is called. By implementing methods in the Activity that override these methods, one can provide a full customized behavior for all scenarios. [RLMM09], [And13c], [Lee12]

- `protected void onCreate(Bundle savedInstanceState)` After an instance of an Activity is created, this is the first method that is being called. The passed argument is a bundle of saved data from a previous life of the Activity, if there was one otherwise it is just `null`. However, this is the place where all the general initialization of the application happens. The necessary Views can be created and if needed populated with data.
- `protected void onStart()` This follows `onCreate` and is being called as the views become visible.
- `protected void onRestart()` If the activity has stopped and is being opened again, this method will be called. It is followed by the `onStart` method.
- `protected void onResume()` This is called after the `onStart` method if the user is still interacting with the application.
- `protected void onPause()` This method is called, when some other Activity becomes active and the current one is not visible anymore. Animation and other memory

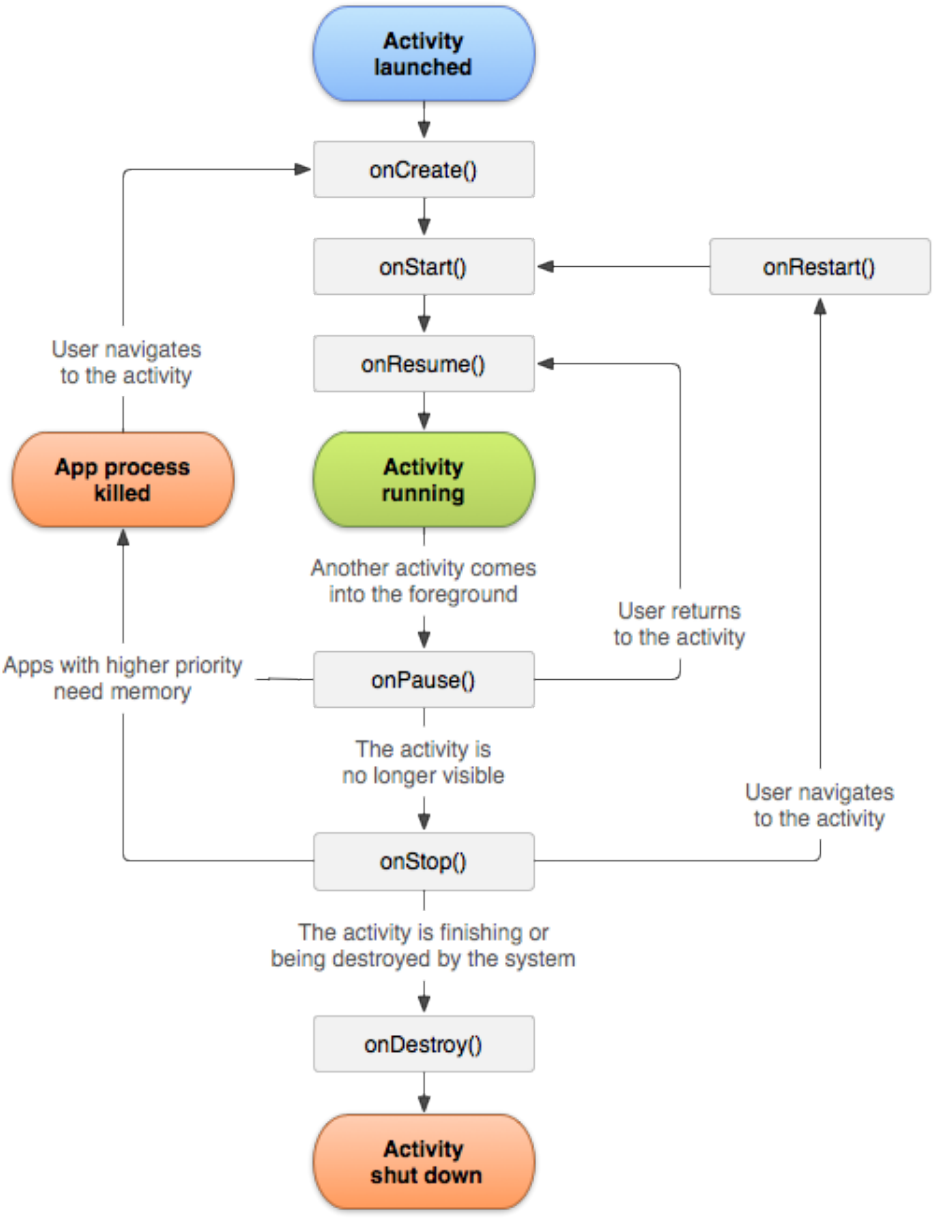

Figure 2.3: Android Activity Lifecycle [And13c]

consuming operations should be paused, and data that will be needed for resuming the Activity should be saved.

- `protected void onStop()` This gets called when the Activity shuts down or some other Activity needs the memory and this one has to be destroyed.
- `protected void onDestroy()` This is the last method in the lifecycle and is called right before the instance of the Activity is going to be destroyed.

It is important to understand the lifecycle and be aware that the application is running on a smartphone. That means the instance might be destroyed because the battery died or a phone call interrupted it. The state of an application can be saved and restored by using the `Bundle` instance. Listing 2.2 shows a simple usage of saving and retrieving data using a `Bundle`.

```
1 outState.putString("data", someDataString);
2 String data = savedInstanceState.getString("data");
```

Listing 2.2: Save and Retrieve from Bundle

Best practice to do so is to use the two methods that are also part of the Activity class and can be overridden:

- `protected void onSaveInstanceState (Bundle outState)` This is called before the app is put to the background. Moreover, if a lot of different data needs to be saved, it might be more convenient to do it in one place instead of spreading it over the Activity code.
- `protected void onRestoreInstanceState (Bundle savedInstanceState)` Is called after `onStart`. Restoring saved data from the bundle can be done here in the `onCreate` method.

Try to hold as little data as possible in the memory, and instead use other possibilities like databases.

[RLMM09], [And13c], [Lee12], [ST10]

2.4.2 Service

This Android component performs, like the name says, some kind of service in the background and does not have a user interface. This could be listening to music, while some other application is in the foreground; or uploading files in the background; or the function is just waiting to get a HTTP notification from a server to perform a task. Pretty much every task that needs to be performed in the background. The application that is triggering it can be either active or not at this moment. However, the Android Service class also has methods implemented that can be overridden to manage the Service's lifecycle that is slightly different from the one from an Activity.

[MDMB12], [And13c], [ST10]

2.4.3 Content Provider

Since Android applications are capsuled from each other, they cannot exchange data directly with each other. The Content Provider component provides a way to exchange data across packages in a standardized modern way. It behaves almost like a database. An application can either access a Content Provider or even integrate a Content Provider interface itself. For example, Android ships already with a few of them, including the following:

- Browser: Stores all the browser related data like history, bookmarks, and so on.
- Calendar: Stores all calendars, events, start and end time, attendees and reminders.
- Calllog: This stores the call details, received calls and so on.
- Contacts: This has all the user's contacts.
- MediaStore: Responsible for all the multimedia files.
- Settings: Stores device settings.

To get information from a Content Provider, a query URI is required. This URI looks generally like this `<standard_prefix>://<authority>/<data_path>/<id>`.

The Content Provider standard prefix is `content`. The authority is basically the identification of the Content Provider. Accessing the built-in Contacts Provider would be `contacts`. Applications that provide a Content Provider will have an individual name, similar to `com.my.apps.content.provider`. The data path specifies the type of data and the id of the particular record. As usually all the built-in Android related queries are saved as constants.

Using Content Provider Content Provider can be accessed with a `ContentResolver` object that provides four methods: `query()`, `insert()`, `update()` and `delete()`. These are used to read, insert, update, and delete data.

```
1 Cursor cursor = getContentResolver().query(
2     ContactsContract.Contacts.CONTENT_URI,  // URI
3     null,                                   // projection
4     null,                                   // selectionClause
5     null,                                   // selectionArgs
6     null);                                  // sortOrder
7 }
```

Listing 2.3: Content Provider Retrieving Data

Listing 2.3 illustrates an example of how data can be fetched. The `getContentResolver` method returns an `android.database.Cursor` interface of a Cursor object. However let us see what is happening next. The `query` function reads some kind of content. The URI is the built-in constant for contacts. Under `projection` the columns can be defined that should

be returned for each row. `selectionClause` and `selectionArgs` are used to define some selection criteria and `sortOrder` defines clearly the sort order of returned rows. Data can be retrieved from the `Cursor` as it is a database Cursor.

Create Own Content Provider Besides using already an existing Content Provider, a custom one can be created. One obvious reason for that is giving other applications the ability to access the data of the developed application. However, it is smart for organizational and simplicity reasons to separate the data storage from the rest of the code by introducing a custom Content Provider.

However, it is wise

Here is an overview of the required steps:

1. Creating a database class by extending the `SQLiteOpenHelper` class.
2. Specifying the database schema and creating the database by overriding the `public void onCreate(SQLiteDatabase db)`.
3. Defining what should happen when the database is upgraded by overriding the `onUpgrade` method.
4. Create a custom Content Provider by extending the `ContentProvider` class.
5. Define the used `CONTENT_URI`.
6. Create the column names of the Content Provider.
7. Declare the column specification strings and define the MIME type.
8. Handle the Content Provider queries by implementing the four functions `query()`, `insert()`, `update()` and `delete()` by overriding them.
9. Declare the `<provider>` in the Android manifest.

After a Content Provider is set up and ready to go, one should think about what type of permissions for it should be enabled in the manifest. If access should be granted to third-party apps, it is required to mark it as `android:exported=true`. Otherwise set it to `android:exported=false` to allow the access just for the developed application. If it is going to be exported for the use of third-party applications, rethink which permissions they will need. A single permission can be set for only reading or writing and one for both reading and writing.

[And13j]

So why should a Content Provider be created for the application? It creates an abstraction of the data model and hides the detailed implementation of the stored data. It helps to separate the code from the actual model by providing an interface to the model. Operations, as opening a connection to a database are done in the `ContentResolver` and do not need to be triggered every time by the application itself. Moreover, certain operations are more easily done using a Content Provider instead of a pure SQLite database or something else.

Broadcast Intent	Description
`ACTION_BOOT_COMPLETED`	This Broadcast can be received after having the `RECEIVE_BOOT_COMPLETED` permission and indicates that the device's boot process has completed.
`ACTION_BATTERY_LOW`	This Broadcast describes that the user just received a warning that the device battery is low. All unnecessary services should be shut down.
`ACTION_TIMEZONE_CHANGED`	Applications that perform any kind of operation that rely on date and time should mandatorily listen to this one. It signifies that the system's timezone has changed.

Table 2.1: Important Broadcast Intents

For example, imagine having a `Listview` that shows some type of data in a list. When data is being added or deleted, the list needs to refresh and show these changes. Using just a pure SQLite database the required queries have to be created every time. By using a Content Provider, the already implemented notification mechanisms can be used.

[MDMB12], [And13e], [Lee12], [Gro13]

2.4.4 Broadcast Receiver

The Broadcast Receiver component is responsible for the system-wide communication. An application or the system itself can send a broadcast announcement about an event. Therefore receiving certain broadcasts require having a proper permission. Two main types of Broadcast Intents are supported by Android. Normal Broadcasts are delivered asynchronously to all registered receivers and disappear afterwards. Ordered Broadcasts however are delivered in a specific order to the receivers. Since the chain of delivery has a certain priority, a receiver could either pass the received broadcast to the next one, or abort it. The operating system is matching a broadcast with Intent Filters and is calling a suitable application. A Broadcast Receiver does not have a user interface and is just listening in the background. So why is this necessary? The first motivation behind the integration of Broadcast Receivers, is being able to respond to broadcasts from the Android system itself. There are a few important system events listed in Table 2.1, that are sometimes worthy to respond to. All system broadcasts are defined in the `Intent` class.

Starting a Service after the phone has booted is an example for using them. Besides just getting notified by system events, one can also broadcast events. This can be used to inform every application about some specific event that occurred and is worthy to broadcast. On the one hand, if the own app wants to let every other application know about some specific event, it is a handy thing, but on the other hand, if it is used to communicate with other parts of the application, this can be not secure. Every application on the device will receive the broadcast message. This can be limited, one way of doing this is to create unique `Intent`

data and specify the broadcast as much as possible. Moreover, a package limitation can be specified, that was introduced in the Android API version 14.

Example of Broadcast Receiver In Listing 2.4 a simple implementation is displayed. The `HelloWorldBroadcastReceiver` extends the `BroadcastReceiver` class and implements the `onReceiver` method. This is the one that is called after receiving a certain broadcast announcement. Long-running operations cannot be performed in it, since it has a ten seconds timeout, after which it will be killed.

```
1 class HelloWorldBroadcastReceiver extends BroadcastReceiver {
2   @Override
3   public void onReceive(Context context, Intent intent) {
4     //do something, call a service for example
5   }
6 }
```

Listing 2.4: Example Broadcast Receiver

The receiver can be registered dynamically in an `Activity` by calling `registerReceiver()` and stop listening for certain broadcasts by calling `unregisterReceiver()`.

[CD12], [And13d], [And13g], [CFGW11]

2.5 Intent and Intent Filter

Intents are the glue between Activities. Generally, they are messages that are asynchronously sent between Activities in order to communicate with each other. An Intent consists primary of `action`, `data` and `extra`. It defines what action should be performed with the data. Additional information can be put into extra.

```
1 Intent intent = new Intent(Intent.ACTION_VIEW);
2 intent.setData(Uri.parse("http://www.google.com"));
3 intent.putExtra("extra_data","justAString");
4 startActivity(intent);
```

Listing 2.5: Intent Example

In Listing 2.5 it is illustrated how to create an Intent, put some data and extra in it and start it. In this example a standard action is used. `Intent.ACTION_VIEW` will display the content of the data depending on what it is. Here, a web browser will open up and call the URL. If a custom Activity that was created should be triggered, this can be performed by using Intent Filters.

```
1 Uri url = getIntent().getData();
2 String some_extra = getIntent().getExtras().getString("extra_data");
```

Listing 2.6: Intent Example 2

It is illustrated in Listing 2.6 how this Activity gets the information that was put into the Intent.

```
1 <activity android:name=".SomeActivity" >
2   <intent-filter>
3     <action android:name="android.intent.action.VIEW" />
4     <data android:scheme="http" />
5   </intent-filter>
6 </activity>
```

Listing 2.7: Intent Filter

Intent Filters are used to state which operations a component, for example Activity, can handle and is willing to receive. Components can have multiple Intent Filters. They are defined in the Android manifest. An example for using an Intent Filter is shown in Listing 2.7.

As presented the Activity `SomeActivity` is going to receive action view Intents that have a http scheme. When multiple Activities are responsible for the same action, the user needs to choose one and can save it as default for the next time. In this case, the user needs to decide between our Activity and the default browser, under the assumption there are no other components installed that are handling this type of Intent.

[Lee12], [And13g], [DK]

2.6 User Interfaces

The Android UI has evolved a lot over the last couple of years. In this process design patterns have appeared, some of them were successful and others not. In contrast to the iOS operating system, Android does not have any mandatory design guidelines that need to be fulfilled in order to be able to publish an application. However, since early 2011, Android released a design initiative called "Android Design"[5]. Recommended user interface principles are presented there. This is more or less a guideline with recommendations from the platform designers. It is supposed to be a baseline for developers to help them but is not required at all. They still can be broken, and new innovative designs can be tried out. But still, since they were announced, the quality of the Android apps has definitely improved. The Android design fundamentals describe the user interface, which is from the user's point of view the most important part.

[5] http://developer.android.com/design/index.html

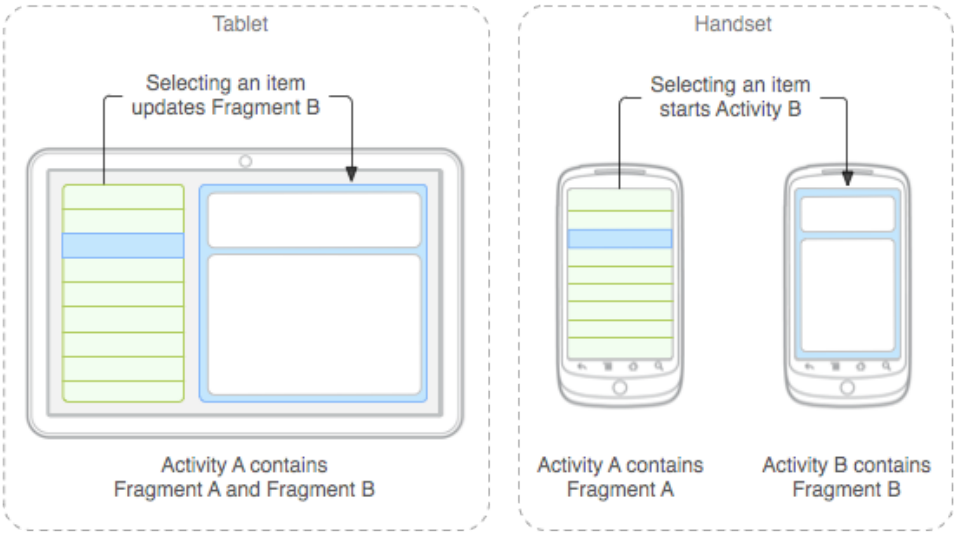

Figure 2.4: Responsive Design by Using Fragments [And13f]

Views The `View` class is the basic building element for all interface elements.

View Groups This is an invisible container that holds other "children" Views or View groups and is the base class for Layouts.

Fragments They were introduced in Android 3.0 and can help to create a flexible UI design. Fragments represent a certain UI segment and can be combined and reused. They are embedded into one Activity and either use their layout or have one for themselves. Moreover, Fragments also have their own lifecycle that looks like the one of Activities.

They are used to create a more responsive design in respect to the many available screen sizes. For example on a tablet there is more space and certain Fragments that would be hidden on a smartphone, can now be displayed. Figure 2.4 from the Android design philosophy shows exactly what that means.

Activities As described in Section 2.4.1, they present the displayed screen.

[And13f], [Mei12], [Ayd12]

2.6.1 Layouts

The Android Layout Managers can either be created dynamically in the source code or as the Android developer guidelines recommend, be described in XML files that are stored under the `res` folder as mention in Section 2.3.3. A Layout consists of a hierarchy of other nested

Layouts and widgets. Widgets are action elements as text fields, buttons, etc. In a Layout, it is defined how all these elements should be positioned and how they fit together. The Android SDK already comes with a few Layouts that can be customized in order to position the child Views in a modern and aesthetic way.

Figure 2.5: Linear Layout [And13h]

Figure 2.6: Relative Layout [And13h]

Frame Layout This one stacks all elements on top of each other. The location of placement can be changed editing the `gravity` attribute.

Linear Layout As mentioned in the name, it positions elements in a linear way. That means it puts them either vertical or horizontal. Figure 2.5 shows an example of that. The size of its columns or rows can be controlled by the `weight` attribute.

Relative Layout This is the most flexible one. The child Views' positions can be defined relatively to each other. In Figure 2.6 a simple representation of that is depicted.

Web View This Layout displays HTML content.

Grid Layout It was introduced in Android 4.0 and puts its child elements in order of a rectangular grid together. It can be used in older Android versions by using the Android Support Library, that is covered in Section 2.7.1. This layout was introduced in order to avoid complex Linear Layouts or Relative Layouts. Having such Layouts can affect the performance and probably nesting more than ten may cause a crash of the application.

List View and Grid View Grid View and List View can be filled dynamically with data and represent this data either as a grid, as Figure 2.7 shows, or as a list, illustrated in Figure 2.8. The data population can be done by using some of the subclasses of the `Adapter` class or by implementing a custom one.

[And13h], [Mei12], [Ayd12]

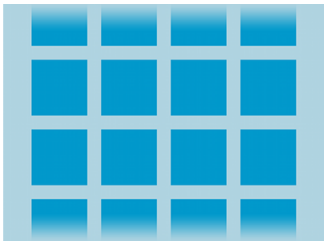

Figure 2.7: Grid View [And13h]

Figure 2.8: List View [And13h]

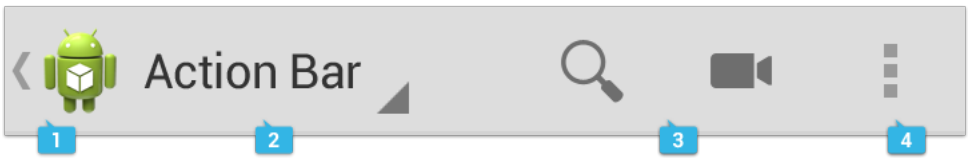

Figure 2.9: Android Action Bar Elements [And13b]

2.6.2 Action Bar

The Action Bar was introduced by Android 3.0 and was first just supported on tablet devices. That changed shortly after it was released. Now, the Action Bar is a fundamental user interface component in Android.

It has four basic areas, which are depicted and enumerated in Figure 2.9:

1. The application's icon, is being displayed here. Depending on the application's hierarchy, the "Up" navigation arrow should be displayed right next to the icon, if the application is not in its top layer state. The Android Design Guide explains in detail how the Android Navigation pattern in relationship to the application's hierarchy works and what should happen when the user presses the "Up" and "Back" button.
2. This area can be used to jump between different views, if multiple views are used. It can also be used to just display some text information as title or something else. However it does not need to display anything, it can be just empty.
3. This is the area where all Action Buttons are displayed. The possibilities to display them are:
   - with Text
   - with no Text
   - with Text, if space is available
4. Action Buttons that are not used often or if there is simply no available room for them, are displayed in the Action Overflow Menu.

In addition, the Action Bar provides even more functionality components:

- Top Bar: Is displayed below the main Action Bar and can be used for example for a tab navigation.
- Bottom Bar: Positioned on the bottom of the application, this area can be used with Action Buttons and Action Overflow Menu.
- Contextual Action Bar: When the user edits context as copying or selecting, a Contextual Action Bar is displayed over the main one.
- View Controls: There are three different control options that can be used to switch between different app Views. Fixed and scrollable tabs, a slide out menu called "Drawer" and a drop down menu called "Spinner".

Since having a lot of different displaying components, the Action Bar can be used to display certain functionality and design components in the best suitable way. In combination with Fragments, it can be used to adopt to certain display sizes and allows to create a responsive app design.

For a deeper look and preview of the Action Bar components, please have a look at the Android Design Guide and Android documentation.

[Mei12], [And13i]

2.7 Compatibility

This is an important topic for Android developers. The Google Play Store lists 2744 different device models at the time of this writing. This means, there is a big variety of different screen sizes and display resolutions that need to be supported. These have to be taken into consideration while creating the application's designs and layouts. As mentioned above there are several ways to adopt to them.

Even though the latest Android release is 4.2, there are a lot of devices that still use older versions, as shown in Section 2.2. Therefore, backward compatibility should always be supported. Up to date data about the different versions' distributions are provided on the Android website. As for now the majority still uses Android 2.3. The Android OS developers are aware of this issue and are providing an Android Support Library for this matter.

2.7.1 Android Support Library

The Android Support Library package includes a collection of static libraries, that help to provide newer API functionality to older Android versions. In particular this library is compatible with all devices running Android 1.6 and above. For example, this enables the use of Fragments, that were introduced in Android 3.0, in lower version. It tries to simplify the development process of supporting multiple Android platforms. In order to use it, the JAR file of the newest version has to be copied to the Project, it is included in the Android

SDK. So this would look something like this:
`cp SDK/extras/android/support/v4/android-support-v4.jar AndroidProject/libs/`

After that, it can be imported in the project files that need the support library as shown below:
`import android.support.v4*;`

To see what exactly is provided and how it can be implemented, please have a look at the Android samples and Android documentation.

However, some of the provided functionality is not really the one developers were hoping for. The best example is the Action Bar. Using the support library does not really provide the same look and feel. It just transforms the Action Buttons in an Overflow Menu. This means that the Action Bar will be not displayed on a device running Android 2.3.

[And13k], [Mei12]

2.7.2 Android Community Libraries

There are many Android libraries provided by the Android community. Some of them provide a backward compatibility of certain Android API functionalities and some of them just provide functionality that is not covered by the Android API at all. However, there are two popular libraries that are widely used by developers.

Action Bar Sherlock

The ActionBarSherlock is a back port of the Action Bar API and is published by Jake Wharton under the Apache License 2.0. It provides the support to use the Android Action Bar design pattern throughout all devices. Now by including this standalone library in a project, one can create an Action Bar for Android 2.x Applications and above, without any problems. If the application is running on an Android platform that already includes the Action Bar, it will automatically use that one instead. For more information please have a look at the official website.

[Act13]

Nine Old Androids

The Nine Old Androids library is also published by Jake Wharton under the Apache License 2.0. It provides a back port of the Android 3.x animation API to Android 1.x and above.

[Nin13]

2.8 Maven

Maven is a cross-platform project management tool that tries to unify certain aspects of the software development process of a project. A big approach doing this is, handling the build process in a uniform build system. A Maven project has a Project Object Model (POM) `pom.xml` file that provides the complete configuration for the build process. A project can have one of them, or multiples where on top of the configuration hierarchy a parent POM file exists that contains global information and combines his child POM files. Besides certain build steps, dependencies can be defined as well through Maven. A package manager is included, that downloads from repositories all defined dependencies required for the project. Maven configures a local repository that is first searched through before reaching to the central repositories on the web.

The usage of Maven is gaining popularity in Android projects. The standalone Maven tool is already powerful and provides rich configuration possibilities. However, it can be extended through plugins. The Android Maven Plugin is one of them and provides the ability to perfect Android projects with Maven. By taking advantage of available plugins, all necessary steps to build and release an application, can be fulfilled.

Among them is the option of signing and packaging Android applications so that they can be published directly to the Google Play Store after that. It even provides a dynamical interaction with the Android manifest. This means that data as the applications name, package, theme, permissions and so on, can all be set through Maven.

Dealing with third party libraries can be complex, especially integrating them into existing projects. Maven is facilitating the management of dependencies a lot by using the repository management system.

[Apa13], [And13s]

3 iOS

3.1 Fundamentals

This thesis is focusing on the native development process of Android and is not covering the aspects of other platforms. Nevertheless a quick overview of iOS is given in this section.

Objective-C is the main programming language that is used to develop applications for iOS or Mac OS X. It is based on the C language and therefore applications written in it can also be compiled against the Objective-C compiler. Cocoa and Cocoa Touch are two APIs that are provided by Apple in order to build software for Mac or iPhone. Apple provides as an IDE XCode that includes the API and other development tools, as device simulators and so on.

Applications written for iOS follow the Model View Controller (MVC) pattern as described further below. In iOS there is not such a large variety of devices and screen resolutions. At the time of this writing there are five different resolutions that cover all iOS devices. However iOS development is influenced by Apple's policies and guidelines. The release of an application always needs to be approved first by Apple. Therefore, it is important that used cross platform techniques are also approved by Apple. Moreover, it can be said that for the compilation process, a Mac computer with OS X is always needed.

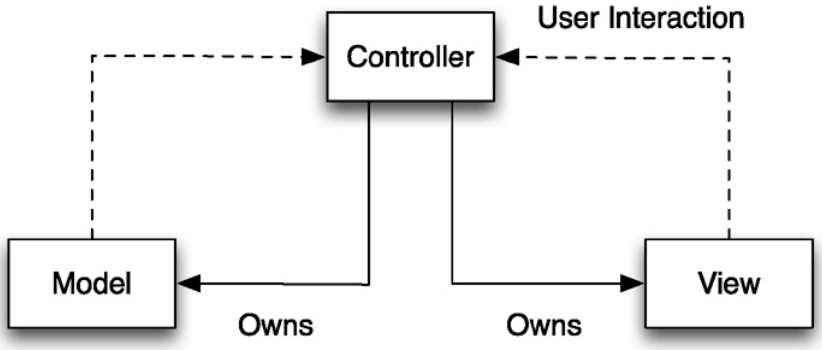

Figure 3.1: Interaction between MVC Components [Fur13]

3.2 Model View Controller

The MVC was invented by Trygve Reenskaug[1] and divides an application into three components. Figure 3.1 illustrates the basics of it. Weak relationships are represented by the dashed lines and strong ones through the solid ones. They can be defined as followed.

Model The Model represents the data layer.

View The View represents the UI and presents the data from the Model in it.

Controller The Controller is between the View and Model. It handles the input of the user and translates it if needed to logical requests to the Model and updates it if necessary. In certain scenarios the Model changes without any user interaction. The Controller is notified in such cases by observing changes in the Model using a Key-Value Observation (KVO). It communicates when the View needs to change its presentation of the Model.

[Fur13]

[1] http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

4 Cross Platform Development

4.1 Hybrid - PhoneGap/Cordova

PhoneGap is a framework for creating mobile applications for various platforms using HTML5, JavaScript and CSS3. It was originally created by Nitobi and acquired by Adobe in 2011. Moreover, they also donated the code of PhoneGap to the Apache Software Foundation. In this process PhoneGap, as an open source project under the Apache License, needed to be renamed. Therefore, the project's name is Apache Cordova. That means that PhoneGap is a distribution of Apache Cordova.

The current stable PhoneGap version is 2.6 as of the time of this writing. In Figure 4.1 the current supported features on different mobile operation systems are displayed.

PhoneGap enables the usage of these components by providing a `JavaScript` API. This forms the uniform module between the web and native components. This is one of the reasons why PhoneGap gained popularity. Developers can use one common programming language to build "apps" and interact with the native OS through a mutual API. These so called "apps" are nothing else than a website, being displayed through a webview component of the device. Almost all popular devices use the WebKit engine for the HTML rendering process. Therefore, it can be implied that the displayed UI will look the same on all devices and just in certain cases will need some minimalistic customization of the code. An exception is Windows Phone, which uses Trident as a web engine. [War12], [Pho13a], [GP12], [Lun11]

4.1.1 Mobile Web Development Frameworks

The PhoneGap/Cordova framework enables the usage of native features through a JavaScript API but a web app using them still needs to be created. Without going into details of mobile web development, it can be assumed that creating a mobile web application without the usage of any framework is hard. One of the biggest issues is that all browsers differ from each other. Using a framework that abstracts these differences is handy. Moreover, the most important part is the user interface. Modern frameworks already have appealing UI elements that can be used or easily customized without having to worry about browser incompatibility. A framework provides the necessary DOM manipulation and AJAX call functionality. Using one means that a lot of work is already done, and this can help focus on the logic and core functionality of a web app. jQuery Mobile and Sencha Touch are two excellent frameworks that are used by the majority of developers. Therefore, these two are going to be explained within the scope of this thesis.

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 5.x	Blackberry OS 6.0+	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	X	✓	✓	X	X	✓	✓	X	✓
Contacts	✓	✓	✓	✓	✓	X	✓	✓	✓
File	✓	✓	✓	✓	✓	X	✓	X	X
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	X	X	X	✓	X	X
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	✓	X

✓ - supported feature
X - unsupported feature due to hardware or software restrictions

Figure 4.1: Supported Device Features [Pho13d]

Sencha Touch 2

Sencha Touch was formed by the merge of the popular JavaScript frameworks Ext JS, JQTouch and Raphaël. It is specially designed to create mobile web applications and follows the MVC pattern as described in Section 3.2. It has a multi license model that can seem complex, but it comes down to one main statement: The core framework can be used for free for commercial or open source projects. For more information on this matter, please have a look at the different licensing options on their official website[1].

The first version had some problems with the performance but since the release of version 2 in March 2012 they were eliminated. This was accomplished by replacing the rendering engine that is now completely based on CSS and is compatible to the class loading system introduced in Ext JS 4.

Sencha Touch 2 is based completely on JavaScript and therefore the whole web app would just have one HTML file where all JavaScript code would be included. Everything else is added through JavaScript to the DOM. Even though the framework is based on the Ext JS standard of writing code that simplifies reading complex code, the framework has a steep learning curve, especially for developers that do not have a lot of experience using JavaScript. The framework is completely optimized for the Webkit engine and therefore performs really well on iOS, Android and Black Berry 10 devices, that use the same engine. This is probably the biggest disadvantage. It is WebKit only and does not support other engines and therefore does not run on other devices like Windows phone.

[1] http://www.sencha.com/products/touch/license/

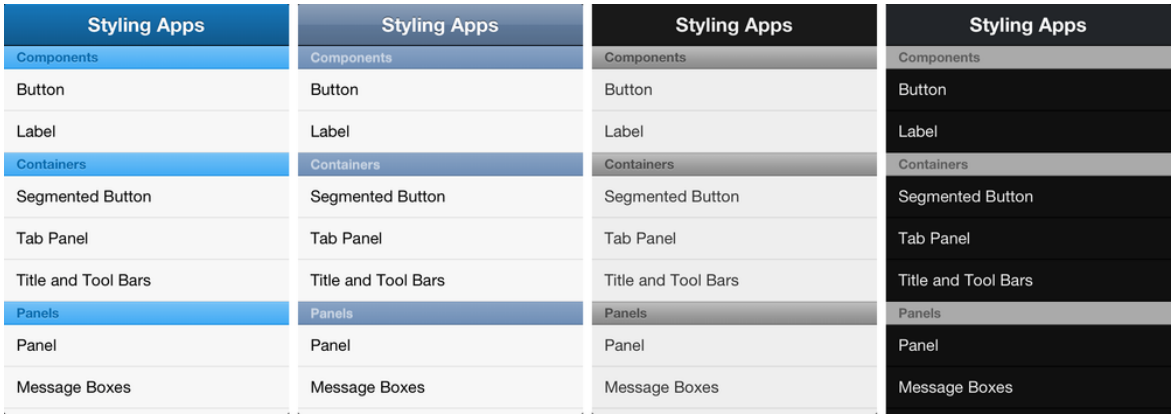

Figure 4.2: Sencha Default Themes Side by Side [Kos13]

The main feature of the Sencha framework is a large UI elements library that is inspired by iOS. It provides buttons that can be themed differently. A big collection of various form elements is included that are used in iOS like sliders, checkboxes, selectors and others. Moreover, it provides a top and bottom toolbar, a menubar and movable tabs. It has a great integration of touch events that perform especially well on the integrated list component. Even a map component is provided that fully supports multi touch gestures.

All graphical elements can be themed differently by using CSS. The framework comes with four different default themes:

- Sencha Touch style
- iOS style
- Android style
- Black Berry style

They are displayed side by side in Figure 4.2 and are supposed to provide a more "native" look of the finished web app.

The actual styling is done in Sencha by SASS. SASS is a meta language to describe CSS. In Sencha this is done automatically by the Compass interpreter that watches a certain styles folder after being started and translates the SASS files into CSS ones as soon as changes occur. Using SASS in combination with Compass actually helps to maintain and organize the different styles. The framework also provides impressive transition effects that are based on CSS3 like slide, fade, cover, reveal, pop and flip. Examples of the animation can be found on their website[2].

Sencha Touch 2 itself provides some native features out of the box, like the access of the device's camera. However, these are very limited and definitely cannot replace a rich framework as Cordova. A nice feature provided is the ability to compile and pack everything in a native app with just one command.

[2] http://dev.sencha.com/deploy/touch/examples/production/kitchensink/

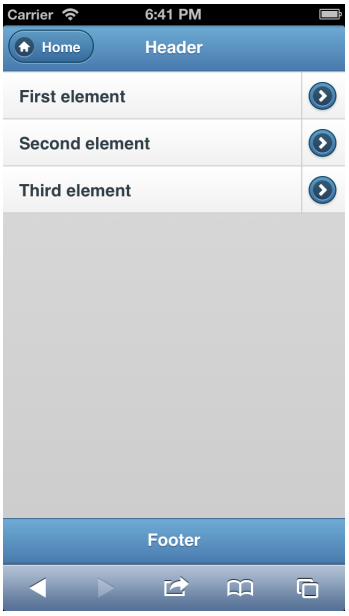

Figure 4.3: Simple jQuery Mobile Example [Gif12]

Moreover, Sencha provides a "market" where extensions can be published and downloaded. As for right now over 167 extensions are available. Everyone of them has its own separate license agreement, but a lot of them are free to use.

[Kos12], [Kos13], [GP12]

jQuery Mobile

jQuery Mobile is a web development framework that is based on jQuery. Both of them underlie the MIT license and can be used for free. The jQuery Mobile is optimized for touch devices and provides full cross browser support. It works for all mobile devices and their browsers, by giving fall back functionality to older ones. Moreover, it categorized them in three support groups: full support, full support without AJAX and basic support.

The framework is light weight and uses HTML5 markups to trigger certain events. However, that does not mean it is not powerful. It has a wide range of UI elements and provides basic ones like buttons, list views, tabs, dialogs and even more. It also includes a top-bar, bottom-bar, navigation bar and a lot more like popups, sliders and so on. It also provides support for the sidebar navigation pattern. jQuery Mobile calls this kind of sidebar a "Panel" and they can be used on the left, right or both sides at the same time. Since the framework is open, there are a lot of third party plugins or extensions that provide additional functionality, as a photo gallery, date picker and much more. Moreover, if they are not enough all plugins from jQuery can also be used.

```
<div data-role="page" id="indexPage" data-theme="b">
<div data-role="header" data-position="fixed" data-theme="b">
        <h1>Header</h1>
        <a href="#"  data-theme="b" data-icon="home">Home</a>
    </div>
  <div data-role="content">
      <ul data-role="listview" data-theme="c">
          <li><a href="#first">First element</a>
              <a href="#setasdone"></a></li>
        <li><a href="#second">Second element</a>
              <a href="#setasdone"></a></li>
        <li><a href="#third">Third element</a>
              <a href="#setasdone"></a></li>
      </ul>
  </div>

    <div data-role="footer" data-position="fixed" data-theme="b">
        <h1>Footer</h1>
    </div>
</div>
```

Listing 4.1: Example index.html

The main section of an application is the HTML area, where markup tags are used. For example, it takes full advantage of the `data-*` attribute. To start developing the jQuery library, jQuery Mobile library, jQuery Mobiles CSS and a file that includes app specific JavaScript code need to be included. In Listing 4.1 a simple example is displayed without any functionality. It uses the `data-role` in the `div` tags to create a View. A View is defined as a `page` and is managed in three elements the `header`, `content` and `footer`. The displayed listview is defined by the `data-role` tag `listview`. jQuery Mobile supports different themes that can be set and changed easily. Figure 4.3 is displaying the result of these few lines of code. The navigation between pages is performed either trough AJAX or Pages can be preloaded by being in the same HTML file. This does not cause problems, since one Page is displayed at a time and other Pages are displayed by putting links to them. In order to do that an `id` can be set in the `div` elements. Through displaying only part of the loaded HTML loading time between the transitions can be reduced.

jQuery Mobile comes with three default themes that can be used. Custom themes can be created by either doing it manually or taking advantage of the "ThemeRoller" provided by jQuery Mobile. This tool gives the ability to configure themes online and provides an import and export function. That way created themes can be edited after that.

If native themes are desired, they can be either made using the "ThemeRoller" and some tweaks after that or themes provided by the community on GitHub can be used. Figure 4.4 displays how this could look by using the second option.

[Kos12], [JQu13], [GP12]

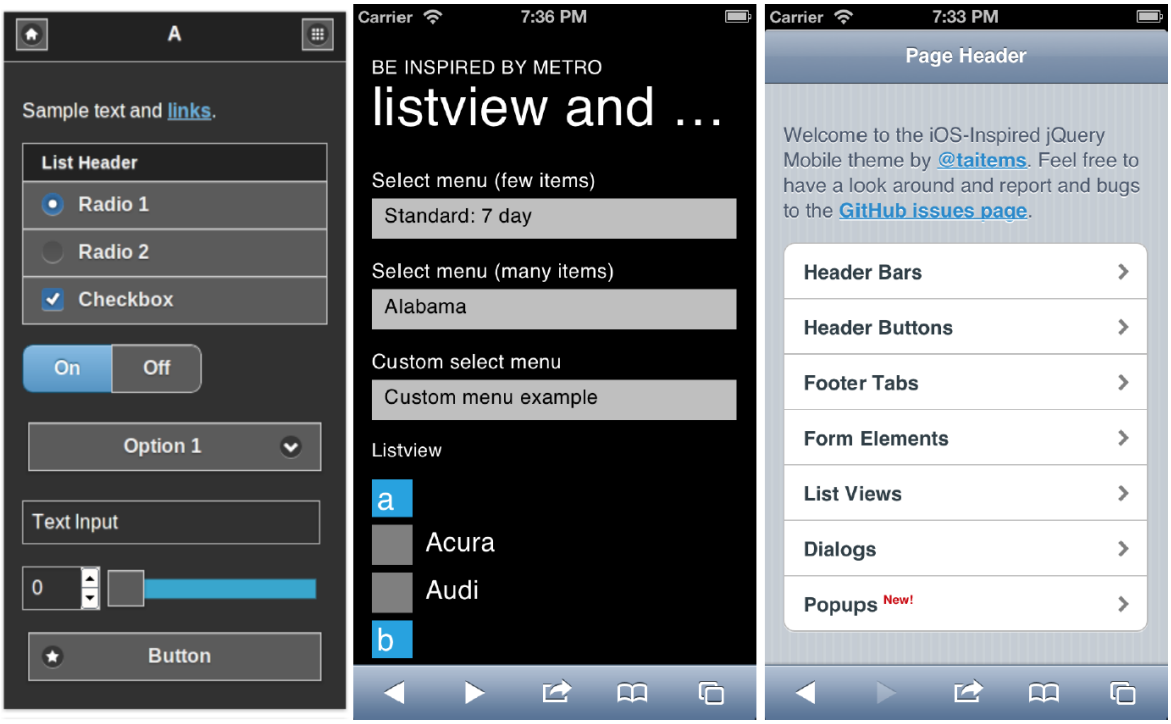

Figure 4.4: Different jQuery Mobile Themes [Git13g], [Git13e], [Git13f]

Conclusion

jQuery Mobile is a great example of providing a library that takes care of UI elements, the navigation between them and covering a wide range of devices. The only problem with jQuery Mobile is while small projects can be done perfectly well with it, bigger ones that are more complex are going to be difficult to create and maintain.

Sencha Touch 2 provides many advantages, as the already included functionality to support AJAX, YAML, JSONP and providing aesthetic UI elements and animations. Moreover, it uses web standards as HTML5 and CSS3 and is applicable for complex web apps. However, it is not suitable for web applications that should run on other web engines, besides the WebKit and for ones that are small and simple and just have a few different views.

A mobile web app with key features can be created with both frameworks. Each one of them has significant advantages over the other one. Table 4.1 summarizes the benefits and disadvantages of them.

[Kos12]

4.1.2 PhoneGap Architecture

Figure 4.5 displays the application architecture of a PhoneGap application. The three different colors symbolize the different layers of it. The first one holds all relevant web application

Framework	Positive	Negative
jQuery Mobile	Cross browser support (fall-back model) Lightweight framework Easy to learn Many plugins & extensions	a few UI widgets Can be slow limited to buttons and forms
Sencha Touch 2	fast & flexible MVC architecture Professional widgets good support for graphics (SVG, charts)	WebKit only Learning curve Heavyweight framework

Table 4.1: Side by Side Comparison of the Two Frameworks

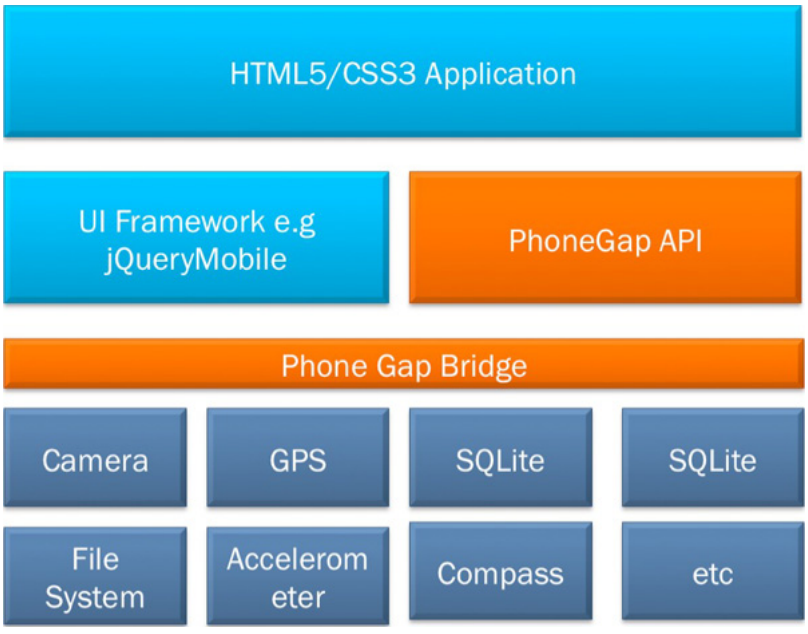

Figure 4.5: PhoneGap Application Architecture [GP12]

parts of it. This is the part that actually is being created by the developers and represents the app. It holds all the customized HTML, CSS and JavaScript files and all the web frameworks as jQueryMobile. The second layer is the PhoneGap framework that is made up of two main components, the PhoneGap JavaScript API and the so called PhoneGap Bridge. The JavaScript API provides common functions in order to use native features of the devices. All of them can be used the same way through all different mobile operating systems. The API provides a common interface and the PhoneGap Bridge actually establishes the communication between JavaScript and the native components. The third layer represents the native components. The functionality of them is implemented in the native programming language of the operating system. The typical work flow is, calling the PhoneGap API, this will trigger the native part of a Plugin. After performing the desired functions, it will return the results back using the PhoneGap Bridge. This way the web application does not need to know anything of the operating system it is running on. [GP12], [Kos12]

4.1.3 PhoneGap Bridge

The so called PhoneGap Bridge actually consists of two bridges, the JavaScript bridge and the native bridge. One of them is responsible for the communication from JavaScript to native and the other one for the correspondence between native to JavaScript. The first one is stationed in the `cordova.js` that also includes the so called PhoneGap JavaScript API and needs to be included as a library in the web application. Since the interpretation of JavaScript through the different operating systems differs, the implementation of it actually differs too. This however does not matter much since the provided interface API still remains the same. It just needs to be assured to include the right file into the right project since the Android `cordova.js` cannot be used in iOS for example.

The native bridge is implemented in the native programming language and uses depending on the mobile OS different approaches to communicate with the JavaScript bridge. For example in Android this would be the `cordova-2.6.0.jar` file that needs to be included. [Kos12]

Android Bridge Components

In Figure 4.6 the PhoneGap Bridge of an Android application is being displayed in detail. On the left side is the overridden `android.webkit.WebView` and on the right the native part of it. The solid lines represent the default communication approaches. As presented the WebView talks to the native part by using the JavaScript prompt function. The Plugin functions that should be triggered are packed in customized prompt commands. On the other side, the overridden `onJsPrompt()` function is catching the prompts. Depending on its signature and included command it is calling the native implemented part. Another way to trigger native code can be done by adding a JavaScript Interface to the WebView. This would bind a JSObject to it that would call a certain native code depending on the arguments of the interface.

The communication between native and JavaScript is done by responding to pull requests that are sent by the WebView constantly to the native side. Depending on the usage frequency

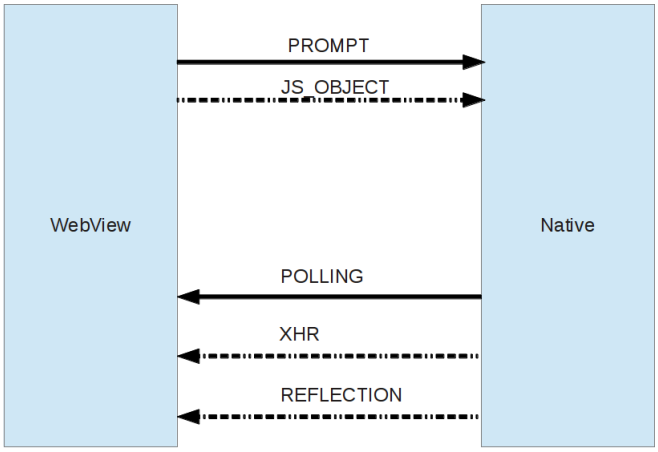

Figure 4.6: Android Bridge [The13]

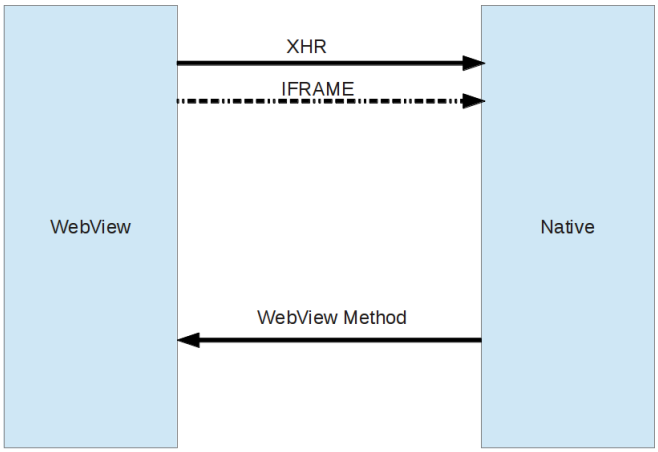

Figure 4.7: iOS Bridge [The13]

of the Plugins this might cause some battery drain. However, if needed alternatives can be used. One of them is using long polling XmlHttpRequests. The native side would register incoming requests in an internal callback server and response to them. Moreover, a Java internal reflection performed on the WebView can be used to respond. This is available since Android version 3.2 and therefore not used that often in respect to backward compatibility. [The13]

iOS Bridge Components

As illustrated in Figure 4.7, the iOS Bridge does not have a lot of possibilities performing the communication between the two components. Actually, it does not need a lot since it is performing this already well. On the left side is a more or less overridden UIWebView, represented by a WebView. So here JavaScript sends XMLHttpRequests to the native

side. This is performed by calling URLs with a specific scheme and packing the necessary commands into them. The native part registers them and calls the appropriate Plugins. In iOS versions lower than 4.3 this is done by using an iframe. The correspondent information is sent back through the `stringByEvaluatingJavaScriptFromString` method that is part of UIWebView. [The13]

4.1.4 Example Implementation

In order to develop an application with PhoneGap the required SDKs of the desired mobile operating systems need to be installed. In this example, an Android based PhoneGap application is going to be produced. The framework includes custom shell scripts that can create a simple project for each platform with all necessary files. In this case, the script located in the Android - bin folder can be executed as stated in Listing 4.2.

```
1 ./create ~/my/project/path/hello at.tu.hello hello
```

Listing 4.2: Install Cordova in Local Maven Repository Example

The created project already includes the libraries files (`cordova-2.7.0.jar` and `cordova-2.7.0.js`).

```
1 ...
2 <script type="text/javascript" src="cordova-2.7.0.js"></script>
3 <script type="text/javascript" src="js/index.js"></script>
4 <script type="text/javascript" src="other.js.functions.or.frameworks.js">
5 </script>
6 ...
```

Listing 4.3: Example index.html

Listing 4.3 illustrates the Cordova and other JavaScript imports in the `index.html` file. In order to ensure stable functionality, the framework should be included first. Other imports can be performed after that. Moreover, this will lead to the web applications being fully functional faster. PhoneGap as mentioned above consists of two parts. While the JavaScript part might be loaded faster, the native part is going to need more time. Therefore, the framework is triggering an internal event to indicate that the web application is completely loaded inside the framework environment. Normally this takes more time than loading some other JavaScript methods or frameworks. Therefore, ensuring that all events are loaded in the correct order is important. Otherwise, the PhoneGap web code might be fully loaded trying to communicate with the native bridge that would still be in the middle of initialization. When the internal wrapper is ready, it would send a "deviceready" signal through the native bridge to indicate that.

```
1 initialize: function() {
2     this.bindEvents();
3 },
4 bindEvents: function() {
5     document.addEventListener('deviceready', this.onDeviceReady, false);
6 },
7 onDeviceReady: function() {
8     //do something
9 }
```

Listing 4.4: Example index.js

In the "HelloWorld" created by the script, this part is in the `index.js` file and is displayed in Listing 4.4. A `document.addEventListener` is added in the `bindEvents` function and is waiting for the `deviceready` event to be triggered. PhoneGap is completely loaded and fully functional only when this is being performed.

4.1.5 PhoneGap API

The PhoneGap API is the interface in front of all the implemented functionality of the framework. It is based on a Plugin architecture. The native functionality is provided by multiple Plugins, that are already included in the framework. They consist of a JavaScript part that can be accessed through a common API and a native part. This means the framework needs to know which of them are available. In order to accomplish that Plugins need to be registered on the native side. For this purpose, there is always a `config.xml` file that provides the framework with this information. In Android the file is located in the `res` folder and under iOS under the root project folder. Moreover, certain permissions need to be set in some operating systems, as Android. Therefore, depending on the OS certain configuration files need to be edited. [Pho13a]

Accelerometer

The Cordova framework provides methods to obtain the current acceleration values of the device's axis or to start obtaining certain orientation changes in a specified time interval. The returned values can sometimes differ depending on OS and device. However, since none of the emulator provides device orientation movement, this feature needs to be tested on an actual device. [War12], [Pho13a], [Kos12]

Camera

The API provides the functionality of taking a picture with the camera or retrieving one from the photo gallery. This means that in particularly when using this function either a URI will

be returned that points directly to an image saved on the device or it will return the content of an image as a base64 encoded string.

```
1 navigator.camera.getPicture(cameraSuccess,cameraError,[cameraOptions]);
```

Listing 4.5: Cordova - Camera

Listing 4.5 shows the command that needs to be performed to do that. The two methods `cameraSuccess` and `cameraError` will be called depending on the result. `cameraOptions` can be used to set the source of the picture and some other options as quality, width, height, and so on. [War12], [Pho13a], [GP12]

Capture

This feature can be used to obtain one or multiple picture(s), video(s) and recording(s). Apparently the implementation of it is based on the W3C Media Capture API. Even though it is based on a standard, it is not supporting it. That means certain parts of it are not implemented. However, the device's default application will be used to obtain these media files.

```
1 navigator.device.capture.captureAudio(captureSuccess,
2                                       captureError,[options]);
3 navigator.device.capture.captureImage(captureSuccess,
4                                       captureError,[options]);
5 navigator.device.capture.captureVideo(captureSuccess,
6                                       captureError,[options]);
```

Listing 4.6: Cordova - Capture

The usage of the capture functions is displayed in Listing 4.6. Depending if the function is going to fail or succeed, one of the functions passed as the first two arguments will be called. The capture `options` can contain three different options. They are `limit`, defining the maximum amount of media that can be recorded, `duration`, defining the maximum duration in seconds and `mode` defining the recording mode depending on the supported capture types. If everything works well, the function will return a `MediaFile` object. This contains, among other information name, path and format specific data.

[War12], [Pho13a], [GP12], [Lun11]

Compass

```
navigator.compass.getCurrentHeading(compassSuccess,
                                    compassError,
                                    compassOptions);

var watchID = navigator.compass.watchHeading(compassSuccess,
                                             compassError,
                                             [compassOptions]);
```

Listing 4.7: Cordova - Compass

Querying the compass function will return an angel between 0 and 359.99 that represents the pointing direction of the phone. The in the device included physical compass chip points to the north, so the value 90 indicates the smartphone is pointing east, 180 refers to south and 270 stands for west. Listing 4.7 illustrates the two most used compass methods. The function `getCurrentHeading` will return a `compassHeading` object on a success. This holds information about the magnetic heading, true heading, accuracy and a time stamp. The `watchHeading` function provides the functionality of performing compass queries operations in a certain time frequency or whenever the heading value changes.

[War12], [Pho13a], [GP12]

Connections

All the possible connection states are saved as constants. So it is enough to get the `navigator.connection.type` property and compare it to the constants. Knowing the current type of connection is a valuable information and is needed to operate correctly. If there is no connection to the Internet, it does not need to perform actions that require one. [Pho13a]

Contacts

The Cordova API provides functionality queries to interact with contacts on the device. Among searching for them, they can be added, edited or deleted.

```
1  var contact = navigator.contacts.create();
2  contact.displayName = "mydisplayname";
3  contact.nickname = "mydisplayname";
4
5  var name = new ContactName();
6  name.givenName = "Max";
7  name.familyName = "Musterman";
8
9  var phoneNumber = new ContactField('mobile', '111-222-3333', true);
10 contact.name = name;
11 contact.phoneNumber = phoneNumbers;
12 contact.save(onSuccess,onError);
```

Listing 4.8: Cordova - Contacts

In Listing 4.8 it is illustrated how to add a new contact. First, a new `contact` object is created by the `create()` function. In order to support all devices a `displayName` and a `nickname` need to be assigned to it. This object contains a variety of properties representing certain contact information. Therefore, in order to add information to it certain objects need to be created and be filled with data. After that they can be put into the `contact` object and actually saved through the `save` method. If a contact should be deleted, first the `contacts.find()` function is performed. In case of succeeding, this will return a `contact` object and using its `remove()` method, will delete it.

[War12], [Pho13a], [GP12]

Device

The framework provides device specific information through a `device` object. The amount of information is limited but still important ones as the device's platform, OS version and model are provided. [Pho13a], [GP12]

Events

Cordova triggers certain events during its lifecycle. They can be obtained by setting an event listener, as show in Listing 4.4. Probably one of the most important events is the `deviceready` one that was already mentioned in Section 4.1.4. It is fired when the complete Cordova framework is loaded and ready to be used. However, other interesting events are for example application specific ones, like putting the Cordova application into the background. When the device changes its network status, an `online` or `offline` event is triggered. The rest of the events are covering the usage of certain buttons.

[War12], [Pho13a], [GP12]

Files

The API handling of file operations is based on the W3C File API and is providing all the essential functionality of it. Therefore, the Cordova Framework supplies functions to read, write, copy, remove and find files.

```
1 var filePath = // retrieving some path before that
2 var fileTransfer = new FileTransfer();
3 var uri = encodeURI("http://www.example.com/somefile");
4 fileTransfer.download(uri, filePath, onSuccess,
5                       onFail, [trustAllHosts] , [options] );
```

Listing 4.9: Cordova - Filetransfer

Actually, the API provides several objects that have different methods and constants regarding file operations. For example, requesting local storage can be done by using the `requestFileSystem` method from the `LocalFileSystem` object. Here for example a temporary or persistent one can be requested. Moreover, through the `DirectoryEntry` an actual file operation can be performed. For example going through the directory, retrieving or creating a file; creating other directories and looking up their path. In Listing 4.9 an example is shown on, how the `download` method from the `Filetrasfer` object can be used. After the path is retrieved, an encoded URI and the object can be created. After that the `download` function with the necessary arguments is available for use. On a success a `FileEntry` object will be returned, that has similar functionality as `DirectoryEntry`. `trustAllHosts` and `options` are optional. The first one controls the confidentiality to the host, and the second one can be used for defining the header. The upload function works almost the same, with the exception that a whole `FileUploadOptions` object can be passed through as an option. Moreover, on a success a `FileUploadResult` object will be returned that holds among other information the server response data. The biggest difference between downloading and uploading a file is, that the first one is only supported on Android and iOS devices. Unfortunately, small restrictions like this are not mentioned in the official overview of features supported, as displayed in Figure 4.1.

[War12], [Pho13a], [GP12], [Lun11]

Geolocation

This part of the API makes the latitude and longitude data of the device available. It either uses GPS or network specific data information and is based on the W3C Geolocation API. The usage is similar to the Compass feature. It can retrieve information of the current position of the device or start a periodical location watch that would asynchronously return data as soon as a position change was detected.

```
1 navigator.geolocation.getCurrentPosition(onSuccess, onError);
```

Listing 4.10: Cordova - Geolocation

Even though almost all new devices have geolocation capabilities, it cannot be guaranteed that the actual position of the device is going to be retrieved. The reason for that can on the one hand be technical on the other hand it can be just as simple as that the user turned off the location retrieving possibility of the device. Moreover, since the HTML5 mobile browsers support the W3C Geolocation standard, the Cordova functionality can be used directly through it. In order to do that after Cordova is fully initialized the function `Geolocation.usePhoneGap()` has to be called. After that the usage, as shown in Listing 4.10, the Geolocation support will invoke Cordova in the background.

[War12], [Pho13a], [GP12], [Lun11]

Globalization

The Globalization feature is a newer Plugin, it was included into the core API with the version 2.2 release. As the name reveals it provides functionality regarding the user's locale and language. This feature can be used to obtain the main used language and therefore facilitates the creation of multi-language apps. Moreover, this Plugin is able to provide correct formated numbers and dates, in respect to the user's country. [Pho13a]

InAppBrowser

The inAppBrowser functionality was added with the release of version 2.3. It is basically a built-in web browser that supports browsing through normal websites. It is based for the most part on the popular ChildBrowser Plugin that can be found on GitHub. As of right now this Plugin only works on iOS and Android.

```
1 var ref = window.open(url, target, options);
```

Listing 4.11: Cordova - InAppBrowser

The usage is pretty simple and depicted in Listing 4.11. There are three different options that can be passed through. "_self" will open the whitelisted URL in a Cordova `WebView`; "_blank" will always open the URL in the InAppBrowser; "_system" will open the website with the device's default web browser. [Pho13a]

Media

This provides the necessary functionality to record and play audio files. The Plugin is old and does not fulfill the W3C standard of capturing media files. The Capture Plugin fulfills some parts of its functionality and therefore clearly overlaps with this one. However, it is not possible to play audio files with the Capture feature.

```
1 var my_media = new Media(url, onSuccess, onError);
2 my_media.play();
```

Listing 4.12: Cordova - Media

In Listing 4.12 it is stated how an URL of an audio file can be played. First a `Media` object pointing to an audio file needs to be created. The other two arguments are callback functions for a success and a fail. After that methods like `play`, `pause`, `stop`, `startRecord` and so on can be called.

[War12], [Pho13a], [GP12], [Lun11]

Notification

Cordova provides several notification possibilities that can be used to give feedback to the user. For example between the `notification.alert` pop up command, the `notification.confirm` method can be used to create a pop up with a message. Then, the user can choose between a "positive" and "negative" button. Moreover, the device's default beep tone can be played or a vibration notification be triggered.

[War12], [Pho13a], [GP12], [Lun11]

Splashscreen

This gives the chance of displaying a certain splash screen and hiding it. Especially web apps that have to load a lot of media content at their startup use this Plugin. Modifications on the native side need to be done, in order to apply it.

```
1 super.setIntegerProperty("splashscreen", R.drawable.splash);
2 super.loadUrl(Config.getStartUrl(), 10000);
```

Listing 4.13: Cordova - Splashscreen

For example in Android the code listened in Listing 4.13 needs to be added to the `onCreate` method of the Activity that extends the `DroidGap`. In this case, the drawable graphic - splash will be displayed for a maximum time of ten seconds. This can be dismissed by calling the `navigator.splashscreen.hide()`, when the Cordova framework is loaded completely.

[War12], [Pho13a], [GP12]

Storage

The Storage plugin is based on the W3C Web SQL Database and W3C Web Storage specifications. HTML5 compatible browsers support these storage options. The functionality is provided by JavaScript and not HTML5. Data can either be saved locally with a key and value or be written to a local SQL database. However, some devices do not support this way of storing data. If the devices do not support the W3C standard, this Plugin will enable itself and provide the same functionality as the standard. This way of storing data bears some risks. Since iOS version 5.1 it is reported that WebKit data would not be backed up. This means that SQLite, indexed DB and local storage data can be deleted anytime by the device if it is low on memory. Despite that, data may not be persistent, since there are always certain database size limitations that cannot be exceeded. Moreover, the data is not actually secure.

Depending on the application and the stored data, alternatives need to be found. One of them is the "Cordova/PhoneGap SQLitePlugin" that can be found on GitHub. It provides an interface to the natively stored SQLite database. The Plugin tries to support as much of the HTML5 Web SQL API as possible, so it can be used without a lot of changes. For example instead of using the command `window.openDatabase()` for creating a database, the command `sqlitePlugin.openDatabase()` can be used. This Plugin supports iOS and Android at the time of this writing. In iOS the database is stored as user data and will be automatically backed up to iCloud, if necessary. Moreover, in both OS SQLCipher can be used. The SQLCipher is an open source extension to SQLite that provides full encryption of the locally saved database. The encryption is performed with a 256bit AES algorithm and claims to have a small overhead of 5-15% .

[War12], [Pho13a], [GP12], [SQL13], [Lun11]

4.1.6 Creating and Using Custom Plugins

The provided Cordova API is rich on functionality and enables the usage of many native features. However, in some cases the core API will not be enough. In this case Cordova supports a great way to create custom Plugins. Since the whole architecture is Plugin-based, it is not difficult to develop one.

As already mentioned before a Plugin consists of two big parts. A JavaScript part that provides a common interface and native code behind it that is being triggered and does the actual work. The communication between them is taken care of by the Bridge. Therefore, as shown in Figure 4.8 in order to create a custom Plugin these two parts need to be created. However, before starting to create any Plugin two things should be considered; is there already a Plugin in the open source community that has the exact same desired functionality and if not, can this functionality actually be performed on all different OS. One has to be aware of the missing features on these devices. Below all necessary steps are explained in order to create a custom Plugin.

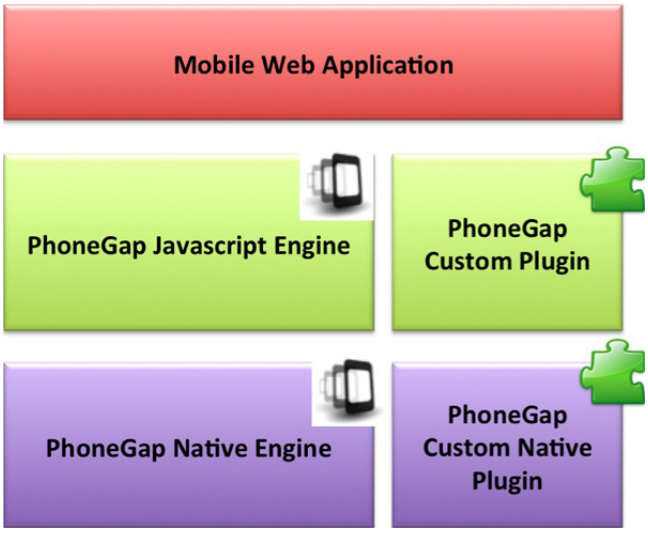

Figure 4.8: PhoneGap Custom Plugin Architecture [GP12]

Native Side of a Custom Plugin First, the name of the Plugin needs to be registered on the native side. In Android this is done under the `res/config.xml` file. An example is depicted in Listing 4.14.

```
1 <plugin name="MyPlugin" value="my.package.name.MyPlugin"/>
```

Listing 4.14: Cordova - Plugin Registration Android

Under iOS this is also done in the `config.xml` file, located in the root of the project and is displayed in Listing 4.15.

```
1 <plugin name="MyPlugin" value="MyPlugin" />
```

Listing 4.15: Cordova - Plugin Registration iOS

The registration looks almost identical. The `name` refers to the JavaScript function name and the `value` to the native class. Since Java organizes code in packages, the full path to the class is provided.

The required process is disclosed first for Android and then for iOS.

The newly created Plugin class has to extend the `CordovaPlugin` class. Moreover, the `execute` function has to be implemented.

```
1  public boolean execute(String action, JSONArray args,
2                 CallbackContext callbackContext) throws JSONException {
3
4      if (action.equals("command1")) {
5          JSONObject params = args.getJSONObject(0);
6          String url = rgs.optString(0);
7          doSomething(url);
8          callbackContext.success();
9          return true;
10     } else {
11     callbackContext.error("action error");
12     return false;
13     }
14 }
```

Listing 4.16: Cordova - Plugin Execute Android

A simple implementation is shown in Listing 4.16. The `execute` function has three arguments. Since a Plugin can handle multiple commands, the first argument is the action that has to be performed. The second one is a `JSONArray` that holds all arguments and the last one is the callback context. Different actions can be filtered first and then executed. The arguments can be retrieved from the `JSONArray`. After that the actual work can be done. Now, the methods implemented in Java that perform the heavy lifting can be called.

In Android the JavaScript does not run on the UI thread. If operations should be performed on it, this has to be enabled.

Achieving this is done by calling `cordova.getActivity().runOnUiThread` and creating a `new Runnable()` to perform the desired operations. If the code should be performed in another thread in order to not block the webcore one, this is not a problem at all. Just run the code in a `Runnable()` and execute it in `cordova.getThreadPool().execute`. After the operations is completed, those need to be communicated to the JavaScript part. For this part the callback context is responsible. Depending if everything went well or an error occured `callbackContext.success();` or `callbackContext.error("action error");` will be called. This will trigger the appropriate JavaScript function. Since the `execute` method is a `boolean` it is required to return either true or false. In the last case a "method not found" error would be triggered.

Under iOS the newly created `MyPlugin` class is a subclass of the Cordova Plugin class `CDVPlugin`.

```
#import <Foundation/Foundation.h>
#import <Cordova/CDVPlugin.h>

@interface MyPlugin : CDVPlugin {
}
- (void) command1:(CDVInvokedUrlCommand*)command;
@end
```

Listing 4.17: Cordova - Custom Plugin iOS (MyPlugin.h)

As illustrated in Listing 4.17, actions will be filtered automatically and methods with the same name will be called to execute them.

```
#import "MyPlugin.h"
@implementation MyPlugin
- (void) command1:(CDVInvokedUrlCommand*)command{
    CDVPluginResult* pluginResult = nil;
    NSString* url=[command.arguments objectAtIndex:0];

    if (url != nil) {
      dosomething(url);
        pluginResult =
        [CDVPluginResult resultWithStatus:CDVCommandStatus_OK];
    } else {
        pluginResult =
        [CDVPluginResult resultWithStatus:CDVCommandStatus_ERROR
                         messageAsString:@"Arg was null"];
    }
    [self.commandDelegate sendPluginResult:pluginResult
                          callbackId:command.callbackId];}
@end
```

Listing 4.18: Cordova - Custom Plugin iOS (MyPlugin.m)

In Listing 4.18 the `MyPlugin.m` is demonstrated, which holds the functionality. The passed argument is a `CDVInvokedUrlCommand` object. This contains all the parameters passed from the JavaScript side. It can be retrieved with `[command.arguments objectAtIndex:0]` indicating to get the first argument on position 0. After this it is advised to check that it is not `nil`. Next other functions can be called to perform the necessary operations. In this example, it is not checked if this was successful or not. After that the `pluginResult` is set to the "ok" status. If the `url` parameter was empty, `pluginResult` is set to the error status. At the end, the result of `pluginResult` is sent back by using an `commandDelegate`. Depending on the result's status the `onSuccess` or `onFail` function will be called on the JavaScript side. The iOS native side of the Plugin is executed in the UI thread. Therefore, it is critical to

make sure that the thread does not block. For example to avoid this issue the code can be run in a separate thread using `[self.commandDelegate runInBackground:^{ .... }];`.

As for the JavaScript side the commands would look similar to the ones from the framework's API.

```
1 cordova.exec(<successFunction>,<failFunction>,<service>,<action>,[<args>]);
2
3 cordova.exec(onSuccess, onFail, "MyPlugin", "command1", [message]);
```

Listing 4.19: Cordova - Simple Custom Plugin JavaScript

The call of the custom Plugin is displayed in Listing 4.19 (line 3). The `[message]` represents the arguments. In line 1 the general syntax of, how a custom Plugin should be called, is shown. As stated earlier, this command can be used after the whole framework has been initiated successfully. Moreover, the whole interface can be structured in a suitable way. The significant part is that at some point the `cordova.exec` method needs to be called. This is the entry point of all Plugins, even the core ones included in the API. As of the arguments, the first two are the callback methods that should be called in the case of the success or failure of the Plugin execution. The native side will trigger one of them automatically. `MyPlugin` is the registered `service` on the native side. This is the name used in the `config.xml` file. This way Cordova knows which class needs to be called on the native side. In the next argument the desired `action` needs to be stated. As mentioned earlier, a Plugin can handle several different actions. In this case `command1` is invoked. The last arguments are the arguments that should be passed to the native side of the Plugin.

As in the Cordova API plain JavaScript is used to invoke the native functionality. The support of custom Plugins is extensive. The whole communication between JavaScript interface and the native side and the other way around is handled by the framework itself.

Handling the JavaScript calls of the Plugins can be intensive without having some structure. Even the core PhoneGap library had problems with that. However, since Cordova 2.0 the usage of JavaScript modules was introduced. Now, the framework uses a `define` and `require` syntax throughout the whole library. This results in many Plugins that are completely independent and focus on doing their own tasks.

Module Pattern in JavaScript This pattern is used to emulate the behavior of object oriented languages. It provides the ability to save public and private methods or variables in one object and shields them from the global scope. This reduces the possibility of values being messed up by other functions. The module is a self contained global objected. The content inside is protected and can only be accessed through the module itself. Logic operations are performed inside, and only one public interface is offered. [Osm12]

```
cordova.define("cordova/plugin/myplugin",function(require,exports,module) {
  var exec = require('cordova/exec');

  var MyPlugin = function() {};

  MyPlugin.prototype.command1 = function(message,
                                         successCallback,
                                         failureCallback) {
      exec(successCallback,failureCallback,'MyPlugin','command1',[message]);
  }

  var myplugin = new MyPlugin();
  module.exports = myplugin;
});
```

Listing 4.20: Cordova - Custom Plugin JavaScript with Modules

This concept can also be accomplished in custom Plugins. In Listing 4.20 an example definition of such a module is shown. First `cordova.define` is called and defines an unique identification of the custom Plugin. The rest of the code needs to be wrapped around the `define` call. A local variable is defined, and a call is made to `require('cordova/exec')`. This will return the registered `exec` method. As described above, this is the part that is actually "executing" the Plugin. Next, a constructor is created and the desired method is registered with it. The execution is performed through this method at the end. The `exec` call is the same, except the method call is performed through the "required" module and not directly through `cordova.exec`. At the end a `MyPlugin` object is created and exported by using `module.exports`.

```
var myplugin = cordova.require("cordova/plugin/myplugin");
myplugin.command1(url, onSuccess, onFail);
```

Listing 4.21: Cordova - Custom Plugin JavaScript Module Usage

After the module is exported successfully it is accessible as stated in Listing 4.21. This code can be used wherever needed, after the `deviceready` event was triggered by Cordova. In order to use modules they first have to be "required". After that the method introduced in the module can be called. The success and fail callback methods are passed through the module to the native side.

[Gif12], [War12], [Pho13a], [GP12], [Lun11], [Pho13b], [Sim13], [Pho13c]

4.1.7 Community Plugins

Cordova is open and therefore its development is driven by the community. The whole project is hosted on the Apache GitHub repository. Custom Plugins created by the community are

hosted on GitHub as well. A collection of these can be found there[3]. Before starting to produce a custom Plugin, this list should always be checked to see if by any chance desired functionality is already available. Even if the desired Plugin cannot be found there, other implementations can be examined. Moreover, it is always appreciated releasing a custom Plugin under an open source license and making it available through GitHub. Not only others profit by this, it will also help to maintain and debug the own work. Two interesting community Plugins are mentioned in the next Section 4.1.8.

4.1.8 Native Elements

A big disadvantage of web apps in the combination with Cordova is the user interface. The elements representing the UI are not native ones. Which leads to a different feel while interacting with them. Moreover, the design and look of all the elements are often not the same as the native UI ones. For example in Android, the user expects to see at least something like an "ActionBar". In addition, the app behavior should be similar to native apps. The same applies to iOS and other mobile platforms.

However, by using the Cordova framework some of the native elements can be used in combination with the web app. Some approaches can be done easily, and other ones are more complex. But, most of the time the native elements are used as a frame around the web application. How deep the interaction with the native elements goes depends heavily on the native side and its operating system, since the implementation of common user interface design patters differs a lot between various OS and their programming languages.

Native Menu in Android The easiest way to use native elements in Android, would be to use the `OptionsMenu`. This kind of menu is displayed when the user presses the hardware menu button on the device.

```
1  <menu xmlns:android="http://schemas.android.com/apk/res/android" >
2    <item
3            android:id="@+id/command1"
4            android:icon="@drawable/command1"
5            android:title="@string/command1"/>
6    <item
7            android:id="@+id/command2"
8            android:icon="@drawable/command2"
9            android:title="@string/command2" />
10 </menu>
```

Listing 4.22: Simple Menu XML File

Therefore, it is easily possible to create a simple menu with an id, icon and title, as shown in Listing 4.22. Next the main class that extends the `DroidGap` class must be edited. There

[3] https://github.com/phonegap/phonegap-plugins

Figure 4.9: ActionBarSherlock Tabs with Cordova [Git13b]

one can override the `onCreateOptionsMenu` function and `inflate` as the menu. Running the app should display a menu. However, there is still no functionality inserted.

```
1  @Override
2  public boolean onOptionsItemSelected(MenuItem item) {
3      switch (item.getItemId()) {
4      case R.id.command1:
5          this.appView.sendJavascript("someJS();");
6          return true;
7      case R.id.command1:
8          this.appView.sendJavascript("implOfMyPlugin();");
9          return true;
10     default:
11         return false;
12     }
13 }
```

Listing 4.23: Example MenuItem

In order to add functionality the `onOptionsItemSelected` method is overridden. In it switch-case blocks distinguish between the different buttons. Finally, the usage of the `sendJavascript` function from the `DroidGap` class, provides the ability to send JavaScript to the WebView. That way JavaScript methods can be triggered inside the WebView. In addition this means, that the JavaScript interface of a Plugin is accessible from the native side.

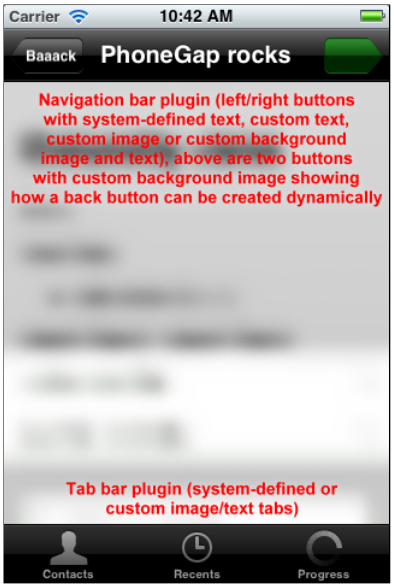

Figure 4.10: Native iOS Navigation Bar in Combination with Cordova [Git13a]

Actionbar Tab Bar for Cordova on Android One great example of the usage of the Plugin functionality, is the one released by "AndiDog" on GitHub. An example screenshot is displayed in Figure 4.9. The Plugin enables the usage of tabs from the ActionBarSherlock that was introduced in Section 2.7.2. The functionality is great but also limited. It provides the ability to display tabs on Android version 2.1 and above. A callback function will be triggered when the selected tab changes and therefore a different content can be displayed. However, the creation of the tabs is done on the native side. This means they are fixed, and the amount of different tabs cannot be changed dynamically through JavaScript. [Git13b]

Navigation Bar for Cordova on iOS There are further two great Plugins that use native elements from "AndiDog". The first enables one to create a tab bar and interact with it. The second one provides the functionality to create and use a native navigation bar. Both of them can be used completely from the JavaScript side and do not need any extra configuration after they are integrated into the project correctly. [Git13a]

4.1.9 Implementation Cordova in Maven

In order to be able to use Cordova with Maven, the library must be installed in the Maven local repository. After downloading the current version of Cordova[4] the content can be published to the local Maven repository. Afterwards a reference to Cordova can be set in the `pom.xml` file as usual.

[4] http://phonegap.com/download

```
1 User:~$ cd Developer/SDK/phonegap-2.4.0/lib/android/
2 User:~$ mvn install:install-file -DgroupId=org.apache.cordova\
3                   -DartifactId=cordova-android\
4                   -Dversion=2.4.0\
5                   -Dfile=./cordova-2.4.0.jar\
6                   -Dpackaging=jar
```

Listing 4.24: Install Cordova in Local Maven Repository Example

4.2 Cross Compilation - Xamarin

4.2.1 History

In order to understand the platform of Xamarin first a brief look at its history should be taken.

- Microsoft announced the .NET framework in 2000. Shortly after that in year 2011, the company Ximian launched the Mono open source project.
- In 2003, Ximian was bought by Novell which continued supporting the Mono project.
- In 2009, MonoTouch was released and supported creating iOS applications in C#.
- First in 2010, Apple changed its terms to not allow third party tools and languages in the development process. Shortly after this Apple reconsidered and again allowed this, as far as code cannot be downloaded.
- In 2011, Mono for Android was released, providing the ability to create Android apps using Mono.
- Later in 2011, Novell was acquired by The Attachmate and laid off the Mono developing team. Shortly after this Xamarin was founded and wanted to continue the support of Mono for iOS and Android on a new code base. However, the two parties were able to reach an angreement so Xamarin was able to reuse the already existing code. At the end of 2011, Xamarin released a new version of Mono, Mono Touch and Mono for Android.
- In 2013, Xamarin released a new product portfolio and rebranded existing products.

In summary it can be said that the creators of Mono are behind Xamarin, they are still supporting the Mono project but offer a product line based on Mono commercially.

4.2.2 General

Xamarin brings cross platform development of mobile applications to a new level and offers a solution to develop everything in one programming language. It provides a commercial product for this where applications are written in the C# language based on the Common Language Infrastructure (CLI) for the following operating systems:

- iOS
- Mac OS
- Windows 8
- Windows Phone
- Android

Mono The open source project is a cross platform implementation of C# and the Common Language Runtime (CLR), or also called CLI. It is binary compatible with the .NET framework from Microsoft and runs on a variety of different systems like Linux, BSD, OS X, Windows, Solaris, PlayStation 3, Wii and Xbox 360. Although it is open and licensed under the MIT, the dual license of GNU Lesser General Public License 2 and GNU General Public License 2[5], it was always supported by various companies.

There is a free package plan from Xamarin, however this is limited to size (32kb) of the app and other features. For all other package plans one need to pay a fee per platform per developer.

Xamarin's platform has the ability to build an application using the C# language and the .NET framework with all benefits it has.

C# features that can be used are:

- Language Integrated Query (LINQ) Support: It can be used for selecting and filtering data from arrays or a database.
- XML Support: The built-in XDocument class allows to handle XML easily.
- Event Handling & Delegates
- many more

However, the biggest advantage is that the resulting application is a native one. This means that all native UI elements can be used to build the interface of the app, and access is gained to the underlying SDK. The goal of using this platform however, is to create and design the application in a way that non user interface specific code can be shared through different platforms. Xamarin sees a mobile application build from multiple layers, as displayed in Figure 4.11. This means that certain layers can be specific to the mobile OS itself and

[5] http://www.mono-project.com/FAQ:_Licensing

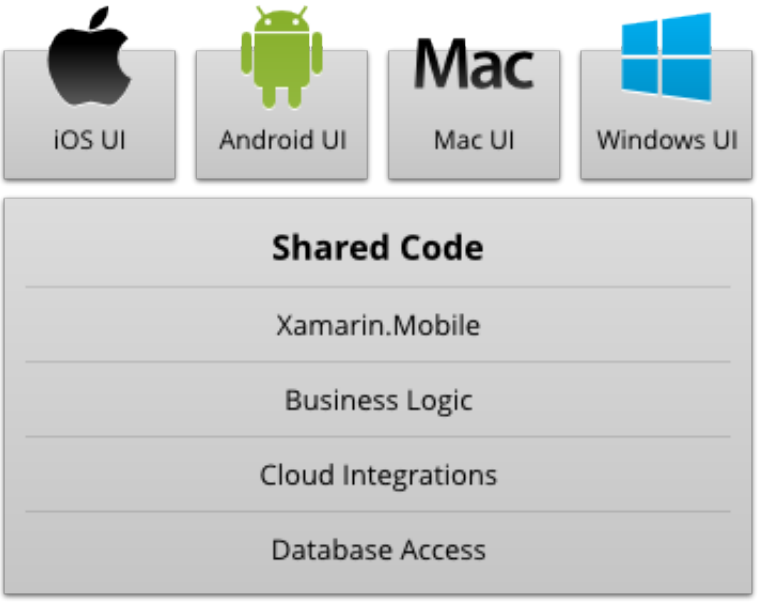

Figure 4.11: Code Reusing Principle [Xam13]

represent the UI. The rest of the layers should be UI independent and give the possibility to be used across all mobile platforms.

These can be achieved by designing UI of a mobile application in respect to the MVC pattern and creating the applications architecture by the principles of the MVVM or MVC approach. By applying these techniques, core functionality can be separated from the platform specific UI, and a maximum of code sharing can be achieved.

The MVC pattern was introduced in Section 3.2 and a description of the MVVM follows below.

[Xam13], [Mon13], [Xam13]

Model View ViewModel

The Model View ViewModel (MVVM) is similar to the MVC. It provides a separation between the logic of an application and its UI. The differences between the MVC and the MVVM are that the user interacts directly with the View. In MVC the input is processed through the Controller, and further there is a reference between the View and the Model. In MVVM the View is not aware of the Model. For the View its Model is the ViewModel. This is also the origin of its name. The ViewModel exposes the data from the Model and holds most of the logic of the View. Moreover, the ViewModel has no information of the View and therefore it can have a one to many relation with it. The View is aware of the ViewModel but however does not have information of the implementation behind it. [gee13]

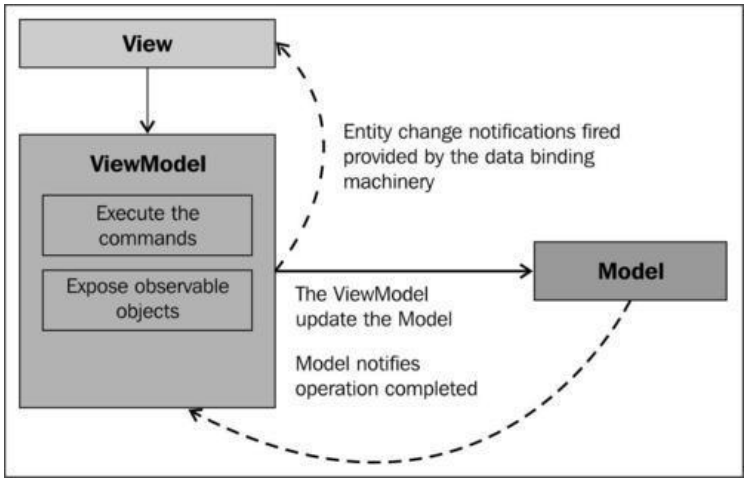

Figure 4.12: Interaction Between MVVM Components [DS12]

4.2.3 Application Architecture

In order to achieve maximization of code sharing the application has to have a well organized structure. Following object oriented programming principles should be obeyed.

1. Polymorphism
2. Separation of concerns
3. Encapsulation

Depending on the data type or class, functions, objects and variables can have multiple forms. In particular this means, that writing abstract classes brings forth the code sharing ability. That way multiple implementations can be supported and this can execute platform specific tasks and features.

An application should be divided in different sections. Each one of them addresses a discrete concern and their functionality should overlap as less as possible. Therefore, components, used in the application, should always perform only well defined and clear tasks. An API can expose this functionality to other classes.

Certain values of objects are protected of the influence of unauthorized other parts of the application. Therefore, objects behave like black boxes, in order to perform certain tasks the consuming part does not need to know how it is done, therefore, the implementation should be hidden. On the architectural level this means, that a simplified API should be provided to execute more complex tasks and simplify them in respect to other abstract layers. For example the UI code never interacts directly with the database, it is only responsible for displaying certain elements and retrieving user input.

[Xam13]

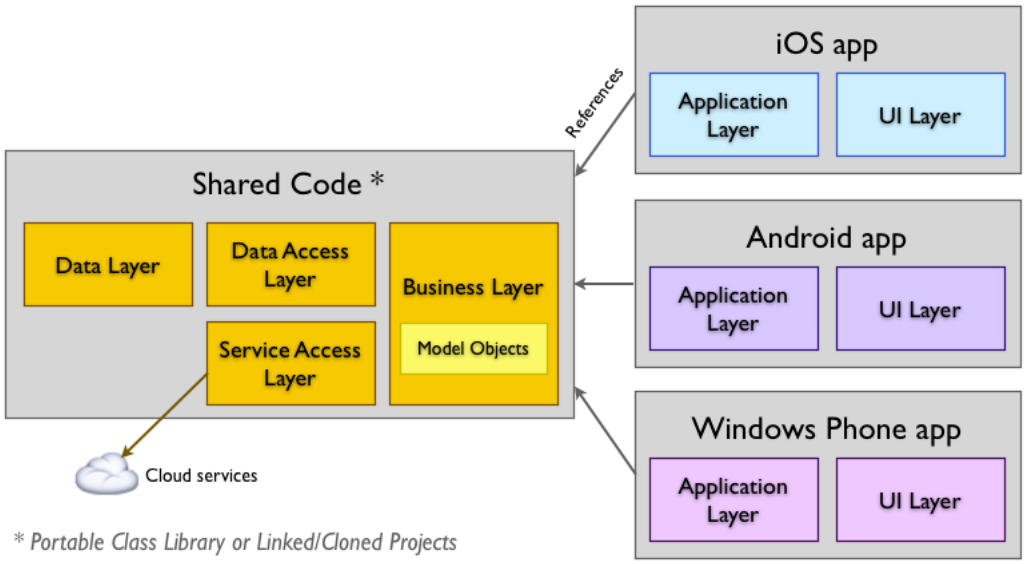

Figure 4.13: Conceptual Architecture [Xam13]

Typical Layers

Figure 4.13 shows a conceptual architecture that represents the different application layers in respect to the interaction between shared code and different mobile platforms.

Application Layer (AL) This part holds all the application or platform specific code that cannot be reused in other applications or platforms.

Business Layer (BL) Pure business logic is stored in here.

Data Layer (DL) The Data Layer represents a database. The storage mechanism behind it can be a SQLite or XML file.

Data Access Layer (DAL) This should provide the functionality to interact with the Data Layer. Methods should be provided like create, read, update and delete. They should be executed in a way that it is not required to know how the DL is actually implemented.

Service Access Layer (SAL) This layer is responsible to encapsulate network specific functionality. A simple API should be provided to achieve complex queries from web services like REST or JSON.

Figure 4.14: Illustration of the Xamarin.Mobile API [Xam13]

User Interface Layer (UI) The UI presents user interacting elements and holds their logic parts. It should only contain code that is responsible for this.

[Xam13]

4.2.4 Platform Abstraction

Xamarin Components

The Xamarin.Mobile .NET library is provided by Xamarin and provides a common API to access native features of a mobile device, as shown in Figure 4.14. It is abstracted from all different operation systems. This provides maximization of the code sharing.

The included extras of the library, at the time of this writing, are listed below:

Xamarin.Contacts provides access to the address book and contact details as phone, email and so.

Xamarin.Geolocation implements access to detailed geo-location data as latitude, longitude, speed, heading, etc.

Xamarin.Media includes the implementation of recording a video, taking pictures or picking one of them.

In addition, an exchange platform is provided where different .NET library components can be exchanged. They provide extra functionality to either one or multiple operation systems.

The collection includes already over 50 of them, whereas half of them are offered for free and the other half can be acquired.

[Xam13]

Cross Platform Libraries

There are a number of libraries that encourage the implementation of cross platform apps or games. They are based on different design patterns to help manage code sharing properly.

- MonoCross[6]
- MvvmCross[7]
- Vernacular[8]
- MonoGame[9]
- CrossGraphics[10]

4.2.5 Xamarin.Android

Xamarin provides the ability to develop Android applications by using the C# language. This is made possible by compiling it down to Intermediate Language (IL) and binding Android callable wrappers to it. The exact process is explained further along in this section. All necessary steps can be performed through the command line. However, it is advised to use an IDE. Applications can be created by the provided Xamarin Studio IDE or by the supported usage of Visual Studio 2012. Regardless the Android SDK and Java have to be installed.

Both development environments provide a visual designer tool. It can be used to create a user interface with a simple drag and drop of elements. The associated layout is saved in the XAML format that is used in the .NET.

[Xam13]

[6] http://code.google.com/p/monocross/
[7] https://github.com/slodge/MvvmCross/
[8] https://github.com/rdio/vernacular/
[9] http://monogame.codeplex.com/
[10] https://github.com/praeclarum/CrossGraphics

SDK Access and Usage

Xamarin.Android provides complete access to the Google Android SDK through namespaces in C#. This is achieved through binding the Android platform specific content to Android assemblies that are included with the application. This assembly library includes everything needed to access the Android API and communicate with the Dalvik VM.

```
protected override void OnCreate (Bundle bundle)
{
    base.OnCreate (bundle);
    var layout = new LinearLayout (this);
    layout.Orientation = Orientation.Vertical;

    var aLabel = new TextView (this);
    aLabel.Text = "Hello, Xamarin.Android";

    var aButton = new Button (this);
    aButton.Text = "Say Hello";
    aButton.Click += (sender, e) => {
        aLabel.Text = "Hello from the button";
    };
    layout.AddView (aLabel);
    layout.AddView (aButton);
    SetContentView (layout);
}
```

Listing 4.25: Example Hello World in Android [Xam13]

This means that Android applications can be created as usually, with the only difference that it is done in C# instead of Java. In Listing 4.25 an "Hello World" example from [Xam13] is displayed. Since both languages have a similar syntax it this should look familiar.

The user interface is created within the code in this particular example. However, it can also be produced through layout XML files. The handling of a button click is the biggest difference between this implementation and one in Java. In Android a new `View.OnClickListener()` with an overridden `onClick()` that handles the click would be created and set to the button to listen when it is clicked. Here a .NET event is handling a click event. Either an anonymous method or a lambda expression can be used to perform the necessary operations. The rest of the code example illustrates basic layout definition as creating a `LinearLayout` and setting it to the current Activity. Running this example in an emulator would result in a button being displayed. Clicking it some text appears above it, as displayed in Figure 4.15.

Regardless the integration of the Android API and the Java language there are limitations, described in at the end of this section.

[Xam13]

Figure 4.15: Conceptual Architecture [Xam13]

Figure 4.16: Application Package Size Without Linker [Xam13]

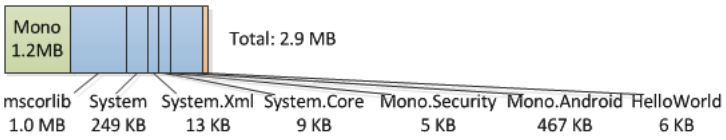

Figure 4.17: Application Package Size Reduced by Linker [Xam13]

Build and Compilation

This section explains how the build and compiling process is performed. In Android resources are stored in a specific folder and references to them are made by resource IDs. Through the compilation process a resources class is generated that contains all references to the included resources.

A similar process is performed in Xamarin.Android. All resources are stored in a particular folder and are as well referred to by IDs. As a pre build step, the `Resources.designer.cs` file is generated that holds all Android resources IDs.

The C# code is compiled down to .NET assembly. Depending on its references Android callable wrappers are created. All this is packaged with the Mono VM and included in the final `apk` file.

In order to establish just in time compilation for Android everything is included inside the package. The application package of a "hello world" example is displayed in Figure 4.16 and includes the Mono runtime and the necessary Base Class Library assemblies. These are mscorlib, System, XML, System.Core, Mono.Security and Mono.Android. However, the size of the application results in almost 16 MB for a simple code that is just 6 KB. The problem is that all included library assemblies require a lot of storage space. Unfortunately, they are necessary to run the application, but they also hold functionality that the app might not actually use. Therefore, it would be suitable to exclude all parts that are not used from the assemblies. For this reason Xamarin provides a process called "Linking" that will be executed when an application is build for distribution. This process examines the app and removes any unused code from the assembly components. The process is similar to the Garbage Collection one, instead of objects, it analyses the code. After running the linker on the "hello world" example, the size of the application reduces enormously. As illustrated in Figure 4.17, all unused functionality is removed and the end size of the app is 2.9 MB.

The release package can now be deployed to the device. The application runs side by side with the Java Dalvik and interacts through the Java Native Interface (JNI) with the native parts. The details of the Java integration are explained further below.

The linking process is time consuming and therefore unsuitable for debugging the application. Moreover, copying and installing packages on Android can be relatively slow as well. Because of this, Xamarin offers a compilation possibility that is optimized for speed of deployment and not the size of it. Deploying applications for debugging will result in copying two packages the first time, Shared Runtime and Shared Platform. The first one contains the Mono runtime and the second one includes the Android API assemblies. Copying them to the device might take some time, since their size is large and normally above 10 MB. The actual application as illustrated in the "hello world" example, has a much smaller size and is deployed quickly after that.

The core components are only copied the first time. On account of this only the application part, since this holds all of the code and its changes, is copied to the device after the first deployment.

[Xam13]

Architecture and Integration

An Android application created with Mono is not executed through the Dalvik VM directly. The execution of it is performed by the Mono environment that runs side by side with it. Both of them are runtime environments written in the C language and expose an API to gain access to the Linux Kernel below them. This means that the direct access from the Mono environment to Linux system components is possible. However, in Android most of them are only exposed through the Dalvik Java API.

In summary it can be said that native components can be accessed either through the exposed Android libraries which provide a bridge to the Java API exposed by the Dalvik VM or through the classes of the .NET framework that provide access to system components.

JNI The JNI is a program interface that enables Java code running in a JVM to access its system host languages and determine the integration with it. In short the Java code can call any other system parts as libraries written in C, C++ and be called by them.

The Java integration is accomplished by the heavy usage of the JNI that has the ability to interact with native parts. Android libraries are binded through its usage. For example, all bindings for the `java.jar` file are the `Mono.Android.dll` that is always included in the final application.

Android Callable Wrappers (ACW) This is a JNI bridge used by the Android runtime environment to call any code from the Mono side.

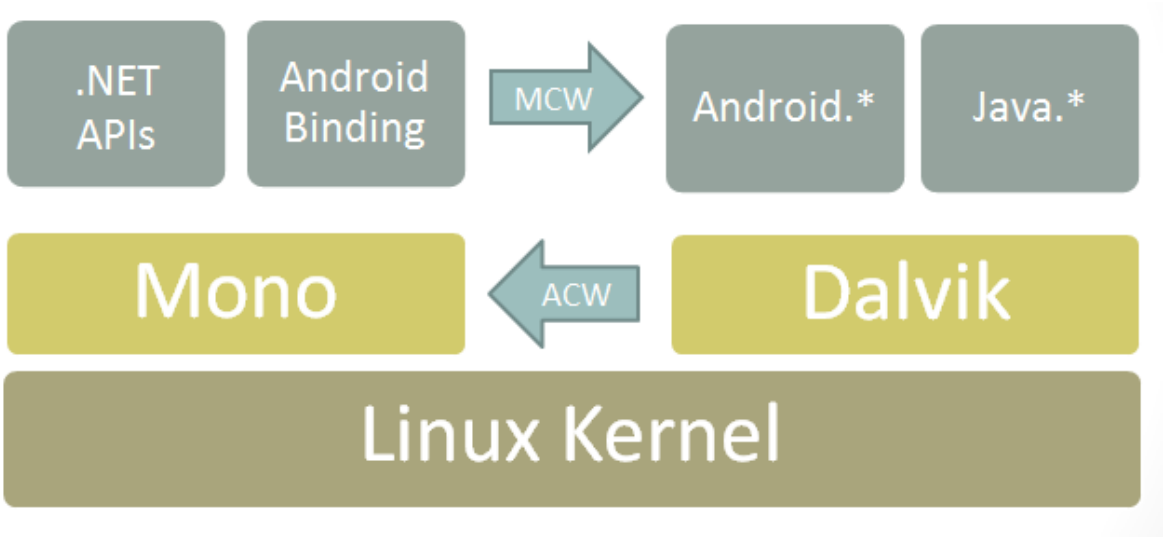

Figure 4.18: Xamarin Android Architecture [Xam13]

Managed Callable Wrappers (MCW) This is another JNI bridge that manages the communication the other way around. It is responsible to invoke Android or Java code when it needs to be done. In order to accomplish this the wrapper holds a Java global reference all the time to ensure the mapping between the instance on the one side and the one on the other side. To make sure instances on the Java side are removed, after they are not needed anymore, `Java.Lang.Object.Dispose()` can be called. This will remove any existing mapping and the Java instance can be collected by the Garbage Collector.

This being said, the architecture can be described as in Figure 4.18.

So far it was explained that the executed Android application runs on a separate VM and uses JNI bridges to establish the communication between two runtime environments and their respective programming languages and libraries.

In addition, this means that none of the existing third party Android library components as mentioned in Section 2.7.2 can be used out of the box. Any Java library that should be used with Xamarin.Android needs to be "ported" to it. In particular Xamarin holds three possible solutions.

The first one is to create to an existing `jar` library an appropriate Java binding library. Xamarin.Android provides a tool that can perform this automatically. It is called "Java Bindings Library". This will created the MCW part for the library. After this is done, the Java library code can be invoked by the Mono side.

The second approach is to create a JNI bridge manually. This is done on a much lower level and is therefore harder to do. However, doing this will give full control of the access and control options of the Java methods.

The last possibility is to port the Java code to C#. This can be performed manually or automatically by using a converter tool as Sharpen[11] that can be found on GitHub[12].

[Xam13]

Limitations

In Xamarin.Android there are several limitations in regard to the usage of Mono. Without going into detail, it can be said that dynamic languages like IronRuby or IronPython are not supported. ACW cannot be created for them during the compilation process.

Third party libraries need to be binded either manually or by the usage of the Java binding library. Even when making use of the possibility to automatically doing this, the process is still time consuming.

[Xam13]

4.2.6 Xamarin.iOS

Xamarin.iOS enables creating iOS applications in the C# language. As mentioned before Xamarin provides an IDE that can be used under the Mac OS to achieve this. Visual Studio 2012 can also be used to do this by using a provided plugin. However, a Mac is still required for the build and licensing process. In particular this means that Windows can be used to develop the application, the rest of the work needs to be done on a Mac that is available through the same network. Regardless of this, the layout design still needs to be performed with the Interface Builder from Apple or can be done programmatically.

Build and Integration

The compilation is driven by the "mtouch" command. This is used to compile an application and deploy it either to a device or a simulator. This thesis is focused more on the Android part therefore the process will not be described in detail.

However, it can be said that the C# code is compiled in Ahead-of-time (AOT), since Apple does not allow just-in-time compilation (JIT) to Advanced RISC Machines (ARM) assembly language. During this process all the code that the JIT compiler would generate is produced and, as in Android, a linker removes classes that are not used to reduce the size of the application.

The core library of Xamarion.iOS uses an interop engine as a bridge between the different language environments. However, all necessary libraries are binded and can be accessed from the C# through certain namespaces. That represents something like an API to all native parts. The available namespaces are:

[11] http://community.versant.com/documentation/reference/db4o-8.0/net20/Content/sharpen.htm
[12] https://github.com/slluis/sharpen

- MonoTouch.ObjCRuntime: This binding can be used to bridge from C# to Objective-C.
- MonoTouch.Foundation: This provides access to the Objective-C Foundation framework.
- MonoTouch.UIKit: This contains a one to one mapping to all UI elements of Cocoa-Touch.
- OpenGLES: This exposes the CoreGraphics functionality.

[Xam13]

Usage

The usage is quite similar to the traditional one. Applications follow the MVC pattern and are split up in their components. The obvious difference is that the development is performed in the C# language. This brings features of the .NET Base Class framework, as mentioned earlier. Besides, this and the ability to reuse code, the fact that developing can be performed without Objective-C knowledge is for many the biggest benefit. Learning the syntax and the usage with all the benefits of the Objective-C language can take some time for developers that do not have experience with it. There are relatively more developers that are more comfortable with the C# language.

Either way there is a learning curve for everybody. The one part has to get used to performing tasks using the C# language while using Objective-C libraries, provided through an API and the other part has to get to know these libraries.

However, the graphical interface if not created programmatically still requires to be performed through the provided Interface Builder from Apple, included in Xcode.

[Xam13]

5 Testing

5.1 General

5.1.1 Agile Testing

First, a brief look needs to be taken over the traditional developmental process, the Waterfall model. This defines a series of phases that are all proceeded sequentially through the developing. After a phase is finished, another one can be started. It is however allowed to go backwards and make some changes based on acquired feedback of the current phase. A clear benefit of this model is that it focuses at the beginning on the specification analysis and design structure. This can provide a clear vision and structure for a software project.

Generally in this model the test phase is assigned at the end, right before the release of the software. In theory there should be as much time for testing as for developing provided. For the most part this is an idealistic scenario. Whereas in reality in a lot of software projects, testing gets squished at the end. Developers are more focused on bug fixing and finishing up the code.

The Agile developing methodology is incremental and iterative. It focuses on planning and realizing small parts of a project, instead of defining it and its details at the beginning. First a basic set of deliverables are fulfilled, followed by developing and testing another set of features. It is all performed in subsequent iterations. In particular this means, that a bit of code is built and tested after that. In this case, programmers and testers always work on the same feature until it is finished. The time period of an iteration differs and can be from one week to a month, usually depending on the feature set or the project itself. Besides the functionality testing in each iteration other factors are controlled as well. For example, other critical factors are tested as well, as usability, security and performance. In summary each iteration realizes a feature completely in respect to all factors.

[CM13], [CG08]

5.1.2 Test Driven Development

The TDD process is suppose to drive the development of software, hence its name. In particular this means, that tests are written before production code. Generally this results in thinking of the implementation before writing it and thereby defining tests in advance. In short, tests are written for non existing functionality, followed by the implementation of

it. However, this is always performed in a loop for a minimalistic operation code until the complete functionality is reached.

Robert C. Martin ([Mar09]) therefore defines following laws of TDD that should be considered:

Three Laws of TDD:

- Your are not allowed to write production code until you write a unit test that is failing.
- Your are not allowed to write more of a unit test than it is sufficient to fail.
- Your are not allowed to write more production code than it is sufficient to pass the failing test.

With this approach the process of writing testing and functionality code is closely connected and the importance of them is equally valued. Consequently tests have to be fast and clean. In addition Robert C. Martin ([Mar09]) characterizes five rules that should be carried out.

F.I.R.S.T.

Fast: Tests should be fast. If test are not fast, you will not run them frequently.

Independent: Tests should not depend on each other. It should be possible to run tests in any order. This way a downstream of failure can be avoided.

Repeatable: Tests should be repeatable without any reliability on the testing environment. Test should perform and have the same result in any chosen environment.

Self-Validating: Tests should either fail or pass. You should not need to compare the output with some other data to determine if the test passed or not.

Timely: Tests should be written at the appropriate time. This is right before you write the production code to make them pass.

It might not be reasonable to practice TDD in smaller projects that have just one or maybe two developers. However, in bigger projects with more developers it provides confidentiality to a single member to make changes of the code. Since all tests are performed after a change, the observation of their results shows if other functionality is being affected or not.

Unfortunately in the developing process of mobile applications TDD can be hard to accomplish, depending on the application. Especially tests that deal with the UI are difficult to implement. Usually, tests are executed automatically with the help of a framework built for such tasks. In conclusion this framework needs to provide support for the testing process of the production code. Generally, the support for UI elements is limited. If the application uses any customized UI elements, the testing frameworks usually have to be extended as well.

Moreover, certain graphical interactions and animations are hard to be described and therefore tested. However, using TDD definitely requires more development time, but it guarantees that individual components and their inclusion in to the project have already been tested. TDD adds certainly a great value to the agile development process.

[Mar09]

5.1.3 Behavior Driven Development

Behavior-driven development (BDD) was developed by Dan North in order to overcome encountered issues of the TDD usage. In conclusion, it can be said that BDD is an extension of TDD. Tests are described more naturally in sentences and therefore explain the behavior of a test. This makes it especially much easier for non developers to understand them. It adds even advantages for developers. For example, a failing test describes what should have happened and allows to analyze if an error occurred in a specific sector of the software or if the general behavior of it was wrong. Sometimes developers concentrate on one functionality only, forget to think of the software as a whole and therefore make mistakes. Such bugs can be presented by using BDD in a logical and easy to understand way. Test methods are described in complete sentences and in conclusion can be written much clearer. [Dan13]

5.1.4 Continuous Integration

A Continuous integration (CI) system builds an application and runs its tests each time a source code change was committed. This brings all benefits of TDD. This maintains to always have a stable and compilable application. It can be made sure that added changes do not affect other sections of the software. Above all the CI system should differ between two tests suits. One that can be completed fast and one that might take a longer time period to finish. The one first should cover a wide range of the code and focus on important functionality that can be tested fast. The second one can concentrate on testing a wider range of the code and should include tests that take a long time. This way it can be guaranteed that tests run fast during the daily development activities. For example, longer running tests can be executed over the night and ensure the quality. A popular open source CI solution is called Jenkis. Its setup is easier and provides a graphical overview of all results.

5.2 Testing under Android

The Android testing framework is based on JUnit. That means non Android classes can be tested with normal JUnit tests that only require a JVM. However, Android components cannot be tested on it. As mentioned in Section 2.3.3, every Android project contains a `android.jar` file that holds Android classes needed for the development process. However, this `android.jar` file does not actually include the complete Android framework and its classes per se. It contains necessary stub methods, types etc that are needed for the compilation of the app. This `jar` file is not actually included in the app. When the app runs on an Android device the `android.jar` from the device itself is being used. Therefore, running tests on Android components need to be done on an actual device or at least an emulator. This results in a longer execution time of tests and can be especially time consuming when there are a lot of them and that can happen quickly performing TDD. Some approaches to solve this problem are explained in Section 5.3. [And13l]

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3    package="com.example.app.test" android:versionCode="1"
4    android:versionName="1.0-SNAPSHOT">
5    <uses-sdk android:minSdkVersion="8"/>
6
7    <application
8    android:icon="@drawable/icon" android:label="@string/app_name">
9      <uses-library android:name="android.test.runner" />
10   </application>
11
12   <instrumentation android:targetPackage="com.example.app"
13     android:name="android.test.InstrumentationTestRunner" />
14
15 </manifest>
```

Listing 5.1: Example of an AndroidManifest.xml File of a Test Project [Mil11]

Listing 5.1 shows the manifest file of an Android test project. The tag `instrumentation` defines which test runner is used and the `targetPackage` of the application being tested.

5.2.1 Assertions

Assertion methods are called to assert if some condition is valid. If the condition results in false, the assertion will fail and display a message. JUnit Assert class contains some useful methods that can be used like `assertEquals`, `assertFalse`, `assertNotNull`, etc. Most of them are self-explanatory and most of the time they are going to be enough to test basic core functionality. However, when it comes to the Android Interface one will start using the `ViewAsserts` class that contains more suitable methods. With them the exact position of Views and their relation to each other can be tested. The visibility of the View on the screen can be inspected with it. Performing this kind of test on multiple devices with different screen sizes and resolutions can be very helpful to detect misbehavior. As a bonus Android also provides the class `MoreAsserts` with a handful of more expanded assert methods. For example, the method `assertMatchesRegex` can be used to check the correctness of a String.

5.2.2 Mocking Object Classes

Mocking object classes are used to isolate the tests from the real system. Some mocking classes are already provided and just have stub methods, enough to test them. If some of their methods are being called, an `UnsupportedOperationException` is being thrown. For example some important ones that are included are: `MockApplication`, `MockContext`, `MockContentProvider`, `MockCursor`, `MockPackageManager` and `MockResources`.

5.2.3 Android Testing API

The Android Testing API supports JUnit 3 code style and extends it with an instrumentation framework. Instrumentation testing is basically testing the user interface by interacting with it. The Android specific testing classes are extended from the JUnit `Assert` and `TestCase` classes. Different methods are provided giving the possibility to check permissions and track memory leaks, besides the typical `setUp()` and `teardown()` methods. For each Android component there is a suitable testing class.

Activity

The Activity component is probably the most important one, since it is representing the UI and interacts with the user, as described earlier. Therefore, testing it properly is even more important. The main class used for that is the `ActivityInstrumentationTestCase2`. It provides functionality to test a single Activity. It is based on the main class `InstrumentationTestCase`. Moreover, there are two other important subclasses, `ActivityUnitTestCase` which can be used to test an isolated Activity and `SingleLaunchActivityTestCase` that can be used for Activities that are started in non standard mode. Instrumentations are used to trigger lifecycle methods of an Activity, this helps to check its behavior. Moreover, mock objects can be created with instrumentations to isolate the test environment and check dependencies. Interactions with the user interface are also done by instrumentations. So what should be tested? Lifecycle events of the application should be tested. For example is the application's state saved correctly if `onDestroy` or `onPause()` is called; or even if everything is initiated correctly in `onCreate()`. Moreover, check the correct behavior of events triggered by the user as `onCLick()`. Certain keystroke sequences can be triggered by using the `sendKeys()` method. Buttons can be pressed by first obtaining them using `findViewById()` and `performClick()`. Checking if the Activity handles user input correctly is significant. If Intent Filters for an Activity are specified in the manifest, they can be tested by sending mock Intents to the Activity. Tests regarding the display size, resolution and orientation should be performed. It should be verified that the content is displayed correctly regardless of the devices screen size and resolution.

[And13l], [Mil11]

Content Provider

The functionality of an implemented Content Provider can be tested in an isolated environment by using the `ProviderTestCase2` class. By using this class the typical methods that a Content Provider offers can be tested. The class itself provides two mock object classes `IsolatedContext` and `MockContentResolver` that offer the necessary functionality and have stub methods for everything else. If the Content Provider can be accessed by other applications, all exposed methods should also be tested. Other operations performed by the Content Provider, besides providing some content, need to be checked as well. These could be some calculations or elimination of data.

```
1 public void testSimpleQuery() {
2     ContentProvider contentProvider = getProvider();
3     Uri uri = Uri.withAppendedPath(MyProvider.CONTENT_URI, "dummy");
4     final Cursor cursor=contentProvider.query(uri, null, null, null, null);
5     final int expected = 10;
6     assertEquals(expected, cursor.getCount());
7 }
```

Listing 5.2: Simply Query Test [Mil11]

In Listing 5.2 an example is depicted of a simple query test. Usually obtaining a reference to the Content Provider by calling `getProvider()` is done in the `setUp()` method so the same provider can be used in multiple tests. However, in this test the returned cursor is expected to have two rows. Since the context is just a mock, adding or deleting elements would not have any effect on the real content.

Application and Service

Android Services can be tested by using the `ServiceTestCase` and Application classes can be tested with the usage of the `ApplicationTestCase` class. Here it is important to test the lifecycle methods again, and in addition any other functionality performed inside.

[Mil11], [And13l], [Sad11]

5.2.4 Monkey

The Monkey is a shell application provided by the Android SDK. It runs an Android application on devices or emulator and generates a certain amount of random user events.

```
1 adb shell monkey -p com.example.package -v 1000
```

Listing 5.3: Example Usage of Monkey

In Listing 5.3 an example call of Monkey is illustrated that tests the application with the package `com.example.package` and performs 1000 random events. That is a great way to execute a few random stress tests to see if any error is going to occur. [And13r]

5.2.5 Monkeyrunner

The Monkeyrunner provides an API with which user input programs be can written in Python and can be performed on a device or an emulator. One of the most used features is taking screenshots of the user interface and storing them. [And13m]

5.2.6 Strict Mode

Strict Mode was introduced in Android API level 9 and can be turned on in code to detect faulty operations. Certain policies can be set for example to detect network operations performed in the UI thread or to find memory leaks. However, this is just a developer tool and should always be turned off before deploying an application to the Google Play Store.

```
1  if(DEBUG && Build.VERSION.SDK_INT >= Build.VERSION_CODES.GINGERBREAD){
2      StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder()
3              .detectAll()
4              .penaltyDialog()
5              .build());
6      StrictMode.setVmPolicy(new StrictMode.VmPolicy.Builder()
7              .detectAll()
8              .penaltyLog()
9              .penaltyDeath()
10             .build());
11 }
```

Listing 5.4: Example Turning Strict Mode On

Listing 5.4 shows how to turn on the Strict Mode. `DEBUG` is a boolean that indicates the debugging modus is on if it is true. Since it was introduced with Gingerbread, there is also a check to confirm the device is running at least Gingerbread. `.detectAll()` detects as it says all policies, `.penaltyLog()` stands for logging the penalties and `.penaltyDeath()` stands for killing the application if a penalty occurs.

5.3 Alternative Android Testing Environments

5.3.1 Robotium

Robotium is an extension for the Android testing framework that allows easy black box testing. It is licensed under the Apache License 2.0 and supports Activities, Dialogs, Toasts, Menus and Context Menus. Using Robotium allows to create tests without knowing the source code. Since it is more like an add-on, the tests are still extending from the `ActivityInstrumentationTestCase2`. In order to use Robotium, the `jar` library needs to be downloaded from the project's website and included in the testing project's dependencies.

As displayed in Figure 5.1, Robotium is extending the tests by wrapping the instruction against the testing framework and extending the functionality of the main test. Since it is a black boxing framework, there is no need to know names of the Views. All visible Views and elements on the screen are being indexed. Therefore, referring to them by their index is possible.

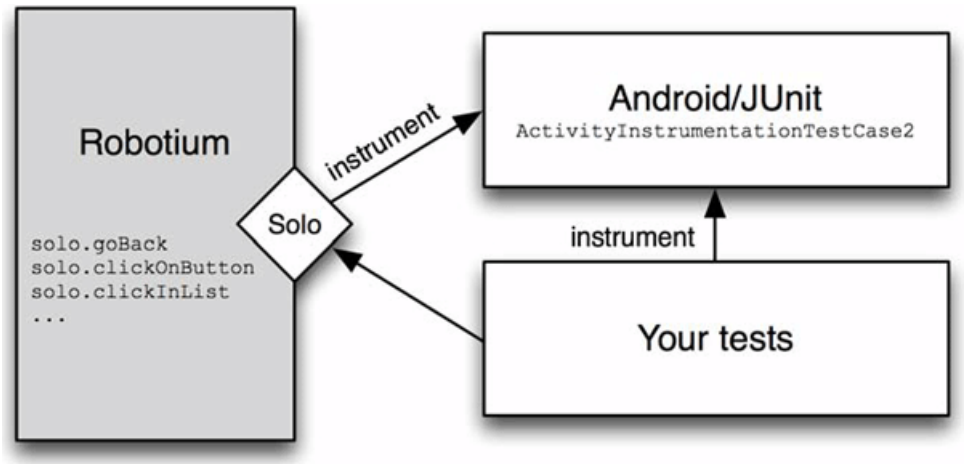

Figure 5.1: Android Robotium [CGK11]

```
1  public class SomeClassTest extends
2    ActivityInstrumentationTestCase2<SomeClassActivity> {
3    private Solo solo;
4
5    public SomeClassTest() {
6      super(SomeClassActivity.class);
7    }
8
9    public void setUp() throws Exception {
10     solo = new Solo(getInstrumentation(), getActivity());
11   }
12
13   public void testAdd() {
14     solo.enterText(0, "10");
15     solo.enterText(1,"2");
16     solo.clickOnButton("Divide");
17     assertTrue(solo.searchEditText("5.0"));
18   }
19
20   @Override
21   public void tearDown() throws Exception {
22       solo.finishOpenedActivities();
23   }
24 }
```

Listing 5.5: Example Robotium

The test looks almost like any other as shown in Listing 5.5. The interesting part here is the Robotium `Solo` instrument's object. A new `Solo` object is created with the current `Instrumentation` in the `setUp()` method. From here on easy readable methods can be executed and Robotium will take care of the rest. Since Robotium does not know how

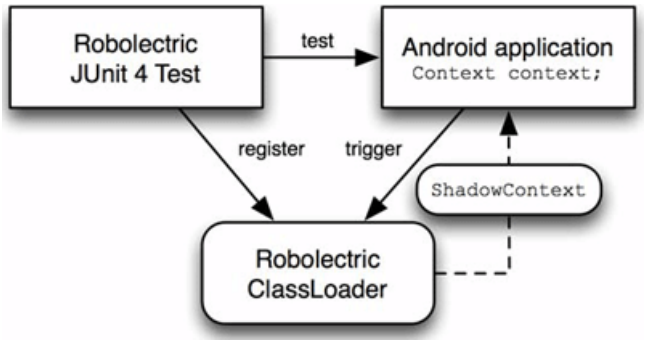

Figure 5.2: Android Robolectric [CGK11]

the source code or the actual program looks like, it searches for all elements and gives them an index. Therefore, when putting some values in a text field like in the example `solo.enterText(0, "10")`, the first number refers to the index, starting with zero. However, the tests do not require an explanation since they are self-explanatory. Test is performed by putting two numbers in two separate text fields, clicking the button "Divide" and searching for another field with the value "5.0" that asserts the calculation being valid. As stated, writing tests can be done quickly and easily. Writing traditional tests can become more complicated and time consuming as the implemented functionality that should be tested, itself. Unfortunately using black box testing results in longer execution times of test cases.

5.3.2 Robolectric

Robolectric is another great open source project that helps to run tests fast on a normal JVM. As explained in Section 5.2 the `android.jar` used while developing is full of stub methods and to be able to actually run tests against it, the Android project needs to be run on a device or emulator that provides the actual Android code. Robolectric redefines some of the Android framework classes and their methods by providing so called shadow classes for them. Figure 5.2 indicates that tests are registered with Robolectric, so it knows which classes need to be provided.

Shadow classes behave the same way normal Android classes would, but they are purely implemented in standard Java code. Therefore, executing them on a normal JVM is possible. They do not actually provide a lot of functionality, they are also basically just mock objects. For example, the `ShadowActivity` class provides the `findViewById` method and the inflation of it. Rendering views or layouts are not actually supported, they just provide enough functionality so they can be tested. Moreover, all methods in a shadow class that are not implemented, will be rewritten to do nothing.

So lets point out the differences between a test written for the Android test framework and one for Robolectric. First of all, Robolectric uses JUnit4 that uses Java annotation. Therefore, it registers a JUnit4 test runner via `@RunWith(RobolectricTestRunner.class)` above the test class. Testing methods do not need to start with "test" as they do in JUnit3 and are registered via the `@test` annotation above them. Besides not using actual Android classes

Testing Method	Type of Test	Test Runtime
Android Instrumentation	Integration test	5.629 sec
Standard Environment	Unit test	0.69 sec
Robolectric	Unit test	1.16 sec

Table 5.1: Comparison Android Test Execution Times [ZMEQ11]

but their shadows instead, user interactions cannot be triggered per se by pressing a button. Instead Android methods, for example `onOptionsItemSelected` that would be triggered by such an user event, need to be called. However, the end results are impressive, the tests run fast on the local JVM.

Table 5.1 shows a comparison of an example Android application from [ZMEQ11]. As illustrated Robolectric tests run significantly faster than Android integration tests. However, Robolectric also has a downside, there are not even nearly all Android framework classes covered by shadow classes that could be invoked running tests. Unfortunately implementing an own shadow class needs to be done in the Robolectric source code and this can become quickly confusing.

One approach is to keep the source code as clean as possible, Android specific code should only be used if necessary, by trying to separate it from the pure Java code. Then, pure unit tests can be performed on the code part that does not use the Android API at all. Robolectric unit tests can be run on the source code that has some small Android dependencies and can be satisfied by shadow classes. The Android specific code may be run at the end with the Android test framework.

[CGK11], [Mil11], [BHH+12]

5.4 Testing Hybrid Applications

As described in Section 4.1 the Cordova Framework transforms a web app into a native app. The web application is displayed through the device's internal web view, and additional native features are supported through a JavaScript API.

Unfortunately, testing applications created with the Cordova Framework is not as easy as testing pure web apps or pure native apps. The difficulty is that the web app is displayed through the device's web view. This means that debugging and testing needs to be done on the device. Besides, the fact that doing this on the device is complicated, it would also be time consuming. However, there are still a view approaches that can be used to make debugging easier.

5.4.1 Usage of Desktop Browser

One technique is debugging it in a desktop browser. When doing this, it needs to be considered that not all mobile OS use the same web engine. Android, iOS, Bada, BlackBerry 6+ and

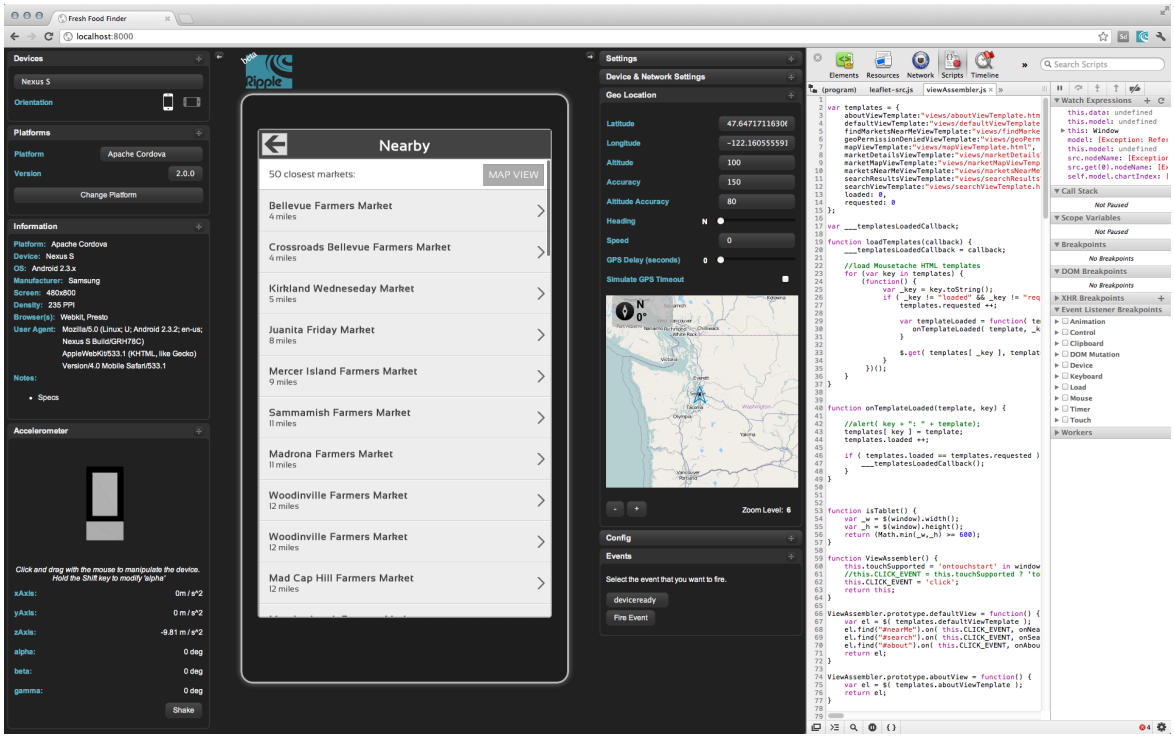

Figure 5.3: Ripple Emulator in Chrome [And13a]

webOS use the WebKit engine and can be tested with the Apple Safari and Google Chrome browser that are compatible to it. Windows Phone uses the same web engine as the Internet Explorer and should therefore be tested with it.

All these browsers come with a built-in developer tool. However, to simulate mobile environments some tweaks need to be made. For example, a desktop browser will perform click events instead of touch events. In the Chrome developers tools override options can be set. Among other an "Emulate touch events" can be activated. This will add a touch event in addition to the mouse events. Moreover, there are options to simulate another user agent and a different resolution. Besides, this the Chrome browser should be opened up through the terminal with `--disable-web-security` and `--allow-file-access-from-files` as addition arguments in order not to have any problems opening local files or security issues. After doing all this debugging a web application can be done with the built-in developer tools. This means that non Cordova related functionality can be tested. But, what if certain native features should be tested?

Apache Ripple

The Apache Ripple simulates various mobile environments such as Apache Cordova and BlackBerry WebWorks. It can be used for free and is licensed under the Apache License

2.0[1]. In order to use it with Chrome, the Ripple Emulator (Beta)[2] extension needs to be downloaded and installed. Ripple emulates Corodva events and API calls. This means that Cordova applications do not need to be deployed to the device in order to test them. They can be run inside the Ripple emulator and events like `deviceready` will be triggered automatically. Moreover, all Chrome debugging tools can be used as resource inspection, the JavaScript console, and so on. This of course cannot eliminate testing it on a device but surely provides a configuration that is fast and familiar to web developers. Testing native plugins is not supported. Instead a JSON string can be passed through and a callback function, either success or fail, can be chosen.

In Figure 5.3 the Ripple Emulator is depicted. It provides several panels with different options to simulate certain device events.

Device A number of pre-configured devices can be selected. Depending on which device is selected it will simulated its specifications as resolution, user agent and so on. The orientation of the device can also be changed here.

Platform Here the emulated platform can be chosen. It supports PhoneGap 1.0, Cordova 2.0, various versions of WebWorks and mobile web configuration.

Information Here all the important information of the simulated device as user agent, pixel density, resolution and so on, are displayed.

Accelerometer It represents the phone as a 3D device. The orientation of it can be manipulated, and it can be rotated in all directions. Under that all the values of the alpha, beta, gamma and all three axises are listed. An option is also provided for shaking the phone.

Settings In this section configuration of the Ripple emulator can be made. Enabling/Disabling the cross domain policy is an important one of them.

Device and Network Settings The network connection that should be simulated can be selected here.

Geo Location All the values regarding the geographical position as latitude, longitude, speed, GPS tweaks and so on can be adjusted here.

Config This represents the `config.xml` file of the Cordova project.

[1] http://ripple.incubator.apache.org/
[2] https://chrome.google.com/webstore/detail/ripple-emulator-beta/geelfhphabnejjhdalkjhgipohgpdnoc

Events This panel provides the ability to fire several Cordova specific events like `deviceready`, `backbutton` and so on.

PhoneGap Desktop

Is a project published on GitHub under the Apache License 2.0. PhoneGap Desktop gives the ability to mock all API calls, and return defined values. It can be used fairly easy. The original `cordova.js` file should be replaced by the `phonegap-desktop.js` file. This represents the mocking library and loads all data from a JSON file in a memory object. Now, API calls will return exactly this data. The JSON values can be changed without a haze to project specific ones. [Git13d]

5.4.2 Remote Debugging

Using a mocking library like PhoneGap Desktop or an emulator like Ripple definitely is helpful and boosts the development work flow. But still, there is no way around testing on an actual device. The complexity is that web views inside a by Cordova created application are like a black box. They are mostly isolated, and important debugging information cannot be returned back directly. Therefore, remote debugging is used to get to this information.

Safari Remote Debugging

Debugging applications with the Safari developing tools that run on the iOS simulator was made possible with the release of iOS 6. In order to do so the application needs to be started in the iOS simulator. After that the Safari browser has to be opened and the developing option "iPhone Simulator" is to be chosen. Remote debugging on a real device is pretty much the same. The device is required to be connect through USB. Before running the application the Web Inspector is enabled on the iOS device under Settings, Safari. After that the application can be run on the real device and in the Safari browser on the computer the real device is to be chosen. Now remote debugging applications are the same as any other web based ones. The tools included in Safari are as good as the ones from Chrome, and the functionality is basically the same. Breakpoints can be set, resources can be watched, the JavaScript console can be used and so on.

Weinre Remote Debugging

Weinre is licensed under the Apache License 2.0 and stands for WEb INspector REmote. The black box problem as mentioned above is solved here by including a JavaScript code that will report back to the Weinre server. Now, the Weinre server can be opened up and the DOM elements will be presented. They can be changed and manipulated directly. Weinre can either be run on a local machine or through `http://debug.phonegap.com`. In order to save transmission time and keep project information save, it should be run locally. Unfortunately it does not support JavaScript source code debugging.

WebKit Remote Debugging

BlackBerry BlackBerry[3] supports remote debugging by starting the Web Inspector on the device. The device needs to be either in the same network as the desktop computer or be connected to it. After that the displayed web content can be retrieved by calling the IP address of the device using the 1337 port. Now WebKit based browsers as Chrome or Safari and its developing tools can be used.

iWebInspector The iWebInspector[4] provides an unofficial way to perform remote debugging on iOS 5 devices. After installing and setting it up, any web app or `UIWebView` can be debugged remotely by opening the device's IP address under the port 9999. However, this is more like a workaround on the older iOS version and does not work on the new Mac OS 10.8.

Android As of right now a real remote web inspector is only available for the Chrome[5] app. This means it is not working for Cordova, since it uses a `WebView` to display the content.

5.4.3 Web Testing Frameworks

Web applications packed into a native app have certain testing and debugging restrictions. However, they can still be tested with consideration to the solution approaches explained above and can either be using a framework for the creation of the web application or not. An application build with Cordova, can on the one hand use JavaScript for its business logic or on the other hand interact with the Cordova API. With all web application being heavily based on JavaScript, there are already established frameworks that support testing them. In addition to pure testing frameworks there is a variety of test runner frameworks. They concentrate on executing test suits automatically on one or multiple browsers. Whereas all these frameworks work well on pure mobile web applications, testing hybrid applications like the Cordova ones, reaches the limits of them. Therefore, not all of them will be covered by this thesis.

Jasmine

Jasmine is a BDD inspired standalone JavaScript testing library. It is distributed under the MIT license and is available on GitHub[6]. The main focus of the library is that it can be used without any other requirements. It is purely implemented in JavaScript and is independent of the DOM, browser or other frameworks. Therefore, it is very suitable to perform tests on Cordova/PhoneGap-based applications.

[3] http://docs.blackberry.com/en/developers/deliverables/38982/
[4] http://www.iwebinspector.com/
[5] https://developers.google.com/chrome-developer-tools/docs/remote-debugging
[6] https://github.com/pivotal/jasmine

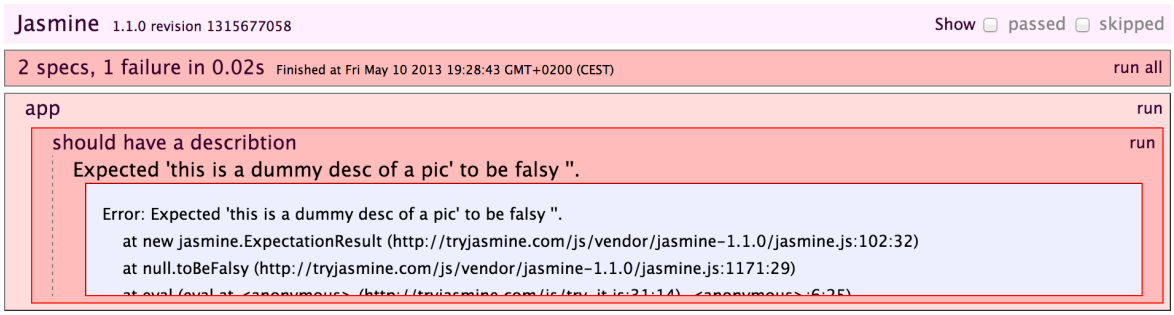

Figure 5.4: Jasmine Output Results

```
describe('app',function(){
  it('should have a name',function(){
    expect(gallery).toBe('My Gallery');
  });
  it('should have a describtion',function(){
    expect(describtion).toBeFalsy('');
  });
});
```

Listing 5.6: Example Jasmine Test

Jasmin tests are organized in a `describe()` method, this groups tests together, and in a `it()` method, which actually runs the test. This way tests can be written in the "it should be or not be ..." form and is becoming self-explanatory. A suitcase of tests are called specs, this further shows that it is a BDD oriented library.

Performing the test will result in returning visual output of the results, displayed in Figure 5.4.

Moreover, there is a jasmine-jquery[7] extension for the framework that provides jQuery functionality and a API that provides HTML, CSS and JSON functionality. This can be useful when testing a jQuery Mobile-based application.

In general, the execution of tests is performed fast, but these still depend on the JavaScript engine they are being run against. The framework is suitable to be integrated into build automation, continues integration or test runner frameworks.

However, performing this on mobile applications created by Cordova might be difficult in some cases. For example, running the tests through a CI server on Android devices will not work out of the box, since the WebView of the OS does not support remote debugging.

[Kos12]

[7] https://github.com/velesin/jasmine-jquery

Siesta

Siesta is a commercial JavaScript testing framework produced by the company Bryntum[8]. It supports pure JavaScript testing, DOM manipulation tests and the simulation of user interactions, like click, touch events and so. Siesta performs well on the Ext.js framework. Therefore, automatic integration tests can be run on a Sencha Touch web application and through touch events, the whole navigation and interface can be included into the testing process. Two versions are available, a lite version that is free and a standard version. The payed version supports premium support of the integration of cross page testing, Selenium intergraton and the integration of PhatomJS.

PhantomJS is a headless implementation of the WebKit engine and provides native support to interact with the DOM, CSS, JSON, SVG and Canvas. However, it is not a testing frameworks, it can be used in combination with various testing framework to launch tests through test runners. In order to be able to analyze this test, it provides automatically captured pseudo screenshots.

Node.js is a server side implementation of JavaScript that uses the Google V8 JacaScript engine.

Siesta can carry out the tests through the command line without the usage of a browser, in combination with Node.js and PhantomJS. Therefore, it can be integrated into CI or just help to perform the tests automatically.

[Kos12], [Pha13]

5.4.4 medic

The project medic is hosted on GitHub and licensed under the Apache 2.0 License. It provides a CI setup for hybrid applications build with Cordova. It currently supports the mobile operating systems Android, iOS and Blackberry 10. The same project is also used by Cordova[9] itself to establishes Continuous integration performing several tests[10] especially designed for this purpose. This project executes JavaScript tests using Jasmine inside the hybrid app. When all test are performed, their results are obtained by an extra Jasmine test suite adapter. However, medic categorizes all test results performed by all different devices into one single CouchDB file that is used as storage.

[8] http://www.bryntum.com/
[9] http://ci.cordova.io
[10] http://github.com/apache/cordova-mobile-spec

Adobe CouchDB is a simple database that uses JSON for storing its documents. The provided API can be used completely through HTTP.

In addition medic provides a dashboard to represent all test results. In order to be able to use this setup in this way, Node.js is required and the project has to be hosted on a git server. The project integrates the mobile spec suite from Cordova, but after setting everything up a custom spec suite can be used to perform tests written with Jasmine.

[Git13c]

5.5 Testing Xamarin Projects

The current testing framework MonoTouch.NUnitLite used in Xamarin is based on the NUnitLite framework. This is a custom lightweight framework that does not need many resources and can be used for mobile platforms. However, NUnitLite is based upon the NUnit framework. This is an open source framework that provides the functionality to perform unit tests on .NET applications. This means in conclusion that a subset of the features of NUnit and NUnitLite can be used as well.

5.5.1 iOS

In Xamarin.iOS the framework works out of the box and test projects can be created either through Xamarin Studio or Visual Studio. The mentioned testing frameworks can be used to write test cases and perform those through an iOS specific test runner MonoTouch.NUnit.UI.TouchRunner.

5.5.2 Android

At the time of this writing there is no official support from Xamarin for the above described testing frameworks on Android. Whereas an unofficial support is provided by the project Andr.Unit[11]. This has a Mono for Android test runner implemented that can be used in combination with the NUnitLite framework.

5.5.3 Test Cloud

Xamarin announced an automated UI testing service in April 2013, that is currently in a beta phase and will be available to Xamarin users until the end of the year.

The released version is going to include following features:

1. Xamarin App Explorer: Random monkey UI interaction tests can be performed on multiple devices by uploading the application to the service.

[11] https://github.com/spouliot/Andr.Unit

2. Cross-Platform UI Test Scripts: UI interaction tests can be written and the test logic can be used through different platforms.
3. Physical Devices: Xamarin is providing the ability to perform the tests on real non jailbroken or rooted devices. This should help against the device fragmentation.
4. CI Support: The service will be executable through the command line or an API interface. This and provided plugins for popular CI systems should give the option to integrate tests in any build system.

The service will support tests written in C#and their integration in Xamarin Studio and Visual Studio. Moreover, a Ruby version of Calabash will also be supported.

Calabash is an open source project originally created by the company LessPainful that was acquired by Xamarin in April 2013. In addition, it supports Cucumber, which is a BDD testing framework. As a result, it is possible to write automated acceptance tests with Calabash and perform them on native or hybrid iOS or Android applications.

6 Conclusion

The most important factors from the user's point of view are the presentation and the usability of an application. Other factors such as speed, performance and functionality are important as well, but they are very often taken for granted by the user. Applications developed natively in the native programming language take full advantage of the platform and the supported SDK. Since users are used to the look and feel of their devices and expect a pleasing experience, the latter can only made sure by taking full advantage of the functionality provided by the native platform. Android and iOS provide already now an impressive set of possibilities for app developers, and the functionality of the provided SDKs is going to grow even more in the coming years. Moreover, code can be maintained much more easily and updates can be applied quicker to existing products.

In Android there is a large variety of different devices with individual specifications. It would be great if software written for the platform could be optimized for all of them. Android provides all necessary options to do that. Due to this support creating an application directly for the OS brings the look and feel a user is expecting. Android is comprised of multiple components, and it provides the full control of replacing them and integrating the software deep into the operating system. In addition applications can be extended, represent themselves as widgets or perform background work as services. The extension of the functionality can easily be achieved through third party open source or commercial libraries. For the most part, the testing and debugging is fully supported through the platform's provided frameworks. However when doing this with extra third party components the success relies heavily on their provided support and can be problematic in certain cases. Usually, natively developed applications require a high knowledge of platform specific programming skills. The documentation of the platform specific SDKs is extensive and provides basic examples of their usage. The greatest disadvantage however is the great amount of developing time that has to be invested repeatedly for every platform.

Hybrid applications developed with Cordova provide a great opportunity to extend a mobile application with native features. These are however limited to the features supported by Cordova. The developers are able to implement custom native functionality with the investment of additional time. As a result, this requires the knowledge of each platform's development process. Generally without the need of custom native functions, applications can be created with just web development skills. Since web programming has been around for longer then the mobile operating systems, many developers already have the necessary knowledge and do not have to learn new platform specific technology. Moreover, web technologies have been perfected for many years now, and the use of HTML5, CSS3 and JavaScript provides the functionality for creating appealing UI elements. The long-term prospect of Cordova is hard to predict, but since it was transformed into an open source

project it can be assumed that the community will preserve and improve its functionality. As for the web part, this is standardized and will probably have the greatest persistence. Performing tests on pure web applications is supported by multiple frameworks and different approaches. However, the testing of Cordova applications is more limited. It relies heavily on the remote debugging support of the web engine and the operating system itself. As mentioned earlier the UI is one of the most important aspects of an application. Developing time as well as expenses can be reduced by using Cordova. Unfortunately, web or hybrid applications cannot replace native elements and their natural look and feel. Moreover, graphical elements and their animations rely heavily on the used mobile development framework, since the latter provides most of them. Even with the usage of a framework, certain UI interaction can be achieved nevertheless, although with a high amount of manual work.

The cross compilation solution of Xamarin provides a native experience with almost no downfalls. Through the use of modern design patterns such as the MVC or MVVM patterns, a high percentage (60% - 80%) of code sharing of the business logic can be achieved. Since the platform specific SDK with all its graphical elements can be used, the user does not notice the difference between an applications produced with this technique and a native one. Xamarin offers all benefits of both worlds together. However, this does not come for free, and therefore a commercial license needs to be acquired. Additionally developers are required to have the knowledge of the platform's SDK and rethink its usage through C#. Moreover, the documentation has a high quality and a wide range of examples, but developers are limited to it. There is not nearly as much support from the community as there is for native or hybrid applications. Moreover, third party libraries that exist for pure native applications either do not exist or can only be ported to it. This can be very difficult in certain cases and, if possible at all, needs a lot of time. Another disadvantage is the limited support of testing possibilities as of now. Xamarin is soon rolling out its cloud testing framework, but this will be most likely involving extra expenses.

Overall it can be said that there is no perfect solution. A hybrid application definitely provides an app-like experience based on web technology standards without the need of platform specific knowledge. If the appropriate resources are available one should consider developing in the native language, since nothing can replace a native application with all its richness and flexibility. The usage of the cross compilation solution of Xamarin brings the native experience to an application and allows the feasibility to share a high amount of the programming code. As for now the best practice would be to focus on the native side and, depending on the application's complexity, try out the approach of Xamarin.

List of Abbreviations

ACW	Android Callable Wrappers
ADT	Android Development Tools
AES	Advanced Encryption Standard
AJAX	Asynchronous JavaScript and XML
AOT	Ahead-of-time
API	Application Programming Interface
APK	Android Application Package File
ARM	Advanced RISC Machines
AVD	Android Virtual Device
BDD	Behavior-driven development
CI	Continuous integration
CLI	Common Language Infrastructure
CLR	Common Language Runtime
CSS3	Cascading Style Sheets Level 3
CSS	Cascading Style Sheets
DOM	Document Object Model
GPS	Global Positioning System
HTC	High Tech Computer Corporation
HTML5	HyperText Markup Language Version 5
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IL	Intermediate Language
IP	Internet Protocol
JIT	just-in-time compilation
JNI	Java Native Interface
JSONP	JSON with padding
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KVO	Key-Value Observation
LINQ	Language Integrated Query
MCW	Managed Callable Wrappers
MIT	Massachusetts Institute of Technology
MVC	Model View Controller
MVVM	Model View ViewModel
OHA	Open Handset Alliance
OS	Operating System

POM	Project Object Model
REST	Representational State Transfer
SASS	Syntactically Awesome Stylesheets
SDK	Software Development Kit
SQLite	Structured Query Language Lite
SQL	Structured Query Language
SVG	Scalable Vector Graphics
TDD	Test Driven Development
UI	User Interface
URI	Uniform resource identifier
URL	Uniform Resource Locator
VM	Virtual Machine
VM	Virtual Machine
W3C	World Wide Web Consortium
XAML	Extensible Application Markup Language
XML	Extensible Markup Language
YAML	YAML Ain't Markup Language

Bibliography

[Act13] ActionBarSherlock. ActionBarSherlock support library, 2013. Available under http://actionbarsherlock.com/; visited on April 3th 2013.

[And13a] Andrew Trice. Emulating PhoneGap/Cordova Apps in the Desktop Browser, 2013. Available under http://www.tricedesigns.com/2012/07/31/emulating-phonegapcordova-apps-in-the-browser/; visited on May 8th 2013.

[And13b] Android. Android Action Bar, 2013. Available under http://developer.android.com/design/patterns/actionbar.html; visited on April 3th 2013.

[And13c] Android. Android Activity Lifecycle, 2013. Available under http://developer.android.com/reference/android/app/Activity.html; visited on March 31st 2013.

[And13d] Android. Android Broadcast Receiver, 2013. Available under http://developer.android.com/reference/android/content/BroadcastReceiver.html; visited on April 3th 2013.

[And13e] Android. Android Content Provider, 2013. Available under http://developer.android.com/guide/topics/providers/content-providers.html; visited on March 31st 2013.

[And13f] Android. Android Fragments, 2013. Available under http://developer.android.com/guide/components/fragments.html; visited on April 3th 2013.

[And13g] Android. Android Intents and Intent Filters, 2013. Available under http://developer.android.com/guide/components/intents-filters.html; visited on April 3th 2013.

[And13h] Android. Android Layouts, 2013. Available under http://developer.android.com/guide/topics/ui/declaring-layout.html; visited on April 3th 2013.

[And13i] Android. Android Navigation Pattern, 2013. Available under http://developer.android.com/design/patterns/navigation.html; visited on April 3th 2013.

[And13j] Android. Android Security Tips, 2013. Available under http://developer.android.com/training/articles/security-tips.html; visited on April 3th 2013.

[And13k] Android. Android Support Library, 2013. Available under http://developer.android.com/tools/extras/support-library.html; visited on April 3th 2013.

[And13l] Android. Android Testing, 2013. Available under `http://developer.android.com/tools/testing/testing_android.html`; visited on April 3th 2013.

[And13m] Android. monkeyrunner, 2013. Available under `http://developer.android.com/tools/help/monkeyrunner_concepts.html`; visited on April 3th 2013.

[And13n] Android. Overview Android Architecture, 2013. Available under `http://developer.android.com/about/versions/index.html`; visited on January 19th 2013.

[And13o] Android. Overview Android SDK Tools, 2013. Available under `http://developer.android.com/tools/help/index.html`; visited on January 28th 2013.

[And13p] Android. Overview Android SDK Tools, 2013. Available under `http://developer.android.com/guide/topics/manifest/manifest-intro.html`; visited on March 31st 2013.

[And13q] Android. Overview Android Versions Usage, 2013. Available under `http://developer.android.com/about/dashboards/index.html`; visited on January 19th 2013.

[And13r] Android. UI/Application Exerciser Monkey, 2013. Available under `http://developer.android.com/tools/help/monkey.html`; visited on April 3th 2013.

[And13s] Android Maven Plugin. Android Maven Plugin, 2013. Available under `https://code.google.com/p/maven-android-plugin/`; visited on May 10th 2013.

[Apa13] Apache. Documentation, 2013. Available under `http://maven.apache.org/guides/`; visited on May 10th 2013.

[Ayd12] M. Aydin. *Android 4: New Features for Application Development*. Community experience distilled. Packt Pub., 2012.

[BHH+12] S. Benli, A. Habash, A. Herrmann, T. Loftis, and D. Simmonds. A comparative evaluation of unit testing techniques on a mobile platform. In *Information Technology: New Generations (ITNG), 2012 Ninth International Conference on*, pages 263–268, 2012.

[CD12] S. Conder and L. Darcey. *Android Wireless Application Development: Advanced Android*. Developer's Library. ADDISON WESLEY Publishing Company Incorporated, 2012.

[CFGW11] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, MobiSys '11, pages 239–252, New York, NY, USA, 2011. ACM.

[CG08] L. Crispin and J. Gregory. *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley Signature Series (Cohn). Pearson Education, 2008.

[CGK11] C.E. Collins, M.D. Galpin, and M. Käppler. *Android in Practice: Includes Free Ebook*. Running Series. Manning Publications Company, 2011.

[CM13] Alex Yau Christian Murphy. Is a rigorous agile methodology the best development strategy for small scale tech startups? In *Is a Rigorous Agile Methodology the Best Development Strategy for Small Scale Tech Startups?*, 2013.

[Dan13] Dan North. Introducing BDD, 2013. Available under `http://dannorth.net/introducing-bdd/`; visited on May 10th 2013.

[DK] Steffen Dienst and Stefan Kühne. In Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, and Jörg Schneider, editors, *Informatik 2011 - Informatik schafft Communities - Proceedings der 41. GI-Jahrestagung*, page 412. Gesellschaft für Informatik e.V. (GI), Bonner Köllen Verlag, 10.

[DS12] B. Diez and R. Serrano. *Mastering Lob Development for Silverlight 5: A Case Study in Action*. Professional expertise distilled. Packt Publishing, 2012.

[Fur13] A. Furrow. *iOS UICollectionView: The Complete Guide*. Mobile Programming. Pearson Education, 2013.

[Gar13] Gartner. World Wide Smartphone Share, 2013. Available under `http://www.gartner.com`; visited on January 13th 2013.

[gee13] geekswithblogs. MVVM Compared To MVC and MVP, 2013. Available under `http://geekswithblogs.net/dlussier/archive/2009/11/21/136454.aspx`; visited on May 10th 2013.

[Gif12] M. Gifford. *PhoneGap Mobile Application Development Guidebook*. Packt Publishing, Limited, 2012.

[Git13a] Github - AndiDog. Navigation bar for Cordova on iOS, 2013. Available under `https://github.com/AndiDog/phonegap-ios-navigationbar-plugin`; visited on April 26th 2013.

[Git13b] Github - AndiDog. Tab bar for Cordova on Android, 2013. Available under `https://github.com/AndiDog/phonegap-android-actionbarsherlock-tabbar-plugin/tree/master/2.2.0`; visited on April 26th 2013.

[Git13c] GitHub - filmaj. medic, 2013. Available under `http://github.com/filmaj/medic`; visited on May 8th 2013.

[Git13d] Github - jxp. Phonegap desktop, 2013. Available under `https://github.com/jxp/phonegap-desktop`; visited on May 8th 2013.

[Git13e] Github - sgrebnov. jQM theme for Windows Phone, 2013. Available under `https://github.com/sgrebnov/jqmobile-metro-theme`; visited on May 8th 2013.

[Git13f] Github - taitems. iOS-Inspired Theme for jQuery Mobile, 2013. Available under `https://github.com/taitems/iOS-Inspired-jQuery-Mobile-Theme`; visited on May 8th 2013.

[Git13g] Github - taitems. jQM theme Android 4.0 Theme.Holo, 2013. Available under `https://github.com/jjoe64/jquery-mobile-android-theme`; visited on May 8th 2013.

[GP12] R. Ghatol and Y. Patel. *Beginning PhoneGap: Mobile Web Framework for JavaScript and HTML5*. Books for professionals by professionals. Apress, 2012.

[Gro13] Grokking Android. Grokking Android Content Provider, 2013. Available under `http://www.grokkingandroid.com/android-tutorial-writing-your-own-content-provider/`; visited on April 3th 2013.

[Gun12] S. Gunasekera. *Android Apps Security*. Apressus Series. Apress, 2012.

[Int13a] International Data Corporation. Low Cost Products Drive Forecast Increases in the Tablet Market, According to IDC , 2013. Available under `http://www.idc.com/getdoc.jsp?containerId=prUS24002213`; visited on January 13th 2013.

[Int13b] International Data Corporation. Worldwide Mobile Phone Growth Expected to Drop to 1.4Despite Continued Growth Of Smartphones, According to IDC, 2013. Available under `http://www.idc.com/getdoc.jsp?containerId=prUS23818212`; visited on January 13th 2013.

[JQu13] JQuery Mobile. JQuery Mobile, 2013. Available under `http://jquerymobile.com/`; visited on May 8th 2013.

[Kos12] A. Kosmaczewski. *Mobile JavaScript Application Development*. Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012.

[Kos13] A. Kosmaczewski. *Sencha Touch 2 Up and Running*. Oreilly & Associates Incorporated, 2013.

[Lee12] W.M. Lee. *Beginning Android 4 Application Development*. ITPro collection. Wiley, 2012.

[Lun11] A. Lunny. *Phonegap Beginner's Guide*. Packt Publishing, Limited, 2011.

[Mar09] R.C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Robert C. Martin Series. Prentice Hall, 2009.

[MDMB12] Z.R. Mednieks, L. Dornin, G.B. Meike, and M. Bakamura. *Programming Android*. Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012.

[Mei12] R. Meier. *Professional Android 4 Application Development*. ITPro collection. Wiley, 2012.

[Mil11] D.T. Milano. *Android Application Testing Guide*. Community experience distilled. Packt Publishing, Limited, 2011.

[Mon13] Mono. Cross platform, open source .NET development framework, 2013. Available under `http://www.mono-project.com/`; visited on May 10th 2013.

[Nin13] NineOldAndroids. NineOldAndroids animation library, 2013. Available under `http://nineoldandroids.com/`; visited on April 3th 2013.

[Osm12] A. Osmani. *Learning JavaScript Design Patterns*. JavaScript and jQuery developer's guide. O'Reilly Media, Incorporated, 2012.

[Pha13] PhantomJS. Full web stack No browser required, 2013. Available under `http://phantomjs.org/`; visited on May 8th 2013.

[Pho13a] Phonegap. API REferences, 2013. Available under `http://docs.phonegap.com/`; visited on April 26th 2013.

[Pho13b] Phonegap. Defining Your Cordova Plugin As A Cordova Module, 2013. Available under `https://github.com/phonegap/phonegap-plugins/wiki/Defining-Your-Cordova-Plugin-As-A-Cordova-Module`; visited on April 26th 2013.

[Pho13c] Phonegap. Introducing Cordova-JS, 2013. Available under `http://phonegap.com/2012/03/21/introducing-cordova-js/`; visited on April 26th 2013.

[Pho13d] Phonegap. Supported Features, 2013. Available under `http://phonegap.com/about/feature/`; visited on April 26th 2013.

[RLMM09] R. Rogers, J. Lombardo, Z. Mednieks, and G.B. Meike. *Android Application Development*. Oreilly Series. O'Reilly Media, 2009.

[Sad11] Ben Sadeh. A Study on the Evaluation of Unit Testing for Android Systems. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 4(1), 2011.

[Sim13] Simon MacDonald. So You Wanna Write a PhoneGap 2.0.0 Android Plugin?, 2013. Available under `http://simonmacdonald.blogspot.co.at/2012/08/so-you-wanna-write-phonegap-200-android.html`; visited on April 26th 2013.

[SQL13] SQLCipher. SQLCipher, 2013. Available under `http://sqlcipher.net/`; visited on April 26th 2013.

[ST10] J. Steele and N. To. *The Android Developer's Cookbook: Building Applications with the Android SDK*. Developer's Library. Pearson Education, 2010.

[The13] Thejaswi Puthraya. Dissecting Phonegap's architecture, 2013. Available under `http://agiliq.com/blog/2012/09/dissecting-phonegaps-architecture/`; visited on April 26th 2013.

[War12] J.M. Wargo. *PhoneGap Essentials: Building Cross-platform Mobile Apps*. ADDISON WESLEY Publishing Company Incorporated, 2012.

[Xam13] Xamarinj. Documentation, 2013. Available under `http://docs.xamarin.com/`; visited on May 10th 2013.

[Yag13] K. Yaghmour. *Embedded Android: Porting, Extending, and Customizing*. Oreilly and Associate Series. O'Reilly Media, 2013.

[ZMEQ11] J.M. Zain, W.M.W. Mohd, and E. El-Qawasmeh. *Software Engineering and Computer Systems, Part III: Second International Conference, ICSECS 2011, Kuantan, Pahang, Malaysia, June 27-29, 2011, Proceedings*. Communications in Computer and Information Science. Springer, 2011.